Three-dimensional Hydraulic Conductivity Upscaling in Groundwater Modelling

Master Thesis submitted by
Haiyan Zhou

Advisors:
J. Jaime Gómez-Hernández

January 2009

Three-dimensional Hydraulic Conductivity Upscaling in Groundwater Modelling

Master Thesis submitted by
Haiyan Zhou

Advisors:
J. Jaime Gómez-Hernández

Master Program:
Ingeniería Hidráulica y Medio Ambiente

Department:
Ingeniería Hidráulica y Medio Ambiente

Instituto de Ingeniería del Agua y Medio Ambiente
Universidad Politécnica de Valencia

January 2009

Acknowledgemetns

Abstract

Resumen

Resum

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and background

1.2 Thesis structure

The three-dimensional upscaling numerical code is presented in Appendix A and an example for parameter file is shown in Appendix B. Appendix C explained the format for input conductivity data file and output.

submitted to Computer & Geoscience

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 program. Flow rate and hydraulic gradient are selected to relate the fine scale and the coarse scale, and in turn, guarantee the flow reproduction and head equality. Block conductivity not only at the centre of the block but also at the interface are available. This "interblock conductivity" algorithm eliminates some artificial errors in coarse scale groundwater modelling resulting from the averaging between blocks. This program presents high accuracy in two synthetic examples: isotropic and anisotropic conductivity fields, especially when the impact of cells around the target block are considered. A further comparison of the numerical block conductivity with analytic solution enhanced the validity of the proposed algorithm.

2.1 Introduction

Numerical simulation of groundwater flow and solute transport is nowadays widely employed in predicting available groundwater resources and fate of pol-

lution plumes, which consequently provids information for decision-making. Hydraulic conductivity no doubt plays a dominant role in the simulation. However, hydraulic conductivities observed in the field are conventionally deviated from the object numerical simulation scales and thus they are rare to be used directly in the numerical model. *Dagan* (1986) introduced three levels to describe this scale property: the laboratory, the local, and the regional scale. The technique used to transform hydraulic conductivity from measured scale to the target model scale is termed "upscaling" which has been an ongoing subject of interest in groundwater study for a long time. Put it another way, scale deviation of hydraulic conductivity between observation and modelling serves as the main cause of upscaling. Computation intensity can be another reason why upscaling is so necessary, despite the rapid development of computer techniques. Many excellent reviews about upsclaing are available (say *Wen and Gómez-Hernández*, 1996; *Renard and de Marsily,* 1997; *Farmer*, 2002; *Sánchez-Vila et al.*, 2006), which presented general overviews of upscling approaches in different stages.

Concepts delineating the coarse scale conductivity contain effective hydraulic conductivity, equivalent conductivity, interpreted conductivity, upscaled conductivity, block conductivity, homogenized conductivity, etc., of which effective conductivity and equivalent conductivity are more representative. Generally speaking, effective conductivity is defined from the stochastic point of view, *i.e.,* $E(u) = -K_{ef}E[grad(h)]$ (*Matheron*, 1967), where u is flow rate, *E(.)* represents the expectation and *grad(h)* is the hydraulic head gradient. On the other hand, equivalent conductivity is an average of the spatial variability of hydraulic conductivity within a block satisfying certain criteria such as flow conservation, head equality and conservation of energy dissipation (*Renard and de Marsily*, 1997; *Vermeulen et al.*, 2006). Block conductivity acts as an equivalent conductivity practically in a finite-size block, and relates the spatial average of flow to the spatial average of the gradient (*Gómez-Hernández*, 1991; *Renard and de Marsily,* 1997). Further, the block conductivity tends to coincide with effective conductivity if the block over which the upscaling is conducted is large enough, which has been demonstrated by several authors (e.g., *Gómez-Hernández*, 1991; *Renard and de Marsily*, 1997). This paper mainly applies the terms "block conductivity" or "interblock conductivity" depending on the actual location of the conductivity (Fig. 2.1).

Some conclusions have been drawn about the property of upscaled conductivities. In one-dimensional flow, harmonic mean of hydraulic conductivity is employed as the equivalent conductivity between the blocks (*Freeze and Cherry*, 1978). In two-dimensional flow, geometric mean behaves well in isotropic media under such constraints as infinite block with log-normal conductivity distribution (*Matheron*, 1967; *Gómez-Hernández and Wen*, 1994), and as for the layered aquifer, the effective conductivity ranges from harmonic

Figure 2.1. Schematic diagram of data read for calculating the block conductivity in the center (left, "ib"=1) and interbloack conductivity at the interface (right, "ib"=3). Two Outer skins and one inner skin are indicated with shade in the diagram.

mean (perpendicular to the layer direction) to arithmetic mean (parallel to the layer direction) (*Matheron*, 1967). In most other aquifers, we have to resort to numerical approximations since the exact solution does not exist. In three-dimensional or multi-dimensional cases, however, much effort is devoted to analytic solutions such as *Gutjahr et al.*, (1978), *De Wit* (1995) and *Green and Paterson* (2007). More detailed analytic approaches in three-dimensional flow is available in section 3.2. Nonetheless, the drawback of analytic method is that their application is quite limited due to many assumptions in distribution of conductivity, property of the media and flow pattern (*Renard and de Marsily*, 1997). Another shortcoming of analytic method is well known that it can only yield scaler or diagonal tensor aligned with the block sides rather than full tensor. These disadvantages obviously prevent the analytic solutions from reflecting the property of aquifers (e.g. cross-bedded sediments, *Bierkens et al.*, 1994).

As for three-dimensional numerical upscaling techniques, Journel et al., (1986) proposed a power average law to describe the block conductivity when they investigated a three-dimensional sand-shale formation ($K_V = (\frac{1}{V}\int_V k^p dV)^{1/p}$). *Holden and Lia* (1992) were the first, to our knowledge, to present the full tensor upscaling estimator albeit quantitative error assessment was absent besides the comparison between full tensor and diagonal tensor. A procedure based on the concept of "border region" which was defined as the blocks around the target block was successfully proposed to improve the permeability upscaling by *Wen et al.*, (2003).

The objective of this paper is to extend the two-dimensional hydraulic conductivity full tensor upscaling technique to three-dimension, in which block

conductivity at the center and interblock conductivity at the interface are both available. The general structure of this paper is as follows: the principle of upscaling technique employed in this paper is explained in section 2. Section 3 is devoted to synthetic examples to assess the performance of proposed program. Finally, several conclusions are summarized in section 4.

2.2 Program algorithm and implementation

2.2.1 Upscaling methodology

Block conductivity ($\vec{K}_V$) is defined by *Rubin and Gómez-Hernández* (1990) as

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

where $\vec{q}$ and $\vec{\nabla h}$ are the specific discharge and piezometric head gradient over a block of volume V, respectively, solution of the flow problem within the heterogeneous block. The definition can be interpreted as Darcy's law at block scale $\bar{q} = -\vec{K}_V\bar{\nabla h}$, where the bar denotes spatial average. At the same time, $\vec{q}$ and $\vec{\nabla h}$ are also related by Darcy's law but at cell scale $\vec{q} = -\vec{k}\vec{\nabla h}$. Here we use "block" and "cell" to denote the basic elements in coarse scale and fine scale.

$\vec{K}_V$ is a three-dimensional full tensor described as below.

$$\begin{pmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{pmatrix}$$

Traditionally, diagonal tensor is acquired in hydraulic conductivity upscaling. However, many natural geologic formations, e.g., cross-bedded sediments (*Bierkens et al.,* 1994), in deed challenge the validation of diagonal tensor and prompt the consideration of full tensor, which has the capacity of accounting for the conductivity whose principal components are not aligned with the block sides. Further more, $\vec{K}_V$ is assumed symmetric and positive definite (*Bear*, 1972 and *Farmer*, 2002), that is, $K_{xy} = K_{yx}, K_{xz} = K_{zx}, K_{yz} = K_{zy}$ and $K_{xx}, K_{yy}, K_{zz} > 0, K_{xx}K_{yy} > K_{xy}^2, K_{xx}K_{yy}K_{zz} + 2K_{xy}K_{xz}K_{yz} - K_{xx}K_{yz}^2 - K_{yy}K_{xz}^2 - K_{zz}K_{xy}^2 > 0$.

It is worthwhile to note that $\vec{K}_V$ is a function of conductivity and boundary condition, which is combined in solving the flow problem within each block. Hence, the design of boundary condition is of great significance. Conventionally, permeameter type boundary condition (two sides constant head while the others no-flow) is utilized to attain diagonal tensor while uniform boundary (prescribed head linearly varied around the block) or periodic conditions

is required when full tensor is preferred (*Pickup et al.*, 1994; *Renard and de Marsily*, 1997; *Wen et al.,* 2003). The uniform boundary condition is employed in the proposed procedure.

The second feature of this method is related with the boundary scheme of each block. Blocks within a domain are not isolated each other and rather it is reasonable to consider the influence of the cells around the object block, which inspires the idea of "border ring" or "skin". *Gómez-Hernández* (1991) elaborated the necessity of considering conductivities neighboring the target block. The "skin" here is defined as cells around the object block, in other words, one skin along the row direction means two more columns outsides the target block (Fig. 2.1).

Last but most important point is that either block conductivity at the center or interblock conductivity at the interface is available with the code. This interblock alternative protects the solution of flow problem at coarse scale from certain artificial errors introduced during the calculation of conductance at the interface. *Li et al.,* (2008) demonstrated that interblock conductivity yielded more accurate flow simulation than block centered flow simulator in the case of full tensor.

2.2.2 Implementation details

Figure 2.2 is the flow chart of proposed program, which can be briefly summarized as several steps:

(1) Parameters required in the program are read, including the information about cells at the fine scale and blocks at the coarse scale, number of skins and directions, hydraulic gradient and boundary conditions in each direction.

(2) Value of "ib" combined with "itt" determines whether the block conductivity at the center (ib=1) or interblock conductivity at the interface (ib=3) is calculated, which subsequently decides the data to be read are within one block or interblock (Fig. 2.1). Cells acting as skin are also read together with the object block.

(3) For each block/interblock, harmonic mean is employed to obtain the conductance based on the fine scale hydraulic conductivities. Then the initial head and boundary conditions are prepared for the immediate flow solver. Prescribed head boundary conditions are applied here and the head varies linearly around the sides of the block.

(4) We assume an incompressible, single phase, steady-state groundwater modeling in each block. Preconditioned conjugate-gradient 2 (PCG2) subroutine is amended according to *Hill.* (1990) to solve the flow equation in each direction, normally four directions in 2D and nine in 3D to account for different flow scenarios. From another point of view, it is necessary to solve the flow equation in more than one direction to overcome the problem of "inde-

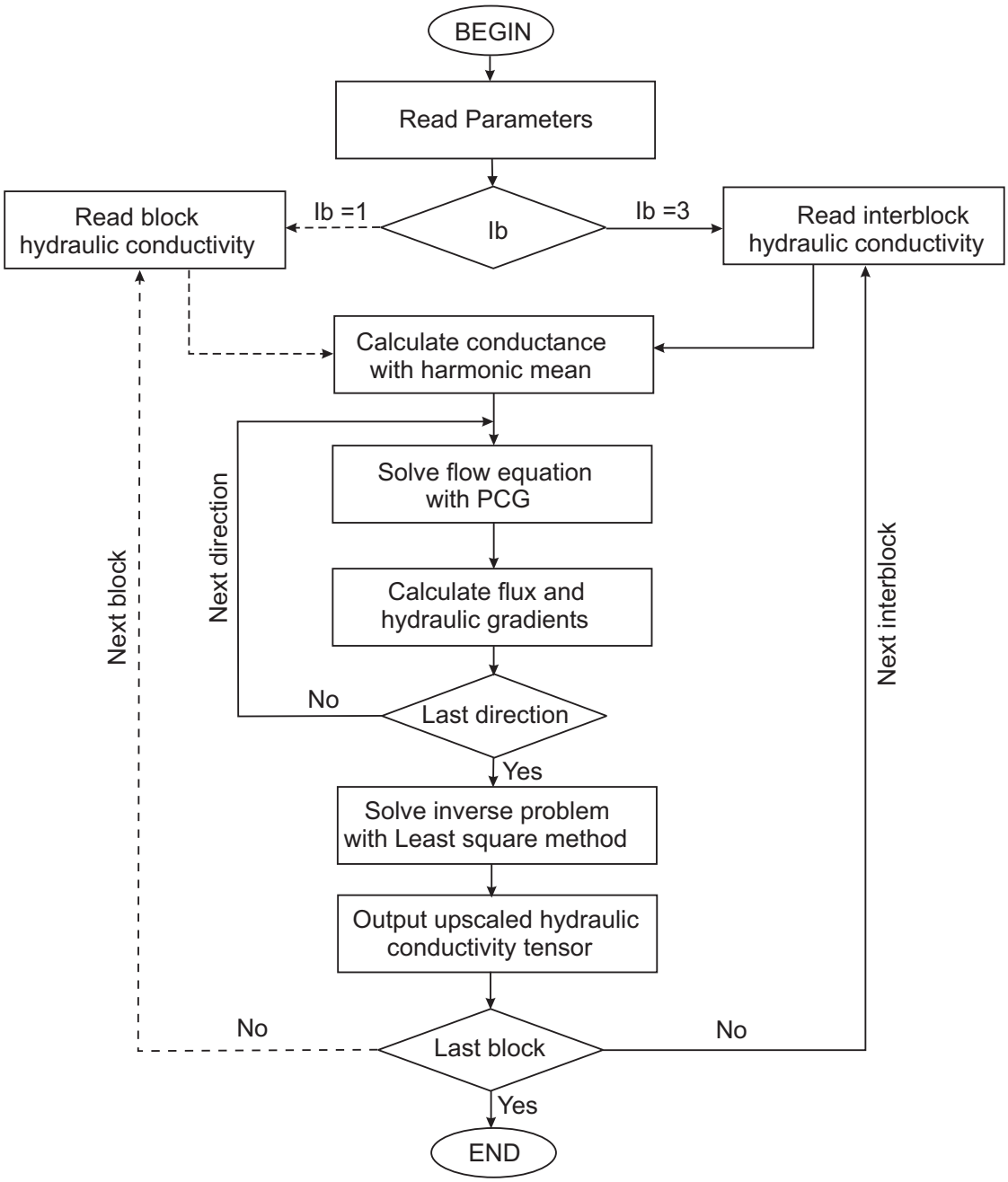

Figure 2.2. Flowchart of the algorithm

terminacy" as described by *White and Horne* (1987). In other words, there are three unknowns in 2D case and six in 3D case ($\vec{K}_V$ being symmetric as we have mentioned before), which follows that the flow equation needs solving at least in two directions to guarantee a unique solution of $\vec{K}_V$. Take 3D flow as an example. In each direction within one block, solving the flow equation in fine scale results:

$$\begin{pmatrix} \bar{q}_x \\ \bar{q}_y \\ \bar{q}_z \end{pmatrix} = - \begin{pmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{xy} & K_{yy} & K_{yz} \\ K_{xz} & K_{yz} & K_{zz} \end{pmatrix} \begin{pmatrix} \nabla \bar{h}_x \\ \nabla \bar{h}_y \\ \nabla \bar{h}_z \end{pmatrix} \tag{2.2}$$

where K_{xx}, K_{xy}, K_{xz}, K_{yy}, K_{yz}, K_{zz} are components of unknown block conductivity $\vec{K}_V$, $\bar{q}_x$, $\bar{q}_y$, $\bar{q}_z$ and $\nabla \bar{h}_x$, $\nabla \bar{h}_y$, $\nabla \bar{h}_z$ are the elements of specific discharge $\vec{q}$ and gradient $\nabla \vec{h}$, respectively, which can be calculated as follows:

$$\begin{aligned}
\bar{q}_x(K, I, J + \frac{1}{2}) &= \frac{1}{n_c} \sum q(k, i, j + \frac{1}{2}) \\
\nabla \bar{h}_x(K, I, J + \frac{1}{2}) &= \frac{1}{n_c} \sum h(k, i, j + 1) - h(k, i, j) \\
\bar{q}_y(K, I + \frac{1}{2}, J) &= \frac{1}{n_c} \sum q(k, i + \frac{1}{2}, j) \\
\nabla \bar{h}_y(K, I + \frac{1}{2}, J) &= \frac{1}{n_c} \sum h(k, i + 1, j) - h(k, i, j) \\
\bar{q}_z(K + \frac{1}{2}, I, J) &= \frac{1}{n_c} \sum q(k + \frac{1}{2}, i, j) \\
\nabla \bar{h}_z(K + \frac{1}{2}, I, J) &= \frac{1}{n_c} \sum h(k + 1, i, j) - h(k, i, j)
\end{aligned} \tag{2.3}$$

where capital letters K, I, J denote coarse scale and the lowercase k, i, j mean fine scale, n_c refers to the number of cells within one block, $q(k, i, j + \frac{1}{2})$ stands for the flow rate through the block interface between (k, i, j) and $(k, i, j + 1)$ calculated by Darcy's law, and $h(k, i, j)$ means the piezometric head.

(5) Once solving the flow problem in all nine directions and gaining the elements of $\vec{q}$ and $\nabla \vec{h}$, we can populate the coefficients matrix, and accordingly

configure the linear equation system for solving $\vec{K}_V$ as follows:

$$-\begin{pmatrix} \nabla\bar{h}_{x1} & \nabla\bar{h}_{y1} & \nabla\bar{h}_{z1} & 0 & 0 & 0 \\ 0 & \nabla\bar{h}_{x1} & 0 & \nabla\bar{h}_{y1} & \nabla\bar{h}_{z1} & 0 \\ 0 & 0 & \nabla\bar{h}_{x1} & 0 & \nabla\bar{h}_{y1} & \nabla\bar{h}_{z1} \\ \nabla\bar{h}_{x2} & \nabla\bar{h}_{y2} & \nabla\bar{h}_{z2} & 0 & 0 & 0 \\ 0 & \nabla\bar{h}_{x2} & 0 & \nabla\bar{h}_{y2} & \nabla\bar{h}_{z2} & 0 \\ 0 & 0 & \nabla\bar{h}_{x2} & 0 & \nabla\bar{h}_{y2} & \nabla\bar{h}_{z2} \\ \cdots & & & & & \end{pmatrix}_{27\times 6} \begin{pmatrix} K_{xx} \\ K_{xy} \\ K_{xz} \\ K_{yy} \\ K_{yz} \\ K_{zz} \end{pmatrix}_{6\times 1} = \begin{pmatrix} \bar{q}_{x1} \\ \bar{q}_{y1} \\ \bar{q}_{z1} \\ \bar{q}_{x2} \\ \bar{q}_{y2} \\ \bar{q}_{z2} \\ \cdots \end{pmatrix}_{27\times 1} \tag{2.4}$$

the subscript "1, 2..." indicates the number of directions. Solving flow equation in nine directions within one block assures far more sufficient equations than needed to obtain $\vec{K}_V$, which causes another problem "overdetermination", since a linear system consisting of twenty-seven equations is designed for only six unknowns (Eq. 2.4).

(6) Standard least squares procedure is employed to solve the inverse problem (*Press et al.*, 1992). Block conductivities are handled sequentially until the last one. Nonetheless, if we aim at the interblock conductivity, two more cycles are needed simply because the interblock conductivity involves three parts, *i.e.*, interblock between columns, rows and layers, which are demanded in the flow simulator dealing with full tensor (*Li et al.,* 2008). Accordingly, the procedure inevitably results in more computation overload in the process of upscaling.

2.3 Synthetic experiments

2.3.1 Experiment description

To demonstrate the accuracy of the program, two synthetic experiments are carried out in this section. The first hydraulic conductivity field is of isotropy and the second of anisotropy under the assumption of multi-Gaussian. The experiments proceed as follows:

(1) Generate the stochastic conductivity field with Sequential Gaussian Simulation (SGSIM in GSLIB. *Deutsch and Journel*, 1998). The computational domain is discretized into 100 columns, 150 rows and 50 layers with cell size $\triangle x = \triangle y = \triangle z = 1m$ for the fine scale. The conductivities in both

examples follow standard log-normal distribution, *i.e.*, $lnK \sim N(0,1)$ and the conductivity field can be described normally by exponential variogram model. Fig. 2.3 shows the isotropic conductivity field used in the first experiment, where the range is set as 20m in all directions. In the anisotropic case the ranges in the greatest, medium and least continuity directions are 30m (30° deviation from Y axis, *i.e.*, α=30° in SGSIM), 20m (β=45°) and 5m (θ=0°), respectively (Fig. 2.4).

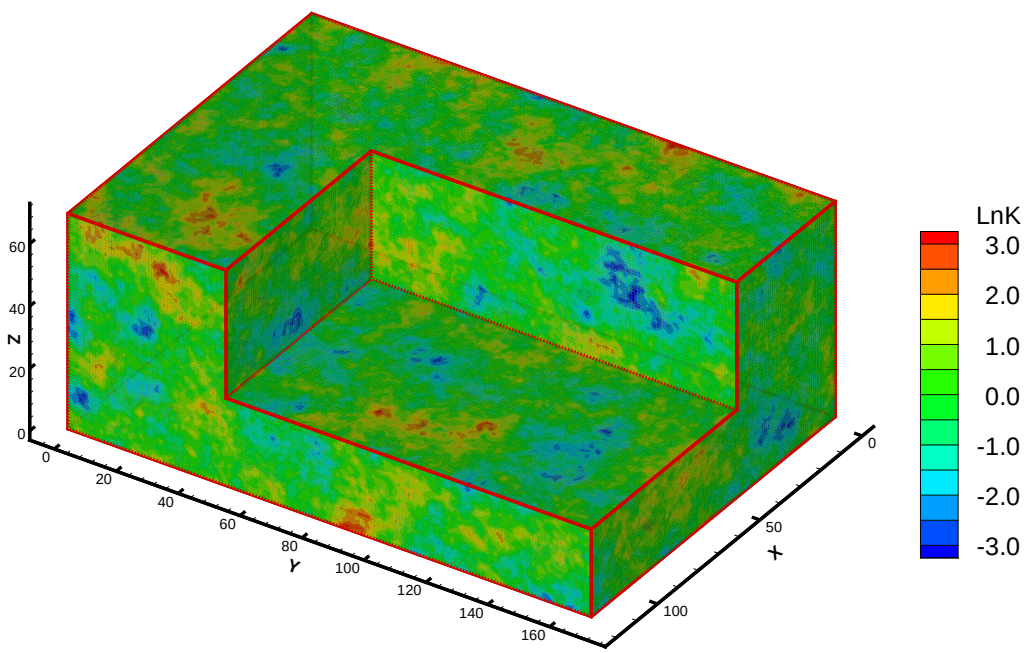

Figure 2.3. The hypothetic isotropic conductivity field

(2) Solve the flow system at fine scale. The aquifers in both synthetic examples are assumed to be confined and are characterized by prescribed head boundary conditions. Considering the sensitivity of upscaling to the flow direction (*Vermeulen et al.*, 2006) and the generality of the aquifers, we simulate a diagonal flow from the lower left corner of the top layer to the up right corner of the bottom layer. MODFLOW2000 (*Harbaugh et al.*, 2000) is chosen to solve the flow system. Then the average specific discharges across each

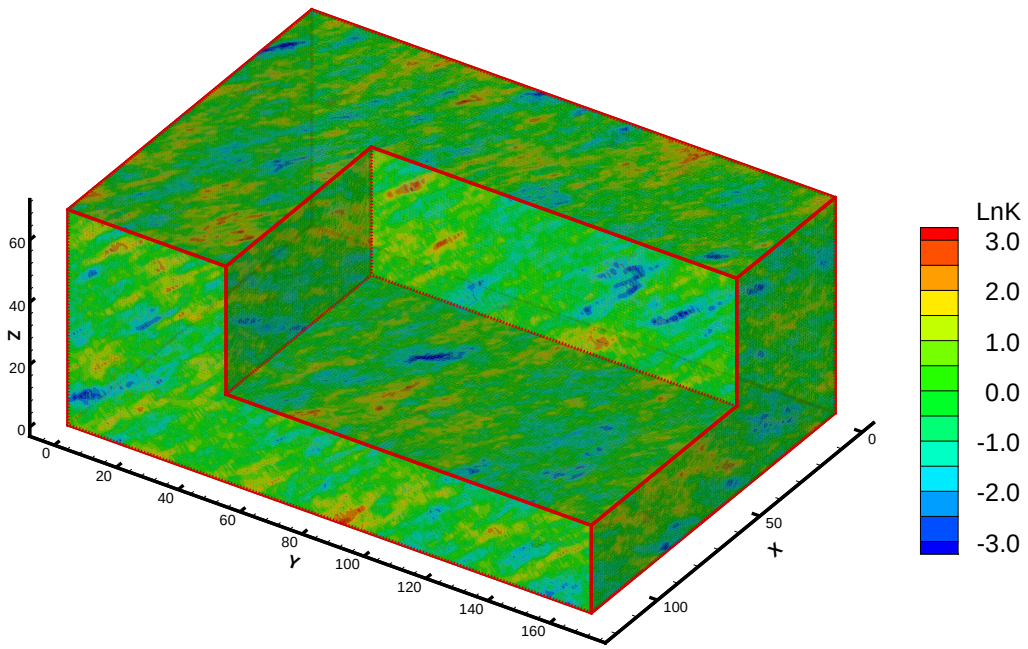

Figure 2.4. The hypothetic anisotropic conductivity field

block interface, serving as the reference, are calculated based on the solution of MODFLOW.

(3) Compute the interblock/block conductivity. The fine scale conductivity generated in step (1) is upscaled to the coarse scale of 10 columns, 15 rows and 5 layers. To eliminate the influence of artificial errors, interblock conductivities at the interface are preferred. In addition, four different skin scenarios are considered when the interblock conductivity is calculated, that is, 0 skin, 2 skins, 5 skins and 10 skins, in order to illuminate the effect of skins. Meanwhile, block conductivity at the center is computed to facilitate the comparison with existing analytic solutions.

(4) Solve flow problem at coarse scale with the interblock conductivity from step (3) and obtain the corresponding specific discharges, which will be compared with the reference obtained in step (2). Another flow simulator dealing with interblock conductivity full tensor is adopted to solve the flow system. Refer to *Li et al.,* (2008) for the algorithm details about that simulator.

(5) Compare the specific discharge across the block interface of coarse scale with those of fine scale to shed light on the quality of this program. To reduce the influence of upscaling boundary conditions, blocks near the boundary of the domain are not taken into account in comparison. Here the specific discharge is selected to examine the performance of the upscaling program since the fundamental purpose of upscaling in groundwater flow modelling is to reproduce the flux which plays a key role in solute transport modelling. "Root Mean Square Error" (*RMSE* for short) is employed to quantify the amount by which the estimated value deviates from the true value.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(CF_i - FF_i)^2} \tag{2.5}$$

where FC_i and FF_i are the ith interblock specific discharge at coarse scale and fine scale, respectively.

2.3.2 Comparison with analytic solution

To justify the validity of proposed code, a comparison of the block conductivity to the well-founded analytic solutions is provided. *Matheron* (1967) derived a conjecture about effective conductivity (K_{ef}) in D dimensional, isotropic, stationary porous media with log-normal conductivity distribution:

$$K_{ef} = K_g exp\,[\sigma^2_{lnk}(\frac{1}{2} - \frac{1}{D})] \tag{2.6}$$

where σ^2_{lnK} stands for the variance of lnK, and K_g refers to geometric mean. *Nætinger* (1994) rewrote equation 2.6 for isotropic log-normal media in this

form:

$$K_{ef} = \langle K^{1-\frac{2}{D}} \rangle^{[1-\frac{2}{D}]^{-1}} \tag{2.7}$$

Desbarats (1992) arrived at the same conclusion by deducing that the exponent equals 1/3 in three dimensions and calibrated it with a numerical experiment. The complex structure of anisotropic media impeded the upscaling technique involved. *Dagan* (1989) provided solutions for axisymmetric anisotropic filed and *Ababou* (1990) suggested an empirical formula:

$$K_{ef,\,i} = \langle K^{p_i} \rangle^{\frac{1}{p_i}}, \quad p_i = 1 - \frac{2\lambda_H}{n\lambda_i}, \quad \frac{1}{\lambda_H} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\lambda_i} \tag{2.8}$$

where λ_i refers to the correlation length in the ith direction and n is dimension. This formula coincided with equation 2.6 in the isotropic case.

Effective conductivity formulas 2.7 and 2.8 are adopted as the analytic solution reference for three-dimensional isotropic and anisotropic hydraulic field. The correlation diagrams between analytic solution and numerical solution are plotted in Fig. 2.5 and 2.6.

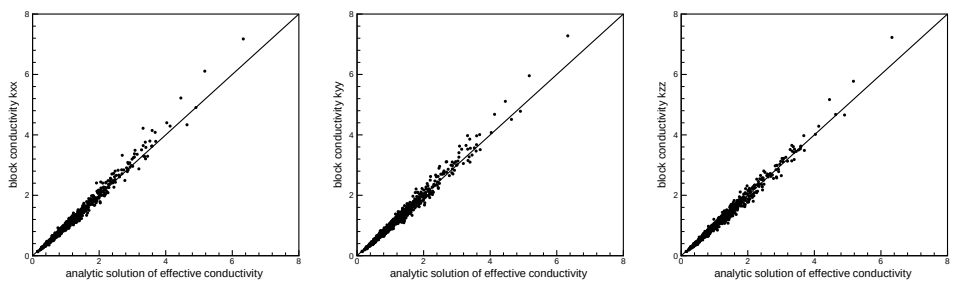

Figure 2.5. Comparison between analytic solution and numerical solution (with 5 skins) in the isotropic field

Note that an angle rotation of the conductivity tensor in the anisotropic field is conducted before the comparison is carried out so that the principal components of the block conductivity are aligned with the block sides. A pretty good match is displayed for the isotropic case but for the anisotropic case the block conductivity exhibited departure from the analytic solution, that is, K_{xx}, K_{yy} and K_{zz} are overestimated in the analytic approach. The overestimation can be attributed to the deviation of continuity direction from the block sides (Fig. 2.4), while the feature of direction dependence is not taken into account in the analytic formula. If considering another anisotropic conductivity field where the greatest continuity direction is parallel to the block sides, we found that the analytic procedure performed well (Fig. 2.7).

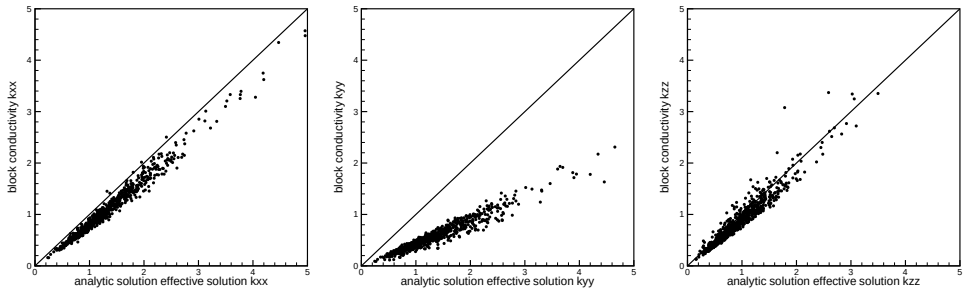

Figure 2.6. Comparison between analytic solution and numerical solution (with 5 skins) in the anisotropic field

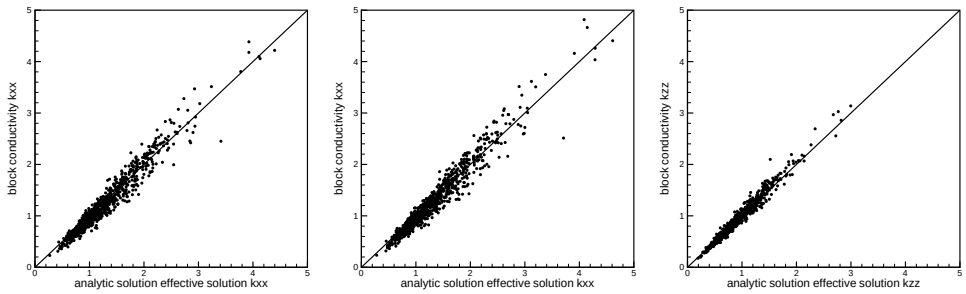

Figure 2.7. Comparison between analytic solution and numerical solution (with 5 skins) in the anisotropic field where the continuity direction coincides with the block sides

Figs. 2.5 - 2.7 indicate that the proposed program are equivalent with analytic approach in isotropic field and anisotropic cases where the continuity direction coincides with the block sides. Moreover, the numerical program is more reliable when the diagonal components are not aligned with the original coordinates. This correlation further reinforced the conjecture between effective conductivity and block conductivity (*Gómez-Hernández*, 1991; *Renard and de Marsily*, 1997). To further quantify the performance of the program, flow rate reproduction at coarse scale is examined.

2.3.3 Flow rate reproduction verification

Isotropic conductivity field

The specific discharge comparison between coarse scale and fine scale is presented in Fig. 2.8. It can be found readily from Fig. 2.8 that the specific discharge is overestimated when there is no skin or only a few skins although the two fluxes are correlated well. With the increase of skins, the specific discharges at coarse scale approximate those of fine scale better. The *RMSE* of specific discharge under different skins are listed in Table 2.1. *RMSE*s of $\bar{q}_x$, $\bar{q}_y$, $\bar{q}_z$ decline with the growth of skins, confirming the fact that computation of block conductivity should account for the conductivity around. It is worthy noting that the *RMSE*s under 5 skins are almost the same with that under 10 skins, but the computation amount of the latter is far more than the former, which means we have to balance between the upscaling quality and computation cost. In this example, 5 skins are sufficient to capture the properties of the conductivity field, which brings a conjecture about size of skins, that is, skins as large as half of the block size could be enough.

Anisotropic conductivity field

Fig. 2.9 is used to illustrate the performance of the upscaling program in anisotropic conductivity field. Compared with the isotropic example, the upscaled conductivity in the anisotropic field is more sensitive to skins and the estimation is noticeably improved when skins are employed. *RMSE*s of $\bar{q}_x$, $\bar{q}_y$, $\bar{q}_z$ in this case are presented in Table 2.2. We found that the *RMSE*s in anisotropic filed (Table 2.2) are comparable with that in isotropic field (Table 2.1), which indicates that the proposed upscaling technique is not influenced seriously by the spatial continuity changes. Further, block conductivity with 5 skins in anisotropic case also yields acceptable results, which enhanced to some extent our inference about the size of skins.

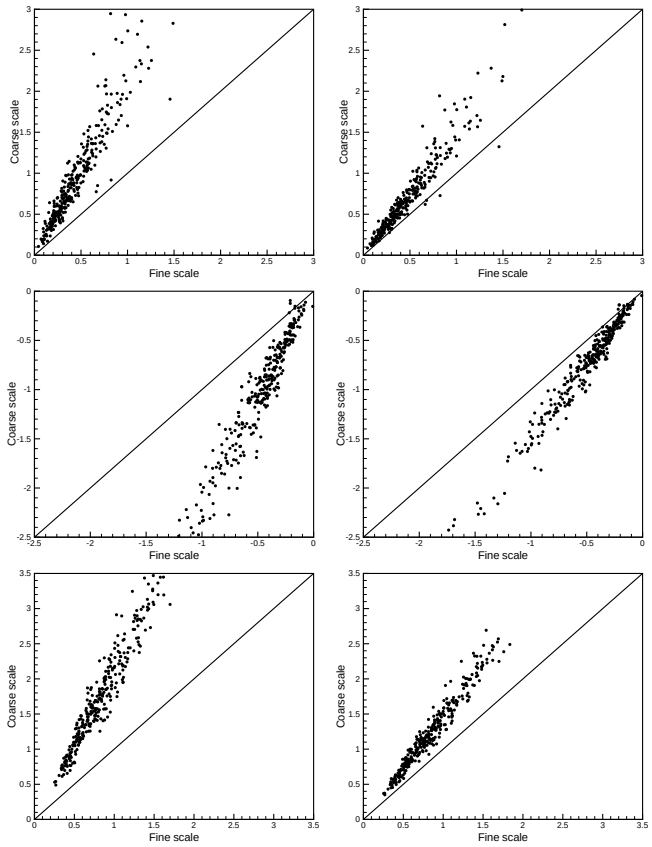

Figure 2.8. Specific discharge comparison between fine scale and coarse scale with different skins (0, 2 and 10 skins from left to right and $\bar{q}_x$, $\bar{q}_y$ and $\bar{q}_z$ from top to bottom) in an isotropic conductivity aquifer

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

	0 Skin	2 Skins	5 Skins	10 Skins
$\bar{q}_x$	0.76	0.34	0.08	0.07
$\bar{q}_y$	0.74	0.31	0.05	0.05
$\bar{q}_z$	1.08	0.46	0.06	0.06

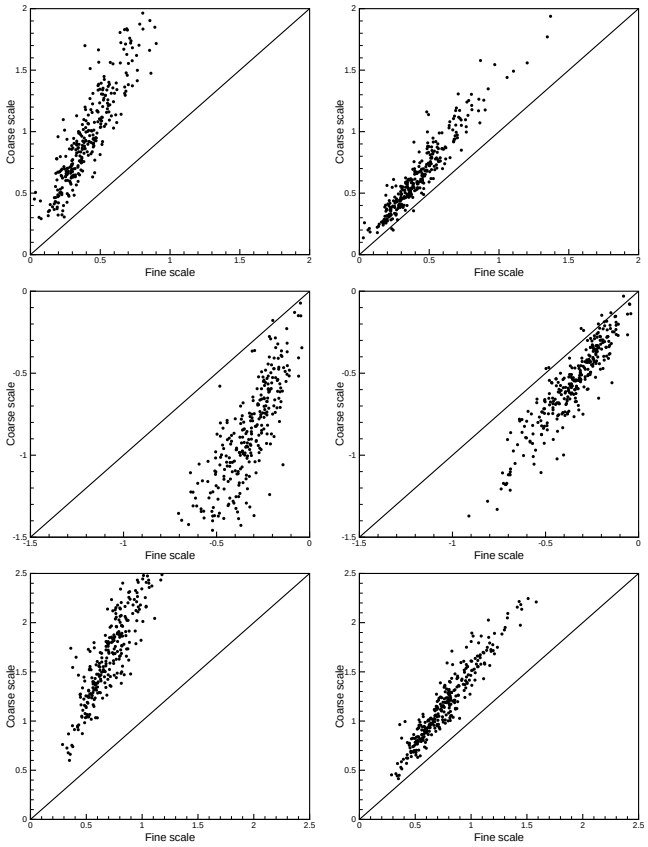

Figure 2.9. Specific discharge comparison between fine scale and coarse scale with different skins (0, 2 and 10 skins from left to right and $\bar{q}_x$, $\bar{q}_y$ and $\bar{q}_z$ from top to bottom) in an anisotropic conductivity aquifer

Table 2.2. Root Mean Square Error *(RMSE)* of the coarse scale specific discharge in the anisotropic case

	0 Skin	2 Skins	5 Skins	10 Skins
$\bar{q}_x$	0.67	0.26	0.05	0.05
$\bar{q}_y$	0.67	0.24	0.05	0.04
$\bar{q}_z$	1.14	0.44	0.06	0.05

2.4 Conclusions

A three-dimensional hydraulic conductivity full tensor upscaling program was proposed in this paper to cope with the problems of scale deviation in heterogeneous media and computation burden in groundwater modelling. Actually, we extended the work of *Gómez-Hernández* (1991) in two dimension. The critical point of this program resided in the capacity of obtaining not only block conductivity at the center of blocks but also interblock conductivity at the interface. This algorithm prevented to some extent the artificial errors by providing the interblock conductivity directly instead of the averaging between blocks during solving flow equations with full tensor at coarse scale. Even if the averaging (normally harmonic mean) was acceptable, it was not clear how the off diagonal component should be averaged. Employment of interblock conductivity full tensor leaded to the development of another flow simulator different from the traditional block-centered approach. This work has been accomplished by *Li et al.,* (2008).

Two synthetic hydraulic conductivity fields were generated to assess the performance of the upscaling program. It was found that the program was of high accuracy in both hydraulic conductivity of isotropy and anisotropy, especially when skins were considered. Stratified media may be examined in the future work. Further, the numerical solution of block conductivity was compared with the analytic model, which exhibited fine agreement.

It is important to note that the employment of skins and interblock conductivity scheme augmented the computation. This can be the disadvantage of this program. So it is reasonable to select appropriate skins and seek some optimization algorithm. For the first problem, skins as large as half of blocks are recommended; and for the alternative, *Holden and Lia*'s work (1992) about convergence speed in their upscaling procedure can provide some clues.

3

Upscaling program performance in layered media

To qualify further the performance of the proposed algorithm, a bimodel consisting of sand and shale is fabricated.

3.1 Conductivity field generation

Discretization of the domain is the same with that in Chapter 2. However, the aquifer in this case is composed of sand and shale cross-bedded, that is, isotropic sand media with range of 20m superposed by shale with ranges of 35m (α=45°), 15m (β=0) and 5m (γ=0), respectively (Fig. 3.1). The shale layers are set horizontal to represent the natural sedimentation in aquifers. The sand and shale conductivity fields are generated independently with SGSIM and SISIM in GSLIB (*Deutsch and Journel*, 1998), where the conductivity in the sand field follows log-normal standard distribution, in another word, it is another realization of the isotropic fields in Section 2.3.1(Fig. 2.3); and the conductivity in the shale case can be described as

$$I(x) = \begin{cases} 1, & \text{if x is shale;} \quad \text{(20\% of the aquifer is occupied by shale)} \\ 0, & \text{if x is sand.} \end{cases} \tag{3.1}$$

The variogram of the shale conductivity field is presented in Fig. 3.2 to examine the reproduction. The shales are assigned a log-conductivity of -8.

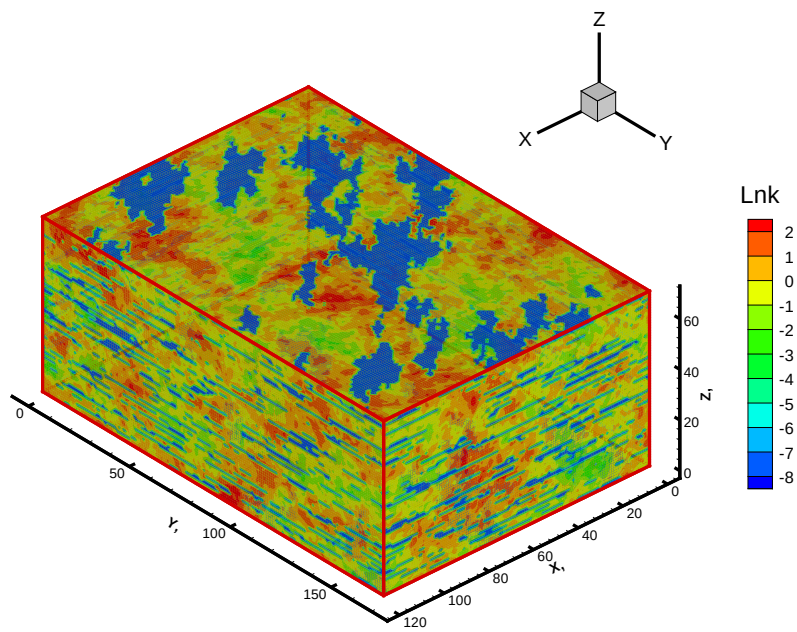

Figure 3.1. The hypothetic bimodel conductivity field

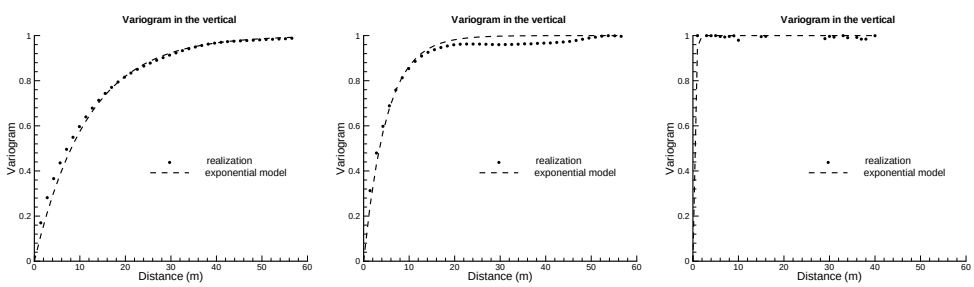

Figure 3.2. Variogram of the shale conductivity field in three direction

3.2 Boundary condition

Uniform boundary is applied again similar with the two synthetic examples in Section 2.3 except for the direction of the hydraulic gradient, i.e., a diagonal flow from the up left corner of the bottom layer to the lower right corner of the top layer. Change of the hydraulic gradient aims simulating different flow scenarios and checking the stability of the code.

3.3 Result and discussion

The performance of the proposed upscaling algorithm is reinforced with Fig. 3.3, in which the local specific discharge across block interface is well reproduced. Root Mean Square Error (*RMSE*) of the block flux (Table 3.1) is presented to illustrate the accuracy of the upscaling program in biomodel aquifer. One drawback when the code is utilized in bimodel lies the computation load since more iterations are needed to reach convergence in the two-components media characterized by several order magnitude difference of conductivity, in that, mean of Lnk of sand and shale medium in this case are 0 and -8. Comparing the flux reproduction and *RMSE*s with skin 5 and 10 (Fig. 3.3 and Table 3.1) and balancing the computation burden and upscaling precision, we think 5 skins, half of a block, are sufficient to reflect the feature of the media. We arrived at the same conclusion in the isotropic and anisotropic synthetic experiments in Chapter 2.

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

	0 Skin	2 Skins	5 Skins	10 Skins
$\bar{q}_x$	0.507	0.203	0.034	0.028
$\bar{q}_y$	0.533	0.220	0.041	0.031
$\bar{q}_z$	0.478	0.174	0.040	0.033

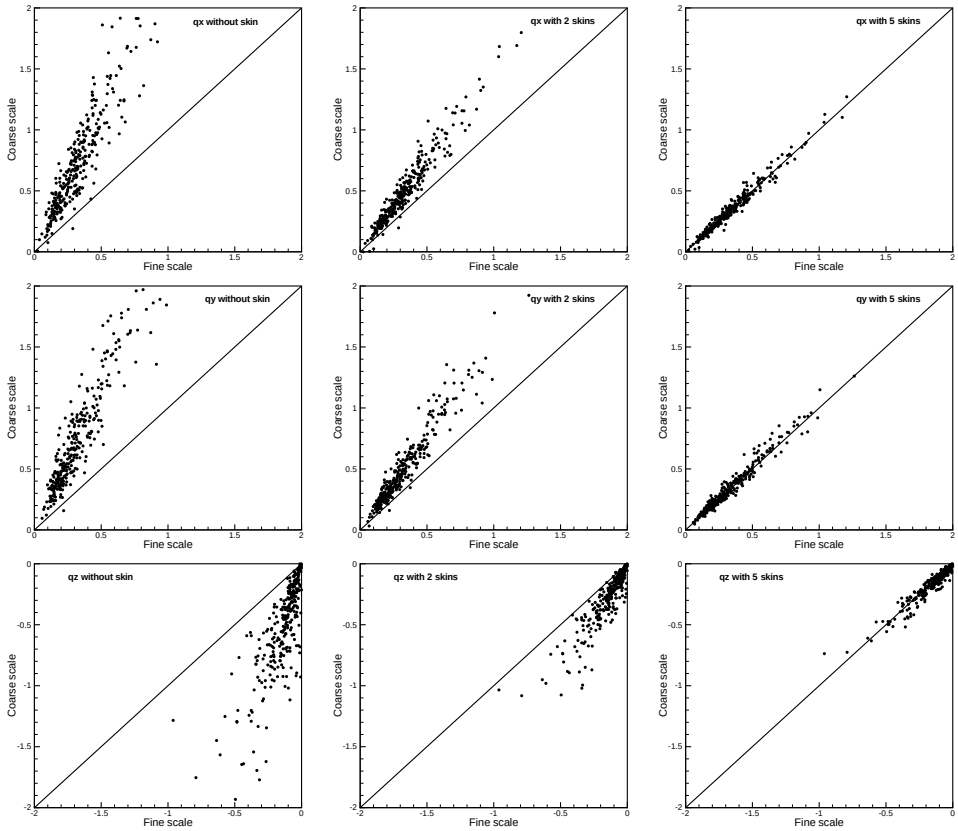

Figure 3.3. Specific discharge comparison between fine scale and coarse scale with different skins (0, 2 and 5 skins from left to right and $\bar{q}_x$, $\bar{q}_y$ and $\bar{q}_z$ from top to bottom) in the sand-shale cross-bedded aquifer

4
Boundary condition effect

4.1 Permeameter-type boundary condition

4.2 Uniform boundary condition

4.3 Periodic condition

5
Summary and Conclusions

Bibliography

[1] Ababou, R., 1990. Identification of effective conductivity tensor in randomly heterogeneous and stratified aquifers, in *Parameter Identification and Estimation for Aquifers and Reservoirs*, edited by S. Bachu, pp. 155–157, Water Well J. Publ., Dublin, Ohio.

[2] Bear, J., 1972. Dynamics of Fluids in Porous Media, Elsevier, New York.

[3] Bierkens, M. F. P., and Weerts H. J. T., 1994. Block hydraulic conductivity of cross-bedded fluvial sediments, *Water Resour. Res.*, 30(10), 2665-2678.

[4] 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(9S), 120S-134S.

[5] Dagan, G., 1989. Flow and Transport in Porous Formations, 465 pp., Springer, New York.

[6] Desbarats, A. J., 1992. Spatial Averaging of Hydraulic Conductivity in Three-Dimensional Heterogeneous Porous Media. *Math. Geol.,* 24(3), 249-267.

[7] De Wit, A., 1995. Correlation structure dependence of the effective permeability of heterogeneous porous-media, *Phys. Fluids.*, 7(11), 2553-2562.

[8] Farmer, C. L., 2002. Upscaling: A review, Int. J. Numer. *Methods Fluids*, 40, 63-78.

[9] Freeze, R.A. and Cherry, J.A., 1978. Groundwater. Prentice-Hall.

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

[11] Deutsch, C.V. and Journel, A.G., 1998. GSLIB: Geostatistical Software Library and User's Guide, 2nd edn. Oxford University Press, New York, 369 pp.

[12] Gómez-Hernández, J.J. and Wen, X.H., 1994. Probabilistic assessment of travel times in groundwater modeling. *J. Stochastic Hydrology and Hydraulics*, 8(l), 19-55.

[13] Green, C. P., and Paterson, L., 2007. Analytical three-dimensional renormalization for calculating effective permeabilities. *Transp. Porous. Med.,* 68, 237-248.

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

[15] Harbaugh, A. W., Banta, E. R., Hill, M. C., and McDonald, M. G., 2000. MODFLOW-2000, The U.S.Geological Survey modular groundwater model-user guide to modularization concepts and the ground-water flow process. U.S. GEOLOGICAL SURVEY. Open-File Report 00-92

[16] Hill, M. C., 1990. Preconditioned conjugate-gradient 2(PCG2), a computer program for solving ground-water flow equation. U.S.Geological Survey. Water-Resources-Investigations Report 90-4048.

[17] Holden, L. and Lia, O., 1992. A tensor estimator for the homogenization of absolute permeability. *Transport in Porous Media*, 8: 37-46.

[18] Journel, A.G., Deutsch, C.V. and Desbarats, A.J., 1986. Power averaging for block effective permeability. SPE 15128.

[19] Li, L.P., Gómez-Hernández, J.J., and Zhou, H.Y., 2008. A C program for flow modeling with full tensor by finite difference. Submitted.

[20] Matheron, G., 1967. Eléments pour une Théorie des Milieux Poruex, Masson, Paris.

[21] Pickup, G. E., Ringrose, P. S., Jensen, J. L., and Sorbie, K. S., 1994. Permeability tensors for sedimentary structures: *Math. Geol.*, 26(2), 227-250.

[22] Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P., 1992. Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press.

[23] Renard, P. and de Marsily, G., 1997. Calculating equivalent permeability: a review. *Advance in Water Resources,* 20(Nos 5-6), 253-278.

[24] Rubin, Y., and 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.

[25] Sánchez-Vila, X., Guadagnini, A., and Carrera, J., 2006. Representative hydraulic conductivities in saturated groundwater flow, *Rev. Geophys.*, 44, RG3002, doi:10.1029/2005RG000169.

[26] Vermeulen, P. T. M., te Stroet, C. B. M. and Heemink, A. W., 2006. Limitations to upscaling of groundwater flow models dominated by surface water interaction, *Water Resour. Res.*, 42, W10406, doi:10.1029/2005WR004620.

[27] Wen, X.-H., and Gómez-Hernández,J.J., 1996. Upscaling hydraulic conductivities in heterogeneous media: An overview, *Journal of Hydrology*, 183, ix-xxxii.

[28] Wen, X. H., Durlofsky, L. J. and Edwards,M. G., 2003. Use of border regions for improved permeability upscaling, *Math. Geol.*, 35(5), 521-547.

A

Three-dimensional Upscaling Code

```
# 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,int&wcmf,int&wrmf,int&wlmf,
        int&ncpmfx,int&ncpmfy,int&ncpmfz,int&ncpmix,int&ncpmiy,int&ncpmiz,
        int&ncmg,int&nrmg,int&nlmg,vector<int>&wcmg,vector<int>&wrmg,
        vector<int>&wlmg,int&ib,int&itt,int&ngra,int&px,int&py,
        vector<vector<int>>&gra,vector<vector<short>>&icon,
        vector<vector<vector<double>>>&tmf,short&debug);
void caltra01(int nuca,intib,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,int wlmf,int wrmf,int wcmf,
        vector <vector<vector<double>>>tmf,int ngra,int itt,
        vector <int> wcmg,vector<int> wrmg, vector<int> 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,int dz,int dy,int 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,int wlmf,int wrmf,int wcmf,
        vector<vector<vector<double>>>tmf,vector<vector<int>> gra,
        int ncpmiy,int ncpmiz,int wlmf,int wrmf,vector <int>wcmg,
        vector<int> wrmg,vector<int> 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,int wlmf,int wrmf,
        int wcmf,vector <vector<vector<double>>>tmf,
        vector<vector<int>> gra,vector <int> wcmg,vector<int>wrmg,
        vector<int> 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,int wlmf,int wrmf,int wcmf,
        vector <vector<vector<double>>>tmf,vector<vector<int>> gra,
        vector <int>wcmg,vector<int> wrmg,vector<int> wlmg, intngra,
        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,int wlmf,int wrmf,int wcmf,
        vector <vector<vector<double>>>tmf,vector<vector<int>> gra,
        vector <int> wcmg,vector<int>wrmg,vector<int> 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,int dz,int dy,int 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,
        intncmg,int &pnl,int &pnr,int &pnc, int nlmf,intnunk);
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,int wlmf,int wrmf,int wcmf,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, int dx,int 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, int dx,int 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,
        int dx,int dy, int dz,double &qx,double &qy,double &qz,double &xjx,
        vector <vector<vector<double>>> hh,double &yjy,double &zjz,
        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,int dx,int dy,int 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 &nl,int &nr,int &nc,int &kz, int &ix,int&jy,int nlmi,int nrmi,
        int ncmi,vector <vector<vector<double>>> &tmi,intnrmg,int ncmg);
void Printc(int &nl,int &nr,int &nc,int &kz, int &ix,int&jy,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,intnr,int nc,int ncpx,
        int ncpy,int ncpz,vector <double>&tmb);
```

```
void cuber(vector <vector<vector<double>>>tmi,int nl,intnr,int nc,int ncpx,
        int ncpy,int ncpz,vector <double>&tmb);
void analy(vector <vector<vector<double>>>tmi,int nl,intnr,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,ncmg,nrmg,nlmg,ncpmfx,ncpmfy,ncpmfz,ncpmix,ncpmiy,
        ncpmiz, wcmf,wrmf,wlmf,ib,itt,ngra,nr,nc,nl,pnl,pnr,pnc,px,py,nuca;
    short debug;px=0;
    vector <int> wcmg; vector <int> wrmg; vector <int> wlmg;
    vector<vector<int>> gra; vector<vector<short>> icon;
    vector <vector<vector<double>>> tmf;
    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,py,gra,icon,tmf,debug);
//.............check tmf.............
    if (debug==1)
        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);
    }
}
//*******************************************************************
//                        Read parameters and conductivity
//*******************************************************************
void Read_para_k(int&ncmf,int&nrmf,int&nlmf,int&wcmf,int&wrmf,int&wlmf,
        int&ncpmfx,int&ncpmfy,int&ncpmfz,int&ncpmix,int&ncpmiy,int&ncpmiz,
        int&ncmg,int&nrmg,int&nlmg,vector<int>&wcmg,vector<int>&wrmg,
        vector<int>&wlmg,int&ib,int&itt,int&ngra,int&px,int&py,
        vector<vector<int>>&gra,vector<vector<short>>&icon,
        vector<vector<vector<double>>>&tmf,short&debug)
{
    int i,j,k,c,r,l; string comments;
    cout<<"Enter of parameter file:"<<" ";
    string para;string conduc;
    cin>>para;
    cout<<"Enter the conductivity of the fine scale:"<<" ";
```

```
    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.......
    int 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');
//......check ib value......
    if((ib!=1) && (ib!=3))
        cout<<"Check ib"<<endl;
    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();
        ifstream inFile2;
        inFile2.open(conduc.c_str());
        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++)
                {
                    inFile2>>f;
                    tmf2.push_back(f);
                }
                tmf1.push_back(tmf2);
            }
            tmf.push_back(tmf1);
        }
        inFile2.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,int wlmf,int wrmf,int wcmf,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,intnr,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,intnr,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])
//*********************************************************************
void analy(vector <vector<vector<double>>>tmi,int nl,intnr,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],0.7647);
    st=st/nd;
    tmb[0]=pow(st,1.3077);
}
//*********************************************************************
//                                  print tmf
//*********************************************************************
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.dbg",ios::out—ios::app);
    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.setf(ios_base::right,ios_base::adjustfield);
                outFile0.precision(8);
                outFile0.setf(ios_base::showpoint);
                ios_base::fmtflags old=outFile0.setf(ios_base::scientific,
                    ios_base::floatfield);
                outFile0.width(20);
                outFile0<<tmf[k][i][j];
            }
            outFile0<<endl;
        }
        outFile0<<endl;
    }
    outFile0.close();
}
//*********************************************************************
//                                print tmi
//*********************************************************************
void Print(int &nl,int &nr,int &nc,int &kz, int &ix,int&jy,int nlmi,int nrmi,
        int ncmi,vector <vector<vector<double>>> &tmi,intnrmg,int ncmg)
{
    int i,j,k;
    ofstream outFile0;
    outFile0.open("check.dbg",ios::out—ios::app);
    outFile0<<"tmi"<<" "<<"block"<<kz/nlmi*nrmg*ncmg+ix/nrmi*ncmg
        +jy/ncmi<<endl;
    for(k=kz;k<kz+nl;k++)
    {
        for(i=ix;i<ix+nr;i++)
        {
            for(j=jy;j<jy+nc;j++)
            {
                outFile0.setf(ios_base::right,ios_base::adjustfield);
                outFile0.precision(8);
                outFile0.setf(ios_base::showpoint);
```

```
                ios_base::fmtflags old=outFile0.setf(ios_base::scientific,
                    ios_base::floatfield);
                outFile0.width(20);
                outFile0<<tmi[k-kz][i-ix][j-jy];
            }
            outFile0<<endl;
        }
        outFile0<<endl;
    }
    outFile0.close();
}
//*********************************************************************
//                              print cc, cr, cv
//*********************************************************************
void Printc(int &nl,int &nr,int &nc,int &kz, int &ix,int&jy,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"<<kz/nl*nrmg*ncmg+ix/nr*ncmg+jy/nc
        <<endl;
    for(k=0;k<nl;k++)
    {
        for(i=0;i<nr;i++)
        {
            for(j=0;j<nc;j++)
            {
                outFile0.setf(ios_base::right,ios_base::adjustfield);
                outFile0.precision(8);
                outFile0.setf(ios_base::showpoint);
                ios_base::fmtflags old=outFile0.setf(ios_base::scientific,
                    ios_base::floatfield);
                outFile0.width(20);
                outFile0<<cc[k][i][j];
            }
            outFile0<<endl;
        }
        outFile0<<endl;
    }
    outFile0.close();
}
```

```
//*********************************************************************
//                                print head
//*********************************************************************
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/nl*nrmg*ncmg+ix/nr*ncmg
        +jy/nc<<" "<<"gra"<<kount<<endl;
    for (k=0;k<nl;k++)
    {
        for (i=0;i<nr;i++)
        {
            for (j=0;j<nc;j++)
            {
                outFile0.setf(ios_base::right,ios_base::adjustfield);
                outFile0.precision(8);
                outFile0.setf(ios_base::showpoint);
                ios_base::fmtflags old=outFile0.setf(ios_base::scientific,
                    ios_base::floatfield);
                outFile0.width(20);
                n=j+i*nc+k*nr*nc;
                outFile0<<hh[k][i][j];
            }
            outFile0<<endl;
        }
        outFile0<<endl;
    }
    outFile0.close();
}
//*********************************************************************
//                                print flux
//*********************************************************************
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/nl*nrmg*ncmg+ix/nr*ncmg
```

```
        +jy/nc<<" "<<"gra"<<kount<<endl;
    outFile0.setf(ios_base::right,ios_base::adjustfield);
    outFile0.precision(8);
    outFile0.setf(ios_base::showpoint);
    ios_base::fmtflags old=outFile0.setf(ios_base::scientific,ios_base::floatfield);
    outFile0.width(20);
    outFile0<<qx<<" "<<qy<<" "<<qz<<" "<<xjx<<" "<<yjy
        <<" "<<zjz<<endl;
    outFile0<<endl;
}
//*********************************************************************
//                        block or interblock (ib,caltra01)
//*********************************************************************
void caltra01(int nuca,intib,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,int wlmf,int wrmf,int wcmf,
        vector <vector<vector<double>>>tmf,int ngra,int itt,
        vector <int> wcmg,vector<int> wrmg, vector<int> 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)
        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)
        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,int dz,int dy,int 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);
        if(px==2)
            cuber(tmi,nl, nr, nc, ncpmix,ncpmiy,ncpmiz,tmb);
        if(px==3)
            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,int wlmf,int wrmf,int wcmf,
        vector<vector<vector<double>>>tmf,vector<vector<int>> gra,
        int ncpmiy,int ncpmiz,int wlmf,int wrmf,vector <int>wcmg,
        vector<int> wrmg,vector<int> 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(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(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[k-kz][i-ix][j-jy]=tmf[k+ncpmfz-ncpmiz]
                                [i+ncpmfy-ncpmiy][j+ncpmfx-ncpmix];
//.........check tmi.........
                if (debug==1)
                    Print(nl,nr,nc,kz,ix,jy,nlmi,nrmi,ncmi,tmi,nrmg,ncmg);
                caltra02(nl,nr,nc,itt,gra,tmi,wlmf,wrmf,wcmf,ngra,icon,ncpmix,
                    ncpmiy,ncpmiz,tmb,debug,ix,jy,kz,nrmg,ncmg,pnl,pnr,pnc,
                    nlmf,px,nunk);
//.........output block k (tmb).........
                for (i=0;i<nunk;i++)
                {
                    outFile1.setf(ios_base::right,ios_base::adjustfield);
                    outFile1.precision(8);
                    outFile1.setf(ios_base::showpoint);
                    ios_base::fmtflags old=outFile1.setf(ios_base::scientific,
```

```
                        ios_base::floatfield);
                    outFile1.width(20);
                    outFile1<<tmb[i];
                }
//.........check positive definite.........
                if(nlmf>1 && px==0)
                {
                    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 && px==0)
                {
                    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();
}
//*********************************************************************
//        read tmf into tmi and run caltra02(genblo02 for ib==2 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,int wlmf,int wrmf,
        int wcmf,vector <vector<vector<double>>>tmf,
        vector<vector<int>> gra,vector <int> wcmg,vector<int>wrmg,
        vector<int> 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(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;   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(2*nl, vector< vector<
                    double>>(2*nr,vector<double>(2*nc,0)));
```

```
                for(k=kz;k<kz+nl;k++)
                    for(i=ix;i<ix+nr;i++)
                        for(j=jy;j<jy+nc;j++)
                            tmi[k-kz][i-ix][j-jy]=tmf[k+ncpmfz-ncpmiz][i+ncpmfy
                                -ncpmiy][j+ncpmfx-ncpmix];
//.........check tmi.........
                if (debug==1)
                    Print(nl,nr,nc,kz,ix,jy,nlmi,nrmi,ncmi,tmi,nrmg,ncmg);
                caltra02(nl,nr,nc,itt,gra,tmi,wlmf,wrmf,wcmf,ngra,icon,ncpmix,
                    ncpmiy,ncpmiz,tmb,debug,ix,jy,kz,nrmg,ncmg,pnl,pnr,pnc,
                    nlmf,px,nunk);
                for (i=0;i<nunk;i++)
                {
                    outFile1.setf(ios_base::right,ios_base::adjustfield);
                    outFile1.precision(8);
                    outFile1.setf(ios_base::showpoint);
                    ios_base::fmtflags old=outFile1.setf(ios_base::scientific,
                        ios_base::floatfield);
                    outFile1.width(20);
                    outFile1<<tmb[i];
                }
//.........check positive definite.........
                if(nlmf>1 && px==0)
                {
                    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 && px==0)
                {
                    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();
}
//*********************************************************************
//        read tmf into tmi and run caltra02(genblo03 for ib==2 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,int wlmf,int wrmf,int wcmf,
        vector <vector<vector<double>>>tmf,vector<vector<int>> gra,
        vector <int>wcmg,vector<int> wrmg,vector<int> wlmg, intngra,
        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 outFile2;
    outFile2.open("upk betw row.dat");
    outFile2<<"block conductivity full tensor"<<"between rows"<<endl;
    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;   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(2*nl, vector< vector<
                    double>>(2*nr,vector<double>(2*nc,0)) );
                for(k=kz;k<kz+nl;k++)
                    for(i=ix;i<ix+nr;i++)
                        for(j=jy;j<jy+nc;j++)
                            tmi[k-kz][i-ix][j-jy]=tmf[k+ncpmfz-ncpmiz][i+ncpmfy
                                -ncpmiy][j+ncpmfx-ncpmix];
//......check tmi......
                if (debug==1)
                    Print(nl,nr,nc,kz,ix,jy,nlmi,nrmi,ncmi,tmi,nrmg,ncmg);
                caltra02(nl,nr,nc,itt,gra,tmi,wlmf,wrmf,wcmf,ngra,icon,ncpmix,
                    ncpmiy,ncpmiz,tmb,debug,ix,jy,kz,nrmg,ncmg,pnl,pnr,pnc,
                    nlmf,px,nunk);
                for (i=0;i<nunk;i++)
                {
                    outFile1.setf(ios_base::right,ios_base::adjustfield);
                    outFile1.precision(8);
                    outFile1.setf(ios_base::showpoint);
                    ios_base::fmtflags old=outFile1.setf(ios_base::scientific,
                        ios_base::floatfield);
                    outFile1.width(20);
                    outFile1<<tmb[i];
                }
//......check positive definite......
                if(nlmf>1 && px==0)
                {
                    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 && px==0)
                {
                    if(tmb[0]<=0 || tmb[2]<=0 ||tmb[0]*tmb[2]-tmb[1]*tmb[1]<=0)
                    {
                        cout<<"No_positive definite!!";
                        system("pause");
                    }
                }
                outFile2<<endl;    jpj=jpj+ncmi;
            }
            ipi=ipi+nrmi1;    jpj=0;
        }
        kpk=kpk+nlmi;    ipi=0;    jpj=0;
    }
    outFile2.close();
}
//*********************************************************************
//        read tmf into tmi and run caltra02(genblo04 for ib==2 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,int wlmf,int wrmf,int wcmf,
        vector <vector<vector<double>>>tmf,vector<vector<int>> gra,
        vector <int> wcmg,vector<int>wrmg,vector<int> 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 outFile3;
    outFile3.open("upk betw lay.dat");
    outFile3<<"block conductivity full tensor"<<"between layers"<<endl;
    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-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(2*nl, vector< vector<
                    double>>(2*nr, vector<double>(2*nc,0)));
                for(k=kz;k<kz+nl;k++)
                    for(i=ix;i<ix+nr;i++)
                        for(j=jy;j<jy+nc;j++)
                            tmi[k-kz][i-ix][j-jy]=tmf[k+ncpmfz-ncpmiz][i+ncpmfy
                                -ncpmiy][j+ncpmfx-ncpmix];
//......check tmi......
                if (debug==1)
                    Print(nl,nr,nc,kz,ix,jy,nlmi,nrmi,ncmi,tmi,nrmg,ncmg);
                caltra02(nl,nr,nc,itt,gra,tmi,wlmf,wrmf,wcmf,ngra,icon,ncpmix,
                    ncpmiy,ncpmiz,tmb,debug,ix,jy,kz,nrmg,ncmg,pnl,pnr,pnc,
                    nlmf,px,nunk);
                for (i=0;i<nunk;i++)
```

```
                    {
                        outFile1.setf(ios_base::right,ios_base::adjustfield);
                        outFile1.precision(8);
                        outFile1.setf(ios_base::showpoint);
                        ios_base::fmtflags old=outFile1.setf(ios_base::scientific,
                            ios_base::floatfield);
                        outFile1.width(20);
                        outFile1<<tmb[i];
                    }
//......check positive definite......
                    if(nlmf>1 && px==0)
                    {
                        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 && px==0)
                    {
                        if(tmb[0]<=0 || tmb[2]<=0 ||tmb[0]*tmb[2]-tmb[1]*tmb[1]<=0)
                        {
                            cout<<"No_positive definite!!";
                            system("pause");
                        }
                    }
                    outFile3<<endl;    jpj=jpj+ncmi;
                }
                ipi=ipi+nrmi;    jpj=0;
            }
            kpk=kpk+nlmi1;    ipi=0;    jpj=0;
        }
        outFile3.close();
}
//*********************************************************************
//                          one block (tengen01)
//*********************************************************************
void tengen01(int nl,int nr,int nc,int itt,vector<vector<int>>gra,
            vector <vector<vector<double>>>tmi,int dz,int dy,int 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,
        intncmg,int &pnl,int &pnr,int &pnc, int nlmf,intnunk)
{
    int iflag,neq,kount,mne,mnunk;
    iflag=1;neq=0;mne=35;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);
    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];
//.........check cc,cv,cr.........
        if(debug==2)
        {
            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);
        }
//.........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);
//.........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
//*********************************************************************
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;
                    }
                }
            }
        }
        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;
        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, int dx,int dy,int ncpx,int ncpy,
        double &qx,double &qy,double&xjx,double &yjy,
        vector <vector<vector<double>>>hh)
{
    int i,j,k;   k=0;   xjx=0;   yjy=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;

    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);
    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, int dx,int 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;   k=0;   qx=0;   qy=0;
    double p1x,p2x,d12,p1y,p2y;
    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;
    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*(nc-2*ncpx);
    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*(nr-2*ncpy);
    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,
        int dx,int dy, int dz,double &qx,double &qy,double &qz,double &xjx,
        vector <vector<vector<double>>> hh,double &yjy,double &zjz,
        int ncpx,int ncpy,int ncpz)
{
    int i,j,k;    xjx=0;    yjy=0;    zjz=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++)
                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;

    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;
    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;
    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,int dx,int dy,int 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*(nc-2*ncpx);// modified 16 October
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*(nr-2*ncpy);// modified 16 October
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*(nl-2*ncpz);// modified 16 October
    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;    mu=10;   maxnu=10;
    double sum;    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
//*********************************************************************
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
//*********************************************************************
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

B.1 Parameter file

```
*************************************************************
**                      Parameter file                     **
*************************************************************
Line 1     100   150   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    9                               ** ngra
Line 12    0     1     0
           1     0     0
           0     0     1
           1     1     0
```

```
                1    0    1
                0    1    1
               -1    1    0
               -1    0    1
                0   -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
               -1   -1   -1   -1   -1   -1   ** icon
Line 14         1                            ** debug
```

Meaning of the parameters in the file above:

Line 1: number of columns, rows and layers in fine scale

Line 2: size of each cell in fine scale

Line 3: outer skins for the domain

Line 4: inner skins for each block. Note the inner skins cannot exceed outer skins.

Line 5: number of blocks in coarse scale

Line 6: width of each block along the row direction ("ncmg" in total)

Line 7: width of each block along the column direction ("nrmg" in total)

Line 8: width of each block along the layer direction ("nlmg" in total)

Line 9 and 10: these two parameters combined to determine whether the upscaled conductivity in the center of block or at the interface ("ib=1 and itt=31" in the center and "ib=3 and itt=32" at the interface)

Line 11: number of directions

Line 12: indicate the direction ("ngra" in total). The direction is defined in Cartesian coordinate with 0 and 1, for instance, "0 1 0" indicates Y axis direction and"0 0 1" means Z direction.

Line 13: boundary condition ("ngra" in total). "-1" means constant head boundary and "0" indicates no flow boundary. No flow boundary are not normally used since the off diagonal elements of $\vec{K}_V$ are influenced negatively by this type of boundary.

Line 14: a flag for debug

This parameter example is aiming at 3D case and for the 2D case the number of layer need to be assigned as 1.

B.2 Conductivity input file

The input data file contains hydraulic conductivity for fine scale. Modflow format is expected in the conductivity data, that is, the origin of coordinates is located at the up left corner of the top layer.

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 3 for the latter case, i.e., interblock conductivity between columns, rows and layers, respectively, for 3D case.