2004

An inverse groundwater model.

Aimin Yan
University of Alabama at Birmingham

AN INVERSE GROUNDWATER MODEL

by

AIMIN YAN

A DISSERTATION

Submitted to the graduate faculty of The Universty of Alabama at Birmingham,
The University of Alabama in Huntsville, and The University of Alabama,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

BIRMINGHAM, ALABAMA

2004

ABSTRACT OF DISSERTATION
GRADUATE SCHOOL, UNIVERSITY OF ALABAMA AT BIRMINGHAM

Degree Ph.D. Program Applied Mathematics

Name of Candidate Aimin Yan

Committee Chair Ian Knowles

Title An Inverse Groundwater Model

A groundwater system can be modeled by the following equations:

$$S(x)\frac{\partial \phi}{\partial t} = -\nabla \cdot \boldsymbol{q} + R(x,t),$$

$$\frac{\partial(\theta c)}{\partial t} = -\nabla \cdot (c\boldsymbol{q}) + \nabla \cdot (\theta \boldsymbol{D} \nabla c) + B,$$

where $\boldsymbol{q}$ is the specific discharge, over x in a bounded region $\Omega \subset R^n$, $n = 2$, or 3, and for $t > 0$.

In this dissertation, we give a descent algorithm to recover all the coefficients of the two equations above. This algorithm is stable and efficient. The method is used in analyzing the Willunga Basin Aquifer in South Australia. A suggestion is given in analyzing the sustainability of the aquifer.

ACKNOWLEDGMENTS

I take this opportunity to give a very special thanks to Dr. Ian Knowles, my Ph.D. advisor, for his wonderful guidance throughout the past few years. I want to also thank him for all the lengthy hours we have spent discussing these mathematical and computational, as well as geological, problems. I have greatly appreciated his immense patience. Dr. Knowles is truly a mentor to me.

I also express my gratitude to Professor Yuan-Ming Li, my previous M.S. advisor in China. Professor Li is one of the best professors I have ever met. It is Professor Li who first led me into the mathematical world.

I thank the other committee members, Drs. Robert Hyatt (The University of Alabama at Birmingham), Tsun-Zee Mai (The University of Alabama), S. S. Ravindran (The University of Alabama in Huntsville), Yanni Zeng (UAB), for their kind guidance and suggestions.

A special thanks goes to Dr. Abe Springer, Professor at the Department of Geology of Northern Arizona University. Dr. Springer, I thank you for your comments and suggestions about the geological concepts.

Last, but certainly not least, I thank my wife, Jie Wei. She is especially deserving of thanks, not only for all of the support that she has given me, but also for having made these past several years much more pleasant than they otherwise would have been.

Contents

List of Figures

CHAPTER 1

Groundwater Hydrology

Groundwater is that portion of the water beneath the surface of the earth that can be collected with wells, tunnels, or drainage galleries. Groundwater can also flow to the earth's surface via seeps or springs. In many places, groundwater is the main source to supply water for people and irrigation.

Not all underground water is groundwater. The term "groundwater" is generally referred to, by the hydrologist, as the water occupying all the voids, saturated, within a geologic stratum. A better understanding about groundwater movements, and the architecture of the aquifer the groundwater moves through, is essential to manage and protect groundwater resources against undue exploitation and pollution. Since the aquifer is generally hundreds of meters below the earth's surface, it is impractical or impossible to directly determine the properties of the aquifer. Our study here uses mathematical modeling equations about a groundwater system and data about the groundwater movements to get the coefficients, the properties of the aquifer, of the modeling equations.

1.1. Aquifers and porous media

Here we introduce some commonly used concepts in groundwater hydrology. For a detailed discussion and examples please refer to textbooks on groundwater hydrology, such as [**9, 13**].

1.1.1. Aquifers. An *aquifer* is a geological formation that contains water and permits significant amounts of water to move through it under ordinary field conditions. The most common aquifer materials are unconsolidated sands and gravels. In contrast, an *aquiclude* is a formation that may contain water but is incapable

of transmitting significant quantities under ordinary field conditions. Clay is such an example. Between the aquifer and aquiclude, an *aquitard* is a semipervious geologic formation that transmits water at a very slow rate as compared to the aquifer. However, over a large (horizontal) area it may permit the passage of large amounts of water between adjacent aquifers, which it separates. It is often referred to as a leaky formation. An *aquifuge* is an impervious formation that neither contains nor transmits water.

The portion in a rock that is not occupied by solid materials may be occupied by water or air. These spaces are called the *void spaces.* Because the void spaces can act as groundwater conduits, they are of fundamental importance to the study of groundwater. Typically, they are characterized by their size, shape irregularity, and distribution. Only connected interstices can act as elementary conduits within the formation.

Aquifers may be regarded as underground storage reservoirs that are replenished naturally by precipitation and influent streams, or through wells and other artificial recharge methods. Water leaves the aquifer naturally through springs or effluent streams and artificially through pumping wells.

The thickness and other vertical dimensions of an aquifer are usually much smaller than the horizontal lengths involved. Aquifers may be classified as *confined* and *unconfined* (or *phreatic*), depending upon the presence or absence of a water table.

A *confined aquifer* is one bounded above and below by impervious formations. In a well penetration of such an aquifer, the water level will rise above the base of the confining formation; it may or may not reach the ground surface. A properly constructed observation well (or a piezometer) has a relatively short screened section (not too short with respect to the size of the openings) such that it indicates the *piezometric head* at a specific point. The water levels in a number of observation wells tapping a certain aquifer define an imaginary surface called the *piezometric surface.* When the flow in the aquifer is essentially horizontal, such that equipotential surfaces

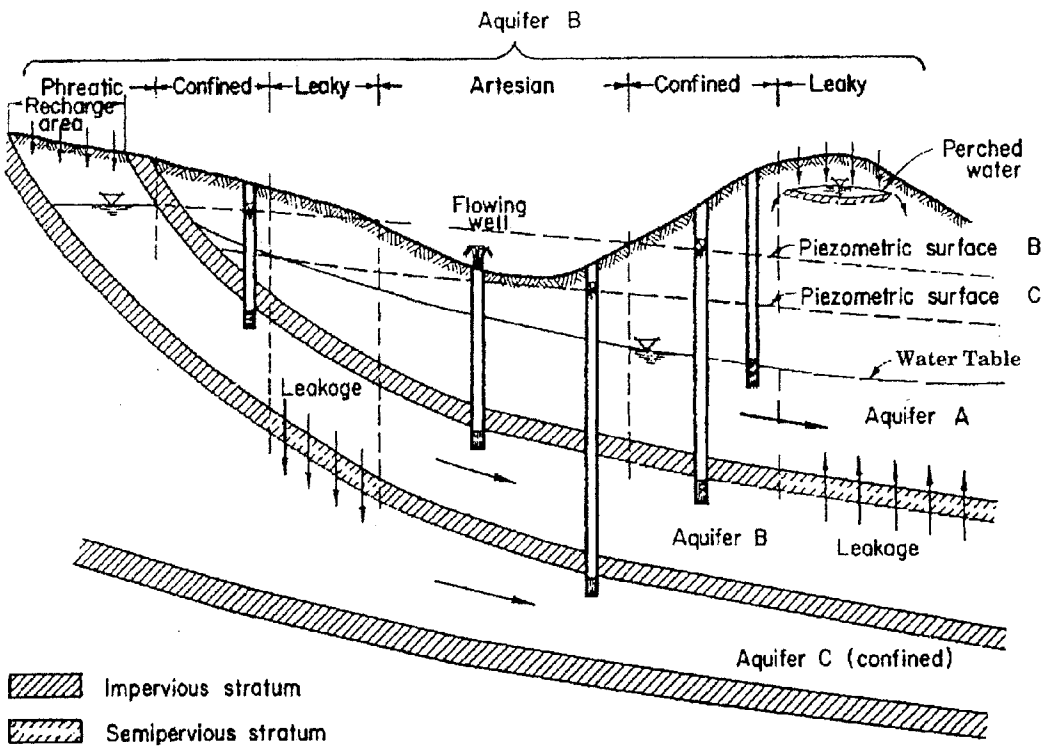
are vertical, the depth of the piezometer opening is immaterial; otherwise, a different piezometric surface is obtained for piezometers that have openings at different elevations. Water enters a confined aquifer through an area between confining strata that rise to the ground surface, or where an impervious stratum ends underground, rendering the aquifer unconfined. The region supplying water to a confined aquifer is called a *recharge area.*

An *unconfined aquifer* (also called a *phreatic aquifer*) is one with a *water table* (*phreatic surface*) serving as its upper boundary. Actually, above the phreatic surface is a *capillary fringe*, often neglected in groundwater studies. A phreatic aquifer is recharged from the ground surface above it, except where impervious layers of limited horizontal area exist between the phreatic surface and the ground surface.

Leaky aquifers are aquifers that can lose or gain water through either or both of the formations bounding them above and below. Although these bounding formations may have a relatively high resistance to the flow of water through them, over the large (horizontal) areas of contact involved significant quantities of water may leak through them into or out of a particular aquifer. The amount and direction of leakage is governed in each case by the difference in piezometric head that exists across the semipervious formation.

A phreatic aquifer (or part of it) that rests on a semipervious layer is a *leaky phreatic aquifer.* A confined aquifer (or part of it) that has at least one semipervious confining stratum is called a *leaky confined aquifer.* Figure 1.1 shows several aquifers and observation wells. The upper phreatic aquifer is underlain by two confined ones. In the recharge area, aquifer B becomes phreatic. Portions of aquifers A, B, and C are leaky, with the direction and rate of leakage determined by the elevation of the piezometric surfaces of each of these aquifers. The boundaries between the various confined and unconfined portions may vary with time as a result of changes in water table and piezometric head elevations. A special case of a phreatic aquifer is the *perched aquifer* that occurs wherever an impervious (or relatively impervious) layer

FIGURE 1.1. Type of aquifers [9]



of limited horizontal area is located between the water table of a phreatic aquifer and the ground surface. Another groundwater body is then built above this impervious layer. Clay or loam lenses in sedimentary deposits have shallow perched aquifers above them. Sometimes these aquifers exist only a relatively short time as they drain to the underlying phreatic aquifer.

1.1.2. The porous medium. The materials forming an aquifer contain *void space* filled with water and/or air. The connected interstices can act as elementary conduits within the formation, allowing water to flow. These materials can be viewed as a *porous medium*, and the flow in the aquifer can be considered as the *flow of fluids through a porous medium*. Soil, porous or fissured rocks, ceramics, and fibrous aggregates are just a few examples of porous materials. All of these materials have some characteristics in common that permit them to be grouped and classified as porous media.

Not all materials containing holes are porous media. For a media to be classified as a porous media, some of the holes adjacent should be connected to allow fluid moving through it. The following is a descriptive definition of a porous medium (Bear, Zaslavsky, and Irmay [14]):

a) A portion of space occupied by *heterogeneous* or *mutiphase* matter. At least one of the phases comprising this matter is not *solid*. There may be gaseous and/or liquid phases. The solid phase is called the *solid matrix*. That space within the porous medium domain that is not part of the solid matrix is referred to as *void space* (or *pore space*).

b) The solid phase should be distributed throughout the porous medium within the domain occupied by a porous medium; solid must be present inside each representative elementary volume. An essential characteristic of a porous medium is that the *specific surface* of the solid matrix is *relatively high*. In many respects, this characteristic dictates the behavior of fluids in porous media. Another basic feature of a porous medium is that the various openings comprising the void space are *relatively narrow*.

c) At least some of the pores comprising the void space should be interconnected. The interconnected pore space is sometimes termed the *effective pore space*. As far as flow through porous media is connected, *unconnected pores* may be considered as part of the solid matrix. Certain portions of the interconnected pore space may, in fact, also be ineffecive as far as flow through the medium is concerned.

1.1.3. Continuum approach to porous media. In an aquifer, water flows through the complex network of pores and channels comprising the void space. This flow is bounded by the (microscopic) solid-water interface. In principle, the flow of a fluid in a porous medium may be treated at the *microscopic level*, at which we focus our attention on what happens at a point within the fluid, regarded as a *continuum* (i.e., overlooking its molecular structure). However, complexity of the pore space will

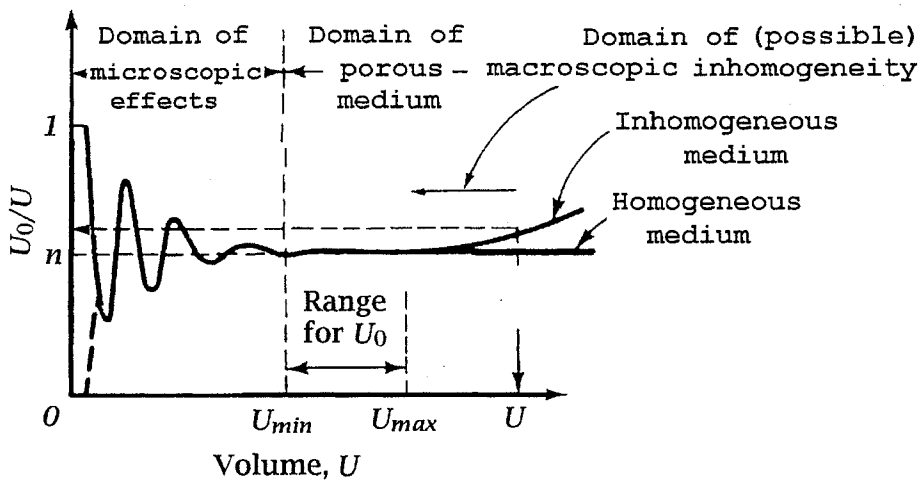
usually make this treatment impossible. Moreover, even if we can solve for the values of state variables, e.g., pressure, at the microscopic level, we could not verify these solutions by measurements at this level.

To circumvent these difficulties, another level of description is needed. This is the *macroscopic level*, at which quantities can be measured and boundary-value problems can be solved. To obtain the description of the flow at this level, we adopt the *continuum approach.* This is the same approach that is also used in order to pass from the molecular level of description to the microscopic one, at which each phase is regarded as a continuum. According to this approach, the real porous medium, in which each phase (solid or fluid) occupies only a portion of the AEV (Arbitrary Elementary Volume), is replaced by a *fictitious model* in which each phase is regarded as a continuum that fills up the entire AEV. We thus obtain within every AEV a set of overlapping and, possibly, interacting, continua. For each of these continua, average values, referred to as *macroscopic values*, can be taken over the AEV and assigned to its centroid, regardless of whether the latter falls within the solid or within one of the fluids that occupy the void space. By traversing the entire porous medium domain with a moving AEV, we obtain *fields* of macroscopic variables, which are differentiable functions of the space coordinates.

The main drawback of the use of an AEV is that every averaged value must be accompanied by a label that specifies the volume over which this average was taken. To circumvent this difficulty, we need a universal procedure that a) is applicable to all porous media and b) will ensure that the averaged values will remain, more or less, constant, at least for a certain range of averaging volumes, that corresponds to the range of variations in instrument sizes. This universal averaging volume is referred to as the *representative elementary volume* (REV).

The size of the REV is selected such that the averaged values of all geometrical characteristics of the microstructure of the void space be a single valued function of the location of that point only, *independent of the size of the REV.*

FIGURE 1.2. Definition of porosity and representative elementary volume. [13]



To illustrate the determination of the size of an REV for a given porous medium domain, D, consider, as an example of a geometrical characteristic of the void space configuration, the ratio $U_v(x_0)/U(x_0)$, where $U(x_0)$ is a volume of a sphere centered at an arbitrary point x_0 within D, and $U_v(x_0)$ is the volume of void space within $U(x_0)$. Figure 1.2 shows the variations of the ratio $U_v(x_0)/U(x_0)$ as U increases. For very small values of U, this ratio is one or zero, depending on whether x_0 happens to fall in the void space or in the solid matrix. As U increases, we note large fluctuations in this ratio due to the random distribution of void and solid within U. However, as U is further increased, these fluctuations gradually decay until above some volume $U = U_{\text{min}}$ they reduce to some small value. If U is further increased beyond some $U = U_{\text{max}}$, we may observe a trend in the considered ratio, due to a systematic variation in the latter, resulting from macroscopic heterogeneity of the porous medium. The size, U_0, of the REV that will make the considered ratio independent of the selected volume, albeit possibly dependent on x, should be in the range $U_{\text{min}} < U_0 < U_{\text{max}}$. For such a volume, the ratio U_{0v}/U_0 represents the medium porosity, n, at x_0.

Once U_0 has been determined, it is used to derive the macroscopic (continuum) description of the flow by averaging the microscopic one over it. Obviously, the selected size of U_0 must be uniform over the entire porous medium domain. The

macroscopic model obtained in this way describes the flow in terms of macroscopic or averaged quantities defined by

$$\overline{G}_\alpha^\alpha(x,t) = \frac{1}{U_{0\alpha}} \int_{U_{0\alpha}(x)} G_\alpha(x',t;x) dU_\alpha(x') \tag{1.1}$$

where G_α is the state variable of the α-phase (such that its volumetric average is physically meaningful), $U_{0\alpha}$ is the volume of the α-phase within U_0, and x' is a point in the REV centered at x. From the discussion presented, we are assured that the macroscopic geometrical characteristics that appear in the macroscopic model represent properties of porous medium at x. The average $\overline{G}_\alpha^\alpha$ of G_α, as defined by (1.1), is called an *intrinsic phase average.*

Another type of average, called a *phase average*, defined by

$$\overline{G}_\alpha(x,t) = \frac{1}{U_0} \int_{U_{0\alpha}(x)} G_\alpha(x',t;x) dU_\alpha(x') \tag{1.2}$$

is also often used. The two types of averages are related to each other by

$$\overline{G}_\alpha = \theta_\alpha \overline{G}_\alpha^\alpha, \tag{1.3}$$

where θ_α is the volumetric fraction of the $\alpha-$phase.

If a volume U_0 cannot be found for a given porous medium domain, the latter cannot be treated as a continuum. In an analogous way, a *representative elementary area* (REA) should also be selected for the porous medium domain, to be used for averaging quantities for which only areal averages are meaningful. Throughout this dissertation, it is assumed that the porous medium can be considered as a continuum.

1.1.4. Isotropic and anisotropic medium. A medium is said to be *homogeneous* with respect to a certain property if that property is *independent of position* within the medium. Otherwise the medium is said to be *heterogeneous.* For example, if the *porosity* of a certain material is constant, then it is a homogeneous property; otherwise it is heterogeneous. In the real world, most of the properties are heterogeneous.

A medium is said to be *isotropic* with respect to a certain property if that property is *independent of direction* within the medium. If at a point within the medium a property of the medium, e.g., permeability or thermal conductivity, *varies with direction*, the medium is said to be *anisotropic* (or *aleotropic*) at the considered point with respect to that property. In natural materials, anisotropy is encountered in soils and in geological formations that serve as reservoirs or aquifers. In most stratified materials the resistance to the flow is smaller (i.e., permeability is greater) along the planes of deposition than across them. Piersol et al. [**80**] mention ratios of horizontal to vertical permeabilities of sandstone of 1.5 : 3. Muskat [**71**, page 111] lists 65 pairs of sand samples, more than two-thirds of which had a larger permeability in the direction parallel to the bedding plane than normal to it. The quotient of the two values ranged from 1 to 42.

Stratified soils are usually anisotropic. The stratification may result from the shape of the particles. For example, plate-shaped particles (e.g., mica) will generally be oriented with the flat side down. Both sedimentation and the pressure of overlaying material cause flat particles to be oriented with their longest dimensions parallel to the plane on which they settle. This later produces flow channels parallel to the bedding plane, differing from those oriented normal to this plane, and the medium becomes anisotropic. Alternating layers of different texture also give rise to anisotropy. However, in order for a stratified formation of this kind to be qualified as an anisotropic homogeneous medium, the thickness of the individual layers should be much smaller than the lengths of interest. There is no use in attempting to determine the permeability of such a formation from a core whose size is smaller than the thickness of the single stratum. In many aquifers, fractures produce very high permeability in the direction along the fracture, whereas the permeability of the rock in the direction normal to the fractures is much smaller. In carbonate rocks, dissolving of the rock takes place by means of the flowing water. This produces solution channels that develop mainly in the direction of the flow; the rock becomes anisotropic, with

a very high permeability in the general direction of these channels. In many soils (e.g., loess), vertical joints, root holes, and animal burrows give rise to anisotropy in permeability, with vertical permeability being greater than horizontal. In some soils, structural fissures may develop more readily in some directions than in others, and the soil will exhibit anisotropy.

1.1.5. The piezometric head. Flow occurs from a place of higher energy to one of lower energy. In groundwater flow, *potential* is a concept describing this energy. The total potential is an algebraic summation of various specific potentials acting on the groundwater flow.

There are many alternate ways of defining a potential function. The ultimate choice depends upon convenience and suitability for the range of problems with which one is concerned. For subsurface water, potential may be defined in such a way that its gradient is proportional to the water-moving forces. Furthermore, because potential is defined relative to an arbitrary datum, one is concerned only with differences of potential between specified points.

Bolt and Miller [**16**] define total potential of soil moisture in a fashion that is extended readily to include groundwater. They define total potential as the minimum energy per gram of water which must expended in order to transport an infinitesimal test body of water from a specific reference state to any point within the liquid phase of a soil-water system that is in a state of rest. Following Bolt and Miller's fashion, Remson [**84**] defines the potential in terms of energy per unit weight of water. With this definition, potential has the dimension of length and is referred to as "head."

In saturated subsurface systems, the total potential is the algebraic summation of the component potentials of the gravitational potential and the hydrostatic pressure potential below the water table [**84**]:

$$\phi = \Psi_g + \Psi_p,$$

where

$$\Psi_g = z,$$

$$\Psi_p = p/\gamma,$$

here z is the height of the water above the reference datum, p is the pressure, and $\gamma = \rho g$ is the specific weight of water (ρ = density, g = acceleration of gravity).

For a homogeneous compressible fluid (i.e., no dissolved components) under isothermal conditions, Hubbert presented a particularly clear derivation of potential defined on a work-per-unit-mass basis for saturated subsurface systems [**45**]. Under this fashion, we have that the potential, $\phi^\star$, at point p (the velocity of which is usually small and is neglected) is

$$\phi^\star = gz + \int_{p_0}^{p} \frac{dp}{\rho(p)}, \tag{1.4}$$

where p is the pressure. This expression is known as Hubbert's "force potential." If we set $\phi = \phi^\star/g$, (1.4) gives the form

$$\phi = z + \int_{p_0}^{p} \frac{dp}{g\rho(p)}. \tag{1.5}$$

When ρ is constant and p_0 is chosen to be 0, (1.5) reduces to

$$\phi = z + p/\gamma, \tag{1.6}$$

where $\gamma = \rho g$ is the specific weight of water.

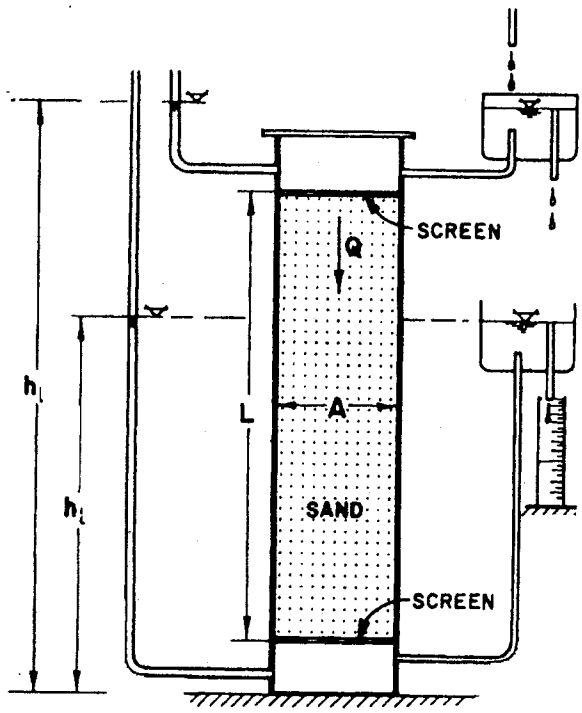
We call ϕ the piezometric head. The gradient $\nabla\phi$ is called the hydraulic gradient; it is proportional to the water-moving forces.

1.2. The equation of groundwater motion

In almost every field of science and engineering the techniques of analysis are based on an understanding of the physical processes, and in most cases it is possible to describe these processes mathematically. Groundwater flow is no exception.

1.2.1. Darcy's law and its extensions. Groundwater moves from levels of higher energy to levels of lower energy, whereby its energy is essentially the result of elevation and pressure. Kinetic energy, proportional to the square of the velocity, is neglected because groundwater velocities are very small, at least in laminar flow. While flowing, groundwater experiences a loss in energy due to friction against the walls of the granular medium along its seepage path. This loss per unit length of distance traveled, or hydraulic gradient, is simply proportional to the velocity of groundwater for laminar flow in sandy aquifers or seepage through earth embankments. When the proportionality of hydraulic gradient and groundwater velocity is expressed by a mathematical equation, a linear law of flow, called Darcy's law, arises.

FIGURE 1.3. Darcy's experiment. [9]



In 1856, Henry Darcy investigated the flow of water in vertical homogeneous sand filters in connection with the fountains of the city of Dijon, France. Figure 1.3 shows the experimental set-up he employed (Darcy [28]). From his experiments, Darcy

concluded that the flow rate Q equals:

$$Q = KA(h_1 - h_2)/L,$$

where $h_1 - h_2$ is the energy loss, and L is the length of the flow path. A is the cross-sectional area filled with sand, and K is a coefficient, called the hydraulic conductivity.

One can easily extend Darcy's law to flow through an inclined homogeneous porous medium column (Figure 1.4). With the nomenclature of this figure, Darcy's law takes the form

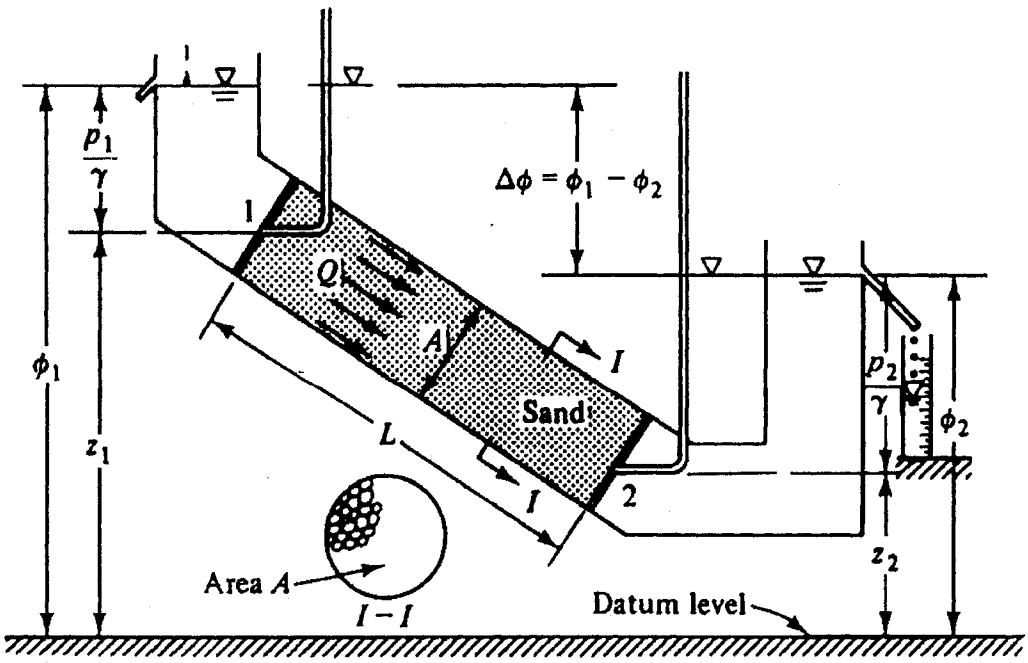
$$Q = KA(\phi_1 - \phi_2)/L,$$

where ϕ is the *piezometric head* defined by

$$\phi = z + p/\gamma,$$

where z is the elevation of the point, p is the pressure, and γ is the volumetric weight of the water. The piezometric head expresses the sum of the potential energy and pressure energy, per unit weight of water.

FIGURE 1.4. Flow through an inclined sand column. [13]



The energy loss $\Delta\phi = \phi_1 - \phi_2$ is due to friction in the flow through the narrow tortuous paths of the porous medium. In Darcy's law, the kinetic energy of the water has been neglected as, in general, changes in the piezometric head along the flow path are much larger than changes in the kinetic energy. Inertial effects have also been neglected.

With the above definition of piezometric head, the quotient $(\phi_1 - \phi_2)/L$ is the *hydraulic gradient* (dimensionless). Denoting this gradient by J and defining the *specific discharge*, q, as the volume of water flowing per unit time through a unit cross-sectional area normal to the direction of flow, we obtain

$$q = KJ.$$

Let us consider a point along the column's axis and a segment of the column of length s along the column's axis on both sides of the point. For this case

$$q_s = K\frac{\phi|_{s-(\Delta s/2)} - \phi|_{s+(\Delta s/2)}}{\Delta s}, \tag{1.7}$$

where the subscript in q_s indicates that the flow is in the s-direction. In the limit, as $\Delta s \to 0$, converging on the point, we obtain

$$\lim_{\Delta s \to 0} \frac{\phi|_{s-(\Delta s/2)} - \phi|_{s+(\Delta s/2)}}{\Delta s} = -\frac{\partial \phi}{\partial s},$$

and (1.7) reduces to

$$q_s = -K\frac{\partial \phi}{\partial s}. \tag{1.8}$$

The experimentally derived form of Darcy's law (for a homogeneous incompressible fluid) was limited to one-dimensional flow. When the flow is three-dimensional, the obvious formal generalization of Darcy's law, is

$$\mathbf{q} = -\mathbf{K}\nabla\phi,$$

where $\mathbf{q}$ is the *specific discharge* with components q_x, q_y, and q_z in the directions of the Cartesian x, y, z coordinates, respectively, and $\nabla\phi = (\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z})$. When the flow takes place through a homogeneous isotropic medium, the coefficient $\boldsymbol{K} = K\boldsymbol{I}$, i.e.,

a scalar times the identity matrix; otherwise, it is a symmetric positive definite 3×3 matrix for the three-dimensional case, or 2×2 matrix for the two-dimensional case.

The coefficient $\mathbf{K}$ is called the *hydraulic conductivity*. The hydraulic conductivity indicates the ability of the aquifer material to conduct water through it under hydraulic gradients. It is a combined property of the porous medium and the fluid flowing through it. When the flow in the aquifer is essentially horizontal, the *aquifer transmissivity* indicates the ability of the aquifer to transmit water through its entire thickness. It is the product of the hydraulic conductivity and the thickness of the aquifer.

As the specific discharge increases, Darcy's law, which specifies a linear relationship between the specific discharge, q, and the hydraulic gradient, $\nabla\phi$, has been shown by many investigators to be invalid. A definition of a range of validity of Darcy's law seems, therefore, appropriate.

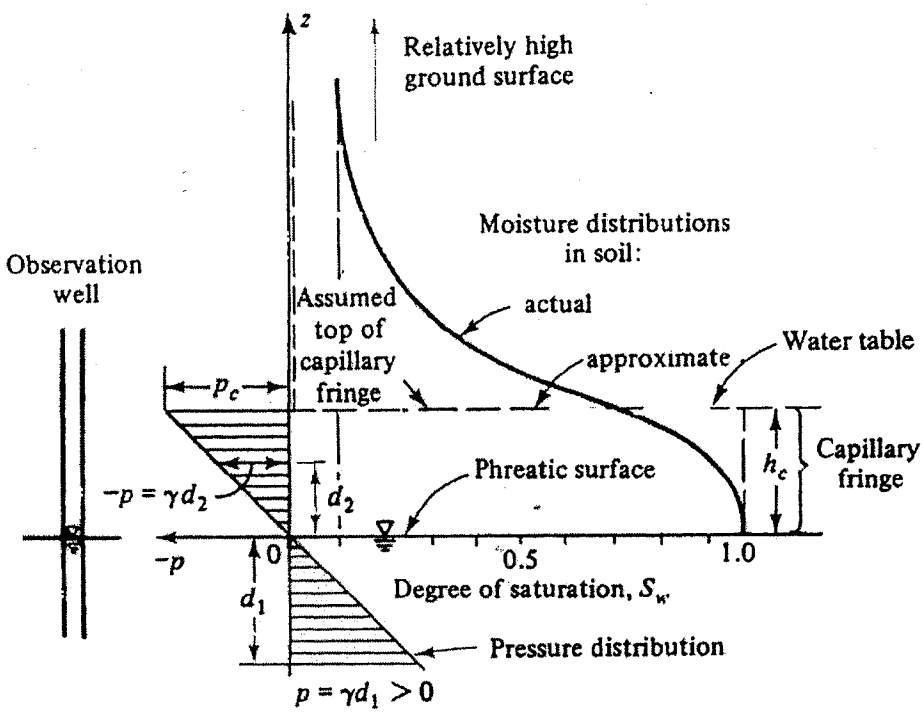
In flow through conduits, the Reynolds number (Re), a dimensionless number expressing the ratio of inertial to viscous forces, is used as a criterion to distinguish between laminar flow occurring at low velocities and turbulent flow. The critical Re between laminar and turbulent flow in pipes is around 2100. By analogy, a Reynolds number is defined also for flow through porous media:

$$Re = \frac{qd}{v},$$

where d is some length dimension of the porous matrix, and v is the kinematic viscosity of the fluid. Although, by analogy to the Reynolds number for pipes, d should be a length dimension representing the elementary channels of the porous medium, it is customary (probably because of the relative ease of determining it) to employ some representative dimension of the grains for d (in an unconsolidated porous medium). Often the mean grain diameter is taken as the length dimension, d. Sometimes d_{10} is used, i.e., the grain size that exceeds the size of 10% of the material by weight. The term d_{50} is also mentioned in the literature as a representative grain diameter.

In practically all cases, Darcy's law is valid as long as the Reynolds number based on average grain diameter does not exceed some value between 1 and 10.

FIGURE 1.5. Approximations of phreatic surface and capillary fringe. [13]



1.2.2. Dupuit assumption. As defined in Section 1.1.1, a phreatic aquifer is one in which a water table (or a phreatic surface) serves as its upper boundary. Above the phreatic surface, which is an imaginary surface, at all points of which the pressure is atmospheric, moisture does occupy at least part of the pore space. The capillary fringe was introduced as an approximation of the actual distribution of moisture in the soil above a phreatic surface.

Figure 1.5 shows how the actual moisture distribution is approximated by a step distribution, assuming that no moisture is present in the soil above a certain level. This step defines the height, h_c, of the capillary fringe. Obviously, this approximation is justified only when the thickness of the capillary fringe thus defined is much smaller than the distance from the phreatic surface to the ground surface. In the capillary

fringe (as in the entire aerated zone above the phreatic surface), pressures are negative; therefore, they cannot be monitored by observation wells which serve as piezometers. A special device, called a *tensiometer*, is needed in order to measure the negative pressures in the aerated zone (Figure 1.6b). Water levels in observation wells that terminate below the phreatic surface give elevation of points on the phreatic surface. Using a sufficient number of such points, we can draw contours of this surface.

Thus, the *capillary fringe approximation* means that we assume a saturated zone up to an elevation h_c above the phreatic surface, and no moisture at all above it. In this case, the upper surface of the capillary fringe may be taken as the *groundwater table*, as the soil is assumed saturated below it. However, when h_c is much smaller than the thickness of an aquifer below the phreatic surface, and this is indeed the situation encountered in most aquifers, the hydrologist often neglects the capillary fringe. He then assumes that the (phreatic) aquifer is bounded from above by a phreatic surface. This is also the assumption below.

An estimate of h_c, can be obtained, for example, from [**67**]

$$h_c = \frac{2.2}{d_H}\left(\frac{1-n}{n}\right)^{3/2}, \tag{1.9}$$

where h_c is in inches, and d_H is the mean grain diameter, also in inches, and n is porosity. Another expression is suggested by Polubarinova-Kochina [**81**]:

$$h_c = \frac{0.45}{d_{10}}\frac{1-n}{n}, \tag{1.10}$$

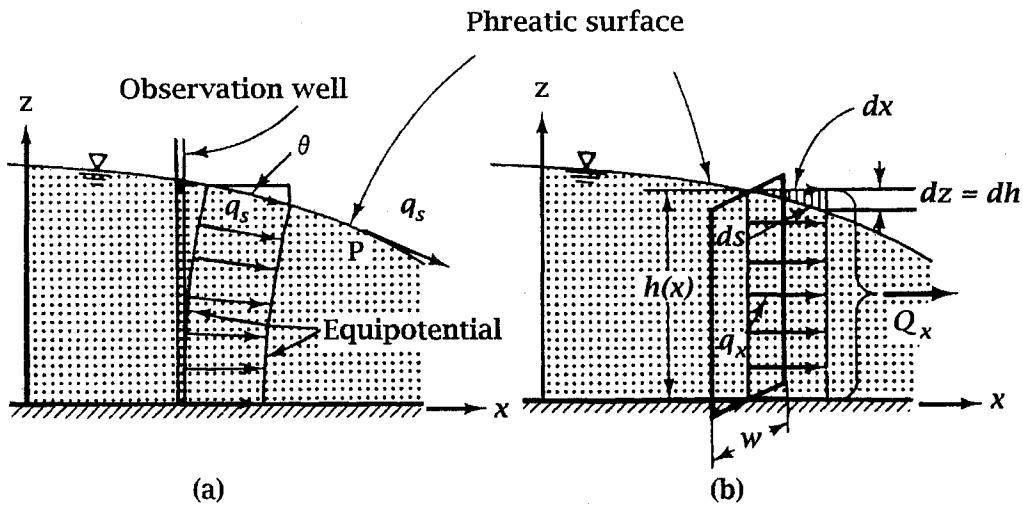
where both h_c and the effective particle diameter are in centimeters. Silin-Bekchurin [**94**] suggested a capillary rise of $2-5$ cm in coarse sand, $12-35$ cm in sand, $35-70$ cm in fine sand, $70-150$ cm in silt, and $2-4$ m and more in clay. Equations (1.9) and (1.10) can be compared with the relationship $h = 2\sigma/r$, which expresses the rise of water in a capillary tube of radius r; a is the surface tension of the water.

Both ϕ and $\boldsymbol{q}$ vary from point to point within a phreatic aquifer. In order to obtain the specific discharge $\boldsymbol{q} = \boldsymbol{q}(x, y, z, t)$ at every point, we have to know the piezometric head $\phi = \phi(x, y, z, t)$ by solving the flow model in a three-dimensional space. An

additional difficulty stems from the fact that the location of the phreatic surface, which serves as a boundary to the three-dimensional flow domain in the aquifer, is a *priori unknown*. In fact its location is part of the sought solution. Once we solve for $\phi = \phi(x, y, z, t)$ within the flow domain, we use the fact that on the phreatic surface, the pressure is zero to obtain $\phi(x, y, z, t) = z$ on the phreatic surface. Hence, the equation that describes the phreatic surface is

$$F(x, y, z, t) \equiv \phi(x, y, z, t) - z = 0. \tag{1.11}$$

FIGURE 1.6. The Dupuit assumption. [**13**]



From the above considerations it follows that this procedure is not a practical one for solving common problems of flow in phreatic aquifers.

In view of this inherent difficulty, Dupuit [**33**] observed that in most groundwater flows, the slope of the phreatic surface is very small. Slopes of 1/1000 and 10/1000 are commonly encountered. In steady flow without accretion in the vertical two-dimensional xz-plane (Figure 1.6a), the phreatic surface is a streamline. At every point, P, along this streamline, the specific discharge is in a direction tangent to the streamline and is given by Darcy's law

$$q_s = -\boldsymbol{K}\frac{\mathrm{d}\phi}{\mathrm{d}s} = -\boldsymbol{K}\frac{\mathrm{d}z}{\mathrm{d}s} = -\boldsymbol{K}\sin\theta, \tag{1.12}$$

since along the phreatic surface $p = 0$ and $\phi = z$. As θ is very small, Dupuit suggested that $\sin\theta$ be replaced by the slope $\tan\theta = \frac{\mathrm{d}h}{\mathrm{d}x}$. The assumption of small θ is equivalent to assuming that equipotential surfaces are vertical (i.e., $\phi = \phi(x)$ rather than $\phi = \phi(x, z)$) and the flow is essentially horizontal. Thus, the Dupuit assumption leads to the specific discharge expressed by

$$q_x = -\boldsymbol{K}\frac{\mathrm{d}h}{\mathrm{d}x}, \qquad h = h(x). \tag{1.13}$$

In general, $h = h(x, y)$ and we have

$$q_x = -\boldsymbol{K}\frac{\partial h}{\partial x}, \qquad q_y = -\boldsymbol{K}\frac{\partial h}{\partial y}. \tag{1.14}$$

Since $\boldsymbol{q}$ is thus independent of elevation, the corresponding total discharge through a vertical surface of width W (normal to the direction of flow; Figure 1.6b) is

$$Q_x = -\boldsymbol{K}Wh\frac{\partial h}{\partial x}, \qquad Q_y = -\boldsymbol{K}Wh\frac{\partial h}{\partial y}, \qquad h = h(x, y), \tag{1.15}$$

or, in the compact vector form

$$\boldsymbol{Q} = -\boldsymbol{K}Wh\nabla h. \tag{1.16}$$

Per unit width, we obtain

$$\boldsymbol{Q}' \equiv \boldsymbol{Q}/W = -\boldsymbol{K}h\nabla h. \tag{1.17}$$

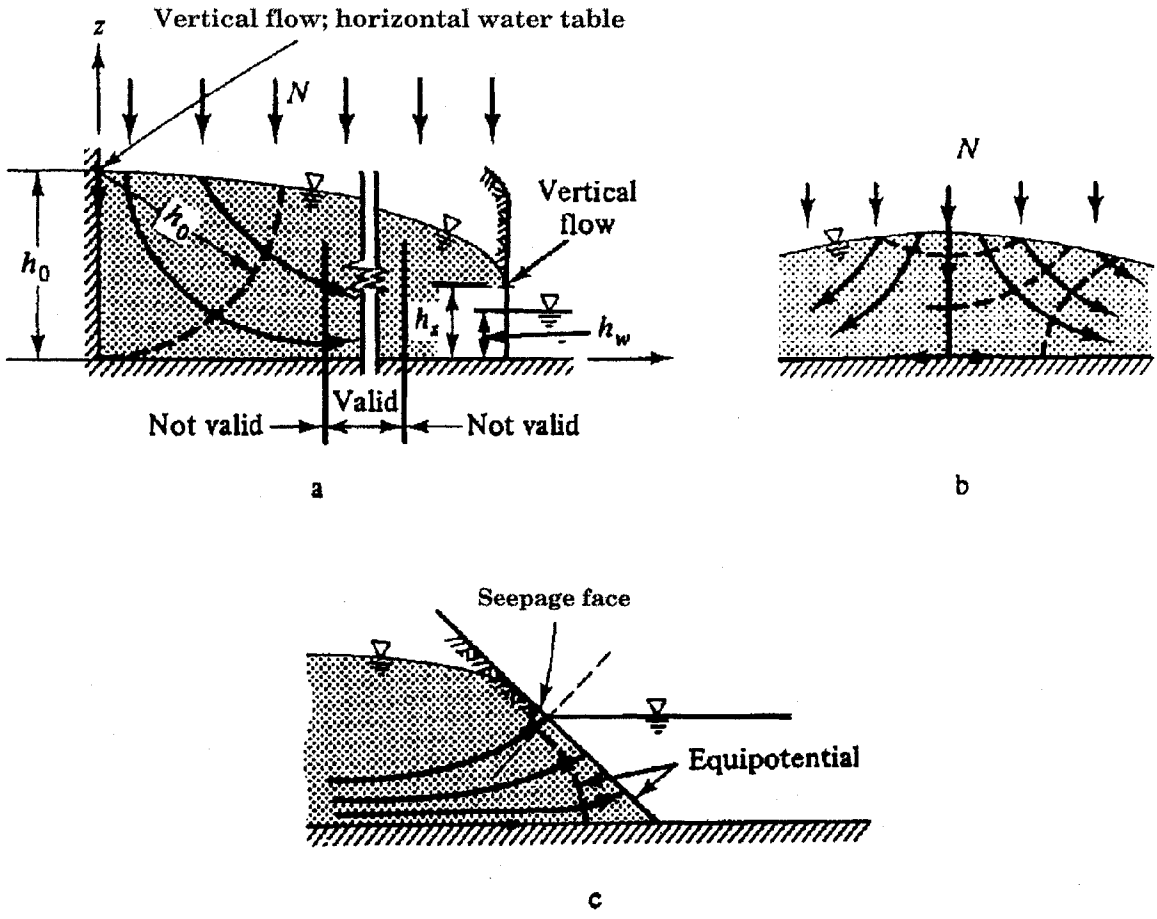
In (1.15) through (1.17), the aquifer's bottom is horizontal. it should be emphasized that the Dupuit assumption may be considered as a good approximation in regions where θ is indeed small and/or the flow is essentially horizontal. We note that the assumption of horizontal flow is equivalent to the assumption of *hydrostatic pressure distribution* $\partial p/\partial z = -\rho g$.

The important advantage gained by employing the Dupuit assumption is that the state variable $\phi = \phi(x, y, z)$ has been replaced by $h = h(x, y)$, i.e., z no longer appears as an independent variable. In addition, since at a point on the free surface, $p = 0$ and $\phi = h$, we assume that the vertical line through the point is also an equipotential line on which $\phi = h =$const. In general, h varies also with time so that

$h = h(x, y, t)$. In this way, the complexity of the problem has been greatly reduced. It is two-dimensional rather than three-dimensional, and the unknown location of the phreatic surface is no longer an extra complication.

The Dupuit assumption presented above is probably the most powerful tool for treating unconfined flows. In fact, it is the only simple tool available to most engineers and hydrologists for solving such problems.

FIGURE 1.7. Regions where Dupuit assumption is not valid. [13]



The Dupuit assumption should not be applied in regions where the vertical flow component is not negligible. Such flow conditions occur as a seepage face is approached (Figure 1.7c) or at a crest (*water divide*) in a phreatic aquifer with accretion (Figure 1.7b). Another example is the region close to the impervious vertical

boundary of Figure 1.7a. It is obvious that the assumption of vertical equipotentials fails at, and in the vicinity of, such a boundary. Only at a distance $x >\sim 2h_0$ have we equipotentials that may be approximated as vertical lines, or surfaces. It is important to note here that in cases with accretion, a horizontal (or almost so) water table is not sufficient to justify the application of the Dupuit assumption. One must verify that vertical flow components may indeed be neglected, before applying the Dupuit assumption.

1.2.3. Equation of continuity. The equation of continuity is a statement of the law of conservation of matter. When applicable, it states that mass can be neither created nor destroyed. It can be derived from the fact that the change in mass stored in a small, elemental, rectangular parallel-piped equals the difference between the mass entering and the mass leaving.

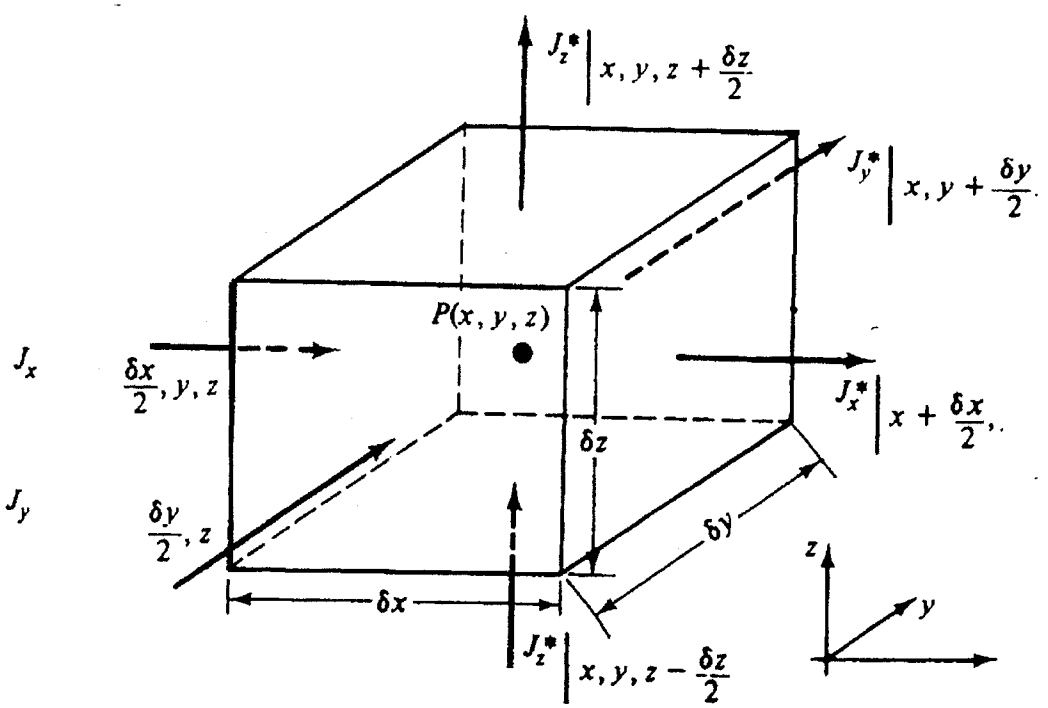
Consider a *control box* having the shape of a rectangular parallel-piped of dimensions dx,dy,dz centered at some point $P(x, y, z)$ inside the flow domain in an aquifer. A control box may be any arbitrary shape, but once its shape and position in space have been fixed, they remain unchanged during the flow, although the amount and identity of the material in it may change with time. In the present analysis, water and solids enter and leave the box through its surfaces. Our objective here is to write a balance equation for the mass of water entering, leaving, and being stored in the box. Let the vector $\mathbf{J} = \rho \boldsymbol{q}$ denote the mass flux (i.e., mass per unit area per unit time) of water of density ρ at point $P(x, y, z)$. It is easy to see that $\boldsymbol{q}$ is the specific discharge in the Darcy's law. Referring to Figure 1.8, the excess of inflow over outflow of mass during a short time interval dt, through the surfaces which are perpendicular to the x, y and z direction, can be expressed by the differences

$$dt\{J_x|_{x-dx/2,y,z} - J_x|_{x+dx/2,y,z}\}dydz,$$

$$dt\{J_y|_{x,y-dy/2,z} - J_y|_{x,y+dy/2,z}\}dzdx,$$

$$dt\{J_z|_{x,y,z-dz/2} - J_z|_{x,y,z+dz/2}\}dxdy.$$

FIGURE 1.8. Nomenclature for mass conservation for a control volume. [13]



The sum of the three expressions, for all three directions, is the total excess of mass inflow over outflow during dt. So the excess of inflow over outflow per unit volume of medium (around P) and per unit time is

$$-(\frac{\partial J_x}{\partial x}+\frac{\partial J_y}{\partial y}+\frac{\partial J_z}{\partial z})=-\nabla\cdot\rho\boldsymbol{q}.$$

On the other hand,

$$\lim_{\triangle t\to 0}\frac{(n\rho)|_{t+\triangle t}-(n\rho)|_t}{\triangle t}=\frac{\partial(n\rho)}{\partial t}$$

is the rate of change of the mass of the fluid per unit volume of porous medium where n is the porosity. So we get the fundamental balance equation

$$-\nabla\cdot\rho\boldsymbol{q}=\frac{\partial(n\rho)}{\partial t}. \tag{1.18}$$

With some assumptions about the fluid flow [**13**], $\frac{\partial(n\rho)}{\partial t}$ can be expressed as $\rho S_0 \frac{\partial \phi}{\partial t}$ where S_0 is called the *specific storativity* and is defined by

$$S_0 = \rho g(\alpha + n\beta), \tag{1.19}$$

where ρ is the density, g is the acceleration of gravity, n is the porosity of the porous medium, and β and α are derivatives of ρ and n with respect to the pressure p respectively. The specific storativity indicates the ability for the medium to hold the fluid. Substituting it into the balance equation, we get

$$-\nabla \cdot \rho \boldsymbol{q} \nabla \phi = \rho S_0 \frac{\partial \phi}{\partial t}. \tag{1.20}$$

If ρ is constant we have

$$-\nabla \cdot \boldsymbol{q} = S_0 \frac{\partial \phi}{\partial t}. \tag{1.21}$$

1.2.4. The flow equations. The balance equations, (1.18), (1.20), and (1.21) in section 1.2.3, do not include the recharge. If the distributed rates of artificial recharge, $R(x,t)$, and the pumping, $P(x,t)$, are added, the balance equation can be modified to

$$S_0 \frac{\partial \phi}{\partial t} = -\nabla \cdot \boldsymbol{q} + R - P. \tag{1.22}$$

Applying Darcy's law $\boldsymbol{q} = -\boldsymbol{K}\nabla\phi$, we get the flow equation in a confined aquifer

$$S_0 \frac{\partial \phi}{\partial t} = \nabla \cdot \boldsymbol{K} \nabla \phi + R - P. \tag{1.23}$$

We can also deduce the flow equation in an unconfined aquifer by applying the Dupuit assumption $\boldsymbol{q} = -\boldsymbol{K}h\nabla h$:

$$S_0 \frac{\partial h}{\partial t} = \nabla \cdot \boldsymbol{K} h \nabla h + R - P. \tag{1.24}$$

The equations above are three-dimensional. We can also get the two-dimensional equations by integrating the above equations over the z direction (see [**13**]). For

example, the two-dimensional flow equation for a confined aquifer is:

$$S\frac{\partial h}{\partial t} = \nabla \cdot (\boldsymbol{T}\nabla h) + q_{v1} - q_{v2} + R^{\star} - P^{\star}, \tag{1.25}$$

where

$$S(x,y) = \int_{b_1}^{b_2} S_0(x,y,z)dz \tag{1.26}$$

is the aquifer storativity,

$$T(x,y) = \int_{b_1}^{b_2} K(x,y,z)dz \tag{1.27}$$

is the aquifer transmissivity,

$$\begin{aligned} R^{\star}(x,y,t) &= \int_{b_1}^{b_2} R(x,y,z,t)dz, \\ P^{\star}(x,y,t) &= \int_{b_1}^{b_2} P(x,y,z,t)dz, \end{aligned}$$

are the source/sink terms and q_{v2}, q_{v1} denote the leakage rates of the upper and lower aquifers. Here b_2 and b_1 denote the elevations of the aquifer's top and bottom. For a confined aquifer without leakage, q_{v1} and q_{v2} will be zero. For an unconfined aquifer, the term ∇h in the above equation will be replaced by $h\nabla h$.

Note that the only term of the parameters that is dependent on the time in the flow equations is the source term $R - P$, and we usually denote it by one symbol R.

1.3. Hydrodynamic dispersion

One major problem, of interest in the development and management of any water resources system, is water quality. With the increased demand for water, the quality problem becomes the limiting factor in the use and development of water resources. Although it may seem that groundwater is more protected than surface water, it is still subject to pollution, and when this occurs, the restoration to the original, nonpolluted state is usually more difficult and lengthy.

We consider the mass of some substance contained in the groundwater as the transport mass that moves with the water in the interstices of a porous medium. The mechanisms affecting the transport of a pollutant in a porous medium are as follows: advective, dispersive, and diffusive fluxes; solid-solute interactions; and various chemical reactions and decay phenomena, which may be regarded as source-sink phenomena for the solute.

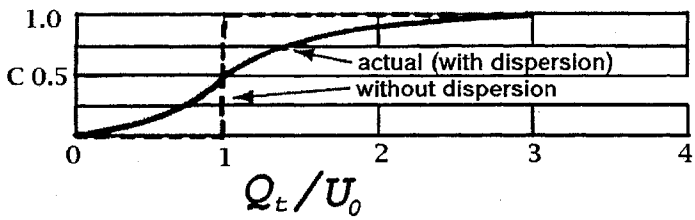
Consider saturated flow through a porous medium and let a portion of the flow domain contain a certain mass of solute. This solute will be referred to as a *tracer.* The tracer, which is a labeled portion of the same liquid, may be identified by its density, color, electrical conductivity, etc. Experience shows that, as flow takes place, the tracer gradually spreads and occupies an ever-increasing portion of the flow domain, beyond the region it is expected to occupy according to the average flow alone. This spreading phenomenon is called *hydrodynamic dispersion* in a porous medium. It is a nonsteady, irreversible process (in the sense that the initial tracer distribution cannot be obtained by reversing the flow), in which the tracer mass mixes with the unlabeled portion of the liquid. If initially the tracer-labeled liquid occupies a separate region, this interface does not remain an abrupt one. Instead, an ever-widening transition zone is created, across which the tracer concentration varies from that of the tracer liquid to that of the unmarked liquid.

One of the earliest observations of this phenomenon is reported by Slichter [**95**], who used an electrolyte as a tracer in studying the movement of groundwater. Slichter observed that at an observation well downstream of a (continuous) injection point, the tracer's concentration increases gradually, and that even in a uniform (average) flow field, the tracer advances in the direction of the flow in a pear-like shape that becomes longer and wider as it advances.

The dispersion phenomenon may also be demonstrated by a simple laboratory experiment. Consider steady flow in a cylindrical column of homogeneous sand, saturated with water. At a certain instant, $t = 0$, tracer-marked water (e.g., water

with NaCl at a low concentration, so that the effect of density variations on the flow pattern is negligible) starts to displace the original unlabeled water in the column. Let the tracer concentration, $C = C(t)$, be measured at the end of the column and presented in a graphic form, called a *breakthrough curve*, as a relationship between the relative tracer concentration and time, or volume of effluent, U.

FIGURE 1.9. Breakthrough curve in one-dimensional flow in a sand column. [13]



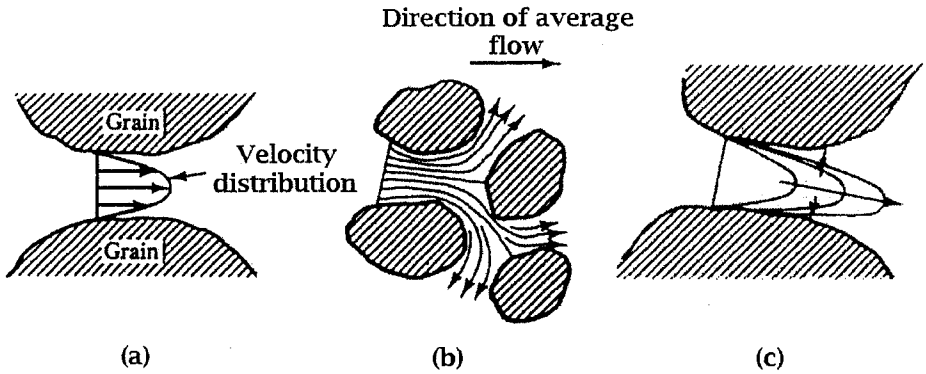
In the absence of dispersion, the breakthrough curve should have taken the form of the broken line shown in Figure 1.9, where U_0 is the pore volume of the column and Q is the constant discharge. Actually, owing to hydrodynamic dispersion, it will take the form of the S-shaped curve shown in full line in Figure 1.9.

We cannot explain all of the above observations on the basis of the average water flow. We must refer to what happens at the microscopic level, i.e., inside the pore cross-section. We usually assume zero fluid velocity on the solid surface, with a maximum velocity at some internal point (compare with the parabolic velocity distribution in a straight capillary tube). The maximum velocity itself varies according to the size of the pore. Because of the shape of the interconnected pore space, the (microscopic) streamlines fluctuate in space with respect to the mean direction of flow (Figure 1.10). This phenomenon causes the spreading of any initially close group of tracer particles; as flow continues, they occupy an ever increasing volume of the flow domain. The two basic factors that produce this kind of spreading are, therefore, flow and the presence of a pore system through which flow takes place.

Although this spreading is in both the longitudinal direction, namely that of the average flow, and in the direction transversal to the average flow, it is primarily in

the former direction. Very little spreading can be caused in a direction perpendicular to the average flow by velocity variations alone. Such velocity variations alone also cannot explain the ever-growing width of the zone occupied by dispersed tracer particles normal to the direction of flow. In order to explain this spreading, we have to refer to *molecular diffusion*, an additional phenomenon that take place in the void space.

FIGURE 1.10. Spreading due to mechanical dispersion (a,b) and molecular diffusion(c). [13]



Molecular diffusion, caused by the random movement of molecules in a fluid, produces an additional flux of tracer particles (at the microscopic level) from regions of higher tracer concentrations to those of lower ones. This means, for example, that as the marked particles spread along each microscopic streamtube, as a result of velocity variations, a concentration gradient of these particles is produced, which in turn produces a flux of tracer by the mechanism of molecular diffusion. The latter phenomenon tends to equalize the concentrations along the steamtube. Relatively, this is a minor effect. However, at the same time, a tracer concentration gradient will also be produced between adjacent streamlines, causing *lateral molecular diffusion across streamtubes* (Figure 1.10c), tending to equalize the concentration across pores. It is this phenomenon that explains the observed transversal dispersion.

In addition to the role played at the microscopic level by molecular diffusion in enhancing the transversal component of mechanical dispersion, it produces macroscopic

flux of its own. This is easily demonstrated by letting the velocity vanish. Then the tracer is transported by (macroscopic) molecular diffusion only.

We shall refer to the spreading caused by the velocity variations at the microscopic level, enhanced by molecular diffusion, as *mechanical dispersion.*

We use the term *hydrodynamic dispersion* to denote the spreading (at microscopic level) resulting from both mechanical dispersion and molecular diffusion. Actually, the separation between the two processes is rather artificial, as they are inseparable. However, molecular diffusion alone does also takes place in the absence of motion (both in a porous medium and in a fluid continuum). Because molecular diffusion depends on time, its effect on the overall dispersion is more significant at low velocities. It is molecular diffusion that makes the phenomenon of hydrodynamic dispersion in purely laminar flow irreversible.

In addition to inhomogeneity on a microscopic scale (i.e., presence of pores and grains), we may also have inhomogeneity on a macroscopic scale, due to variations in permeability from one portion of the flow domain to the next. This inhomogeneity also produces dispersion of marked particles, but on a much larger scale.

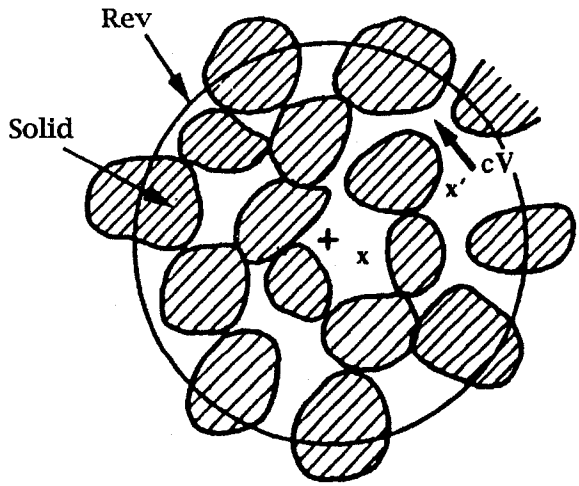
Dispersion may take place both in a laminar flow regime, where the liquid moves along definite paths that may be averaged to yield streamlines, and in a turbulent regime, where the turbulence may cause yet an additional mixing. In what follows, we shall focus our attention only on flow of the first type.

In addition to advection (at average velocity), mechanical dispersion, and molecular diffusion, several other phenomena may affect the concentration distribution of a tracer as it moves through a porous medium. The tracer (say, a solute) may interact with the solid surface of the porous matrix in the form of absorption of tracer particles *on the solid surface*, deposition, solution of the solid matrix, or ion exchange. All these phenomena cause changes in the concentration of a tracer in a flowing liquid. Radioactive decay and chemical reactions within the liquid also cause tracer concentration changes.

In general, variations in tracer concentration cause changes in the liquid's density and viscosity. These, in turn, affect the flow regime (i.e., velocity distribution) that depends on these properties. We use the term *ideal tracer* when the concentration of the latter does not affect the liquid's density and viscosity. At relatively low concentrations, the ideal tracer approximation is sufficient for most practical purposes. However, in certain cases, for example in the problem of sea water intrusion, the density may vary appreciably, and the ideal tracer approximation should not be used.

1.3.1. Advective, dispersive, and diffusive fluxes. As explained above, at every (microscopic) point within a porous medium domain, we have a velocity $\boldsymbol{V}$ and a concentration, c, of some considered substance; c expresses the mass of the substance per unit volume of the liquid. Figure 1.11 shows a point x' belonging to an REV centered at point x. The product $c\boldsymbol{V}$ at x' denotes the local flux (= quantity of the considered substance per unit area of liquid) vector at that point. However, we already know that we cannot predict values of $\boldsymbol{V}$ and c at this microscopic level, and that, instead, we should aim at predicting the average concentration, $\bar{c}$, and the average tracer flux, $\overline{c\boldsymbol{V}}$, at the macroscopic level. To achieve this goal, without going

FIGURE 1.11. Nomenclature for the dispersive flux. [**13**]



into the details of the continuum approach to transport in porous media, let the liquid's velocity at an arbitrary point, x', within the liquid that completely occupies

the pore space, be denoted by $\boldsymbol{V}(x', t; x)$. The symbol x in this parenthesis indicates that point x' belongs to an REV centered at x (Figure 1.11). The velocity, $\boldsymbol{V}$, can be decomposed into two parts: the average velocity, $\overline{\boldsymbol{V}}$, of the liquid within the REV, and a deviation, $\boldsymbol{V}^o$, from that average. Thus

$$\boldsymbol{V}(x', t; x) = \overline{\boldsymbol{V}}(x, t) + \boldsymbol{V}^o(x', t; x), \tag{1.28}$$

$$c(x', t, x) = \overline{c}(x, t) + c^o(x', t; x). \tag{1.29}$$

In both cases, the average has the meaning of an intrinsic phase average as defined by (1.1).

To obtain the average flux, we write

$$\overline{c\boldsymbol{V}} = \overline{(\overline{c} + c^o)(\overline{\boldsymbol{V}} + \boldsymbol{V}^o)} = \overline{\overline{c}\,\overline{\boldsymbol{V}}} + \overline{\overline{c}\boldsymbol{V}^o} + \overline{c^o\overline{\boldsymbol{V}}} + \overline{c^o\boldsymbol{V}^o}. \tag{1.30}$$

However, in view of (1.1), $\overline{c^o\overline{\boldsymbol{V}}} = 0$ and $\overline{\overline{c}\boldsymbol{V}} = 0$. Hence

$$\overline{c\boldsymbol{V}} = \overline{c}\overline{\boldsymbol{V}} + \overline{c^o\boldsymbol{V}^o}, \tag{1.31}$$

i.e., the average flux of the considered substance is equal to the sum of two macroscopic fluxes:

a) An *advective flux*, $\overline{c}\overline{\boldsymbol{V}}$, expressing the flux carried by the water at the latter's average velocity, $\overline{\boldsymbol{V}}$, as determined by Darcy's law.

b) A flux $\overline{c^o\boldsymbol{V}^o} = \overline{c\boldsymbol{V}^o}$ expressing an additional flux resulting from the fluctuating velocity in the vicinity (i.e., within the REV) of the considered point. Recalling the discussion in the previous section, this is the flux that produces the spreading, or dispersion. We refer to it as the dispersive flux. It is a macroscopic flux that expresses the effect of the microscopic variations of the velocity in the vicinity of a considered point. We note that this flux is created by the averaging procedure. It does not exist at the microscopic level. In employing this flux, we are losing the information about the behavior at the microscopic level (which we do not have anyway).

1.3.2. Mechanical dispersion. Our next objective is to express the dispersive flux in terms of averaged (and measurable) quantities, such as averaged velocity and averaged concentration. Investigations over a period of about two decades, starting from the mid-50s (see review, for example, in Bear, [**9**]), have led to the *working assumption* that the dispersive flux can be expressed as a Fickian type law; i.e., in the form

$$\overline{c^o \boldsymbol{V}^o} = -\boldsymbol{D} \cdot \nabla \bar{c}; \qquad \overline{c^o V_i^o} = -D_{ij} \frac{\partial \bar{c}}{\partial x_j}; \tag{1.32}$$

where $\boldsymbol{D}$ is a second rank symmetric tensor called the *coefficient of (mechanical) dispersion.* We recall that $\bar{c}$ denotes the mass of the dispersing substance per unit volume of water, and $\overline{c^o \boldsymbol{V}^o}$ represents a flux per unit area of the water. Equation (1.32) indicates that the dispersive flux is linearly proportional to the gradient of the average concentration and that this flux takes place from high concentrations to lower ones.

Several authors (e.g., Nikolaevskii [**76**], Bear [**8**], Scheidegger [**92**], Bear and Bachmat [**10**]) derived the following expression for the relationship between the coefficient $\boldsymbol{D}$ and microscopic porous matrix configuration, flow velocity, and molecular diffusion

$$D_{ij} = \sum_{k,m} a_{ijkm} \frac{\overline{V}_k \overline{V}_m}{\overline{V}} f(Pe, \delta), \tag{1.33}$$

where $\overline{V} = |\overline{\boldsymbol{V}}|$ is the average velocity, Pe is the Peclet number defined as $Pe = L\overline{V}/D_d$, L is some characteristic length of the pores, D_d is the coefficient of molecular diffusion of the solute in the liquid phase, δ is the ratio of the length characterizing the individual pores of a porous medium to the length characterizing their cross-section, and $f(Pe, \delta)$ is a function which introduces the effect of tracer transfer by molecular diffusion between adjacent streamlines at the microscopic level. In this way, molecular diffusion affects mechanical dispersion. One should not identify this effect with the macroscopic flux due to molecular diffusion (see below), but with the

transfer between streamtubes at the microscopic level, as explained in the definition of mechanical dispersion in the previous section. Bear and Bachmat [**10**] suggested the relationship $f(Pe,\delta) = Pe/(Pe + 2 + 4\delta^2)$. In most cases, it is assumed that $f(Pe,\delta) \approx 1$. Henceforth, we shall also make this assumption.

The coefficient a_{ijkm}, (dims. L), called the dispersivity of the porous medium, is a fourth-rank tensor which expresses the microscopic configuration of the solid-liquid interface. Bear and Bachmat [**10**] and Bear [**9**, page 614] express a_{ijkm} by

$$a_{ijkm} = [\overline{\sum_p (VT'_{ij})^o (BT'_{jp})^o / (BT'_{lk})^o (BT'_{pm})^o}]L, \tag{1.34}$$

where B is the conductance of an elementary medium channel, BT'_{ij} is an oriented conductance of a channel, $\overline{T'_{ij}}$ is the medium's *tortuosity*, $n\overline{BT'_{ij}} = k_{ij}$ is the medium's permeability, and L is a characteristic length of the medium. Thus, the medium's dispersivity is related to the variance of $\overline{(BT'_{ij})}^o$, while its permeability is related to the average, $\overline{BT'_{ij}}$, of BT'_{ij}.

A fourth rank tensor has 81 components in a three-dimensional space (and 16 in a two-dimensional one). Scheidegger [**92**] and Bear [**9**] showed that a_{ijkm} has a number of symmetries that reduce the number of nonzero components of the dispersivity tensor, in a three-dimensional space, to only 36.

For an *isotropic porous medium*, the number of nonzero components is further reduced to 21. Furthermore, these 21 components are related to two parameters: a_L (dim. L), called the *longitudinal dispersivity* of the isotropic porous medium, and a_T (dim. L), called the *transversal dispersivity*. In the theoretical developments mentioned above, it is shown that a_L expresses the heterogeneity of the porous medium at the microscopic scale due to the presence of pores and solids. Hence, in laboratory experiments in homogeneous sand columns it was found that a_L is of the order of magnitude of the average sand grain. The transversal dispersivity is estimated as being 10 to 20 times smaller than that of a_L.

With a_L and a_T, the components of the dispersivity for an isotropic porous medium can be expressed in the form

$$a_{ijkm} = a_T \delta_{ij}\delta_{km} + \frac{a_L - a_T}{2}(\delta_{ik}\delta_{jm} + \delta_{im}\delta_{jk}), \tag{1.35}$$

where

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{otherwise,} \end{cases}$$

is the Kroenecker delta. For an isotropic porous medium, the components a_{ijkm} do not change under rotation of the coordinate system.

For an *anisotropic porous medium with axial symmetry*, e.g., a medium made up of a large number of thin layers normal to the axis of symmetry, the dispersivity can be expressed in the form

$$\begin{aligned} a_{ijkm} &= a_I \delta_{ij}\delta_{km} + a_{II}(\delta_{ik}\delta_{jm} + \delta_{im}\delta_{jk}) \\ &+ a_{III}(\delta_{ij}h_k h_m + \delta_{km}h_i h_j) \\ &+ a_{IV}(\delta_{ik}h_j h_m + \delta_{jk}h_i h_m + \delta_{im}h_j h_k + \delta_{jm}h_i h_k) \\ &+ a_V h_i h_j h_k h_m, \end{aligned}$$

where a_I,a_{II},a_{III},a_{IV} and a_V are five independent parameters and h is a unit vector directed along the axis of symmetry. Similar expressions can be written for other types of anisotropy.

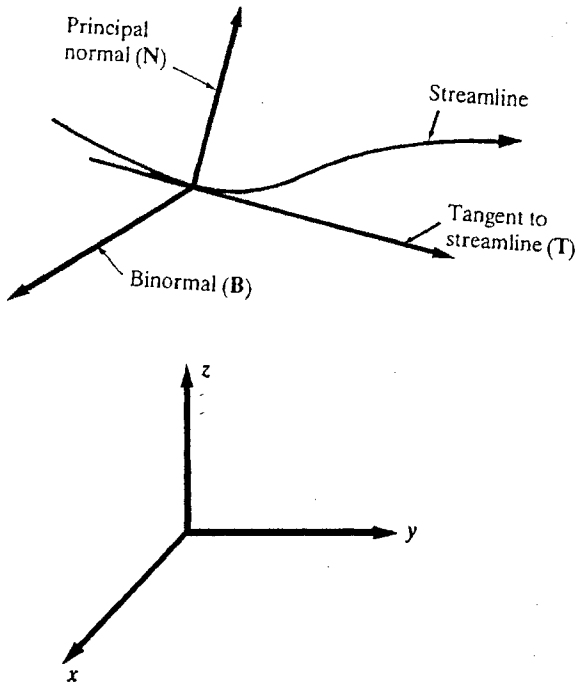
By combining (1.33) with (1.35) for $f(Pe, \delta) = 1$, we obtain

$$D_{ij} = a_T V \delta_{ij} + (a_L - a_T)V_i V_j / V, \tag{1.36}$$

where here, and henceforth, we have omitted the overline symbol that indicates the velocity is an average one.

The permeability, k_{ij}, of a porous medium is also a second-rank symmetric tensor. However, there is a basic difference between tensors k_{ij} and D_{ij}. In an isotropic porous medium, any three mutually orthogonal directions in space may serve as principal directions. However, due to the effect of the velocity pattern, one of the principal

FIGURE 1.12. Principal axes of the coefficient of dispersion. [13]



axes of the dispersion coefficient, D_{ij}, at a point, is always in the direction of the tangent of the streamline passing through that point. The other two principal axes are in the directions of the two principal normals to this direction. Figure 1.12 shows these directions. The unit vectors $\boldsymbol{N}$, $\boldsymbol{T}$, and $\boldsymbol{B}$ are called the *principal normal*, the *tangent*, and the *binomial* to the curve.

Thus, although the porous medium is isotropic, we have a distinct set of principal directions at every point of a flow domain. As the velocity varies from point to point, so do the principal axes of the dispersion. Furthermore, at every point, these directions may vary continuously as the flow pattern varies. This dependence of the dispersion coefficient on the velocity introduces a major difficulty in the solution of pollution problems, especially under unsteady flow conditions and when the velocity is density (and hence, concentration) dependent.

1.3.3. Molecular diffusion. At the microscopic level, the flux vector, $\boldsymbol{J}^{(d)}$, due to molecular diffusion is expressed by Fick's law

$$\boldsymbol{J}^{(d)} = -D_d \nabla c, \tag{1.37}$$

where D_d is the coefficient of molecular diffusion in a fluid continuum (equals about $10^{-5}\text{cm}^2/\text{sec}$ in dilute systems). By averaging (1.37) over the REV, and introducing certain simplifying assumptions, Bear and Bachmat [**11, 12**] derived an expression for the macroscopic flux in the form

$$\overline{\boldsymbol{J}^{(d)}} = -D_d \boldsymbol{T}^\star \cdot \nabla \overline{c} = -\boldsymbol{D}_d^\star \cdot \nabla \overline{c}, \tag{1.38}$$

where $\boldsymbol{D}_d^\star = \boldsymbol{T}^\star D_d$ is the *coefficient of molecular diffusion in a porous medium* and $\boldsymbol{T}^\star$ is a second-rank symmetric tensor that expresses the effect of the configuration of the water-occupied portion of the REV. We used the averaging symbol in (1.38) in order to emphasize the difference between this equation and (1.37).

The coefficient $\boldsymbol{T}^\star$, often referred to as a tortuosity, is defined by (Bear and Bachmat [**11, 12**])

$$T_{ij}^\star = \frac{1}{U_{0w}} \int_{S_{ww}} (x_j - x_{0j}) \nu_i dS, \tag{1.39}$$

where S_{ww}, denotes the water-water portion of the bounding surface of the REV, x_0 is the centroid of the REV, $\mathbf{v}$ is the outwardly directed normal to the surface S_{ww}, and U_{0w} denotes the volume occupied by water within the REV.

For an isotropic porous medium, $T^\star$ reduces to

$$T_{ij}^\star = \frac{\theta_w^s}{\theta_w} \delta_{ij}, \tag{1.40}$$

where $\theta_w^s = S_{ww}/S_0$, $\theta_w = U_{0w}/U_0$, and δ_{ij} is the Kroenecker delta.

1.3.4. Coefficient of hydrodynamic dispersion. By adding the dispersive flux, expressed by (1.32), and the diffusive flux, expressed by (1.38), we obtain

$$\overline{c^o \boldsymbol{V}^o} + \overline{\boldsymbol{J}^{(d)}} = -(\boldsymbol{D} + \boldsymbol{D}_d^\star) \cdot \nabla \overline{c} = -\boldsymbol{D}_h \cdot \nabla \overline{c}, \tag{1.41}$$

where the coefficient $\boldsymbol{D}_h = \boldsymbol{D} + \boldsymbol{D}_d^\star$ is called the *coefficient of hydrodynamic dispersion.*

The total flux, $\boldsymbol{q}_{c,\text{total}}$, of a pollutant, by advection, dispersion, and diffusion, can now be written in the form

$$\boldsymbol{q}_{c,\text{total}} = \theta_w(\bar{c}\overline{\boldsymbol{V}} - \boldsymbol{D}_h \cdot \nabla\bar{c}). \tag{1.42}$$

This is the amount per unit time of the pollutant passing through a unit area of porous medium.

1.3.5. Balance equation for a pollutant. Five components should be taken into account in the construction of a balance equation for a constituent ([**13**]).

a) The quantity of the pollutant entering and leaving a control volume around a considered point by advection dispersion and diffusion, or the total flux, $\boldsymbol{q}_{c,\text{total}}$, expressed by (1.42).

We recall that in Section 1.3.1, using a parallelpiped control box, we have shown that *the negative divergence of a flux (of any extensive quantity) represents the excess of inflow (of that quantity) over outflow, per unit volume of porous medium, per unit time.* Hence, here $-\text{div}\,\boldsymbol{q}_{c,\text{total}}$, total represents the excess of inflow of a considered pollutant over outflow, per unit volume of porous medium, per unit time.

b) Pollutant leaving the fluid phase through the water-solid interface as a result of chemical or electrical interactions between the pollutant and the solid surface. Phenomena of *ion exchange* and *absorption* may serve as examples. Let f denote the quantity of pollutant that leaves the water by such mechanisms, per unit volume of porous medium, per unit time.

c) Pollutant added to the water (or leaving it) as a result of chemical interactions among species *inside* the water, or by various decay phenomena. Let Γ denote the rate at which the mass of a pollutant is added to the water per unit mass

of fluid, and θ be the moisture content (so that $\theta\rho\Gamma$ denotes the mass added by such phenomena, per unit volume of porous medium per unit time).

d) Pollutant may be added by injecting polluted water into a porous medium domain, e.g., as part of artificial recharge or waste disposal operations. Pollutant may be removed from a porous medium domain by withdrawing (polluted) water, e.g., by pumping. With $P(x,t)$ and $R(x,t)$ denoting the rates of water withdrawn or added, respectively, per unit volume of porous medium per unit time, and $c(x,t)$ and $c_R(x,t)$ denoting the pollutant's concentration in the water present in the porous medium and in the water added by injection, respectively, the total quantity of pollutant added per unit volume of porous medium per unit time is expressed by $Rc_R - Pc$.

e) As a result of the above components, the quantity of the pollutant is increased within a control box. With θc denoting the mass of a pollutant per unit volume of porous medium, $\frac{\partial(\theta c)}{\partial t}$ denotes the rate at which this quantity increases.

Combining all the components, we obtain

$$\frac{\partial \theta c}{\partial t} = -\nabla \cdot \boldsymbol{q}_{c,\text{total}} - f + \theta\rho\Gamma - Pc + Rc_R, \tag{1.43}$$

or, using (1.42) to express $\boldsymbol{q}_{c,\text{total}}$,

$$\frac{\partial \theta c}{\partial t} = -\nabla \cdot (c\boldsymbol{q} - \theta \boldsymbol{D}_h \cdot \nabla c) - f + \theta\rho\Gamma - Pc + Rc_R. \tag{1.44}$$

Equation (1.44) is the (macroscopic) mass balance equation of a pollutant, expressed in terms of $c = c(x,t)$. It is often called the *equation of hydrodynamic dispersion*, or the *advection - dispersion equation.*

The previous equation is a general case of unsaturated flow. For saturated flow, θ is replaced by the porosity, n.

CHAPTER 2

The Mathematical Model

2.1. Introduction

As stated in the previous chapter, the saturated flow and single-phase solute transport in groundwater systems can be modelled by the equations

$$S(x)\frac{\partial \phi}{\partial t} = -\nabla \cdot \boldsymbol{q} + R(x,t), \tag{2.1}$$

$$\frac{\partial(\theta c)}{\partial t} = -\nabla \cdot (c\boldsymbol{q}) + \nabla \cdot (\theta \boldsymbol{D} \nabla c) + B, \tag{2.2}$$

over x in a bounded region $\Omega \subset R^n$, $n = 2$, or 3, and for $t > 0$. Here, $\phi(x,t)$ is the piezometric head, $c(x,t)$ is the solute concentration, S is the specific storativity, θ is the porosity, $\boldsymbol{D}$ is the hydrodynamic dispersion tensor, and $R(x,t)$, B are the source/sink terms for the flow and solute, respectively. The term $\boldsymbol{q}$ is the specific discharge. When applying Darcy's law $\boldsymbol{q} = -\boldsymbol{K}\nabla\phi$, we get the model equations in a *confined* aquifer:

$$S(x)\frac{\partial \phi}{\partial t} = \nabla \cdot (\boldsymbol{K} \nabla \phi) + R(x,t), \tag{2.3}$$

$$\frac{\partial(\theta c)}{\partial t} = \nabla \cdot (c\boldsymbol{K} \nabla \phi) + \nabla \cdot (\theta \boldsymbol{D} \nabla c) + B. \tag{2.4}$$

The model equations in an *unconfined* (*phreatic*) aquifer can be obtained by applying the Dupuit assumption $\boldsymbol{q} = -\boldsymbol{K}\phi\nabla\phi$,

$$S(x)\frac{\partial \phi}{\partial t} = \nabla \cdot (\phi \boldsymbol{K} \nabla \phi) + R(x,t), \tag{2.5}$$

$$\frac{\partial(\theta c)}{\partial t} = \nabla \cdot (c\phi \boldsymbol{K} \nabla \phi) + \nabla \cdot (\theta \boldsymbol{D} \nabla c) + B. \tag{2.6}$$

Note: we changed some symbols here so that it is convenient for us in the later discussion.

A fundamentally important part of the modelling process is the full reconstruction problem, i.e., the problem of obtaining reliable estimates for *all* of the various coefficient functions appearing in equations (2.1) and (2.2) from field measurements of the quantities ϕ and c (together with some ancillary data, such as boundary data on $\boldsymbol{K}$ and $\boldsymbol{D}$).

Many of the methods that have been employed on the inverse groundwater problem typically focus only on the recovery of the scalar (*isotropic*) hydraulic conductivity. These methods range from educated guesswork (referred to as "trial and error calibration" in the hydrology literature, the method still preferred by some practitioners [5, page 226]) to various attempts at "automatic calibration" ([**7, 62, 102, 103**] for survey materials; see also [**19, 20, 23, 69, 78**]). Some people [**89, 91**] have tried the direct approach of viewing the steady state version of (2.3) as a first-order hyperbolic equation in the conductivity; in addition to the fact that one must somehow integrate, in stable fashion, along the characteristic curves (which depend on $\nabla\phi$), this requires that one know the inflow part of the boundary, the determination of which is itself a non-trivial ill-posed problem. Another approach is to reformulate the problem as an optimization exercise, which can be done in several ways. One can work directly to minimize the "equation error," as in [**34, 35, 36, 47, 48, 62, 72, 91, 97**], or minimize over an "output error," as in [**21, 22, 37, 41, 65**]. The output error methods are applicable when the number of observations is limited, but suffer badly from non-uniqueness problems, as well as numerical instabilities. Another optimization route makes use of the general idea of Tikhonov regularization [**70, 74**]; examples include [**1, 18, 63, 98**]. All Tikhonov regularization methods make use of a regularization parameter whose critical value must be known quite accurately for the method to be effective. This general class of methods is less effective because of the lack of reliable methods for determining this critical value in practical situations; this problem can be even more pronounced in the aquifer case due to the uncertainties in the available data. A different regularization, asymptotic regularization, is employed in [**4, 43**]. In

the last two decades much work has appeared with the aim of applying geostatistical techniques [**46**] to the aquifer problem; examples include [**2, 27, 42, 48, 74, 89, 96**].

A further point worthy of note is that in the current literature there are few universally applicable techniques for recovering the specific storage and even fewer viable methods available [**5**, page 153] for objectively assigning values to a time-dependent recharge term. Once again, rainfall is not readily measured as a local phenomenon, and the effect of supply and discharge from underground sources is even more difficult to measure directly. There are also essentially no viable methods for objectively obtaining the full hydraulic conductivity tensor.

It is evident that obtaining the dispersion tensor $\boldsymbol{D}$ in equation (2.2) is even more difficult [**48**, page 2219]. Recall that the movement of the contaminant fluid in a groundwater system can be divided into three mechanisms (see Section 1.3): *advection*, *convective dispersion*, and *molecular diffusion*. Advection is represented by the first term on the right-hand side of equation (2.2); the sum of the convective dispersion and molecular diffusion is the coefficient $\boldsymbol{D}$ of equation (2.2). Note that the convective dispersion itself is a combination of the *longitudinal dispersivity*, a_L, and *transversal dispersivity*, a_T, for an isotropic porous medium. It is a combination of five independent parameters for the anisotropic case (see Section 1.3). So the full reconstruction problem for the groundwater model is a computationally formidable inverse problem.

In this dissertation, we will extend the work of [**64**] to a full groundwater model that can recover all the parameter functions in the flow equation (2.1) and the transport equation (2.2) for both confined and unconfined aquifers. Observe that a major difficulty encountered in the process is that these parameters can be very poorly represented by measurements taken at a fixed collection of points in an aquifer. This is because quantities such as hydraulic conductivity, for example, can vary by up to 12 orders of magnitude at a given site [**5**], due in part to the presence of significant geological inhomogeneities. In order to reliably model the flow of materials through

a porous medium, one has to somehow assign appropriate averaged values for these parameters determined in a suitable way from the flow itself [**5**, page 329].

2.2. The flow equations

Recall that the source term R in the flow equation is time dependent. This makes the problem more complicated. For simplicity, we assume that R is a piecewise constant function with respect to the time variable t, i.e.,

$$R(x,t) = \sum_{i=1}^{N} R_i(x)\chi_{[t_{i-1},t_i]} \tag{2.7}$$

where $0 = t_0 < t_1 < \cdots < t_N = 1$. This assumption is justified since

a) the data is available for only a limited time period, and it is difficult to monitor field data changing continuously in time;

b) R is generally a slowly varying function of time; and

c) from a mathematical point of view, it will converge to the real case when the time step is tends to 0.

Laplace transforming equation (2.3) in t over $[t_{i-1}, t_i]$, $i = 1, 2, \cdots, n$, we get N equations

$$-\nabla \cdot (\boldsymbol{K}(x)\nabla u_i) + \{\lambda u_i + \alpha_{i,1}(x,\lambda)\}S(x) = \beta_{i,1}(x,\lambda)R_i(x), \tag{2.8}$$

where

$$u_i(x,\lambda) = \int_{t_{i-1}}^{t_i} e^{-\lambda t}\phi(x,t)dt, \tag{2.9}$$

$$\alpha_{i,1}(x,\lambda) = \phi(x,t_i)e^{-\lambda t_i} - \phi(x,t_{i-1})e^{-\lambda t_{i-1}}, \tag{2.10}$$

$$\beta_{i,1}(x,\lambda) = \frac{1}{\lambda}(e^{-\lambda t_{i-1}} - e^{-\lambda t_i}), \tag{2.11}$$

$i = 1, 2, \cdots, N$.

For the flow equation (2.5) of an unconfined aquifer, after the Laplace transformation above, we have

$$-\nabla \cdot (\boldsymbol{K}(x)\nabla w_i) + \{\lambda w_i + \alpha_{i,2}(x,\lambda)\}S(x) = \beta_{i,2}(x,\lambda)R_i(x), \tag{2.12}$$

where

$$w_i(x,\lambda) = \int_{t_{i-1}}^{t_i} e^{-\lambda t}\phi^2(x,t)dt, \tag{2.13}$$

$$\alpha_{i,2}(x,\lambda) = \lambda(2u_i - w_i) + 2\{\phi(x,t_i)e^{-\lambda t_i} - \phi(x,t_{i-1})e^{-\lambda t_{i-1}}\}, \tag{2.14}$$

$$\beta_{i,2}(x,\lambda) = \frac{2}{\lambda}(e^{-\lambda t_{i-1}} - e^{-\lambda t_i}), \tag{2.15}$$

$i = 1, 2, \cdots, N$.

It is simple to compute values $u_i(x,\lambda)$ and $w_i(x,\lambda)$ from the known data $\phi(x,t)$ with fixed λ. Thus we arrive at a new problem: given data $u_i(x,\lambda)$ (as well as $w_i(x,\lambda)$ for an unconfined aquifer) for x in Ω and all $\lambda > 0$ (and the boundary value of $\boldsymbol{K}$), determine the functions $\boldsymbol{K}$, S, and R_i, $i = 1, 2, \cdots, N$.

2.3. The transport equations

Consider the transport equations (2.4) and (2.6). Assuming that the hydraulic conductivity $\boldsymbol{K}$ is known, then the first term of the right-hand side of equations (2.4) and (2.6) are known data. The coefficient hydrodynamic dispersion tensor, $\boldsymbol{D}$, is a very complicated combination of some components (see Section 1.3). Here we adopt a somewhat different approach. Consider $\boldsymbol{D}$ as a function of $\boldsymbol{q}$; i.e., $\boldsymbol{D} = \boldsymbol{D}(\boldsymbol{q})$, where the specific discharge $\boldsymbol{q}$ is dependent on time (although the hydrodynamic dispersion tensor $\boldsymbol{D}$ itself is time independent). So the term $\boldsymbol{D}(\boldsymbol{q})$ in the transport equations (2.4) and (2.6) are actually time dependent. Adopting the technique as in the flow equations, we assume

$$\boldsymbol{D}(x,t) = \sum_{i=1}^{N_1} \boldsymbol{D}_i(x)\chi_{[t_{i-1},t_i]}. \tag{2.16}$$

The source/sink term B is very complicated (see Section 1.3.5). For simplicity, we assume here that $B = B^1(x,t)c + B^2(x,t)$. Similar to the discussion above, we assume

$$B^1(x,t) = \sum_{i=1}^{N_2} B_i^1(x)\chi_{[t_{i-1},t_i]}, \tag{2.17}$$

$$B^2(x,t) = \sum_{i=1}^{N_3} B_i^2(x)\chi_{[t_{i-1},t_i]}. \tag{2.18}$$

Without loss of generality, we can assume $N_1 = N_2 = N_3$. Now, applying the finite Laplace transformation to equation (2.2) in t over $[t_{i-1}, t_i]$, we have

$$-\nabla\cdot(\theta(x)\boldsymbol{D}_i(x)\nabla v_i) + \{\lambda v_i + \alpha_i(x,\lambda)\}\theta(x) = \beta_i(x,\lambda)B_i^1(x) + \gamma_i(x,\lambda)B_i^2(x) + \delta_i(x,\lambda), \tag{2.19}$$

where

$$v_i(x,\lambda) = \int_{t_{i-1}}^{t_i} e^{-\lambda t}c(x,t)dt, \tag{2.20}$$

$$\alpha_i(x,\lambda) = c(x,t_i)e^{-\lambda t_i} - c(x,t_{i-1})e^{-\lambda t_{i-1}}, \tag{2.21}$$

$$\beta_i(x,\lambda) = v_i(x,\lambda), \tag{2.22}$$

$$\gamma_i(x,\lambda) = \frac{1}{\lambda}[e^{-\lambda t_{i-1}} - e^{-\lambda t_i}], \tag{2.23}$$

$$\delta_i(x,\lambda) = -\int_{t_{i-1}}^{t_i} \nabla\cdot(c\boldsymbol{q})e^{-\lambda t}dt, \tag{2.24}$$

$i = 1, 2, \cdots, N_1$. When the specific discharge is replaced by the Darcy flow (Dupuit assumption) we get the transformed transport equation in the confined (unconfined) aquifer. With the assumption that $c(x,t)$ and $\boldsymbol{K}(x)$ are known, we have u_i, α_i, β_i, γ_i and δ_i are all known, $i = 1, 2, \cdots, N_1$. So our problem becomes as follows: given data $v_i(x,\lambda)$ for $x \in \Omega$ and all $\lambda > 0$ (together with some boundary values of $\boldsymbol{D}_i$ and θ), determine the parameters $\boldsymbol{D}_i$, θ, B_i^1 and B_i^2, $i = 1, 2, \cdots, N_1$.

2.4. The inverse problem

If we regard $\theta(x)\boldsymbol{D}_i(x)$ in equation (2.19) as one (matrix) function $\boldsymbol{K}(x)$, then compare the equations (2.8), (2.12), and (2.19), we have that these three equations can be written in the following form:

$$-\nabla \cdot (\boldsymbol{K}(x)\nabla v) + (\lambda v + \alpha)Q(x) = \beta R(x) + \gamma S(x) + \delta, \tag{2.25}$$

where $\alpha = \alpha(x, \lambda)$, $\beta = \beta(x, \lambda)$, $\gamma = \gamma(x, \lambda)$, and $\delta = \delta(x, \lambda)$ are all known ($\gamma = \delta = 0$ for flow equations). And our problem is to find the coefficient functions $\boldsymbol{K}$, Q, R, and S from the known solution data $v = u(x, \lambda)$ and $\boldsymbol{K}|_{\partial\Omega}$.

According to [**38**, Chapter 8], the generalized Dirichlet problems associated with (2.26-2.29) below are uniquely solvable, and the solutions v lie in the Sobolev space $W^{1,2}(\Omega)$:

$$Lv = -\nabla \cdot \boldsymbol{p}(x)\nabla v + \lambda v q(x) = -\alpha q(x) + \beta r(x) + \gamma s(x) + \delta, \tag{2.26}$$

where $x \in \Omega$, Ω is a $\mathcal{C}^2$ domain in R^n; $\lambda > 0$; and

$$\alpha(x, \lambda), \beta(x, \lambda), \gamma(x, \lambda), \delta(x, \lambda) \in \mathcal{L}^2(\Omega); \tag{2.27}$$

$$\boldsymbol{p} = (p_{ij}) \text{ symmetric, strictly positive with } p_{ij} \in \mathcal{L}^\infty; \tag{2.28}$$

$$q(x), r(x), s(x) \in \mathcal{L}^2(\Omega). \tag{2.29}$$

Now let $\mathcal{D}$ be the set of all $c = (\boldsymbol{p}, q, r, s)$, such that $\boldsymbol{p}, q, r, s$ satisfy (2.28) and (2.29). Denote $u = u(x, \lambda)$ the solution of equation (2.25) and $u_c = u_c(x, \lambda)$ the solution of equation (2.26) corresponding to $c = (\boldsymbol{p}, q, r, s) \in \mathcal{D}$ such that

$$u_c|_{\partial\Omega} = u|_{\partial\Omega}. \tag{2.30}$$

The functional G is defined as follows

$$G(c, \lambda) = \int_\Omega \boldsymbol{p}(x)\nabla(u - u_c) \cdot \nabla(u - u_c) + \lambda(u - u_c)^2 q. \tag{2.31}$$

For a fixed number of λ values, $\lambda_1, \lambda_2, \cdots, \lambda_M$, we define functional H as the sum of G at those fixed λs, i.e.,

$$H(c) = \sum_{k=1}^{M} G(c, \lambda_k), \tag{2.32}$$

where $c = (\boldsymbol{p}, q, r, s) \in \mathcal{D}$. We will prove that the functional H is convex and has a global minimum point at $(\boldsymbol{K}, Q, R, S)$. Thus the recovery of $(\boldsymbol{K}, Q, R, S)$ becomes a procedure of minimization of the functional H.

2.5. The uniqueness

Before we give the method of recovery, we first show that the coefficients in equation (2.25) can indeed be uniquely determined under some assumptions, provided we already know the solutions. To this end we assume in this section that the matrix function $\boldsymbol{p}$ in (2.28) has entries in $\mathcal{C}^2(\Omega)$.

We proved, in [**59**], that if $\boldsymbol{p}$ is the only unknown parameter, then it can be uniquely determined by two solutions $u(x, \lambda_1)$ and $u(x, \lambda_2)$, $\lambda_1 \neq \lambda_2$, provided that one of the entries is known and

$$\begin{vmatrix} u_x(x, y, \lambda_1) & u_y(x, y, \lambda_1) \\ u_x(x, y, \lambda_2) & u_y(x, y, \lambda_2) \end{vmatrix} \neq 0$$

throughout the region Ω. We also give a uniqueness assumption which states that, in a two-dimensional case, $\boldsymbol{p}_1 = \boldsymbol{p}_2$ if and only if $u(\boldsymbol{p}_1, \lambda) = u(\boldsymbol{p}_2, \lambda)$ for three distinct values of λ, if all the other parameters are known. These results also apply to equation (2.26).

Now let $c = (\boldsymbol{p}, q, r, s) \in \mathcal{D}$, $u_i = u(x, \lambda_i)$, $i = 1, \cdots, 6$, be solutions of (2.26) with 6 distinct values of λ. If there is $\tilde{c} = (\tilde{\boldsymbol{p}}, \tilde{q}, \tilde{r}, \tilde{s}) \in \mathcal{D}$, which also gives solutions u_i, then

$$-\nabla \cdot \boldsymbol{p}(x) \nabla u_i + (\lambda u_i + \alpha_i) q(x) = \beta_i r(x) + \gamma_i s(x) + \delta_i,$$

$$-\nabla \cdot \tilde{\boldsymbol{p}}(x) \nabla u_i + (\lambda_i u_i + \alpha_i) \tilde{q}(x) = \beta_i \tilde{r}(x) + \gamma_i \tilde{s}(x) + \delta_i,$$

where $\alpha_i = \alpha(x, \lambda_i)$, $\beta_i = \beta(x, \lambda_i)$, $\gamma_i = \gamma(x, \lambda_i)$, $\delta_i = \delta(x, \lambda_i)$, $i = 1, \cdots, 6$. Thus we have

$$-\nabla \cdot (\boldsymbol{p}(x) - \tilde{\boldsymbol{p}}(x))\nabla u_i + (\lambda_i u_i + \alpha_i)(q(x) - \tilde{q}(x)) = \beta_i(r(x) - \tilde{r}(x)) + \gamma_i(s(x) - \tilde{s}(x)),$$

$i = 1, \cdots, 6$, or in matrix form

$$-\boldsymbol{A}\boldsymbol{\mu}_x - \boldsymbol{B}\boldsymbol{\mu}_y - \boldsymbol{C}\boldsymbol{\mu} = \boldsymbol{M}\boldsymbol{\nu}, \tag{2.33}$$

where $\boldsymbol{\mu} = (a, b, c)^T$, $\boldsymbol{\nu} = (q - \tilde{q}, r - \tilde{r}, s - \tilde{s})^T$,

$$\boldsymbol{p}(x) - \tilde{\boldsymbol{p}}(x) = \begin{pmatrix} a & b \\ b & c \end{pmatrix},$$

$$\boldsymbol{M} = \begin{pmatrix} -\lambda_1 u_1 - \alpha_1 & \beta_1 & \gamma_1 \\ -\lambda_2 u_2 - \alpha_2 & \beta_2 & \gamma_2 \\ -\lambda_3 u_3 - \alpha_3 & \beta_3 & \gamma_3 \\ -\lambda_4 u_4 - \alpha_4 & \beta_4 & \gamma_4 \\ -\lambda_5 u_5 - \alpha_5 & \beta_5 & \gamma_5 \\ -\lambda_6 u_6 - \alpha_6 & \beta_6 & \gamma_6 \end{pmatrix}, \qquad \boldsymbol{A} = \begin{pmatrix} u_{1x} & 0 & u_{1y} \\ u_{2x} & 0 & u_{2y} \\ u_{3x} & 0 & u_{3y} \\ u_{4x} & 0 & u_{4y} \\ u_{5x} & 0 & u_{5y} \\ u_{6x} & 0 & u_{6y} \end{pmatrix},$$

$$\boldsymbol{B} = \begin{pmatrix} 0 & u_{1y} & u_{1x} \\ 0 & u_{2y} & u_{2x} \\ 0 & u_{3y} & u_{3x} \\ 0 & u_{4y} & u_{4x} \\ 0 & u_{5y} & u_{5x} \\ 0 & u_{6y} & u_{6x} \end{pmatrix}, \qquad \boldsymbol{C} = \begin{pmatrix} u_{1xx} & u_{1yy} & 2u_{1xy} \\ u_{2xx} & u_{2yy} & 2u_{2xy} \\ u_{3xx} & u_{3yy} & 2u_{3xy} \\ u_{4xx} & u_{4yy} & 2u_{4xy} \\ u_{5xx} & u_{5yy} & 2u_{5xy} \\ u_{6xx} & u_{6yy} & 2u_{6xy} \end{pmatrix}.$$

If $\boldsymbol{p} = \tilde{\boldsymbol{p}}$, from (2.33) we can see that if we want to recover $m = 1, 2$, or 3 coefficients of q, r, s, we need only m solutions, provided the matrix $\boldsymbol{M}$ has rank m. Now for the general case, suppose we want to recover m entries of $\boldsymbol{p}$ and n coefficients of q, r, s. Without loss of generality we assume that $n = 3$. If $\boldsymbol{M}$ has rank 3 and $\boldsymbol{M}_0$ is a 3×3 nonsingular submatrix, denote $\boldsymbol{M}_1$ the submatrix of $\boldsymbol{M}$ whose elements are

those not in M_0. Let A_0, A_1, B_0, B_1, C_0, and C_1 be the submatrices of A, B, and C which take the same rows as M_0 and M_1 corresponding to the same subindices. Then we can get

$$\{M_1M_0^{-1}A_0 - A_1\}\mu_x + \{M_1M_0^{-1}B_0 - B_1\}\mu_y + \\ + \{M_1M_0^{-1}C_0 - C_1\}\mu = 0 \tag{2.34}$$

from (2.33), where M_0^{-1} is the inverse of M_0. One can see that $M_1M_0^{-1}A_0 - A_1$ and $M_1M_0^{-1}B_0 - B_1$ have the form

$$M_1M_0^{-1}A_0 - A_1 = \begin{pmatrix} a_1 & 0 & a_2 \\ b_1 & 0 & b_2 \\ c_1 & 0 & c_2 \end{pmatrix}, \tag{2.35}$$

$$M_1M_0^{-1}B_0 - B_1 = \begin{pmatrix} 0 & a_2 & a_1 \\ 0 & b_2 & b_1 \\ 0 & c_2 & c_1 \end{pmatrix}. \tag{2.36}$$

Similar to the proof in [**59**], we can show that

PROPOSITION 2.1. *In two dimensions $m + n$ distinct solutions u_i are needed to recover $m < 3$ entries of $\boldsymbol{p}$ and $n \leq 3$ coefficients of q, r, s of equation (2.26), provided that M_0 above and a submatrix of (2.35), namely*

$$\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix}$$

are non-singular throughout the region Ω.

We assume

UNIQUENESS ASSUMPTION 2.2. *In two dimensions, if $\boldsymbol{p}$ has entries in $C^2(\Omega)$, then $c = (\boldsymbol{p}, q, r, s) \in \mathcal{D}$ can be uniquely determined by N different solutions u_{x,λ_i} of (2.26), $i = 1, \cdots, N$, where $N \geq 6$.*

Proposition 2.1 states that if the given data is "good enough," then only six solutions would be enough to recover all the coefficients (see [**59**]). In the real world, six solutions is usually not enough. We test for different synthetic data, and find that 20 solutions is a good choice. On the other hand, the choice is justified, since in the finite Laplace transformation of the flow equation, λ acts as the time variable. If we choose more λ values, that means we use more data during the time period, which should certainly gives us more information.

2.6. Properties of functionals G and H

In this section, we will give some properties, which are essential for the numerical algorithms, of the functionals we constructed.

Let v be a solution of the generalized Dirichlet problem (2.26, 2.30), for $\phi \in W_0^{1,2}(\Omega)$, we have

$$(Lv, \phi) = \int_\Omega (\boldsymbol{p}\nabla v \cdot \nabla\phi + \lambda q(x) v\phi)dx, \tag{2.37}$$

by (2.26),

$$\int_\Omega \phi\nabla \cdot (\boldsymbol{p}\nabla v)dx = -\int_\Omega \boldsymbol{p}\nabla v \cdot \nabla\phi dx. \tag{2.38}$$

The latter formula is essentially Green's formula for this situation ("integration by parts") and will be used a great deal in the proof of the properties of the functional G and H.

LEMMA 2.3. (a). *For any* $c = (\boldsymbol{p}, q, r, s) \in \mathcal{D}$,

$$\begin{aligned} G(c,\lambda) = \int_\Omega \{&\boldsymbol{p}(x)\nabla u \cdot \nabla u - \boldsymbol{p}(x)\nabla u_c \cdot \nabla u_c + \\ &+ [\lambda(u^2 - u_c^2) + 2\alpha(u - u_c)]q(x) \\ &- 2(u - u_c)[\beta r(x) + \gamma s(x) + \delta]\}dx. \end{aligned} \tag{2.39}$$

(b). *For* $c_1 = (\boldsymbol{p}_1, q_1, r_1, s_1)$ *and* $c_2 = (\boldsymbol{p}_2, q_2, r_2, s_2)$ *in* $\mathcal{D}$ *we have*

$$G(c_1,\lambda) - G(c_2,\lambda) = \tag{2.40}$$
$$\int_\Omega \{(\boldsymbol{p}_1 - \boldsymbol{p}_2)\nabla u \cdot \nabla u - (\boldsymbol{p}_1 - \boldsymbol{p}_2)\nabla u_{c_1} \cdot \nabla u_{c_2} +$$
$$+ [\lambda(u^2 - u_{c_1}u_{c_2}) + 2\alpha(u - \frac{1}{2}(u_{c_1} + u_{c_2}))](q_1 - q_2)$$
$$- 2[u - \frac{1}{2}(u_{c_1} + u_{c_2})][\beta(r_1 - r_2) + \gamma(s_1 - s_2)]\}dx.$$

(c). *For* $c = (\boldsymbol{p}, q, r, s)$ *in* $\mathcal{D}$ *and* $h = (\boldsymbol{h}_1, h_2, h_3, h_4)$, $\boldsymbol{h}_1$ *symmetric matrix with entries in* $\mathcal{L}^\infty(\Omega)$, $\boldsymbol{h}_1|_{\partial\Omega} = 0$, *and* h_2, h_3, h_4 *in* $\mathcal{L}^2(\Omega)$, *we have*

$$\lim_{\epsilon\to 0} u_{c+\epsilon h} = u_c, \tag{2.41}$$

in $W^{1,2}(\Omega)$.

(d). *For* $c = (\boldsymbol{p}, q, r, s)$ *in* $\mathcal{D}$ *and* $h = (\boldsymbol{h}_1, h_2, h_3, h_4)$, $\boldsymbol{h}_1$ *symmetric matrix with entries in* $\mathcal{L}^\infty(\Omega)$, $\boldsymbol{h}_1|_{\partial\Omega} = 0$, *and* h_2, h_3, h_4 *in* $\mathcal{L}^2(\Omega)$, *and any symetric matrix* $\boldsymbol{\eta}$ *with entries in* $\mathcal{L}^\infty(\Omega)$,

$$||\nabla \cdot (\boldsymbol{\eta}\nabla u_{c+\epsilon h})||_{W^{-1,2}(\Omega)} \leq K, \tag{2.42}$$

where K *is a constant that does not depend on* ϵ *when* ϵ *is not too big.*

PROOF. To prove part (a) first note the identity

$$G(c,\lambda) = \int_\Omega \{\boldsymbol{p}(x)\nabla u \cdot \nabla u - \boldsymbol{p}(x)\nabla u_c \cdot \nabla u_c +$$
$$+ 2\boldsymbol{p}(x)\nabla u_c \cdot \nabla(u - u_c) + \lambda\alpha(u_c - u)^2 q(x)\}.$$

Now, using integration by parts and equation (2.26), together with the fact that the solutions u and u_c share the same boundary data, we have

$$\begin{aligned}
&\int_\Omega \{2\boldsymbol{p}(x)\nabla u_c \cdot \nabla(u_c - u) + \lambda\alpha(u - u_c)^2 q(x)\}dx \\
= &\int_\Omega \{-2(u_c - u)\nabla \cdot (\boldsymbol{p}(x)\nabla u_c) + \lambda\alpha(u - u_c)^2 q(x)\}dx \\
= &\int_\Omega \{2(u_c - u)[-(\lambda u_c + \alpha)q(x) + \beta r(x) + \gamma s(x) + \delta] + \lambda\alpha(u - u_c)^2 q(x)\}dx \\
= &\int_\Omega \{[\lambda(u^2 - u_c^2) + 2\alpha(u - u_c)]q(x) - 2(u - u_c)[\beta r(x) + \gamma s(x) + \delta]\}dx
\end{aligned}$$

and this gives the proof of (a).

To prove part (b), notice that by (a)

$$\begin{aligned}
&G(c_1, \lambda) - G(c_2, \lambda) = \\
= &\int_\Omega \{(\boldsymbol{p}_1 - \boldsymbol{p}_2)\nabla u \cdot \nabla u - (\boldsymbol{p}_1 - \boldsymbol{p}_2)\nabla u_{c_1} \cdot \nabla u_{c_2} + \\
&\quad +[\lambda(u^2 - u_{c_1}u_{c_2}) + 2\alpha(u - \frac{1}{2}(u_{c_1} + u_{c_2}))](q_1 - q_2) + \\
&\quad -2[u - \frac{1}{2}(u_{c_1} + u_{c_2})][\beta(r_1 - r_2) + \gamma(s_1 - s_2)]\}dx \\
+ &\int_\Omega \{-\boldsymbol{p}_1\nabla u_{c_1}\nabla(u_{c_1} - u_{c_2}) - (u_{c_1} - u_{c_2})[(\lambda u_{c_1} + \alpha)q_1 + \beta r_1 + \gamma s_1 + \delta] + \\
&\quad -\boldsymbol{p}_2\nabla u_{c_2}\nabla(u_{c_1} - u_{c_2}) - (u_{c_1} - u_{c_2})[(\lambda u_{c_2} + \alpha)q_2 + \beta r_2 + \gamma s_2 + \delta]\}dx
\end{aligned}$$

By integration by parts, equation (2.26), and the $u_{c_1} = u_{c_2}$ on the boundary $\partial\Omega$, we have that the second part of the right hand side equals to 0. This gives the proof of part (b).

In order to prove part (c), we subtract the equations

$$-\nabla \cdot (\boldsymbol{p}\nabla u_c) + (\lambda u_c + \alpha)q = \beta r + \gamma s + \delta$$

$$\begin{aligned}
&-\nabla \cdot ((\boldsymbol{p} + \epsilon\boldsymbol{h}_1)\nabla u_{c+\epsilon h}) + (\lambda u_{c+\epsilon h} + \alpha)(q + \epsilon h_2) = \\
&\quad = \beta(r + \epsilon h_3) + \gamma(s + \epsilon h_4) + \delta
\end{aligned}$$

to get

$$-\nabla\cdot(\boldsymbol{p}\nabla(u_{c+\epsilon h}-u_c))+\lambda(u_{c+\epsilon h}-u_c)q=$$
$$\epsilon[\nabla\cdot\boldsymbol{h}_1\nabla u_{c+\epsilon h}-(\lambda u_{c+\epsilon h}+\alpha)h_2+\beta h_3+\gamma h_4]$$

Now we multiply $u_{c+\epsilon h}-u_c$ on both sides of the above equation and integrate over Ω we get (after integration by parts)

$$\begin{aligned}
&\int_\Omega\{\boldsymbol{p}\nabla(u_{c+\epsilon h}-u_c)\cdot\nabla(u_{c+\epsilon h}-u_c)+\lambda(u_{c+\epsilon h}-u_c)^2q\}dx\\
=&\int_\Omega\{-\nabla\cdot\boldsymbol{p}\nabla(u_{c+\epsilon h}-u_c)(u_{c+\epsilon h}-u_c)+\lambda(u_{c+\epsilon h}-u_c)^2q\}dx\\
=&\ \epsilon\int_\Omega\{(u_{c+\epsilon h}-u_c)\nabla\cdot\boldsymbol{h}_1\nabla u_{c+\epsilon h}-\\
&\qquad-[(\lambda u_{c+\epsilon h}+\alpha)h_2+\beta h_3+\gamma h_4](u_{c+\epsilon h}-u_c)\}dx\\
=&\ \epsilon\int_\Omega\{-\boldsymbol{h}_1\nabla(u_{c+\epsilon h}-u_c)\cdot\nabla(u_{c+\epsilon h}-u_c)-\boldsymbol{h}_1\nabla u_c\cdot\nabla(u_{c+\epsilon h}-u_c)-\\
&\qquad-\lambda(u_{c+\epsilon h}-u_c)^2h_2-((\lambda u_c+\alpha)h_2+\beta h_3+\gamma h_4)(u_{c+\epsilon h}-u_c)\}dx\\
\le&\ \epsilon\int_\Omega\{\boldsymbol{h}_1\nabla(u_{c+\epsilon h}-u_c))\cdot\nabla(u_{c+\epsilon h}-u_c)\\
&\qquad+\frac{1}{2}(|\boldsymbol{h}_1\nabla u_c|^2+|\nabla(u_{c+\epsilon h}-u_c)|^2)\\
&\qquad+\frac{1}{2}(((\lambda u_c+\alpha)h_2+\beta h_3+\gamma h_4)^2+(u_{c+\epsilon}-u_c)^2)\\
&\qquad+\lambda h_2(u_{c+\epsilon h}-u_c)^2\}dx
\end{aligned}$$

and using the inequality $ab\le(a^2+b^2)/2$.

Now, with the assumption that q is lower bounded by some positive number, we have that the left hand side of the above inequality is bounded below by a constant multiple of $||u_{c+\epsilon h}-u_c||_{W^{1,2}(\Omega)}$. On the other hand, the terms on the right hand side are independent of ϵ except for $u_{c+\epsilon h}$. So when ϵ is small enough, we can move the terms on the right hand side which contain $u_{c+\epsilon h}-u_c$ to the left hand side so that the left side is still lower bounded by a constant multiple of $||u_{c+\epsilon h}-u_c||_{W^{1,2}(\Omega)}$ and the remaining right hand side is $O(\epsilon)$. Then part (c) follows.

In order to prove (d), we define a functional F on $W_0^{1,2}(\Omega)$ by $F(\phi) = \int_\Omega \boldsymbol{\eta} \nabla u_{c+\epsilon h} \cdot \nabla \phi$. Since the entries of $\boldsymbol{\eta}$ are in $\mathcal{L}^\infty(\Omega)$, (2.41) implies that

$$|F(\phi)| \leq K||\phi||_{W^{1,2}(\Omega)}, \tag{2.43}$$

when ϵ is small enough, and K does not depend on ϵ. Thus $F \in (W_0^{1,2}(\Omega))^\star = W^{-1,2}(\Omega)$. The estimate (2.42) then follows from (2.43). □

THEOREM 2.4. (a). *The first Gâteaux differential of G is given by*

$$\begin{aligned} G'(c,\lambda)[h] = \int_\Omega \{ & \boldsymbol{h}_1 \nabla u \cdot \nabla u - \boldsymbol{h}_1 \nabla u_c \cdot \nabla u_c + \\ & + [\lambda(u^2 - u_c^2) + 2\alpha(u - u_c)]h_2 \\ & - 2(u - u_c)[\beta h_3 + \gamma h_4]\} dx, \end{aligned} \tag{2.44}$$

where $h = (\boldsymbol{h}_1, h_2, h_3, h_4)$, $\boldsymbol{h}_1$ is a symmetric matrix with entries in $\mathcal{L}^\infty(\Omega)$, $\boldsymbol{h}_1|_{\partial\Omega} = 0$, and h_2, h_3, h_4 in $\mathcal{L}^2(\Omega)$.

(b). *The second Gâteaux differential of G is given by*

$$G''(c,\lambda)[h,k] = 2(L^{-1}(e(h)), e(k)), \tag{2.45}$$

where $h = (\boldsymbol{h}_1, h_2, h_3, h_4)$, $k = (\boldsymbol{k}_1, k_2, k_3, k_4)$, and $\boldsymbol{h}_1$, $\boldsymbol{k}_1$ are symmetric matrices with entries in $\mathcal{L}^\infty(\Omega)$ with $\boldsymbol{h}_1|_{\partial\Omega} = \boldsymbol{k}_1|_{\partial\Omega} = 0$, and the functions $h_2, h_3, h_4, k_2, k_3, k_4$ lie in $\mathcal{L}^2(\Omega)$,

$$e(h) = -\nabla \cdot \boldsymbol{h}_1 \nabla u_c + [\lambda u_c + \alpha]h_2 - \beta h_3 - \gamma h_4, \tag{2.46}$$

and $(\cdot,\cdot)$ denotes the usual inner product in $\mathcal{L}^2(\Omega)$.

PROOF. By Lemma 2.3 (b), for $\epsilon > 0$,

$$\begin{aligned} \frac{1}{\epsilon}(G(c+\epsilon h, \lambda) - G(c,\lambda)) = \int_\Omega \{ & q\boldsymbol{h}_1 \nabla u \cdot \nabla u - q\boldsymbol{h}_1 \nabla u_c \cdot \nabla u_{c+\epsilon h} \\ & + [\boldsymbol{p} \nabla u \cdot \nabla u - \boldsymbol{p} \nabla u_c \cdot \nabla u_{c+\epsilon h} + \lambda(u^2 - u_{c+\epsilon h} u_c) + 2\alpha(u - \frac{1}{2}(u_c + u_{c+\epsilon h})]h_2 \\ & - 2[u - \frac{1}{2}(u_c + u_{c+\epsilon h})][\beta h_3 + \gamma h_4] + \epsilon[h_2 \boldsymbol{h}_1 \nabla u \nabla u - h_2 \boldsymbol{h}_1 \nabla u_c \nabla u_{c+\epsilon h}]\} dx. \end{aligned}$$

Then part (a) follows from Lemma 2.3 (c) by letting ϵ go to 0.

To prove (b), note that by (a),

$$\begin{aligned}
& G'(c+\epsilon h,\lambda)[k]-G'(c,\lambda)[k]= \\
= & -\int_\Omega \{\boldsymbol{k}_1\nabla u_{c+\epsilon h}\cdot\nabla u_{c+\epsilon h}-\boldsymbol{k}_1\nabla u_c\cdot\nabla u_c \\
& +[\lambda(u_{c+\epsilon h}^2-u_c^2)+2\alpha(u_{c+\epsilon h}-u_c)]k_2 \\
& -2(u_{c+\epsilon h}-u_c)[\beta k_3+\gamma k_4]\}dx \\
= & -\int_\Omega \{\boldsymbol{k}_1\nabla(u_{c+\epsilon h}+u_c)\cdot\nabla(u_{c+\epsilon h}-u_c) \\
& +[\lambda(u_{c+\epsilon h}^2-u_c^2)+2\alpha(u_{c+\epsilon h}-u_c)]k_2 \\
& -2(u_{c+\epsilon h}-u_c)[\beta k_3+\gamma k_4]\}dx \\
= & \int_\Omega (u_{c+\epsilon h}-u_c)\{\nabla\cdot(\boldsymbol{k}_1\nabla(u_{c+\epsilon h}+u_c)) \\
& -[\lambda(u_{c+\epsilon h}+u_c)+2\alpha]k_2+2[\beta k_3+\gamma k_4]\}dx,
\end{aligned}$$

after an integration by parts.

Subtract the equations

$$Lu_c=-\nabla\cdot(\boldsymbol{p}\nabla u_c)+\lambda u_c q=-\alpha q+\beta r(x)+\gamma s(x)+\delta \tag{2.47}$$

and

$$\begin{aligned}
Lu_{c+\epsilon h} &= -\nabla\cdot((\boldsymbol{p}+\epsilon\boldsymbol{h}_1)\nabla u_{c+\epsilon h})+\lambda u_{c+\epsilon h}(q+\epsilon h_2) \\
&= -\alpha(q+\epsilon h_2)+\beta(r+\epsilon h_3)+\gamma(s+\epsilon h_4)+\delta
\end{aligned} \tag{2.48}$$

we have

$$\begin{aligned}
L(u_{c+\epsilon h}-u_c) &= -\nabla\cdot\boldsymbol{p}\nabla(u_{c+\epsilon h}-u_c)+\lambda(u_{c+\epsilon h}-u_c)q \\
&= -\epsilon[-\nabla\cdot\boldsymbol{h}_1\nabla u_{c+\epsilon h}+(\lambda u_{c+\epsilon h}+\alpha)h_2-(\beta h_3+\gamma h_4)],
\end{aligned}$$

or

$$u_{c+\epsilon h}-u_c=-\epsilon L^{-1}(-\nabla\cdot\boldsymbol{h}_1\nabla u_{c+\epsilon h}+(\lambda u_{c+\epsilon h}+\alpha)h_2-(\beta h_3+\gamma h_4)). \tag{2.49}$$

Thus

$$\frac{G'(c+\epsilon h,\lambda)[k]-G'(c,\lambda)[k]}{\epsilon} =$$

$$= \int_\Omega L^{-1}(-\nabla\cdot \boldsymbol{h}_1\nabla u_{c+\epsilon h}+(\lambda u_{c+\epsilon h}+\alpha)h_2-(\beta h_3+\gamma h_4))\times$$
$$\times\{-\nabla\cdot(\boldsymbol{k}_1\nabla(u_{c+\epsilon h}+u_c))+[\lambda(u_{c+\epsilon h}+u_c)+2\alpha]k_2-2[\beta k_3+\gamma k_4]\}dx$$
$$= 2\int_\Omega L^{-1}[e(h)]\{-\nabla\cdot(\boldsymbol{k}_1\nabla u_c)+(\lambda u_c+\alpha)k_2-(\beta k_3+\gamma k_4)\}dx +$$
$$+\int_\Omega L^{-1}[-\nabla\cdot \boldsymbol{h}_1\nabla(u_{c+\epsilon h}-u_c)+\lambda(u_{c+\epsilon h}-u_c)h_2]\times$$
$$\times\{-\nabla\cdot(\boldsymbol{k}_1\nabla(u_{c+\epsilon h}+u_c))+[\lambda(u_{c+\epsilon h}+u_c)+\alpha]k_2-[\beta k_3+\gamma k_4]\}dx +$$
$$+\int_\Omega L^{-1}[e(h)]\{-\nabla\cdot(\boldsymbol{k}_1\nabla(u_{c+\epsilon h}-u_c))+\lambda(u_{c+\epsilon h}-u_c)k_2\}dx.$$

It remains to be shown that the second and third integrals of the last expression tend to zero as $\epsilon\to 0$. As the operator L^{-1} is self-adjoint, if we set

$$w_e=-\nabla\cdot(\boldsymbol{k}_1\nabla(u_{c+\epsilon h}+u_c))+[\lambda(u_{c+\epsilon h}+u_c)+\alpha]k_2-[\beta k_3+\gamma k_4],$$

the second integral may be rewritten as

$$\int_\Omega[-\nabla\cdot \boldsymbol{h}_1\nabla(u_{c+\epsilon h}-u_c)+\lambda(u_{c+\epsilon h}-u_c)h_2]\times L^{-1}(w_e)$$
$$= \int_\Omega \boldsymbol{h}_1\nabla(u_{c+\epsilon h}-u_c)\cdot(L^{-1}(w_e))+\lambda h_2(u_{c+\epsilon h}-u_c)L^{-1}(w_e).$$

From (2.42), w_e is uniformly bounded in ϵ in $\mathcal{L}^2(\Omega)$, and as L^{-1} may be extended uniquely as a bounded linear operator from $\mathcal{L}^2(\Omega)$ to $W^{1,2}(\Omega)$, $L^{-1}(w_e)$ is bounded independently of ϵ in $W^{1,2}(\Omega)$. From the boundedness of ∇ on $W^{1,2}(\Omega)$ to $\mathcal{L}^2(\Omega)\times\mathcal{L}^2(\Omega)$ it follows that $|L^{-1}(w_e)|$ is bounded independently of ϵ in $\mathcal{L}^2(\Omega)$. From (2.41) it now follows that the second integral tends to zero with $\epsilon\to 0$. Finally, note that $L^{-1}[e(h)]$ lies in $W^{1,2}(\Omega)$, and that the third integral vanishes as $\epsilon\to 0$ follows via (2.41) after an integration by parts. This completes the proof of the theorem. □

THEOREM 2.5. (a). *For c in $\mathcal{D}$ and $\lambda>0$*

$$G(c,\lambda)=0 \iff G'(c,\lambda)=0 \iff u_c=u.$$

where u_c is the solution of (2.26), u is the solution of (2.25), i.e., the known data.

(b). *Assume that c lies in $\mathcal{D}$ and $G''(c,\lambda)[h,h] = 0$ for some $h = (\boldsymbol{h}_1, h_2, h_3, h_4)$ where $\boldsymbol{h}_1$ is symmetric matrix with entries in $\mathcal{L}^\infty(\Omega)$, $\boldsymbol{h}_1|_{\partial\Omega} = 0$, and h_2, h_3, h_4 in $\mathcal{L}^2(\Omega)$. Then $u_{c+\epsilon h} = u_c$ for all ϵ small enough.*

PROOF. The assertion in (a) that $G(c,\lambda) = 0$ if and only if $u_c = u$ follows immediately from the definition of G, and one direction of the remaining assertion is obvious. If $G'(c,\lambda)[h] = 0$ for all h, then the $\mathcal{L}^2$ gradient of G, ∇G, satisfies $\nabla G(c,\lambda) = (\gamma_{jk}) = 0$, where, for $1 \le j,k \le n$,

$$\gamma_{jk} = u_{x_j} u_{x_k} - u_{c,x_j} u_{c,x_k}.$$

Consequently, from Theorem 2.4 part (a) and Lemma 2.3 (a),

$$\begin{aligned} G(c,\lambda) &= \int_\Omega \{\boldsymbol{p}(x)\nabla(u-u_c)\cdot\nabla(u-u_c) + \lambda(u-u_c)^2 q(x)\}dx \\ &= -2\int_\Omega \delta(u-u_c)dx. \end{aligned}$$

If we interchange c and C in this formula and note that $u = u_C$, we have

$$\int_\Omega \{\boldsymbol{K}(x)\nabla(u-u_c)\cdot\nabla(u-u_c) + \lambda(u-u_c)^2 Q(x)\}dx = -2\int_\Omega \delta(u_c-u)dx.$$

Adding, we find that

$$\int_\Omega \{(\boldsymbol{p}(x)+\boldsymbol{K}(x))\nabla(u-u_c)\cdot\nabla(u-u_c) + \lambda(u-u_c)^2(q(x)+Q(x))\}dx = 0,$$

from this and the fact that $\boldsymbol{p}$, $\boldsymbol{K}$ are positive definite and $q, Q > 0$, it follows that $u - u_c = 0$.

Part (b) is a consequence of Theorem 2.4 part (b) in that, if we assume that $G''(c,\lambda)[h,h] = 0$, then

$$e(h) = -\nabla\cdot\boldsymbol{h}_1\nabla u_c + [\lambda u_c + \alpha]h_2 - \beta h_3 - \gamma h_4 = 0,$$

so that from (2.26), for all ϵ small enough $\boldsymbol{p} + \epsilon \boldsymbol{h}_1$ is strictly positive and

$$-\nabla \cdot (\boldsymbol{p}(x) + \epsilon \boldsymbol{h}_1)\nabla u_c + (\lambda u_c + \alpha)(q(x) + \epsilon h_2)$$
$$= \beta(r(x) + \epsilon h_3) + \gamma(s(x) + \epsilon h_4) + \delta.$$

But $u_{c+\epsilon h}$ is the unique solution of this equation with the boundary data $u|_{\partial\Omega}$; it follows immediately that $u_{c+\epsilon h} = u_c$. □

THEOREM 2.6. *Assume that the uniqueness assumption of the previous section holds, and in (2.32) set* $M >= 5$. *Then for* c *in* $\mathcal{D}$,

$$H(c) = 0 \iff H'(c) = 0 \iff c = C(= (\boldsymbol{K}, Q, R, S)) \tag{2.50}$$

where $\boldsymbol{K}, Q, R, S$ *are the coefficients in equation (2.25), i.e., the coefficients we intended to recover. And the functional* H *is strictly convex on* $\mathcal{D}$.

PROOF. Noting that $H(c) = 0$ if and only if $G(c, \lambda_i) = 0$ for $1 \leq i \leq M$, the first assertion follows from Theorem 2.5 and the uniqueness assumption. Next, if $H'(c) = 0$, the same proof that was used for G shows that $H(c) = 0$, and the rest follows from the statements above. Finally, let $H''(c)[h, h] = 0$. As the functionals $G(c, \lambda_i)$, $1 \leq i \leq M$, are convex, it follows that $G''(c, \lambda_i)[h, h] = 0$ for $1 \leq i \leq M$. By Theorem 2.5 part (b), for all ϵ small enough , $u_{c+\epsilon h} = u_c$ for $\lambda = \lambda_i$, $1 \leq i \leq M$. The uniqueness assumption now indicates that $h = 0$. □

2.7. A descent algorithm

Theorem 2.6 shows that, under computationally verifiable conditions on the data functions, $u_i = u(c, \lambda_i)$, $1 \leq i \leq M$, and the coefficients $(\boldsymbol{K}, Q, R, S)$ can be uniquely recovered by minimization of the functional H given by (2.32). Recall that by Taylor's expansion,

$$H(c + \epsilon h) \approx H(c) + \epsilon H'(c)[h] = \sum_{i=1}^{M} G(c, \lambda_i) + \epsilon \sum_{i=1}^{M} G'(c, \lambda_i)[h]$$

for the direction h. If we can find a direction h such that the second term of the right-hand side above is negative, then we can get to a point $c_1 = c + \epsilon h$, such that $H(c_1) < H(c)$ when $\epsilon > 0$ is not too big. So the search for a descent direction is an important part in our minimization procedure. First for $h = (\boldsymbol{h}_1, 0, 0, 0)$, Theorem 2.4 gives

$$\begin{aligned} G'(c,\lambda)[h] &= \int_\Omega \boldsymbol{h}_1 \nabla u \cdot \nabla u - \boldsymbol{h}_1 \nabla u_c \cdot \nabla u_c \\ &= \sum_{i,j=1}^{n} \int_\Omega h_{ij}(u_i u_j - u_{ci} u_{cj}) \\ &= \sum_{i,j=1}^{n} \int_\Omega h_{ij} \eta_{ij} \end{aligned}$$

where u is the solution of (2.25), u_c is the solution of (2.26), $u_i = \frac{\partial u}{\partial x_i}$, $u_{ci} = \frac{\partial u_c}{\partial x_i}$, and $\boldsymbol{h}_1 = (h_{ij})$ is a symmetric matrix with h_{ij} in $\mathcal{L}^\infty(\Omega)$ and $\boldsymbol{h}_1|_{\partial\Omega} = 0$. Notice that if we define the $\mathcal{L}^2$ inner product for $n \times n$ matrix functions $\boldsymbol{h} = (h_{ij})$ and $\boldsymbol{g} = (g_{ij})$ to be

$$(\boldsymbol{h}, \boldsymbol{g})_{\mathcal{L}^2} = \sum_{i,j=1}^{n} \int_\Omega h_{ij}(x) g_{ij}(x) dx,$$

and $\boldsymbol{\eta} = (\eta_{ij})$ then $G'(c,\lambda)[h] = (\boldsymbol{\eta}, \boldsymbol{h}_1)_{\mathcal{L}^2}$. Now let h_{ij} be the solution of

$$-\triangle g_{ij} + g_{ij} = (\nabla G)_{ij} = \eta_{ij} \qquad g_{ij}|_{\partial\Omega} = 0, \tag{2.51}$$

then

$$\begin{aligned} G'(c,\lambda)[h] &= \int_\Omega (\boldsymbol{\eta}, \boldsymbol{h}_1)_{\mathcal{L}^2} dx \\ &= \sum_{i,j=1}^{n} \int_\Omega \eta_{ij} g_{ij} dx \\ &= \sum_{i,j=1}^{n} \int_\Omega (-\triangle g_{ij} + g_{ij}) g_{ij} dx \\ &= \sum_{i,j=1}^{n} \int_\Omega \{\nabla g_{ij} \cdot \nabla g_{ij} + g_{ij} \cdot g_{ij}\} dx \\ &= (\boldsymbol{g}, \boldsymbol{g})_{\mathcal{H}^1}, \end{aligned}$$

where $\boldsymbol{g} = (g_{ij})$, $(\cdot,\cdot)_{\mathcal{H}^1}$ is the Sobolev inner product corresponding the $\mathcal{L}^2$ inner product above. So we get a descent direction $\boldsymbol{g}$, which is called the Neuberger gradient [**73**]. Thus for the matrix direction $\boldsymbol{h}_1$, we can choose the Neuberger gradient as the descent direction. Note that we can not choose the $\mathcal{L}^2$ gradient $\nabla G = (\eta_{ij})$ as our descent direction here, because ∇G is generally not zero on the boundary $\partial\Omega$. For other directions, h_2, h_3, h_4, this requirement is not necessary. So we can simply choose the $\mathcal{L}^2$ gradient as the descent direction. If we know the boundary value of a coefficient, such as the storativity, we can choose the Neuberger gradient as the descent direction. This usually leads to a better result. For a more detailed discussion about this descent method, please refer to [**52**].

CHAPTER 3

Numerical Implementation and Results

3.1. The numerical implementation

There are two approaches we can adopt in the actual recovery. A total of six coefficients, $\boldsymbol{K}$, Q, R, and S, are involved in our recovery procedure. We can regard H as a function of six variables and use a minimization method for multiple variable functions, such as the "Powell" method. This method is generally more efficient for the recovery of $\boldsymbol{K}$, [59]. We can also code to recover the variables one by one. The advantage of this method is that we can control the actual recovery for each individual variable. If one variable is more difficult to recover than the others, we can set more iterations for this variable in each step. Our example shows that $\boldsymbol{K}$ is most difficult to recover, compared to other variables. We use the second method in this dissertation. Here is a brief illustration of our minimization step. Assume the six variables to be recovered are denoted by r_i, $1 \leq i \leq 6$. Our algorithm is implemented according to the following scheme:

a) Choose an appropriate initial guess for the functions $c = (r_i)$, $1 \leq i \leq 6$; these can be arbitrary, except that some of them must satisfy some boundary conditions, such as $r_i|_{\partial\Omega} = K_i|_{\partial\Omega}$, $i = 1, 2, 3$, where the boundary values for K_i come from the known boundary values of $\boldsymbol{K}$.

b) Set $i = 1$, i.e., the search variable is r_1. Also set the control variable $flag = 0$.

c) Compute the descent direction h_i. If the boundary values of r_i are known, such as $1 \leq i \leq 3$ for the two-dimensional case, the descent direction is computed as the Neuberger gradient; otherwise, it is the $\mathcal{L}^1$ gradient.

d) Let h be the direction such that the ith component is h_i and all the others are 0. Check to see if H can be minimized in direction h by comparing $H(c)$

and $H(c - \alpha h)$ for some small number α $(= 10^{-7})$. If

$$H(c) > H(c - \alpha h),$$

then set $flag = 1$ and go to the next step. Otherwise, set $i = i + 1$ and go to step g).

e) Do a line-minimization in the direction h, using the one-dimensional search routine to compute $c^{\text{new}} = c - \alpha^{\min} h$.

f) Set $c = c^{\text{new}}$ and $i = i + 1$.

g) If $i \leq 6$, then go back to step c); otherwise, go back to step b) if $flag = 1$, or exit the search (since no search is successful in this iteration step).

One of the advantages of this algorithm is that it is easily parallelized. Recall that in our actual recovery, we need to compute a total of M H_i functionals, and all these functionals can be minimized separately. We can set the program to M processes to recover those H_i separately on different processors. There is a second level of parallelization. Since each $H_i(c)$ is the sum of N different $G(c, \lambda_i)$, and the gradient is also the sum of the N different $\nabla G(c, \lambda_i)$, we can write the program as a master-slave program such that the master process mainly does the line search work, while the slave part deals with the numerical solution of the partial differential equations and the quadrature needed for computing the values of the functionals G and the gradient ∇G. In our program, we use the PVM package [**15**] for message passing between the master and the slave programs. Some of routines are implemented or adopted from the Numerical Recipes [**82**] in our programs. Note also

(1). The elliptic PDE solver. Since the recovery is very sensitive to the error of the numerical solution, a solver is needed that can accurately and efficiently solve elliptic boundary value problems of the type

$$-\nabla \cdot \boldsymbol{K} \nabla u + Qu = F,$$

$$u(x, y)|_{\partial\Omega} = B(x, y).$$

with minimal error. We use the nine-point difference method for the discretization and then employ the band-solving subroutine BANDEC adopted from [**82**] to solve the resulting system of linear equations. This solver was called upon to determine the various solutions u_c during the descent searching procedure and the Neuberger gradient. The solver we implemented is efficient for two dimensions. For three dimensions, a more efficient solver is required.

(2). The numerical differentiation. We use central differences for the numerical derivatives for our synthetic dataset. This is accurate and efficient here, because the solutions being differentiated are sufficiently smooth functions. For practical data with noise, one must apply more sophisticated numerical differentiation techniques. We implemented a routine by using the mollifier function

$$\rho(x) = \begin{cases} \beta \exp(\frac{1}{||x||^2-1}) & \text{if} ||x|| < 1, \\ 0 & \text{otherwise,} \end{cases}$$

where β is chosen so that $\int_{R^n} \rho(x)dx = 1$, to regularize the data function u by

$$u_h(x) = h^{-n} \int_\Omega \rho(\frac{x-y}{h}) u(y) dy, \tag{3.1}$$

for some small $h > 0$. One can then compute the numerical derivatives of u_h using central differences and use these as approximations to the derivatives of u. For a more detailed discussion about this routine please see Section 3.3.

(3). The quadrature. We iterate the Simpson's rule function QSIMP in [**82**] to perform the required quadrature in the formula (2.31) for $G(c, \lambda)$ and the finite Laplace transformation. Given that parts of the integrand lack smoothness, Simpson's rule is an effective choice here.

(4). The line minimization routine. In the minimization search, we adopted the bracketing and line minimization approach in [**82**]. The idea is as follows.

If H can be minimized along a given direction, we use our own (somewhat primitive but safe) bracketing method to find the bracketing points. First we choose an initial stepping distance, and then step along the chosen direction using this stepping size as an increment until either a bracketing is found or a preset stepping limit is encountered. In the latter case the original c is reset to the new c at the stepping limit and a new gradient is computed. In practice we use the actual length of the movement in the previous search to make some adjustment to the stepping distance. Once the bracketing points are found, we adopt the BRENT function [**82**] to find the minimum.

(5). The parabolic PDE solver. In our test program, we need a parabolic solver to solve the flow and transport equations. To this end, we adopted the pde solver PDETWO of [**68**], modified so that it can handle the matrix-valued case with nonzero cross-term coefficients. This solver is efficient and quite accurate, with smooth boundary and initial values. The solution formed our synthetic dataset.

Note that since the elliptic solvers are extremely sensitive to a loss of positivity for $\boldsymbol{p}$, the program would tend to crash when non-positive eigenvalues for $\boldsymbol{p}$ were encountered. Noting that $\boldsymbol{p}$ is positive definite if and only if

$$p_{11} > 0, \quad p_{22} > 0, \quad p_{11}p_{22} - p_{12}^2 > 0,$$

we argue that it is reasonable to set lower bounds on the functions p_{11}, p_{22} with local knowledge of a particular aquifer and with the knowledge [**52**] that the insertion of additional information tends to have a stabilizing effect on an ill-conditioned computation. It is not clear from the physical problem how one might constrain p_{12}; we chose to bound the absolute value of p_{12} by the square root of the product of the lower bounds for p_{11} and p_{22}, so that $p_{11}p_{22} - p_{12}^2 > 0$ is always true. Whenever the computed values of $\boldsymbol{p}$ are under the lower bound in the descent search for p_{11} and p_{22}, or above the bound for p_{12}, we set them equal to the bound. With this arrangement,

the algorithm became extremely stable with respect to allowing a large number of the descent iterations. From our test functions we can see that if we can assign a lower bound for p_{12}, the images are substantially improved.

As described in Section 2.5, we need at least five different λ values to recover $\boldsymbol{K}$, Q, and R for the flow equation and six different λ values to recover $\boldsymbol{D}$, θ, B_1, and B_2 for the transport equations. In practice we found that increasing the number of λ values used substantially improved the images. In our tests, we chose the number of λ values to be 20. This is consistent with the view that the ill-posedness in the computational problem corresponds to a certain loss of information in the data, and, as noted above, the most natural way to offset this is to add as much ancillary information as possible.

As is observed earlier, the recovery of the functions $\boldsymbol{D}_i(x)$, $i = 1, \cdots, N$ is essentially equivalent to the recovery of the function $\boldsymbol{D}(\boldsymbol{v}(x,t))$ where the Darcy flow $\boldsymbol{v}(x,t) = -\boldsymbol{K}\nabla\phi$ may be regarded as already known. The remaining task is to recover the time-independent dispersion function $\boldsymbol{D}(\cdot)$ from the information gathered thus far.

The Darcy flow can be regarded as a function $\boldsymbol{v} = \boldsymbol{h}(x,t)$ from $\Omega \times [0,1]$ onto a vector subset, $\boldsymbol{V}$, of R^m, while the time-independent scalar dispersion $\boldsymbol{D}(\cdot)$ is a function from $\boldsymbol{V}$ to the real line R. So, we can at best recover $\boldsymbol{D}(\cdot)$ restricted to $\boldsymbol{V}$. The other issue is that if h is not one-to-one, the numerically recovered $\boldsymbol{D}(\boldsymbol{v}(x,t))$ will most likely not take equal values on those points (x,t) that map to the same flow vectors under h; we take the average of those values as the value of $\boldsymbol{D}(\cdot)$ at that point.

The algorithm works in the two-dimensional case as follows. Let $\{x_{ij}\}$ represent the grid points in Ω, and let t_k denote a partition of the time interval. The vectors

$$v_{ijk} = \boldsymbol{K}(x_{ij})\nabla\phi(x_{ij}, t_k) = (a_{ijk}, b_{ijk})^T$$

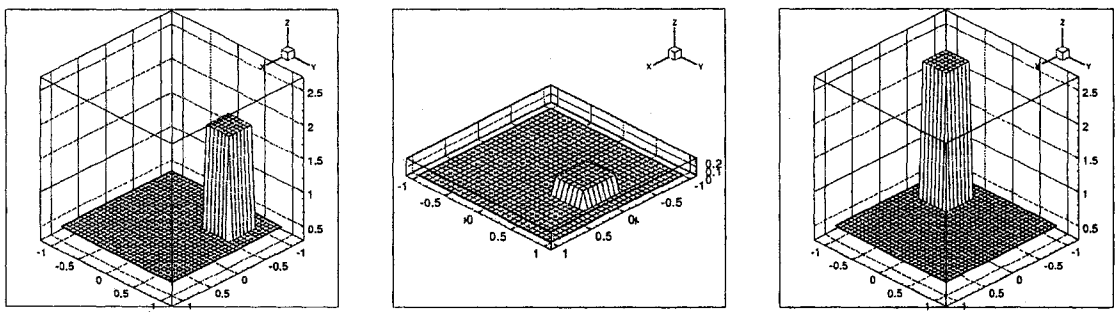
are computed and stored, and the minimum and maximum of the a_{ijk}, $a_{\min}$ and $a_{\max}$ are computed, together with the minimum and maximum of the b_{ijk}, $b_{\min}$ and $b_{\max}$.

The rectangle $\boldsymbol{V} = [a_{\min}, a_{\max}] \times [b_{\min}, b_{\max}]$ is discretized by a grid with stepsize h, the stepsize used in the grid for Ω. Each of the vectors $\boldsymbol{v}_{ijk}$ is then assigned to its grid square in $\boldsymbol{V}$, and for each of the grid squares V_{rs} in $\boldsymbol{V}$, the average of $D(\boldsymbol{v}_{ijk})$ over all of the $\boldsymbol{v}_{ijk}$ in V_{rs} is computed; this is the value of $\boldsymbol{D}(\cdot)$ on V_{rs}. If no v_{ijk} lie in V_{rs} we set $\boldsymbol{D}(V_{rs}) = c_0$, the predefined lower bound for $\boldsymbol{D}(\boldsymbol{v})$ before. The test program shows that this method is effective.

3.2. Results with synthetic data

In our synthetic data test, we assume that the region $\Omega = [-1, 1] \times [-1, 1]$ is overlaid with a 30×30 discretization grid. We deliberately chose all the coefficients to be non-smooth functions, because the non-smooth functions are more difficult to recover, and also because one cannot assume *a priori* that the parameters in a real groundwater system are smooth functions. The time period was set from 0 to 1. The code was written in PGI Fortran 90 in double precision and run on a cluster consisting of Dell PowerEdge 2450 nodes with dual Intel 733MHz Pentium III processors and the Redhat Linux 6.2 Operating System. We used 20 nodes, one for each of the λ-values in the functional H.

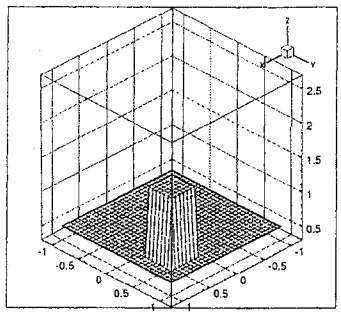
FIGURE 3.1. True parameter functions $\boldsymbol{K}$, Q, and $R - 1$



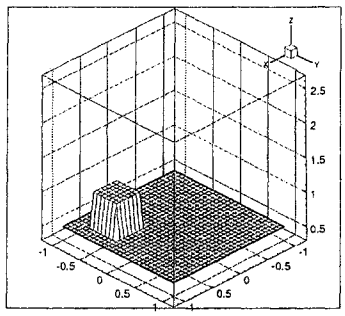
(a) $\boldsymbol{K}_{11}$ (b) $\boldsymbol{K}_{12}$ (c) $\boldsymbol{K}_{22}$

3.2.1. The flow equation. We assume that the time interval $[0, 1]$ is divided into 10 equal subintervals, and the hydraulic conductivity $\boldsymbol{K}$, the storativity S, and

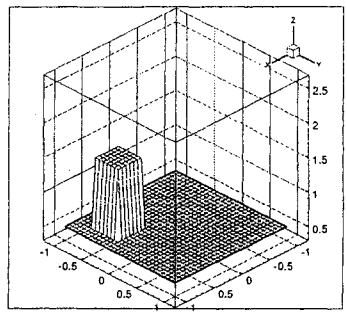
FIGURE 3.2. True parameter functions $\boldsymbol{K}$, Q, and R - 2



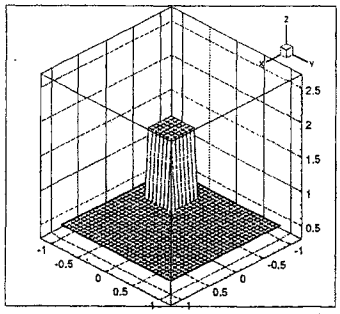
(a) Q



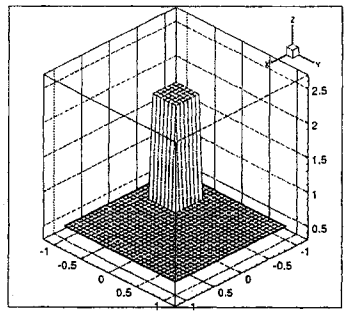
(b) R_1



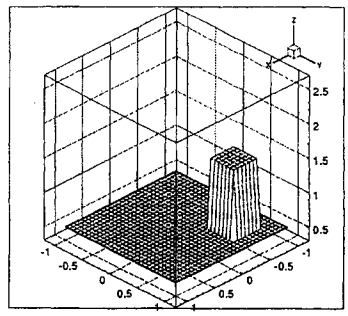
(c) R_2



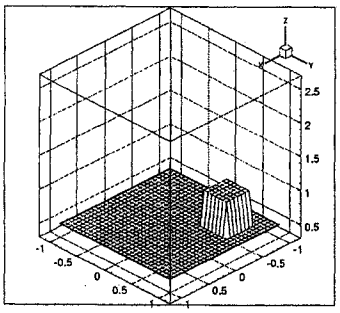
(d) R_3



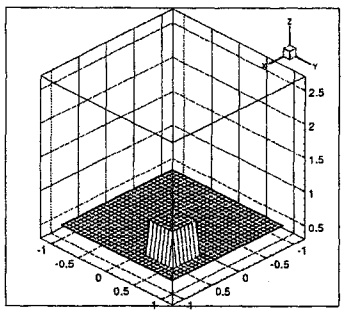
(e) R_4



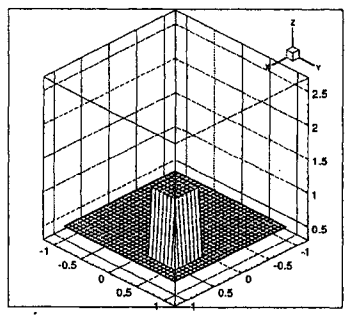
(f) R_5



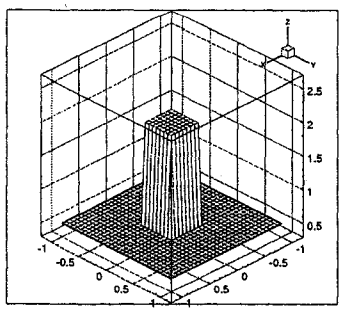
(g) R_6



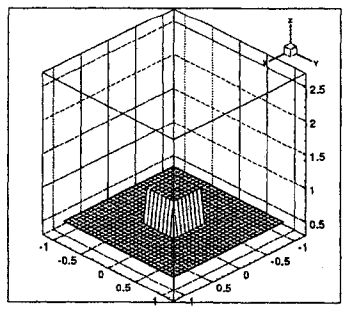
(h) R_7



(i) R_8



(j) R_9



(k) R_{10}

the source/sink term R_k, $1 \leq k \leq 10$, are defined as in Figure 3.2.1 and Figure 3.2.1. And

$$R(x,y,t) = \sum_{k=1}^{10} R_k(x,y)\chi_{[\frac{k-1}{10},\frac{k}{10}]}.$$

The piezometric head data, ϕ, is solved from the flow equation (2.3) with initial condition

$$w(x,y,0) = 2 + 0.5\cos(\pi x)\cos(\pi y),$$

(to simulate slowly varying head data), and boundary conditions

$$w(x,\pm 1,t) = 2 - (0.5 - t)\cos(\pi x),$$

$$w(\pm 1,y,t) = 2 - (0.5 - t)\cos(\pi y),$$

by the PDE package PDETWO [**68**], over the region Ω and time $[0,1]$. Then we use the quadrature implemented with the Simpson's rule to get the data u_i, $1 \leq i \leq 10$.

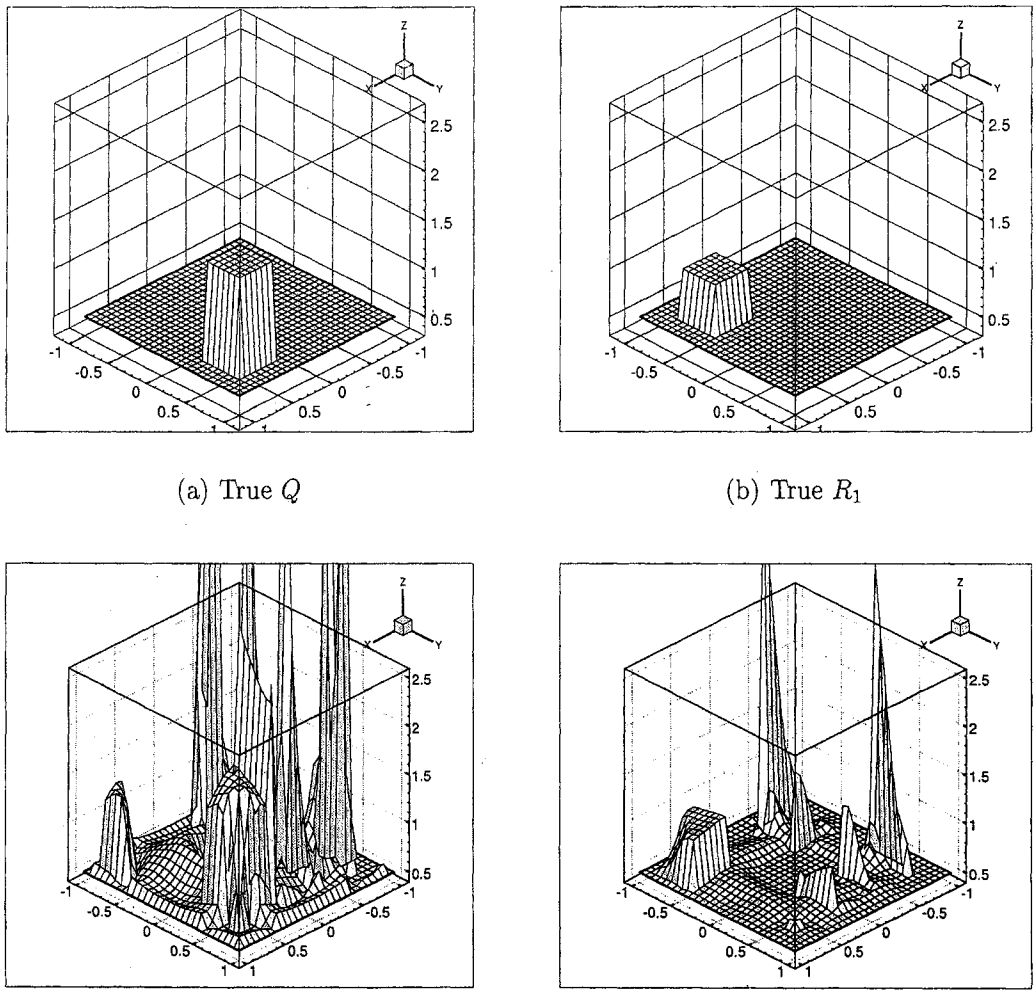
EXAMPLE 3.1. *Assume that the hydraulic conductivity $\boldsymbol{K}$ is known. We recover Q and R simultaneously for* $1,000$ *iteration steps. Figure 3.2.1 shows the search result where we use the $\mathcal{L}^1$ gradient as the descent direction. It can be seen that the result is really bad. If we have some information about the parameters, such as the boundary values of the storativity S, then we can get a much better result (see Figure 3.2.1(a) and Figure 3.2.1(b)) by adopting the Neuberger gradient for the descent direction.*

There is still some "junk" in Figure 3.2.1(a) and Figure 3.2.1(b). This junk comes from the numerical difference of the solutions solved by the two PDE solvers - the parabolic PDE solver PDETWO and our elliptic PDE solver (together with the numerical Laplace transformation). Recall that the source data u_i, $1 \leq i \leq 10$, are computed by the Laplace transformation (2.9) of the solution of the parabolic equation (2.3). Since

$$u_i|_{\partial\Omega} = \int_{t_{i-1}}^{t_i} e^{-\lambda t}\phi(x,t)|_{\partial\Omega} dt,$$

we use this for the boundary values to solve the elliptic equation (2.8), with the true parameters $\boldsymbol{K}$, S, and R, and use these u_k, $1 \leq k \leq 10$, as source data. Then all the

FIGURE 3.3. The recovery of Q and R with $\boldsymbol{K}$ fixed with $\mathcal{L}^1$ gradient



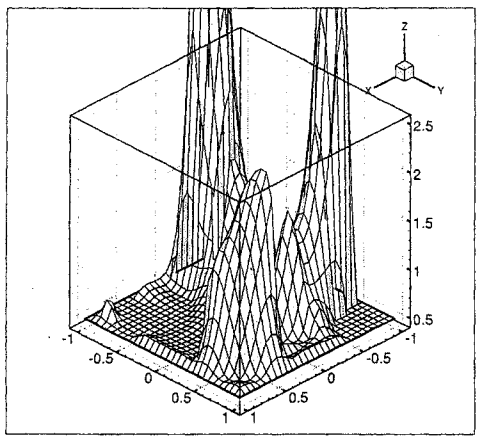
(a) True Q (b) True R_1

(c) Recovered Q (d) Recovered R_1

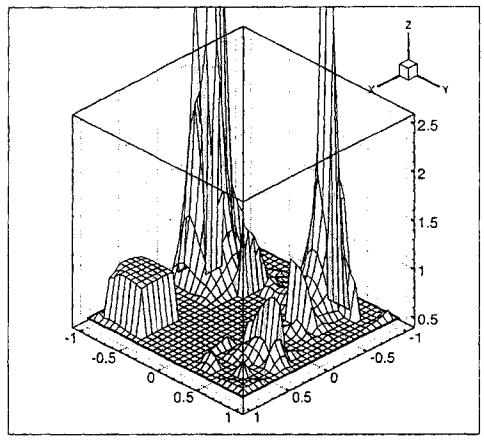
junk disappears, as in Figure 3.2.1(c) and Figure 3.2.1(d). Note that we still need the solution ϕ of equation (2.3) to compute α, but we can regard them as fixed once ϕ is known.

Actually, the difference of the two solutions is very small. But the difference of the numerical derivatives is much bigger than the difference of the solutions. Figure 3.2.1 shows the differences of the solutions and the corresponding numerical derivatives between the solutions, with time period set $k = 1$ and $\lambda = 0.5$. Since PDETWO is only used to generate the synthetic data, and since the source data is gathered by

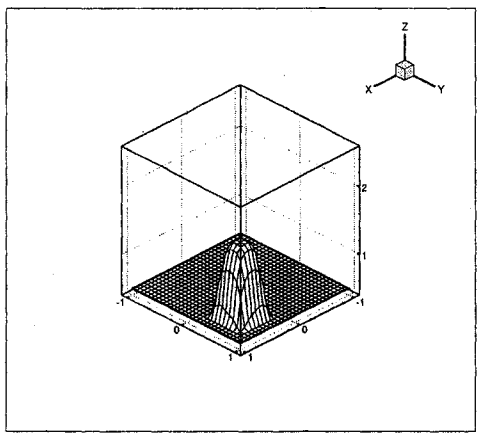
FIGURE 3.4. The recovery of Q and R with $\boldsymbol{K}$ fixed with Neuberger gradient



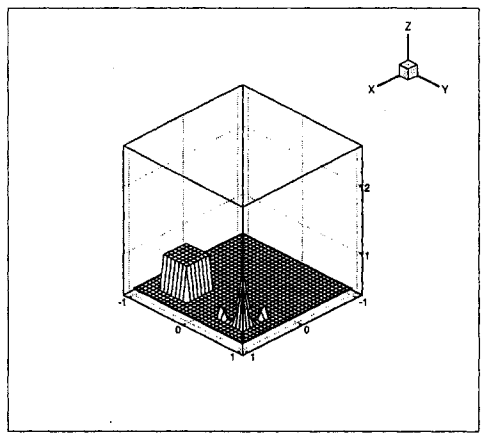
(a) Recovered Q



(b) Recovered R_1



(c) Recovered Q



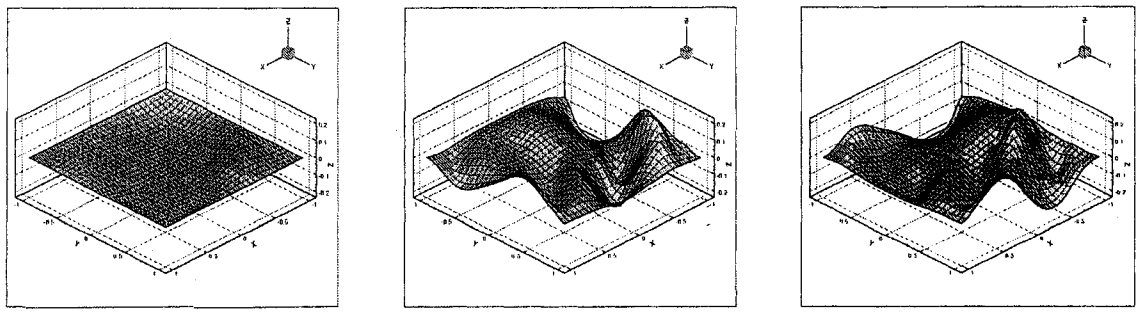
(d) Recovered R_1

field measurements in the real situation, we assume in the following examples that the synthetic data u_i, $1 \leq i \leq 10$, are generated by our elliptic solver together with the boundary values above. Example 3.1 also tells us that the more information we have about the recovered parameters, the more accurate the result. We assume that in the following examples we will use the Neuberger gradient for the descent directions.

EXAMPLE 3.2. *Assuming that the storativity S and the source term R are known. We simultaneously recover the coefficients, K_{11}, K_{12}, and K_{22}, of the anisotropic hydraulic conductivity $\boldsymbol{K}$. After 1,000 descent steps, we get the result shown in*

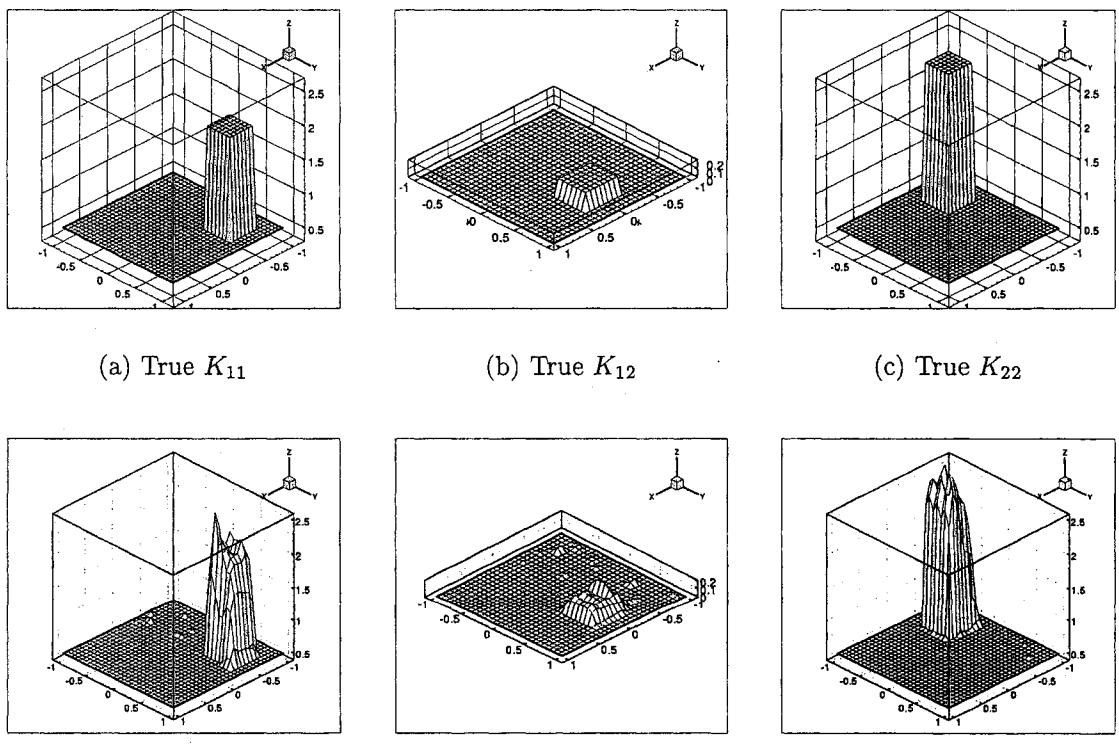
Figure 3.2.1. We can see that the result is good both in shape and height, and the discontinuity is quite clear.

FIGURE 3.5. Difference of solutions between the two PDE Solvers when $k = 1$ and $\lambda = 0.5$



(a) difference of u (b) difference of $\frac{\partial u}{\partial x}$ (c) difference of $\frac{\partial u}{\partial y}$

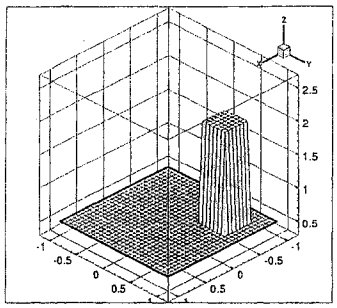
FIGURE 3.6. The recovery of $\boldsymbol{K}$ when Q, R are assumed known



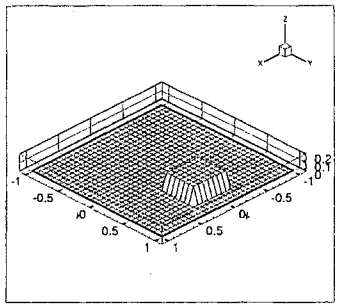
(a) True K_{11} (b) True K_{12} (c) True K_{22}

(d) Recovered K_{11} (e) Recovered K_{12} (f) Recovered K_{22}

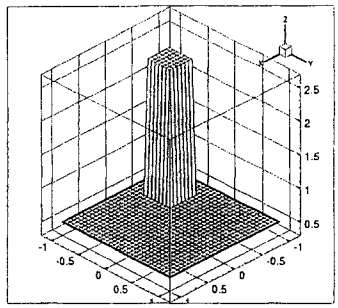
FIGURE 3.7. The recovery of the parameters of flow equation of unconfined aquifer – 1



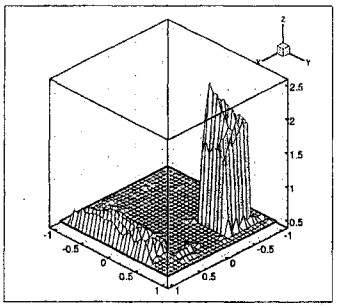
(a) true K_{11}



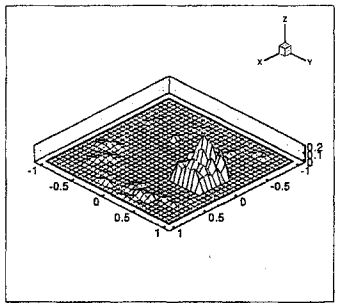
(b) true K_{12}



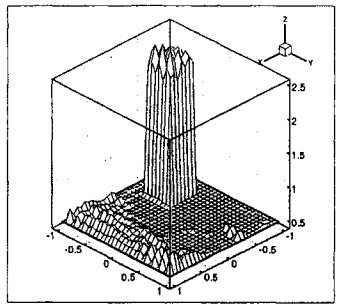
(c) true K_{22}



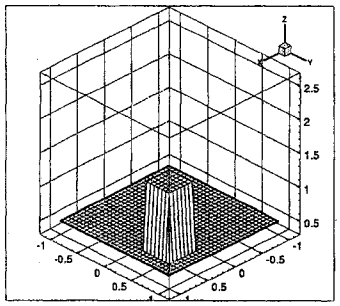
(d) recovered K_{11}



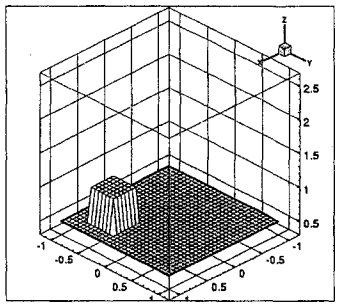
(e) recovered K_{12}



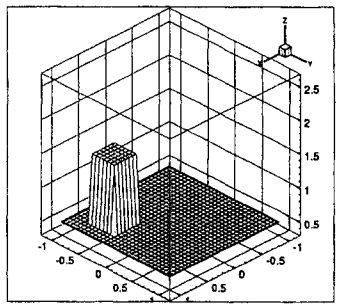
(f) recovered K_{22}



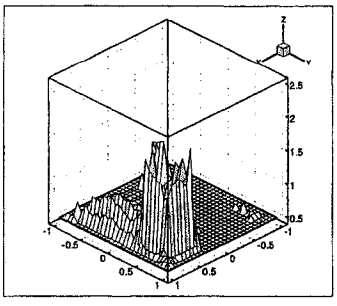
(g) true Q



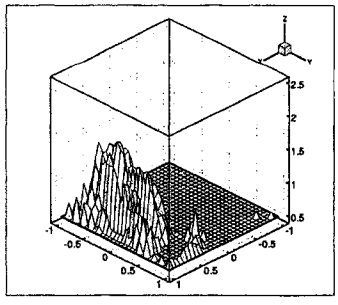
(h) true R_1



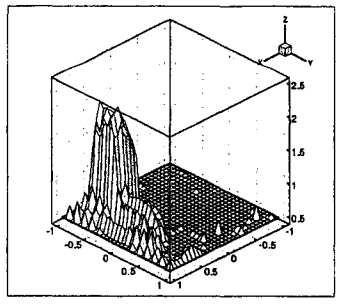
(i) true R_2



(j) recovered Q



(k) recovered R_1



(l) recovered R_2

EXAMPLE 3.3. *As a final example of the flow equation, we simultaneously recovered all the coefficients of the flow equation (2.5) in an unconfined aquifer. The true parameters* $\boldsymbol{K}$, Q, *and* R_i, $1 \leq i \leq 10$, *were as in Figure 3.2.1 and Figure 3.2.1. Our search was scheduled as follows. We set an arbitrary initial position* $c_0 = (\boldsymbol{p}_0, q_0, r_{1_0})$. *With the descent minimization, we searched to* c_3 *by using the source data* u_1 *as follows*

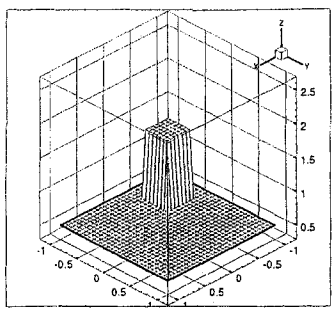
$$c_0^1 = (\boldsymbol{p}_0, q_0, r_{1_0}) \longrightarrow c_1^1 = (\boldsymbol{p}_1, q_0, r_{1_0}) \longrightarrow \cdots \longrightarrow c_3^1 = (\boldsymbol{p}_1, q_1, r_{1_1}).$$

Then we used $c_0^2 = (\boldsymbol{p}_1, q_1, r_{2_0})$, *where* r_{2_0} *was chosen arbitrarily, as the initial position and searched to* $c_3^2 = (\boldsymbol{p}_2, q_2, r_{2_1})$ *with source data* u_2. *Adopting the same procedure, we searched to point* $c_3^{10} = (\boldsymbol{p}_{10}, q_{10}, r_{10_1})$. *In the second iteration, we chose* $c_0^1 = (\boldsymbol{p}_{10}, q_{10}, r_{1_1})$ *as the initial point and made a further search. After a total of* $5,000$ *descent steps, we got the recovered parameters as listed in Figure 3.2.1, Figure 3.3, and Figure 3.3. We can see that all the parameters recovered are quite accurate.*

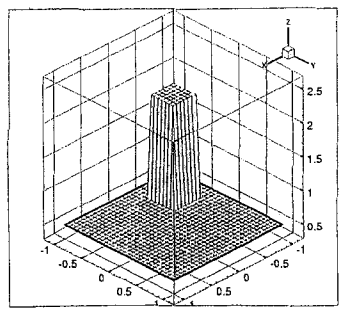
It should be noted that the true S here is not physically reasonable in the groundwater context because the possible values for S are very small, at 0.003ft^{-1} for clay, with very high compressibility and porosity, for example. This arises in the following example.

EXAMPLE 3.4. *In this example, we set* S *to be much smaller, between* 0.0005 *and* 0.0015, *as shown in Figure 3.2.1. The hydraulic conductivity,* $\boldsymbol{K}$, *was set to be isotropic, while the source/sink term,* R *was set as a step function of* R_i, $i = 1, \cdots, 6$. *If we assumed that the aquifer was confined, and we recovered all the parameters simultaneously for* $5,000$ *steps. The recovered parameters are shown in Figure 3.2.1 and Figure 3.2.1. It can be seen that all the parameters except* S *are very accurate. To prove that our recovery was effective, we computed the relative error between the "true" source data and our recovered data. The idea is as follows: regard the source data* ϕ *solved by PDETWO with the true* $\boldsymbol{K}$, S, *and* R_i, $i = 1, \cdots, 6$ *as the true source data; the data solved by PDETWO with the recovered* $\boldsymbol{K}$, S, *and* R_i, $i = 1, \cdots, 6$ *is*

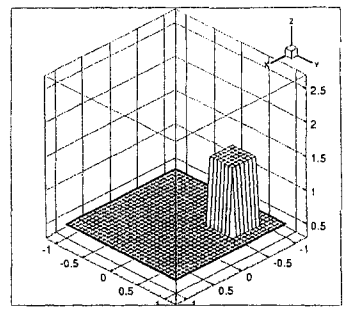
FIGURE 3.8. The recovery of the parameters of flow equation of unconfined aquifer – 2



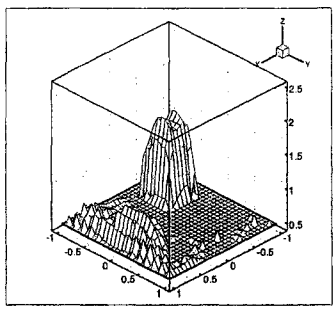
(a) true R_3



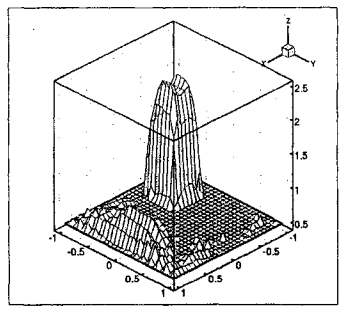
(b) true R_4



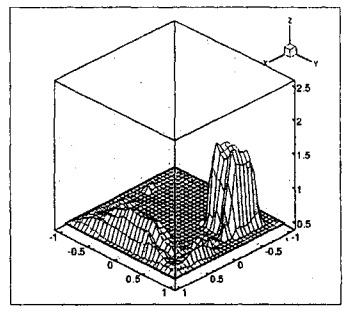
(c) true R_5



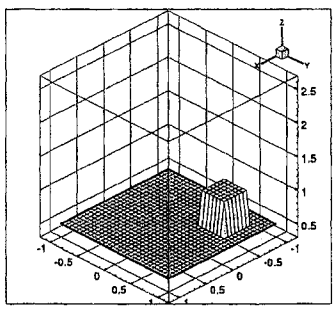
(d) recovered R_3



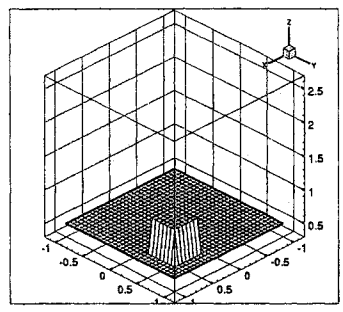
(e) recovered R_4



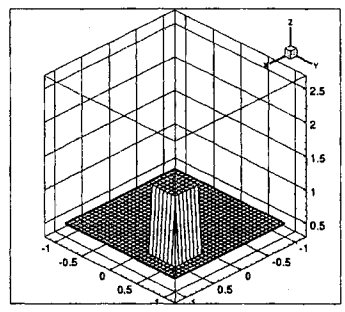
(f) recovered R_5



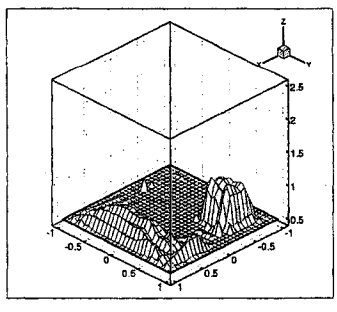
(g) true R_6



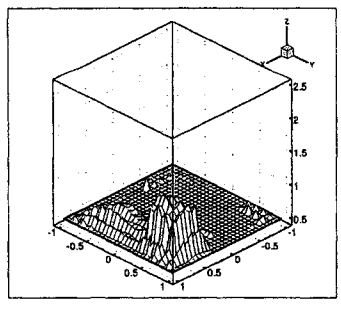
(h) true R_7



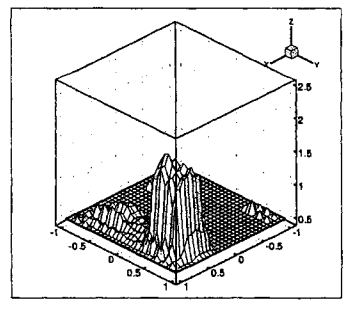
(i) true R_8



(j) recovered R_6

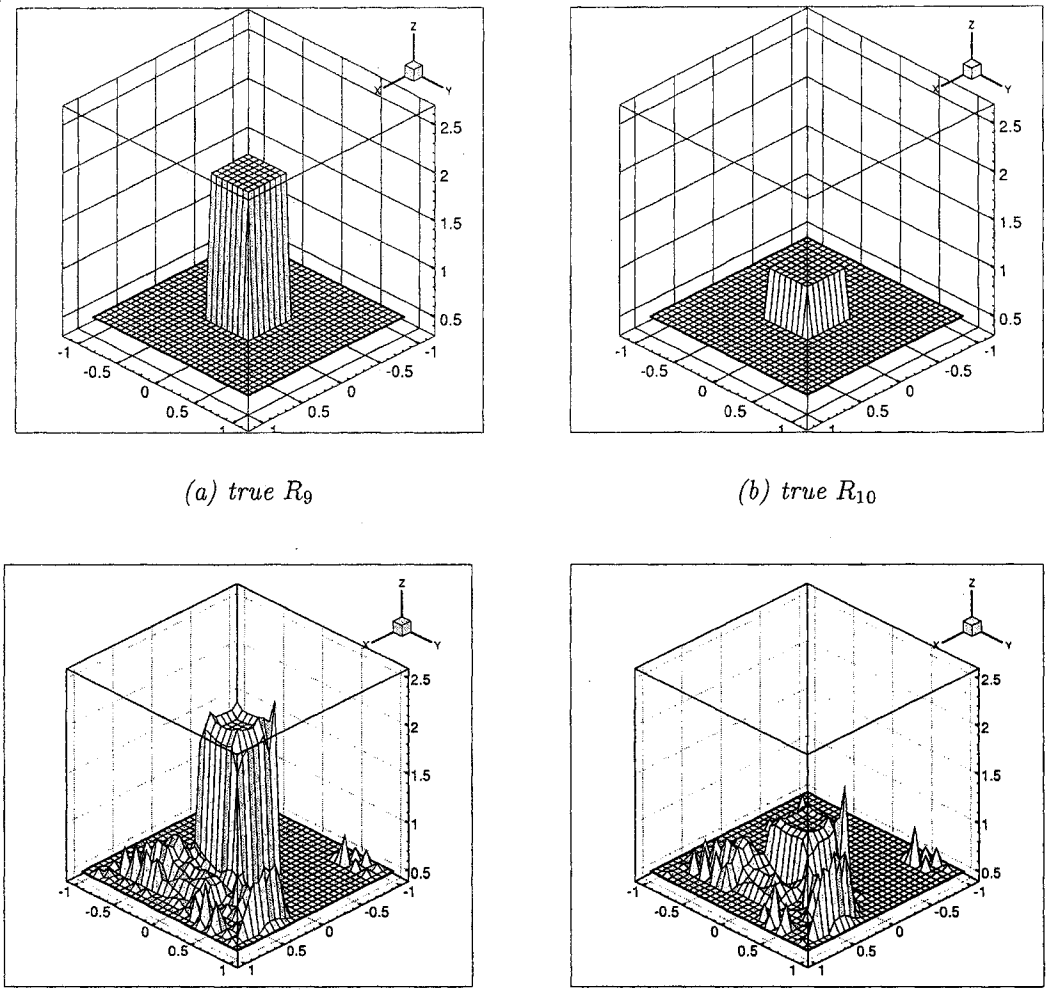


(k) recovered R_7



(l) recovered R_8

FIGURE 3.9. The recovery of the parameters of flow equation of unconfined aquifer – 3



(a) true R_9

(b) true R_{10}

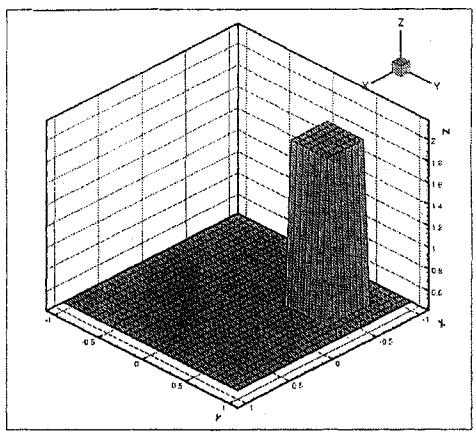
(c) recovered R_9

(d) recovered R_{10}

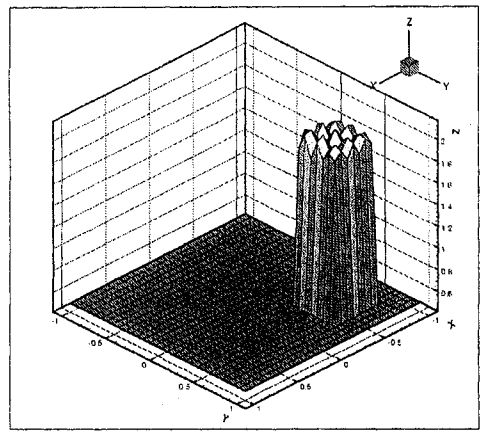
regarded as the recovered data. We then computed the relative error between ϕ and ϕ_1. The result is shown in Figure 3.2.1. It can be seen that the error is below 1%, very small. It also agrees with [**83**]*, in which it is stated that the storativity is insensitive to small changes of piezometric head.*

However, in the real situation, the source/sink term R in the flow equation is also very small. This difficulty can be avoided by applying a variable substitution (see Section 4.6).

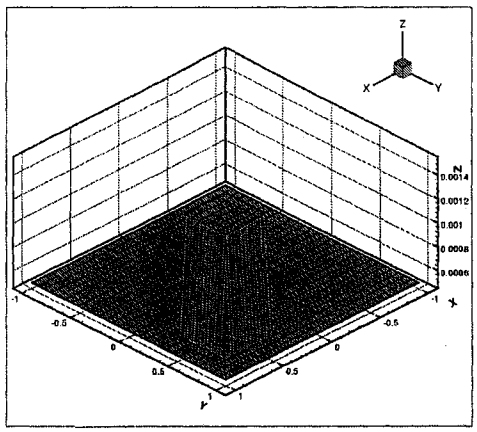
FIGURE 3.10. Recovery with small $S - 1$



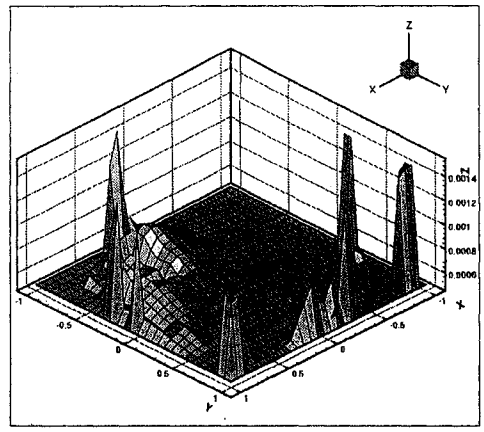
(a) true $\boldsymbol{K}$



(b) recovered $\boldsymbol{K}$



(c) true S

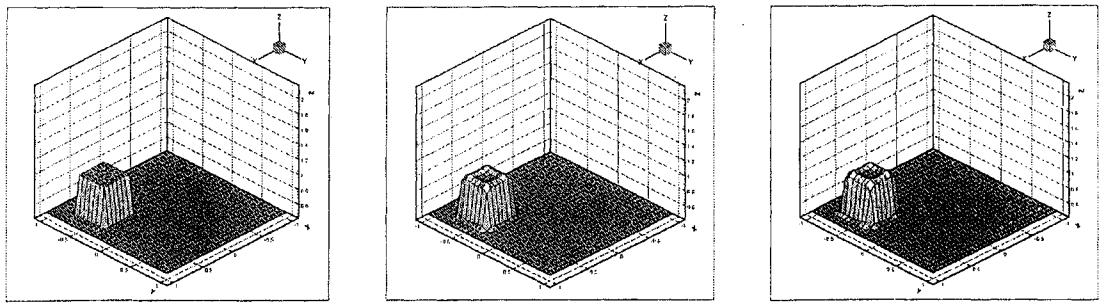


(d) recovered S

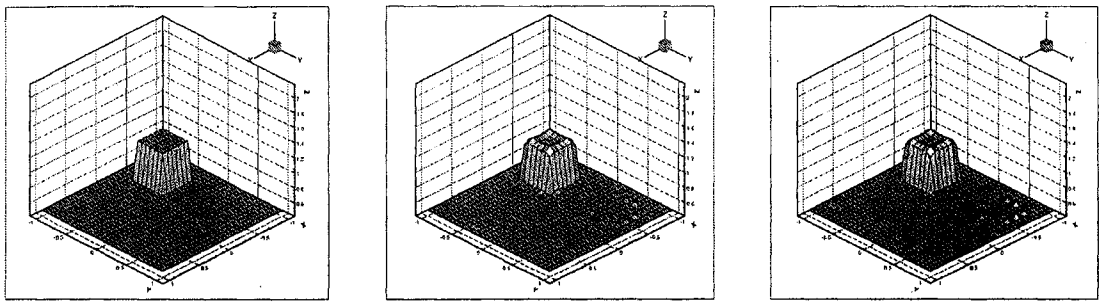
3.2.2. The transport equation. Assume that the parameters in the flow equation are already known and the true parameters for the transport equation are defined as in Figure 3.2.2, Figure 3.2.2, and Figure 3.2.2. We will test the case of an unconfined aquifer here (similar results can be obtained for the confined case). Note that since the parameters $(\boldsymbol{K}, S, R)$ are regarded as known, the parameter δ_k in equation (2.19) can be computed as follows:

$$\delta_k(x, \lambda) = \int_{t_{k-1}}^{t_k} e^{-\lambda t} \nabla \cdot (c\boldsymbol{K}\phi\nabla\phi)dt$$

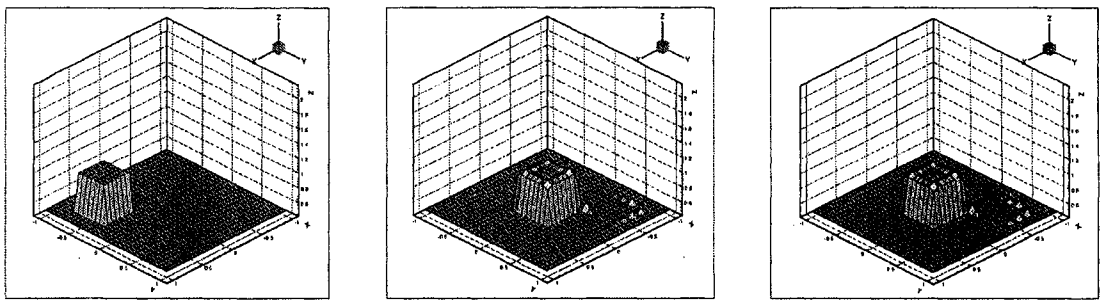
FIGURE 3.11. Recovery with small S - 2



(a) true R_1, R_2 (b) recovered R_1 (c) recovered R_2



(d) true R_3, R_4 (e) recovered R_3 (f) recovered R_4



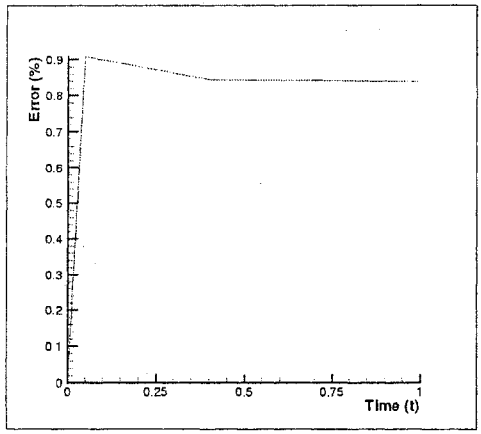
(g) true R_5, R_6 (h) recovered R_5 (i) recovered R_6

$$= \int_{t_{i-1}}^{t_i} e^{-\lambda t}\{c\nabla \cdot \boldsymbol{K}\phi\nabla\phi + \boldsymbol{K}\phi\nabla\phi \cdot \nabla c\}dt$$

$$= \int_{t_{i-1}}^{t_i} e^{-\lambda t}\{c[S(x)\frac{\partial\phi}{\partial t} - R_k(x,t)] + \boldsymbol{K}\phi\nabla\phi \cdot \nabla c\}dt.$$

EXAMPLE 3.5. *In this example, the piezometric head ϕ is solved, as in the flow equation where the parameters $\boldsymbol{K}$, S, and R are those recovered in Example 3.3. Then*

FIGURE 3.12. Error between the recovered data and the true source data with small S



$$||\phi(t,x) - \phi_1(t,x)||_{\mathcal{L}^\infty(\Omega)}$$

we use the parabolic PDE solver PDETWO to solve the transport equation where the parameters are those listed in Figure 3.2.2, Figure 3.2.2, and Figure 3.2.2, with initial condition

$$w(x,y,0) = 1 + 0.5\sin(8\pi(x+y))$$

and boundary values

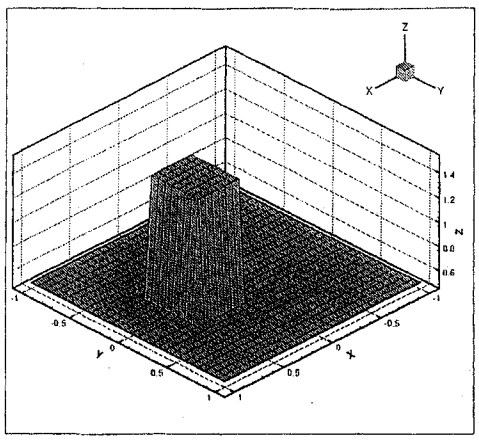
$$w(x,\pm 1,t) = 1 + (0.5-t)\cos(8\pi(x \pm 1)),$$

$$w(\pm 1,y,t) = 1 + (0.5-t)\cos(8\pi(y \pm 1)).$$

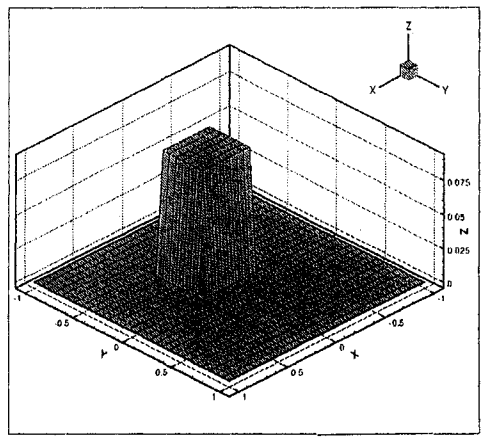
Of those parameters, $\boldsymbol{D}(\cdot)$ is the most difficult to recover. So here we assume that all the B coefficients are known, and we recover D_{11}, D_{12}, D_{22}, and θ simultaneously. After $5,000$ iteration steps, we get the recovered coefficients shown in Figure 3.2.2.

Note that the non-smoothness of $\boldsymbol{K}$, when combined with the non-smoothness of $\boldsymbol{D}$ itself and possible problems with the finite difference solvers, causes increased difficulties with the $\boldsymbol{D}(\boldsymbol{K}\phi\nabla\phi)$ term to recover [59]. *However, we can see in Figure 3.2.2 that the computed $\boldsymbol{D}(\cdot)$ assembled from the recovered D_{ij_k}'s, except D_{12}, is an effective reconstruction.*

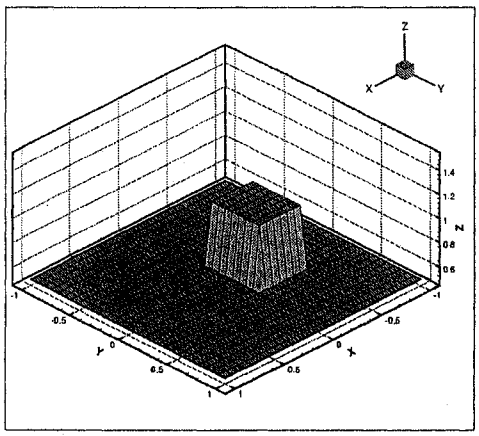
FIGURE 3.13. True parameter $\boldsymbol{D}$, θ of the transport equation



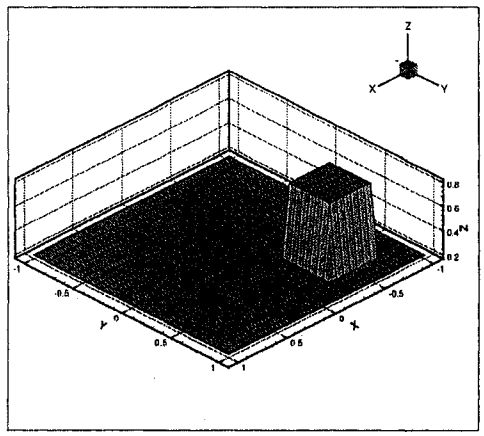
(a) true D_{11}



(b) true D_{12}



(c) true D_{22}

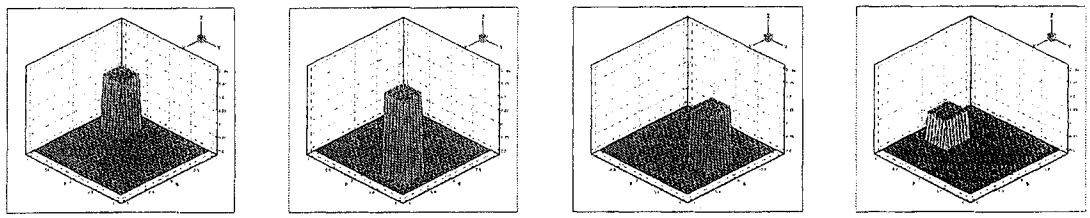


(d) true θ

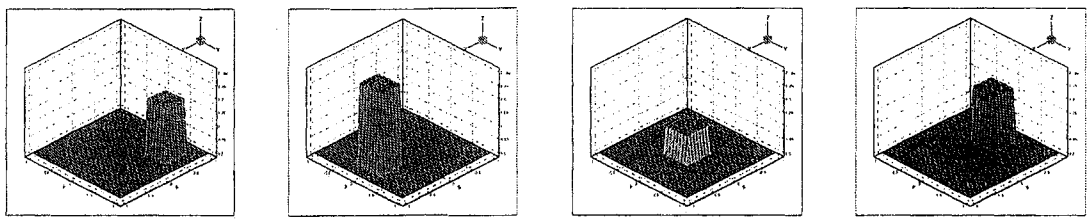
EXAMPLE 3.6. *In this example, all the data and parameters are the same as in Example 3.5. We assume that* $\boldsymbol{D}$ *and* θ *are known and recover the coefficients* B_k^1 *and* B_k^2, $k = 1, \cdots, 20$. *After a total of* $5,000$ *iteration steps, we get the recovered parameters,* B_k^1 *and* B_k^2, $1 \leq k \leq 20$, *shown in Figure 3.2.2, Figure 3.2.2, Figure 3.2.2, Figure 3.2.2, Figure 3.2.2, Figure 3.2.2, and Figure 3.2.2. It can be seen that the recovery is quite accurate.*

In the previous example, we recovered $\boldsymbol{D}(\cdot)$ directly from the $\boldsymbol{D}(\boldsymbol{K}\phi\nabla\phi)$ term. The hydrologists tend to write $\boldsymbol{D}(\cdot)$ as the sum of (D_{ij}) and $(T_{ij}^{\star})D_d$, (Section 1.3),

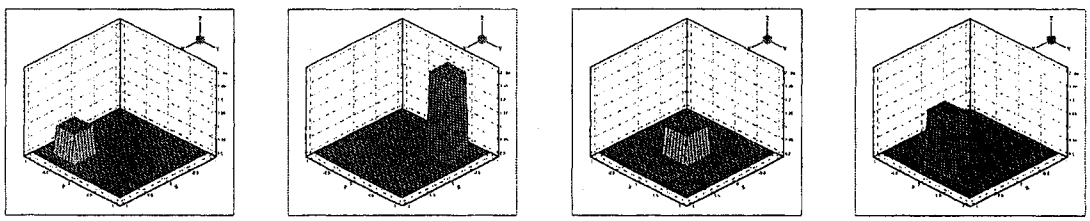
FIGURE 3.14. True parameter B^1 of the transport equation



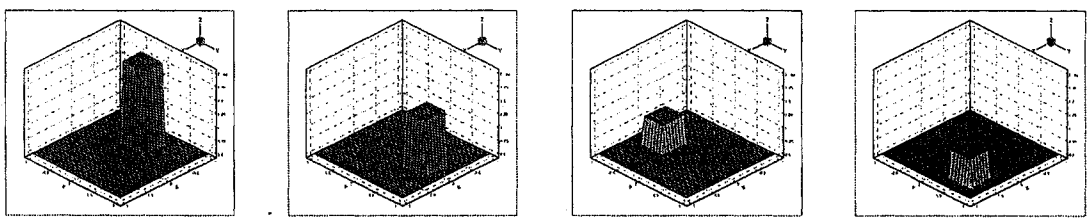
(a) true B_1^1 (b) true B_2^1 (c) true B_3^1 (d) true B_4^1



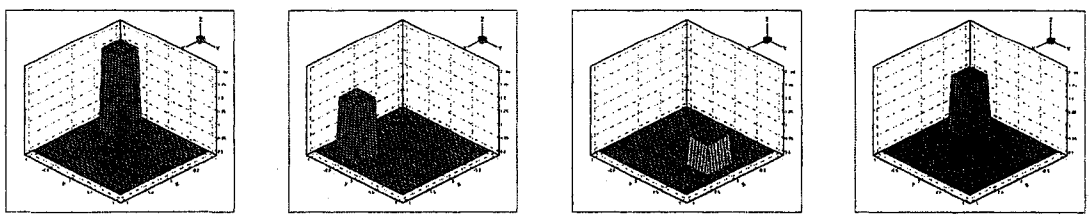
(e) true B_5^1 (f) true B_6^1 (g) true B_7^1 (h) true B_8^1



(i) true B_9^1 (j) true B_{10}^1 (k) true B_{11}^1 (l) true B_{12}^1

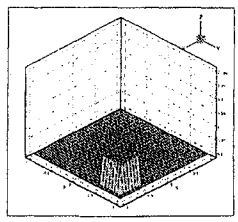


(m) true B_{13}^1 (n) true B_{14}^1 (o) true B_{15}^1 (p) true B_{16}^1

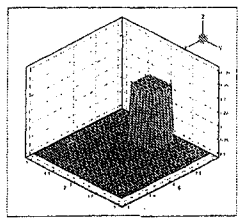


(q) true B_{17}^1 (r) true B_{18}^1 (s) true B_{19}^1 (t) true B_{20}^1

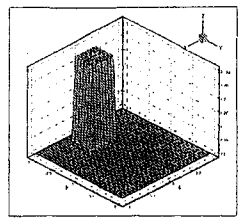
FIGURE 3.15. True parameter B^2 of the transport equation



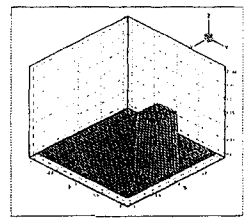
(a) true B_1^2



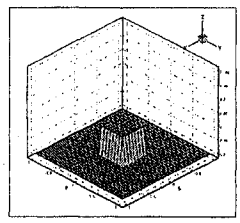
(b) true B_2^2



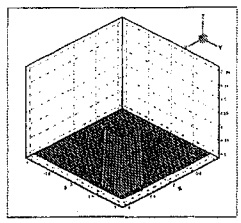
(c) true B_3^2



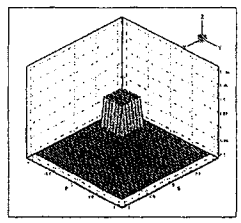
(d) true B_4^2



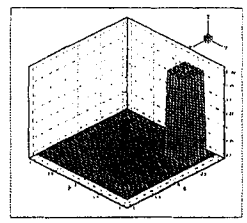
(e) true B_5^2



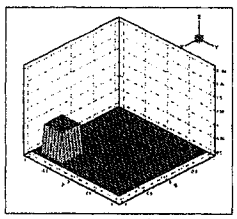
(f) true B_6^2



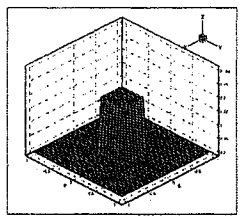
(g) true B_7^2



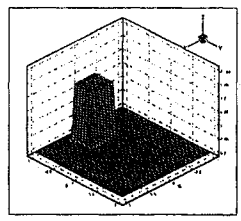
(h) true B_8^2



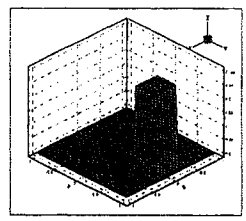
(i) true B_9^2



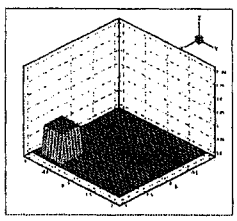
(j) true B_{10}^2



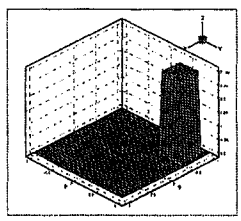
(k) true B_{11}^2



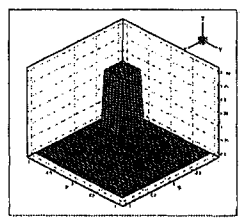
(l) true B_{12}^2



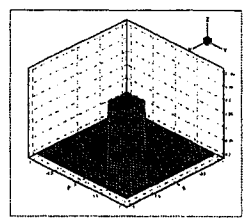
(m) true B_{13}^2



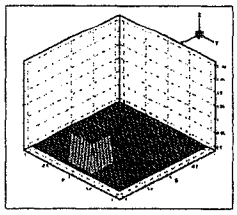
(n) true B_{14}^2



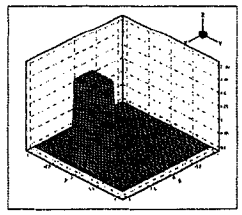
(o) true B_{15}^2



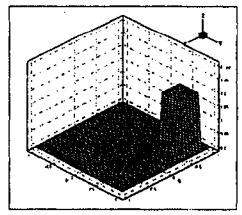
(p) true B_{16}^2



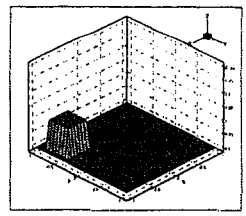
(q) true B_{17}^2



(r) true B_{18}^2

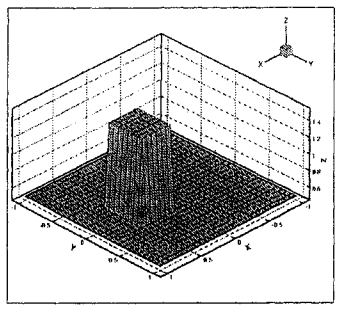


(s) true B_{19}^2

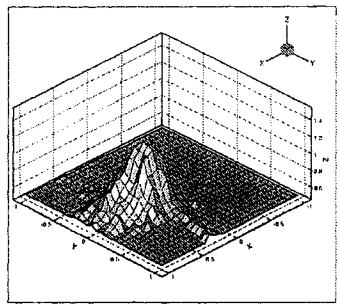


(t) true B_{20}^2

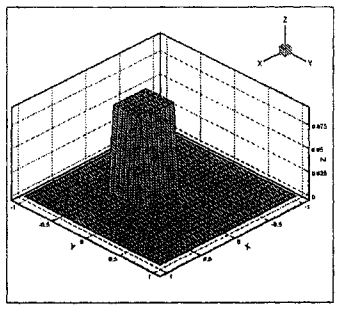
FIGURE 3.16. Recovered $\boldsymbol{D}(\cdot)$ and θ, assuming B known



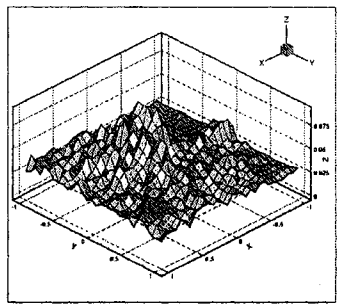
(a) true D_{11}



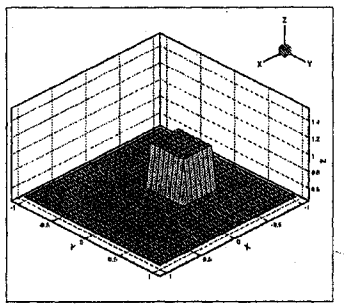
(b) recovered D_{11}



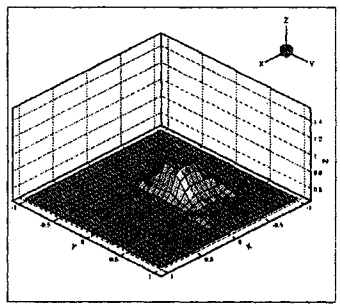
(c) true D_{12}



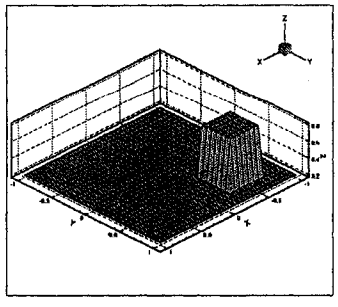
(d) recovered D_{12}



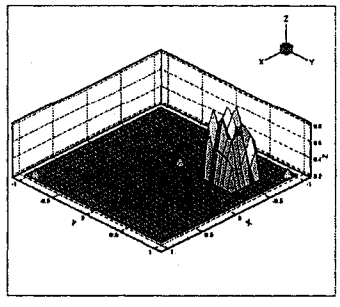
(e) true D_{22}



(f) recovered D_{22}

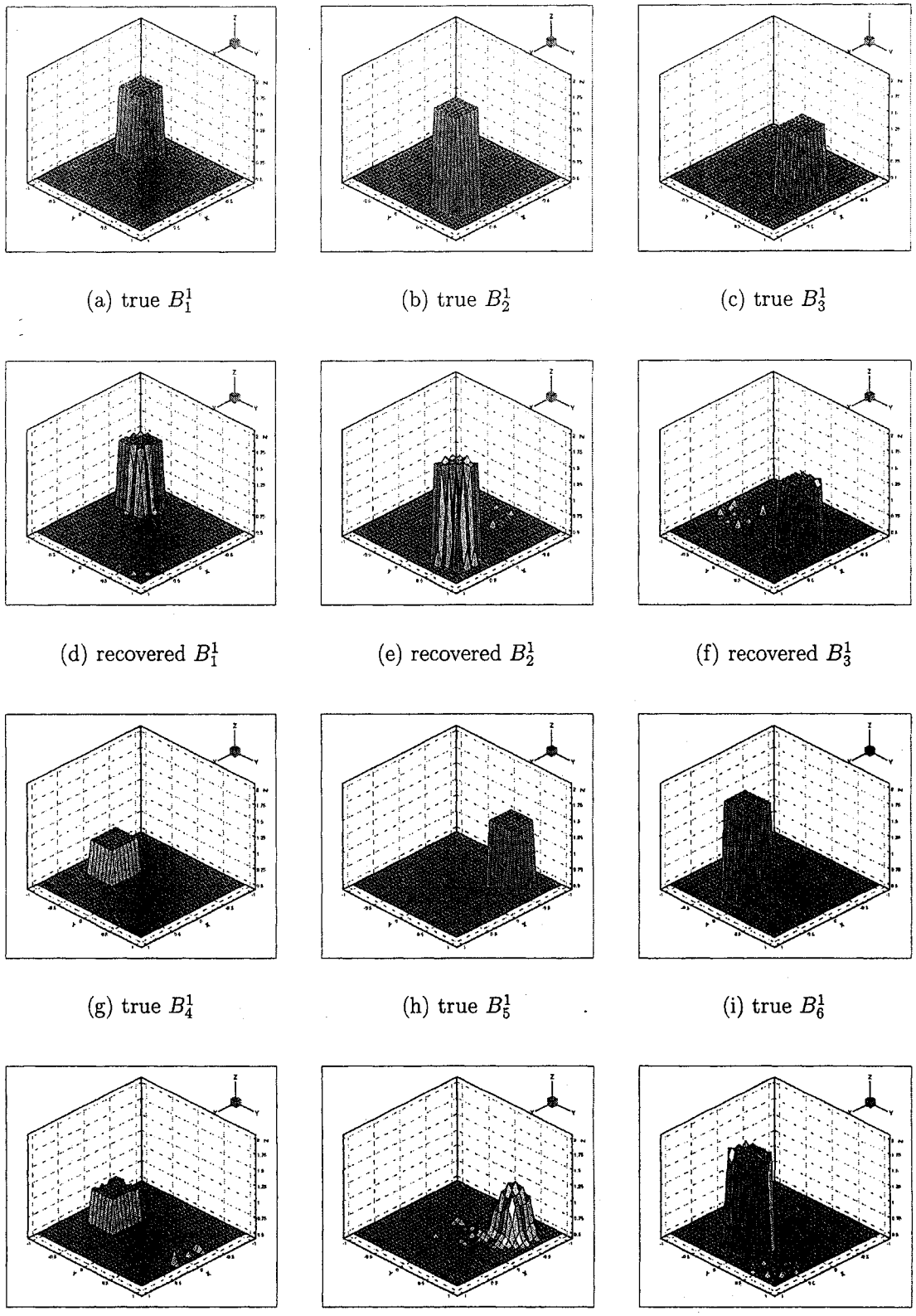


(g) true θ



(h) recovered θ

FIGURE 3.17. Recovered $B_1^1 - B_6^1$, assuming $\boldsymbol{D}$ and θ are known



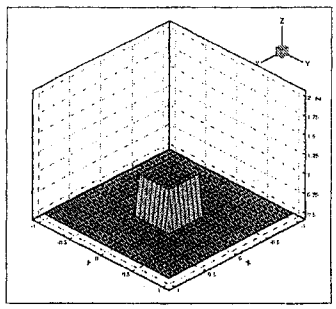
(a) true B_1^1 (b) true B_2^1 (c) true B_3^1

(d) recovered B_1^1 (e) recovered B_2^1 (f) recovered B_3^1

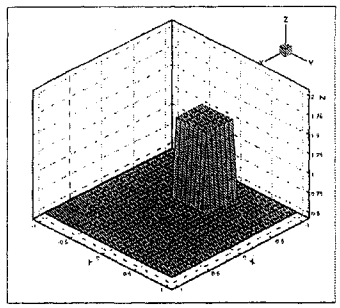
(g) true B_4^1 (h) true B_5^1 (i) true B_6^1

(j) recovered B_4^1 (k) recovered B_5^1 (l) recovered B_6^1

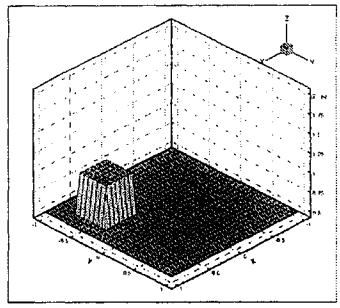
FIGURE 3.18. Recovered B_7^1 – B_{12}^1, assuming $\boldsymbol{D}$ and θ are known



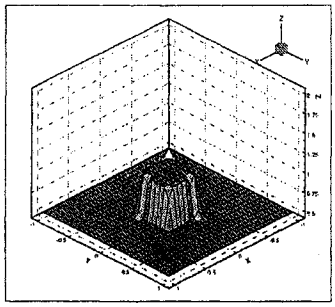
(a) true B_7^1



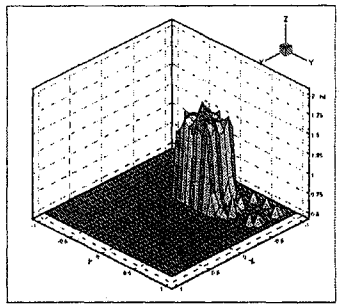
(b) true B_8^1



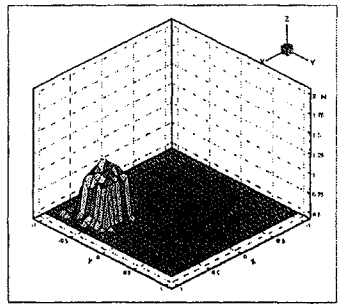
(c) true B_9^1



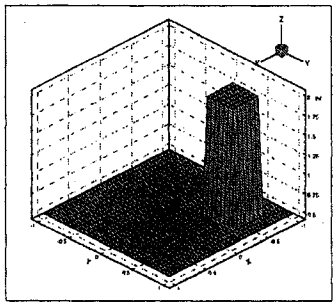
(d) recovered B_7^1



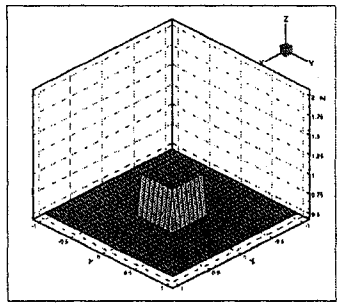
(e) recovered B_8^1



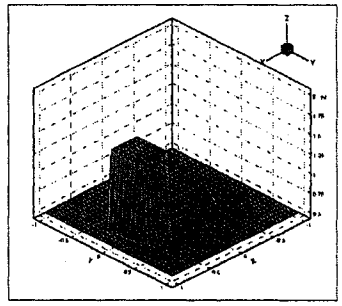
(f) recovered B_9^1



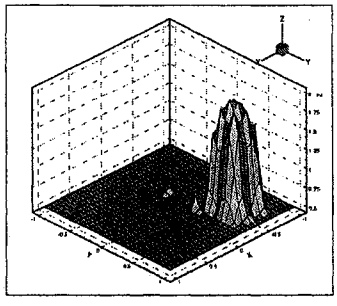
(g) true B_{10}^1



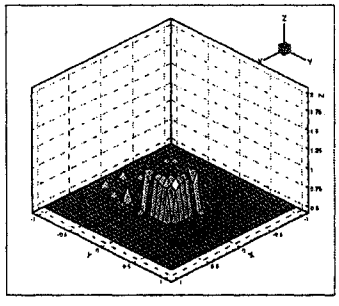
(h) true B_{11}^1



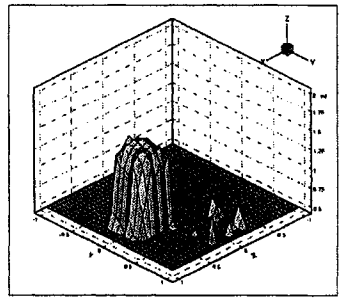
(i) true B_{12}^1



(j) recovered B_{10}^1

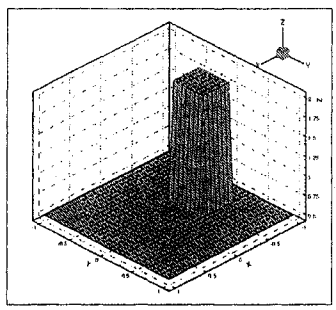


(k) recovered B_{11}^1

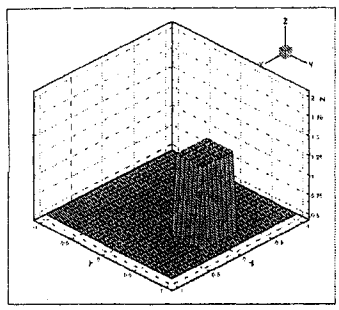


(l) recovered B_{12}^1

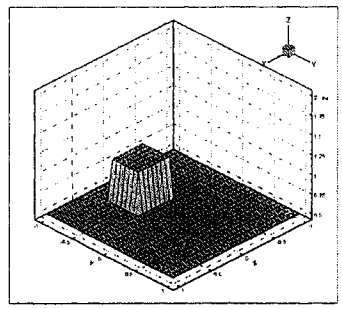
FIGURE 3.19. Recovered B_{13}^1 – B_{18}^1, assuming $\boldsymbol{D}$ and θ are known



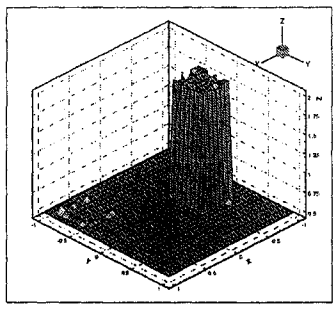
(a) true B_{13}^1



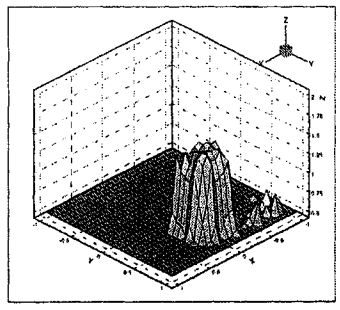
(b) true B_{14}^1



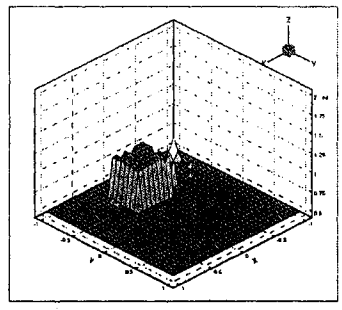
(c) true B_{15}^1



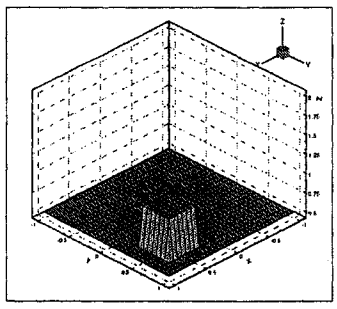
(d) recovered B_{13}^1



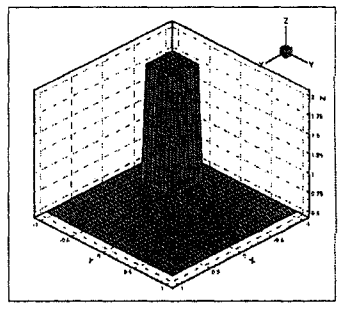
(e) recovered B_{14}^1



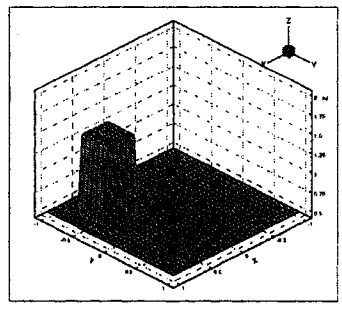
(f) recovered B_{15}^1



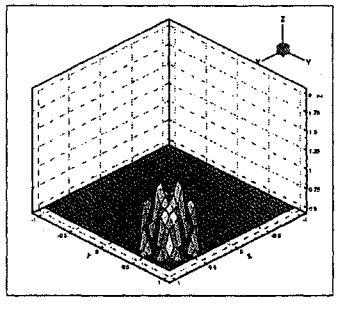
(g) true B_{16}^1



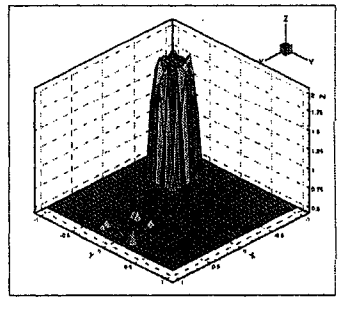
(h) true B_{17}^1



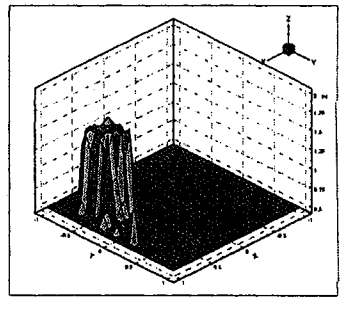
(i) true B_{18}^1



(j) recovered B_{16}^1

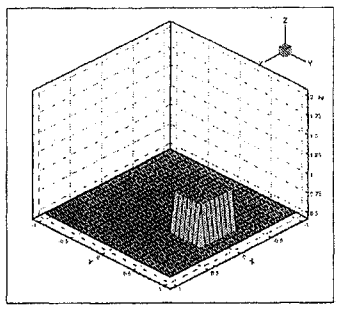


(k) recovered B_{17}^1

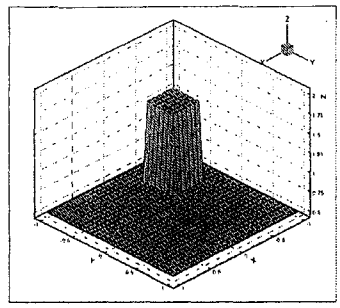


(l) recovered B_{18}^1

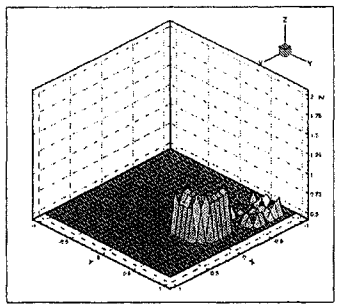
FIGURE 3.20. Recovered B_{19}^1, B_{20}^1, B_{19}^2 and B_{20}^2, assuming $\boldsymbol{D}$ and θ are known



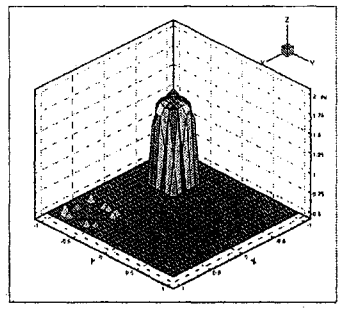
(a) true B_{19}^1



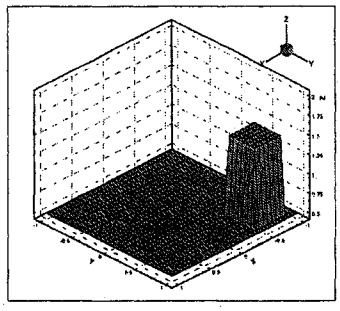
(b) true B_{20}^1



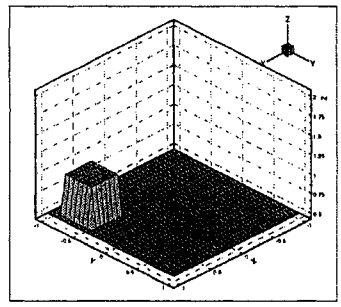
(c) recovered B_{19}^1



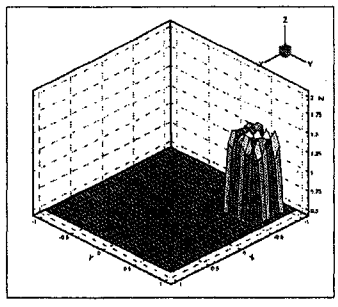
(d) recovered B_{20}^1



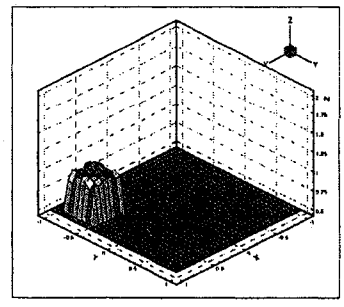
(e) true B_{19}^2



(f) true B_{20}^2

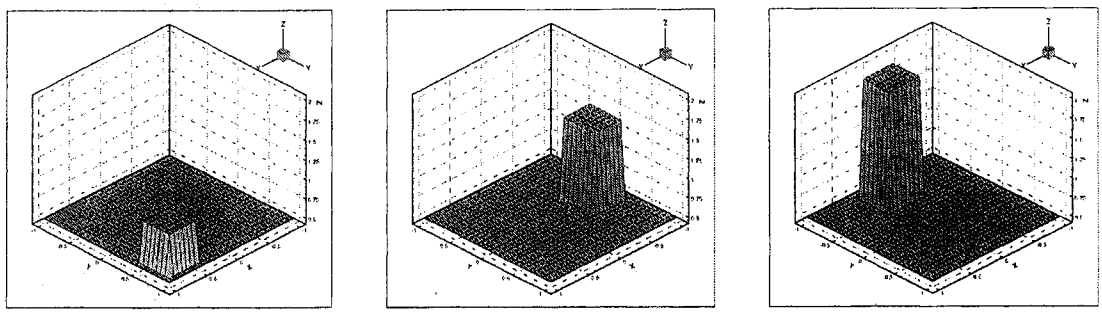


(g) recovered B_{19}^1

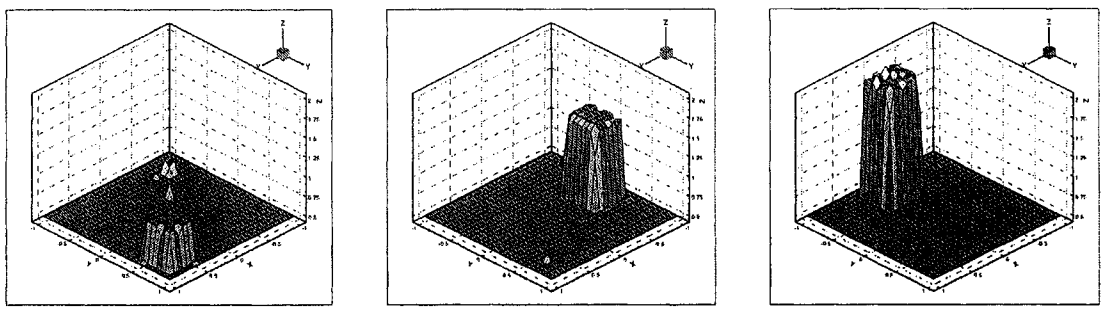


(h) recovered B_{20}^1

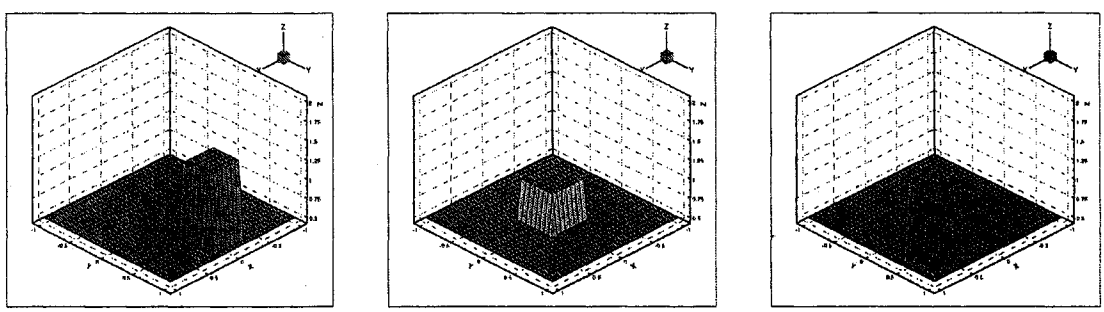
FIGURE 3.21. Recovered $B_1^2 - B_6^2$, assuming $\boldsymbol{D}$ and θ are known



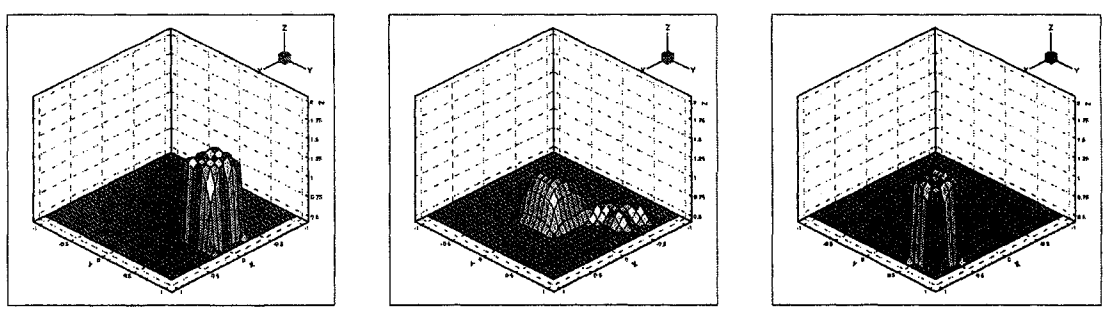
(a) true B_1^2 (b) true B_2^2 (c) true B_3^2



(d) recovered B_1^2 (e) recovered B_2^2 (f) recovered B_3^2

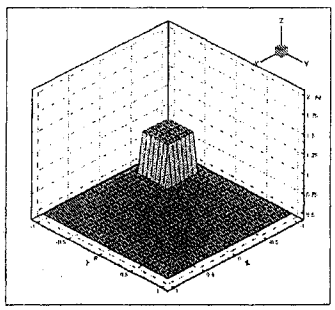


(g) true B_4^2 (h) true B_5^2 (i) true B_6^2

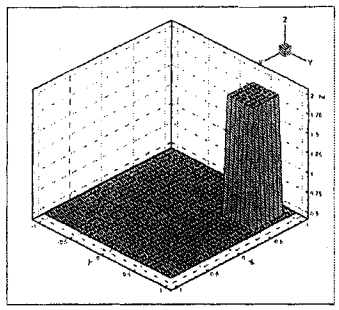


(j) recovered B_4^2 (k) recovered B_5^2 (l) recovered B_6^2

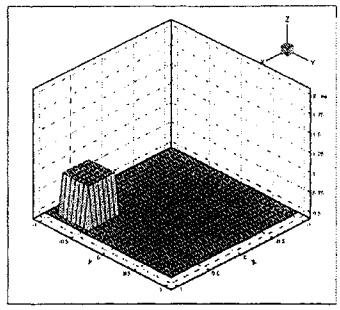
FIGURE 3.22. Recovered B_7^2 – B_{12}^2, assuming $\boldsymbol{D}$ and θ are known



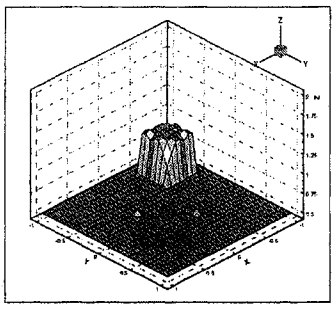
(a) true B_7^2



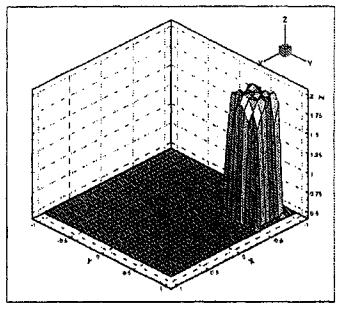
(b) true B_8^2



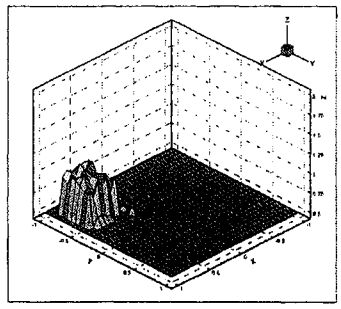
(c) true B_9^2



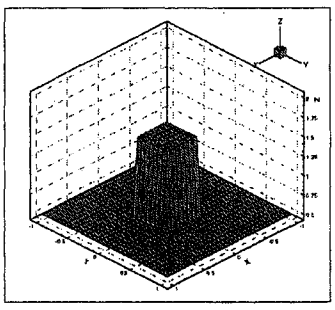
(d) recovered B_7^2



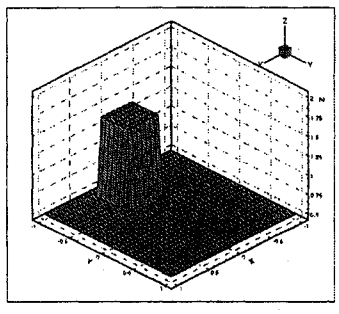
(e) recovered B_8^2



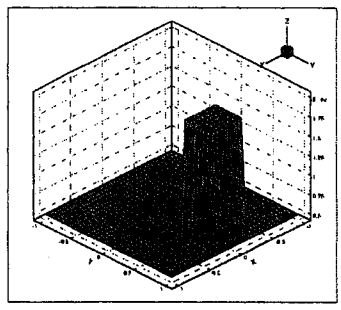
(f) recovered B_9^2



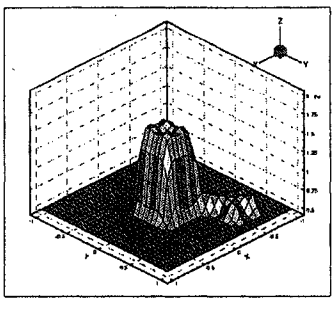
(g) true B_{10}^2



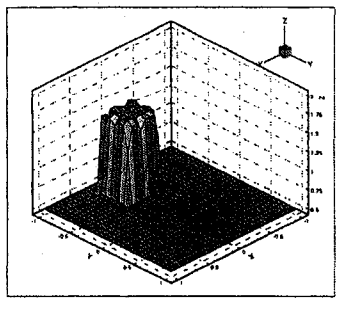
(h) true B_{11}^2



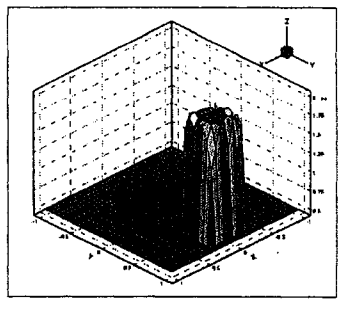
(i) true B_{12}^2



(j) recovered B_{10}^2

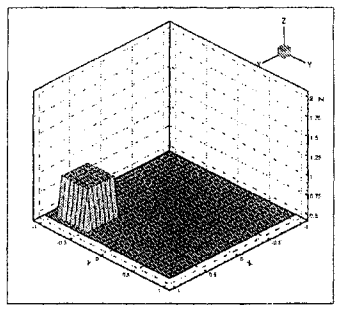


(k) recovered B_{11}^2

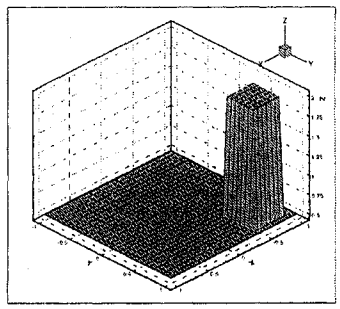


(l) recovered B_{12}^2

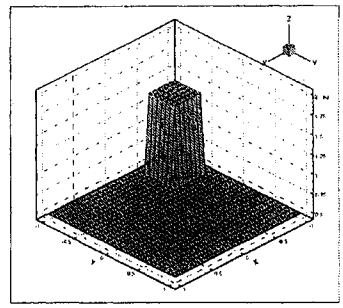
FIGURE 3.23. Recovered B_{13}^2 – B_{18}^2, assuming $\boldsymbol{D}$ and θ are known



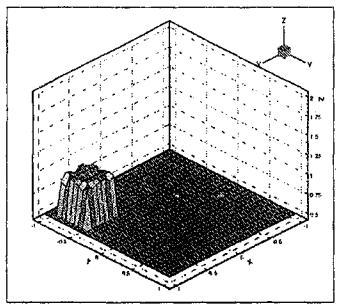
(a) true B_{13}^2



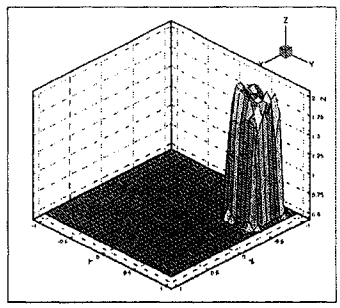
(b) true B_{14}^2



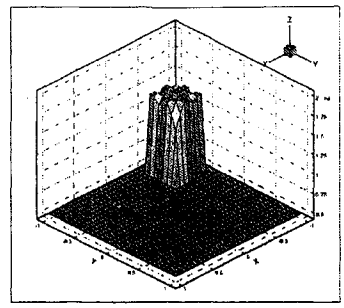
(c) true B_{15}^2



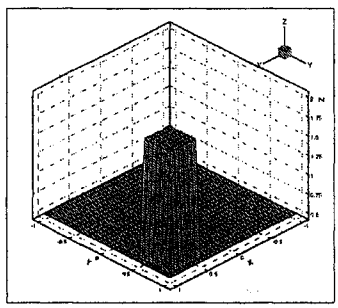
(d) recovered B_{13}^2



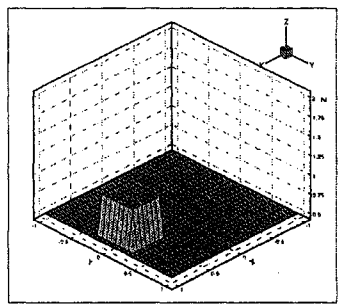
(e) recovered B_{14}^2



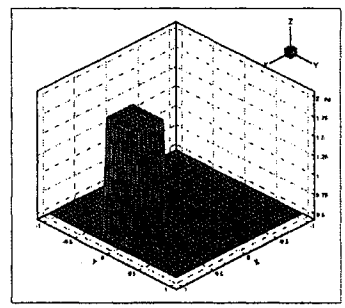
(f) recovered B_{15}^2



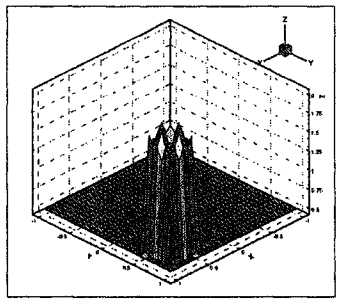
(g) true B_{16}^2



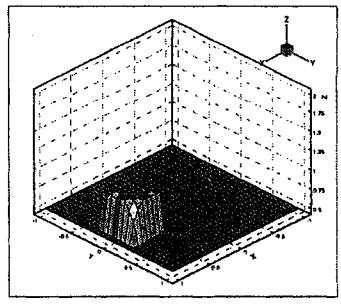
(h) true B_{17}^2



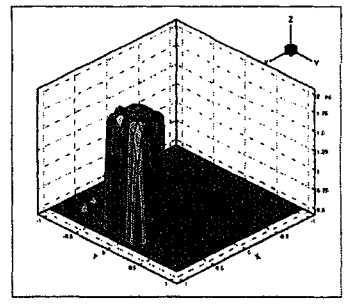
(i) true B_{18}^2



(j) recovered B_{16}^2



(k) recovered B_{17}^2



(l) recovered B_{18}^2

where

$$D_{ij} = a_{ijkm}\frac{V_k V_m}{V} f(Pe, \delta)$$

and $V = |\boldsymbol{V}| = |\boldsymbol{K}\phi\nabla\phi|$, V_k is the kth component of velocity vector $\boldsymbol{V}$, $f(Pe, \delta)$ is a function which introduces the effect of tracer transfer by molecular diffusion between adjacent streamlines, D_d is the coefficient of molecular diffusion, $T_{ij}^{\star}$ is the tortuosity. Mathematically, there are no differences for us to recover the $\boldsymbol{D}$ as above or recover the coefficients a_{ijkm} and D_d directly, but the hydrologists prefer to recover these coefficients because they can be evaluated in the lab. We note here that we can actually recover those coefficients from the source data with a slight modification of our code.

For simplicity, let us consider the isotropic case (in a confined aquifer). From Section 1.3, we know that for isotropic porous media, D_{ij} can be written in the following form (under some assumptions)

$$D_{11} = \rho_1^2 a_L + \rho_2^2 a_T, \tag{3.2}$$

$$D_{12} = \rho_1\rho_2(a_L - a_T), \tag{3.3}$$

$$D_{22} = \rho_2^2 a_L + \rho_1^2 a_T \tag{3.4}$$

for a two-dimensional case, where $\rho_i = V_i/\sqrt{V}$. If we neglect the molecular dispersion (which can be evaluated directly in the lab), the parameter $\boldsymbol{D} = (D_{ij})$ in the transport equation (2.2) becomes a function of a_L and a_T since

$$\rho_1 = \frac{V_1}{\sqrt{V}} = \frac{K_{11}\phi_x + K_{12}\phi_y}{\sqrt{[K_{11}\phi_x + K_{12}\phi_y]^2 + [K_{12}\phi_x + K_{22}\phi_y]^2}}\phi,$$
$$\rho_2 = \frac{V_2}{\sqrt{V}} = \frac{K_{12}\phi_x + K_{22}\phi_y}{\sqrt{[K_{11}\phi_x + K_{12}\phi_y]^2 + [K_{12}\phi_x + K_{22}\phi_y]^2}}\phi,$$

are known once ϕ and $\boldsymbol{K}$ are determined. Now the functional

$$G(c, \lambda) = \int_\Omega \boldsymbol{p}\nabla(u - u_c)\cdot\nabla(u - u_c) + \lambda\theta(u - u_c)^2$$
$$\int_\Omega (a_l\theta M_1 + a_t\theta M_2)\nabla(u - u_c)\cdot\nabla(u - u_c) + \lambda\theta(u - u_c)^2,$$

where $\boldsymbol{p} = a_l M_1 + a_t M_2$, and

$$M_1 = \begin{pmatrix} \rho_1^2 & \rho_1\rho_2 \\ \rho_1\rho_2 & \rho_2^2 \end{pmatrix}, \qquad M_2 = \begin{pmatrix} \rho_2^2 & -\rho_1\rho_2 \\ -\rho_1\rho_2 & \rho_1^2 \end{pmatrix},$$

can be regarded as a functional involving a_l and a_t (assuming that θ and B are known for simplicity). Similar properties in Section 2.6 about G and H can be obtained here. Thus a_L and a_T can be directly recovered.

3.3. Error analysis

From the examples in the previous section, we can see that the recovery is quite accurate, compared to the true parameters. In this section, we will analyse errors of various kinds compared to the original data. Four situations are considered here:

a) How accurate is the method? i.e., what is the error between the data obtained from the recovered parameters and the original data? (Figure 3.24)

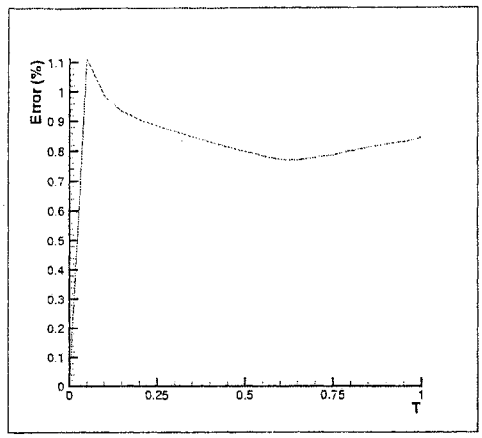
b) Is the method stable?, i.e., with a small change in parameters, does the recovered data also change by a small amount? (Figure 3.3)

c) If the original data contains error, is this method still applicable? It is truly important since the field data will inevitably contain error. (Figure 3.3)

d) Regarding sparse data situation, i.e., if the original data is not sufficient, is the method still applicable? It is also an important issue in economic and geological situations. (Figure 3.3)

EXAMPLE 3.7. *For situation a), we set the parameters as in Figure 3.2.1 and Figure 3.2.1. The solution, $\phi(t,x)$, of the parabolic equation (2.5), solved by PDETWO, is set as the source data. We compute the recovered data, $\overline{\phi}(t,x)$, by solving the parabolic equation (2.5) with the parameters recovered from Example 3.3. The L^∞ error is computed between ϕ and $\overline{\phi}$. Figure 3.24 depicts the error with the time period we defined from* 0 *to* 1. *We can see that the error is very small.*

FIGURE 3.24. Error analysis of situation 1



(a) $||\phi(t,x) - \overline{\phi}(t,x)||_{L^\infty(\Omega)}$

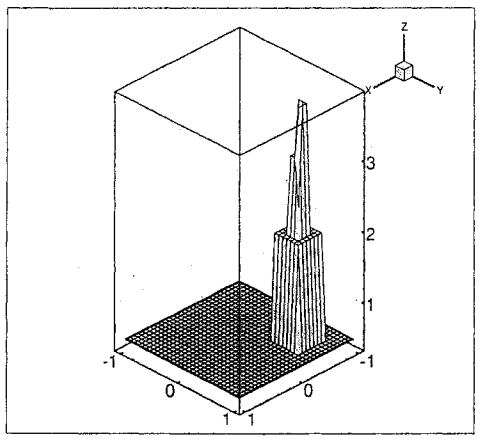
EXAMPLE 3.8. *In this example, we add* 1% *of* L^1 *error and* 100% *of* L^∞ *error to the parameter* $\boldsymbol{K}$ *(see Figure 3.3). Let* $\phi(t,x)$ *be the solution of equation (2.3) with the parameters as in Figure 3.2.1 and Figure 3.2.1, and* $\phi_1(t,x)$ *the solution of equation (2.3) with the modified parameters* $\boldsymbol{K}$ *in Figure 3.3. We use* ϕ_1 *as the source data to recover* K_{11}, K_{12}, *and* K_{22} *for* 10,000 *steps and use this recovered parameter as the coefficient to get the solution of (2.3),* $\overline{\phi}$. *Then the* L^∞ *error is computed between* ϕ *and* $\overline{\phi}$. *We can see in Figure 3.3(d) the error is still very small.*

Since we repeatedly use numerical gradient, an efficient and accurate numerical differentiation method is essential for our numerical implementation. We use the central difference method in our implementation for its efficiency. The method is accurate enough in the situation when the source data is smooth. With the source data contains error, the method fails. To overcome this difficulty, we used a mollifier to smooth the original data and regarded the smoothed data as the source data. The idea is as follows:

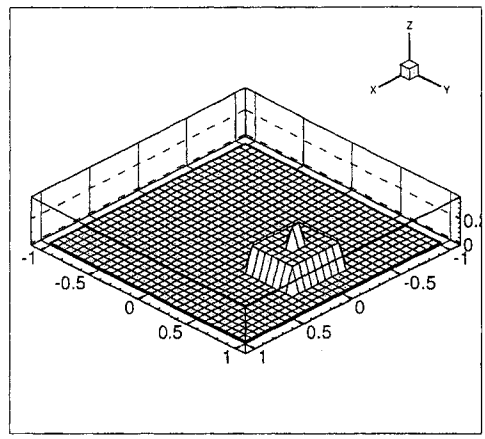
Let ρ be a C^∞ function defined on R^n such that $\rho(x) = 0$ when $||x|| > 1$ and

$$\int_{R^n} \rho(x)dx = 1. \tag{3.5}$$

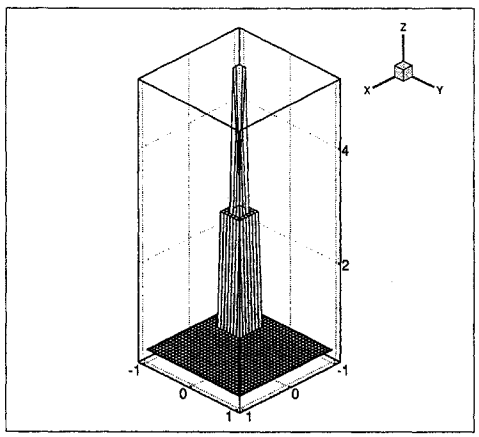
FIGURE 3.25. Error analysis of situation 2



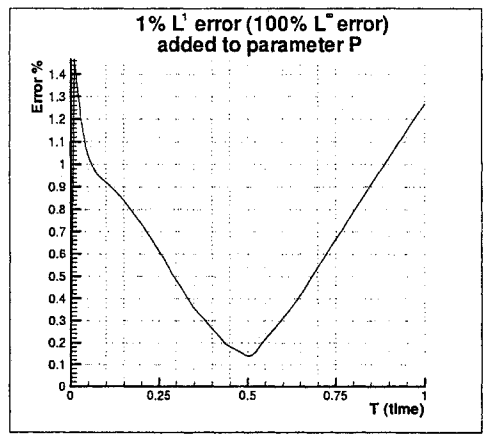
(a) true K_{11}



(b) true K_{12}



(c) true K_{22}



(d) $||\phi - \overline{\phi}||_{L^\infty(\Omega)}$

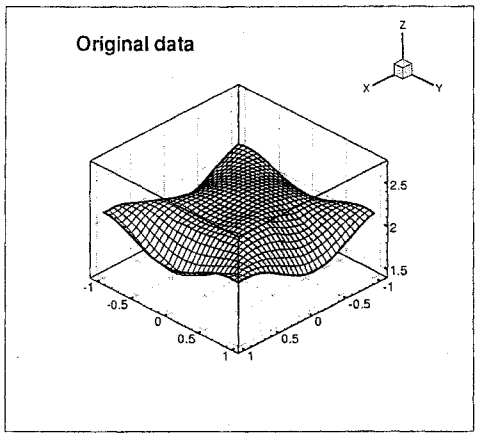
Let Ω be a bounded open subset of R^n and u be a continuous function defined on the compact set $\overline{\Omega}$. Now we add some random (L^∞) error e_0 to the function u and denote the corresponding function by $\overline{u}$. Then $\overline{u}$ can be represented as

$$\overline{u}(x) = u(x) + e(x), \tag{3.6}$$

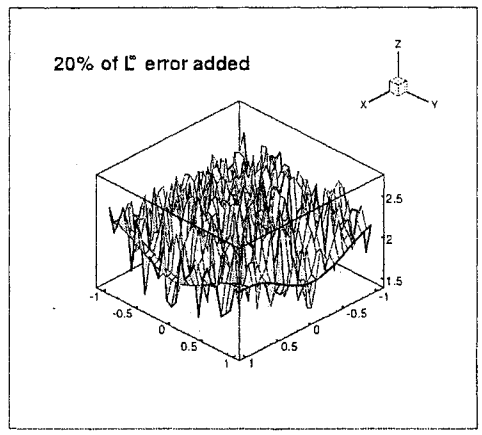
where $||e(x)||_{L^\infty} \leq e_0$. Then $\overline{u} \in \mathcal{L}^1(\Omega)$ with the assumption of $e(x)$ in $\mathcal{L}^1(\Omega)$. For $h > 0$, the regularization

$$\overline{u}_h = h^{-n} \int_\Omega \overline{u}(y)\rho(\frac{x-y}{h})dy, \tag{3.7}$$

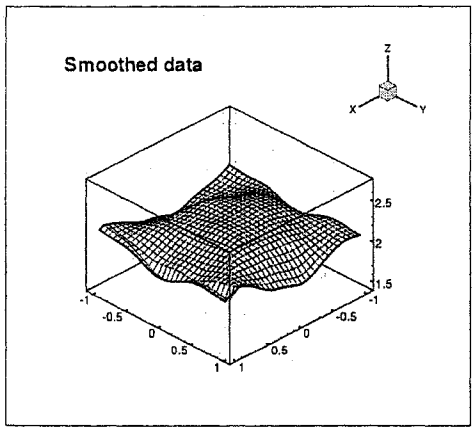
FIGURE 3.26. Error analysis of situation 3



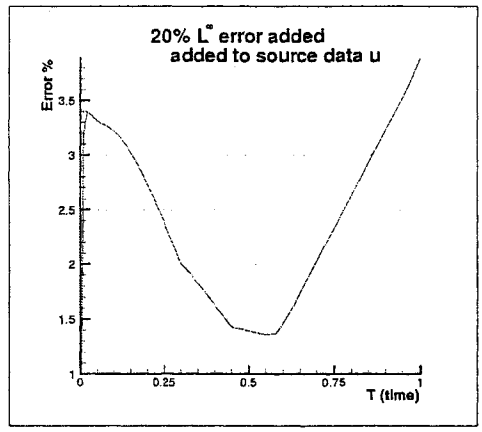
(a) $\phi(0.5, x)$



(b) $\overline{\phi}(0.5, x)$



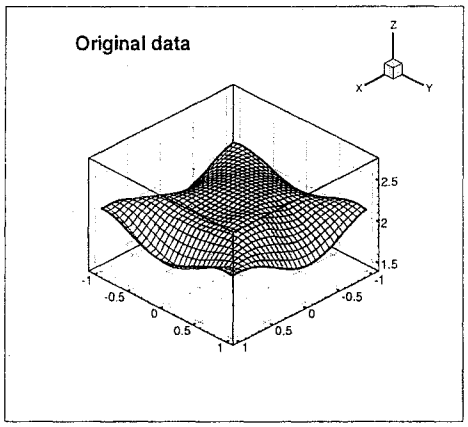
(c) $\phi_1(0.5, x)$



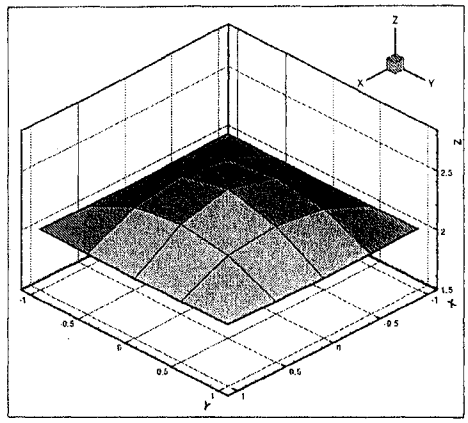
(d) $||\phi - \phi_2||_{L^\infty(\Omega)}$

of $\overline{u}$ lies in $\overline{u}_h \in \mathcal{C}^\infty(\Omega')$ for any $\Omega' \subset \Omega$ with $h < dist(\Omega', \partial\Omega)$ [38]. So the function $\overline{u}_h$ is a sufficiently smooth function, and also the error, e_h, between $\overline{u}_h$ and u, will not exceed e_0 by much. In fact, since u is continuous over $\overline{\Omega}$, for any $\epsilon > 0$, there is some $h > 0$ such that $|u(x) - u(y)| < \epsilon$ for any $x, y \in \Omega$ and $||x - y|| < h$, where $||\cdot||$

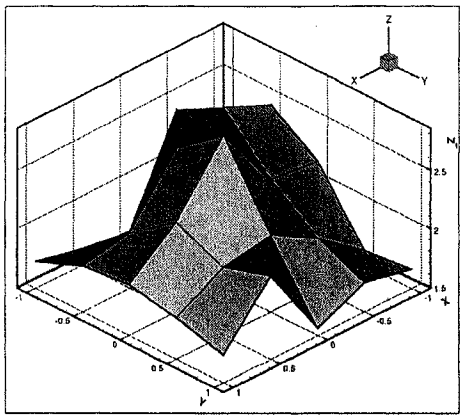
FIGURE 3.27. Error analysis of situation 4



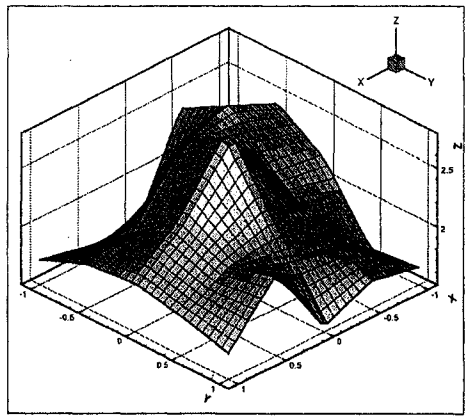
(a) $\phi(0.5, x)$



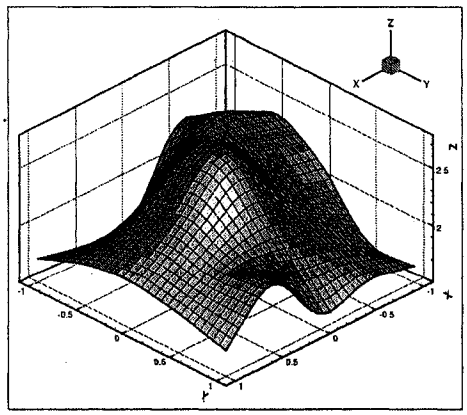
(b) $\overline{\phi}(0.5, x)$



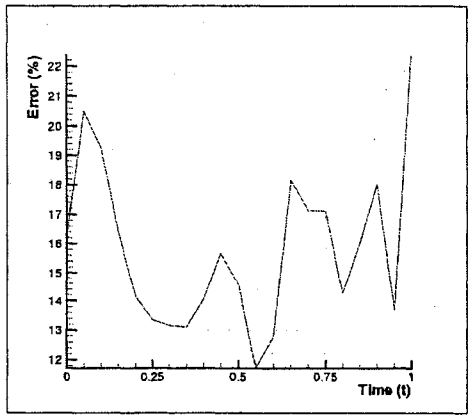
(c) $\phi_1(0.5, x)$



(d) $\phi_2(0.5, x)$



(e) $\phi_3(0.5, x)$



(f) $||\phi(t, x) - \phi_4(t, x)||_{L^\infty(\Omega)}$

denotes the Euclidean norm in R^n. Thus we have

$$\begin{aligned} |\overline{u}_h(x) - u(x)| &= h^{-n} \left| \int_\Omega (\overline{u}(y) - u(x)) \rho(\frac{x-y}{h}) dy \right| \\ &\leq h^{-n} \int_\Omega |u(y) - u(x)| \rho(\frac{x-y}{h}) dy + \\ &\qquad + h^{-n} \int_\Omega |e(y)| \rho(\frac{x-y}{h}) dy \Big| \\ &\leq \epsilon + e_0. \end{aligned}$$

So

$$e_h = \sup\{|\overline{u}_h(x) - u(x)| : x \in \Omega\} \leq \epsilon + e_0. \tag{3.8}$$

Practically, the error e_h will be much smaller than e_0, since

$$\int_\Omega e(y) \rho(\frac{x-y}{h}) dy = \int_\Omega e^+(x) \rho(\frac{x-y}{h}) dy - \int_\Omega e^-(x) \rho(\frac{x-y}{h}) \tag{3.9}$$

where e^+ and e^- denote the positive and negative part of e, and the first and second parts of the right-hand side in the previous expression tend to be equal due to the normal distribution of the errors.

Once the data has been smoothed, we can use any numerical derivative method to compute the derivatives with the smoothed data. Because the smoothed data is C^∞ at every inner point, the simple central difference method would be enough.

EXAMPLE 3.9. *In this example, we add* 20% *of* L^∞ *error to the original data* $\phi(t,x)$ *to get* $\overline{\phi}(t,x)$. *We then use the mollifier to smooth the data* $\overline{\phi}(t,x)$ *to get* $\phi_1(t,x)$. *Then we use the smoothed data* $\phi_1(t,x)$ *as the source data to recover the parameters. For simplicity, we only recovered the parameter* $\boldsymbol{K}$, *and all the other parameters are assumed to be known. After* $10,000$ *descent steps, we use the recovered parameters as the known parameters for equation (2.3). The numerical solution* $\phi_2(t,x)$ *was solved from (2.5). Then we compute the error between* $\phi_2(t,x)$ *and* ϕ, *the original data without errors. The results and those* ϕ*s at time* $t = 0.5$ *are shown in*

Figure 3.3. We can see that the error is quite small. It is much smaller than 20%, *the error we added to the original data.*

EXAMPLE 3.10. *In this example,* $\overline{\phi}(t,x)$ *is the data chosen from* 25 *grid points of the original data* $\phi(t,x)$*;* $\phi_1(t,x)$ *is the data with* 20% *of error that was added to* $\overline{\phi}(t,x)$*. We then use a bilinear interpolation to get data* $\phi_2(t,x)$ *which has values on the* 30×30 *grid, and finally we use the mollifier to smooth the data* $\phi_2(t,x)$ *to get* $\phi_3(t,x)$*. We treat* $\phi_3(t,x)$ *as the source data to recover all the parameters* $\boldsymbol{K}$*,* Q*, and* R_i*,* $i = 1, \cdots, 10$*. After a total of* $5,000$ *steps minimization searching, we treat the recovered parameters as the known parameters and solve equation (2.5) to get data* $\phi_4(t,x)$*. It can be seen that the error between* $\phi_4(t,x)$ *and the original data* $\phi(t,x)$*, shown in Figure 3.3, is still reasonable.*

CHAPTER 4

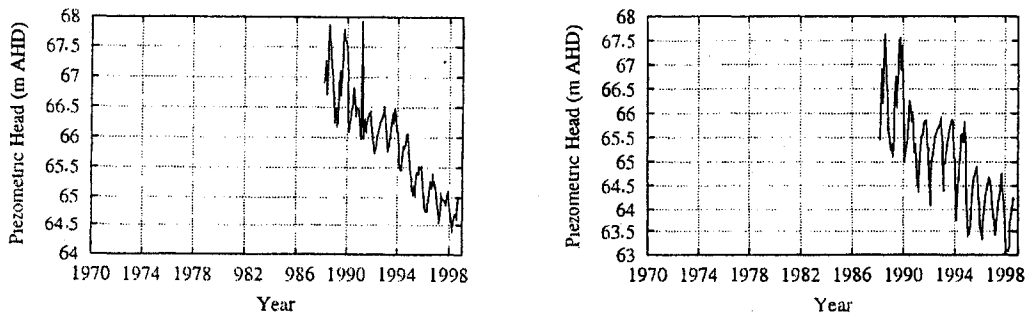
The Willunga Basin, South Australia

4.1. Introduction

The Willunga Basin is located approximately 30 km southeast of Adelaide, South Australia. This is an area of significant agricultural production. Viticulture and almond are the main industries, and groundwater is the most important resources for them. Groundwater is also used to support livestock and some light industrial enterprises.

In the last few decades, the groundwater levels within the Willunga Basin region have declined greatly due to excessive pumping. Figure 4.1 [**83**] shows the decline of the piezometric head over the 10-year period 1988–1998. As is suggested in [**66**], this decline will certainly increase the costs for extraction of groundwater. The quality of the groundwater can also be degraded due to the long-term decline, especially in the coast regions where the salt water may intrude if the groundwater level is too low.

FIGURE 4.1. Hydrographs of piezometric heads over the period 1988–1998 [**83**]



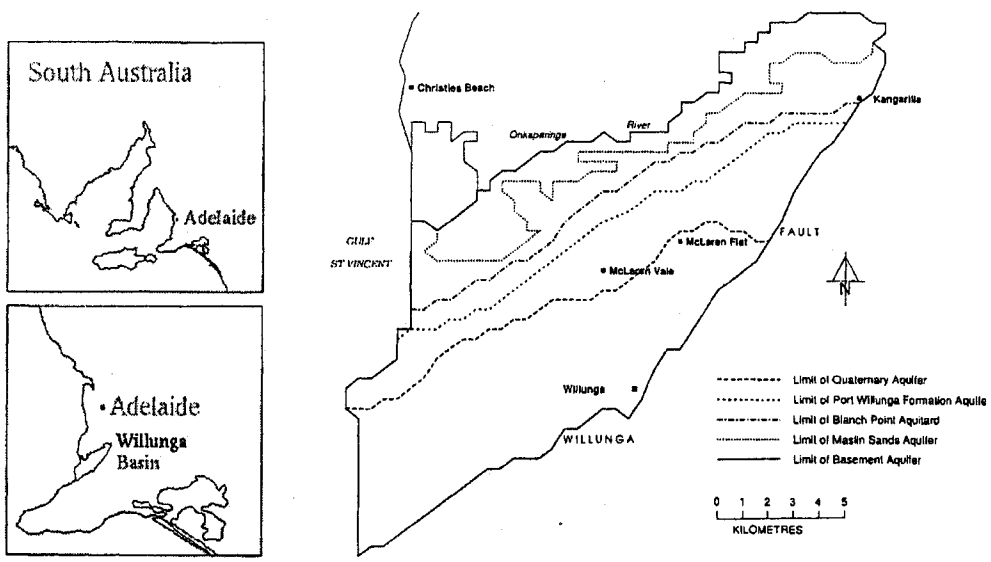
(a) Hydrograph of WLG051

(b) Hydrograph of WLG067

4.2. Hydrogeology

As summarized in [**66**], the Willunga Basin is part of the St. Vincent Basin. The Basin dips noticably to the southwest, and sits upon and is bounded to the north, east, and south by Late Precambrian and Cambrian age rocks belonging to the Adelaide Geosyncline, and consisting of interbedded slates, quartzites, and dolomites. It is wedge-shaped, with the southern and western portions the thickest, and tapers to the north. The groundwater in the Basin flows toward the coast of Gulf St. Vincent from the northeast corner. According to [**3**], the groundwater system in the Willunga Basin may be divided into four aquifer subsystems listed, from the bottom upwards, as the Basement, Maslin Sands, Port Willunga Formation, and Quaternary; see Figure 4.2.

FIGURE 4.2. Location map of the Willunga Basin, South Australia [**83**]



4.3. The Port Willunga Formation Aquifer

The most important source of groundwater within the Willunga Basin is the Port Willunga Formation, which was formed in the late Eocene to the Oligocene period, and is bounded below by marls and marly limestone of the Blanche Point Formation aquatard, and confined from above by a clay layer from the Quaternary period [**26**].

It is recharged by direct rainfall infiltration over the outcropping area north of the town of McLaren Vale to the town of McLaren Flat, and also by streams and outflow from the basement rocks. The Willunga Fault is believed to be impervious along the greater part of its length and thus acts as an obstruction to lateral inflow from the adjacent basement rocks [**66**]. The rainfall infiltration for the Port Willunga Formation is estimated to be 1050 ML/year [**66**]. The broad scale transmissivity of this aquifer is estimated from pumping tests to lie between 45 and 5560 m^2/day, while the storativity is estimated to lie between 2.7×10^{-4} and 0.011 [**83**]. The aquifer is reasonably constant in thickness, averaging around 100 metres [**26**], which makes it a viable candidate for our use here of a confined depth-average two-dimensional flow model.

4.4. Observation wells within the Port Willunga Formation Aquifer

There are about 36 observation wells within the Port Willunga Formation Aquifer [**66**] with the location of each well being shown in Figure 4.4. Piezometric head data from these wells has been collected spasmodically since December 1973.

In our test program, we chose 10 observation wells surrounding a rectangular region (shown in Figure 4.4). The reason we deliberately chose a rectangular region is because our model program is written for a rectangular region $ABCD$. For a more complete model, the finite element method is an ideal choice, since it can handle nonregular boundaries.

4.5. Groundwater levels within the Port Willunga Formation Aquifer

The flow of groundwater within the Port Willunga Formation is from the north-eastern corner to the coast. The piezometric head at the observation wells forms a piezometric surface, with the highest point at the north-eastern corner and sloping downward to the coast [**83**]. The data is from the Primary Industries and Resources, SA (PIRSA) web site. We chose a period of about one year (January 12, 1998 –

FIGURE 4.3. Observation well locations of Port Willunga Formation Aquifer [66]

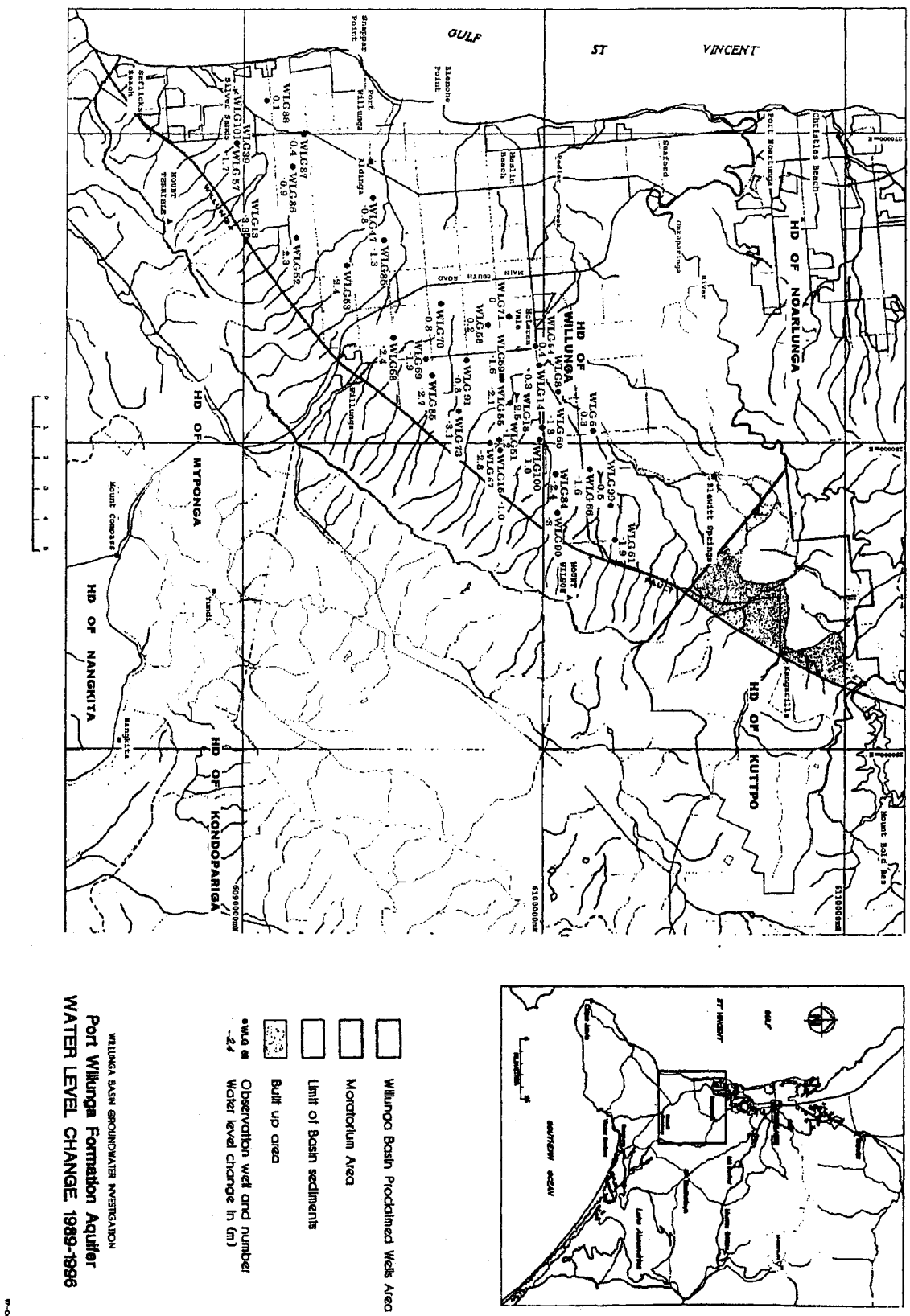
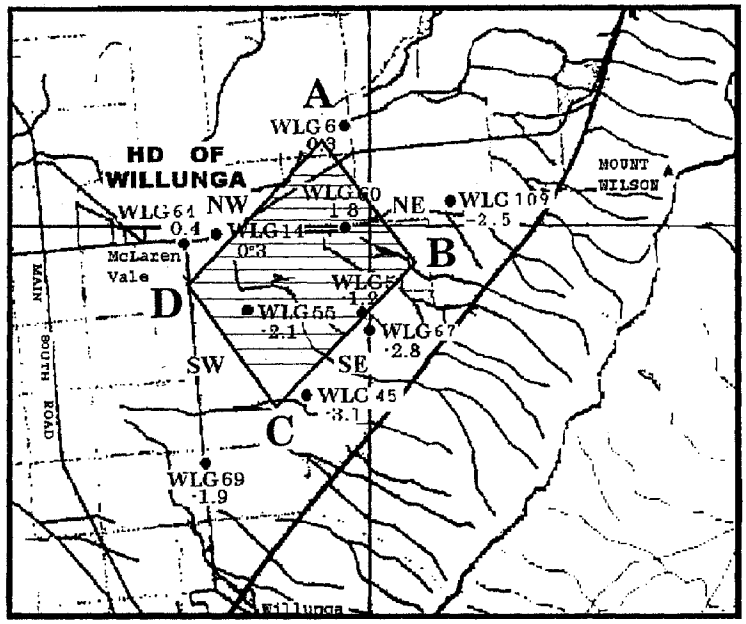


FIGURE 4.4. Test region and observation wells



January 1, 1999) for testing. The piezometric head for all 10 observation wells in this period are shown in Figure 4.7, Figure 4.7, and Figure 4.7.

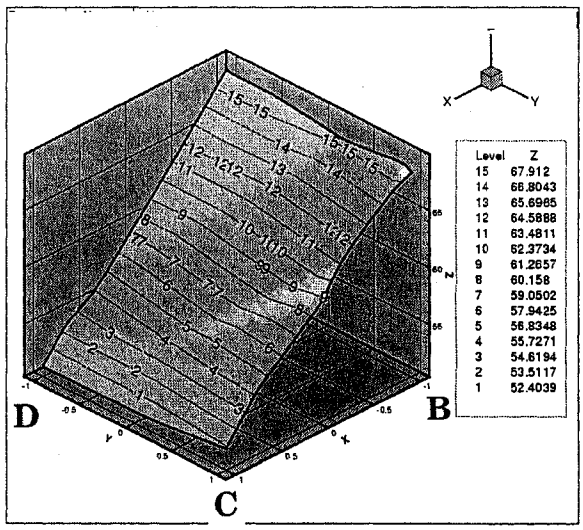
In our test program, the rectangular region $ABCD$ has AB = 1551.62 meters and BC = 2151.52 meters; these measurements were computed from latitude and longitude data obtained from the PIRSA website. This region was scaled to the square $[-1,1]^2$. This scaling is necessary. Recall that the transmissivity $\boldsymbol{T}$, 45 to 5560 (m^2/day) for the Port Willunga Formation Aquifer, for example, is much larger than the storativity, Q, 2.7×10^{-4} to 0.011, for the Port Willunga Aquifer. The recharge, measured in units of $kL/\text{day}/m^2$, is also very small. Our synthetic example shows that the recovery will be not accurate when Q is too small. If we assume $x' = 2M_1^{-1}x - 1$, $y' = 2M_2^{-1}y - 1$, where $M_1 = |BC|$, $M_2 = |AB|$, the flow equation (2.3) can be changed to

$$Q'\frac{\partial \phi}{\partial t} = \nabla^{\text{new}} \cdot (\boldsymbol{K}'\nabla^{\text{new}}\phi) + R'$$

where $\nabla^{\text{new}} = (\frac{\partial}{\partial x'}, \frac{\partial}{\partial y'})$, $Q' = M^2Q(x,y)$, $R' = M^2R(x,y,t)$, $K'_{ij} = \frac{4M^2}{M_iM_j}M_{ij}$, $i,j = 1,2$, and $M = 10^3$.

After scaling the real field to a smaller region, the scaled parameters, Q and R, in the flow equation (2.3) can be made relatively larger, making the recovery more accurate. We divided the square by a 30×30 grid and used triangular interpolation to get the data at those grid points. Figure 4.5 represents the piezometric head on January 12, 1998, at those grid points. Since there is only approximately one observation data for each well every month, in order to simulate the piezometric head as a continuous function of time variable, we used linear interpolation to simulate the daily data.

FIGURE 4.5. Piezometric head in the test region at January 12, 1998



4.6. The test program

In our test program, we divided the time period from January 12, 1998, to January 7, 1999, into 12 subintervals, and assumed that in each subintervals the source/sink term R was constant in time. This reflected the real situation since we had only one observation of the piezometric head each month.

We applied the finite Laplace transformation, using Simpson's rule, to the piezometric head to get the source data. Then we input the source data to our test program to recover the transmissivity $\boldsymbol{T}$, storativity S, and the source/sink terms R_i, $i = 1, \cdots, 12$.

In our test program, the upper and lower bounds for transmissivity, i.e., the estimated range values for transmissivity [**83**] were set to be $5,560$, 45 (m^2/day). The upper and lower bounds for storativity were set to be the estimated values 0.011 and 2.7×10^{-4} times the square of the scaling factor ($M^2 = 10^6$), as we mentioned in the previous section. For the scaled source/sink term, we set the upper and lower bounds to $10,000$ and $-10,000$ respectively. Since we had no information about the source/sink term, this bound had to be large enough. Note that our algorithm requires the knowledge of the transmissivity $\boldsymbol{T}$ at the boundary. Otherwise, it cannot guarantee the uniqueness of the recovery. Relatively accurate upper and lower bounds are therefore especially important here.

Since we could not get the boundary values for the conductivity which were essential in our algorithm, we adopted a trick by propagating the recovered interior values to the boundary in every step of the iteration search. The idea is as follows. At the beginning, we set the initial values between the preset lower and upper bounds. After every step of the iteration search, the boundary values were replaced by the recovered values at the adjacent grids. In our test program we set the lower bounds as the initial values. This is an effective method when we do not know the actual boundary values. As is known from the recovery with synthetic data, better results can be obtained for the storativity S and the source/sink term R if we know the boundary values; we adopted the same method for S and R. The number of λ-values was set to be 20.

The total iteration count for the recovery was $4,000$, which took about a week on our Beowulf cluster using 20 nodes.

4.7. The effectiveness of the recovery

To check the accuracy of the recovery, we used the recovered parameters to solve the parabolic flow equation (2.3) using the PDE solver PDETWO and compared the results with the original well data. The piezometric heads constituting the original and recovered data, as well as the relative error between the original and recovered

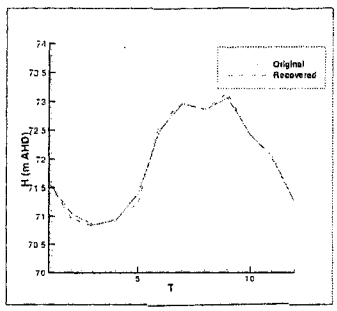
data at all the chosen observation wells, are shown in Figure 4.7, Figure 4.7, and Figure 4.7. We can see that at all the observation wells, except wells WLG069 and WLG045, the data is quite accurate (error below 0.4%) and the shapes of the data are very similar. The errors for well WLG045 and well WLG069 are relatively bigger. One possible explanation for this error is that they are far from the selected test region, especially well WLG069 (see Figure 4.4). We can see from Figure 4.5 that the Piezometric head near corner C of our test region is higher than those around corner D, and this causes the flow turn to the direction away from corner C (see Figure 4.8). The reason for this phenomenon is probably that there are some underground stream recharges near corner C. The well WLG045 is located at a position with a very high transmissivity (see Figure 4.8) compared to other regions. This is probably the other reason that the recovered data at well WLG045 is not as accurate as that at other wells.

The recovered parameters have some "spikes" which make it difficult to see the shape. In order to better represent the shape of the recovered parameters, the figures of the recovered parameters shown in the following sections have been smoothed with our mollifier.

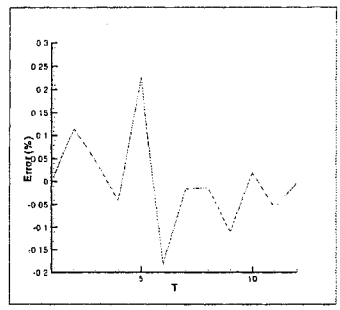
4.8. The transmissivity within the Port Willunga Formation Aquifer

In our recovery, the transmissivity was assumed to be anisotropic. Figure 4.8 shows the recovered transmissivity (the average conductivity of the aquifer can be obtained from the transmissivity by dividing by the aquifer height, approximately 100 meters). We can see that the value of T_{11} is mostly larger, except near the C corner of the region. From this we can conclude that the conductivity in the Port Willunga Formation Aquifer is anisotropic; it is more conductible in the x direction than the y direction. Figure 4.8 shows the actual Darcy flux $\boldsymbol{q} = -\boldsymbol{K}\nabla\phi$ direction.

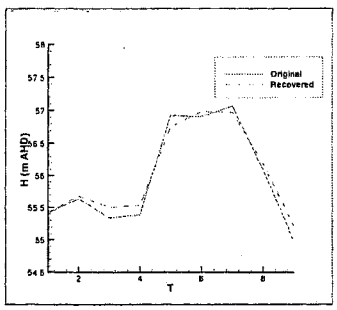
FIGURE 4.6. Accuracy of Recovery – 1



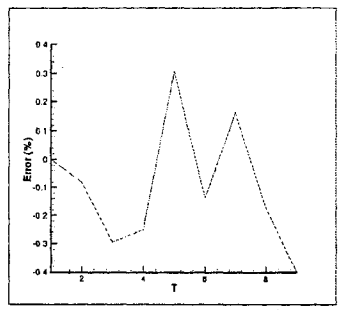
(a) WLG006



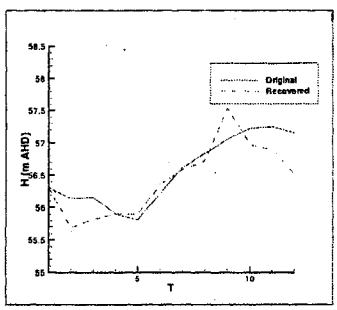
(b) Error



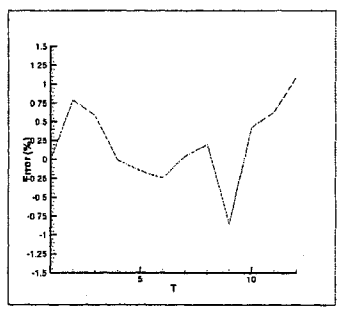
(c) WLG014



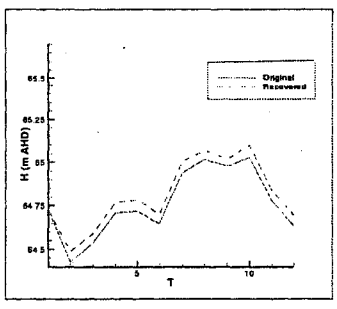
(d) Error



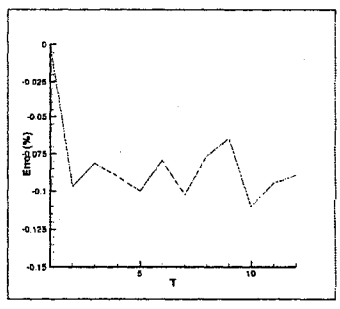
(e) WLG045



(f) Error

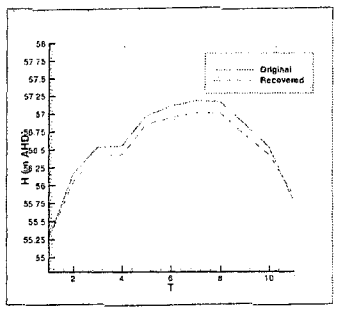


(g) WLG051

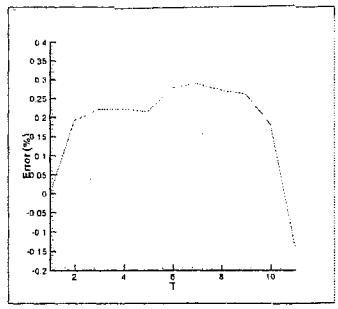


(h) Error

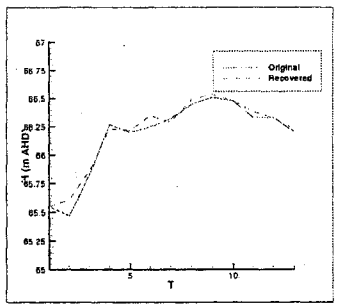
FIGURE 4.7. Accuracy of Recovery – 2



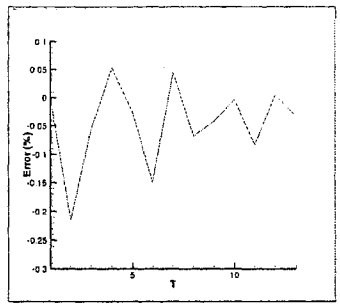
(a) WLG055



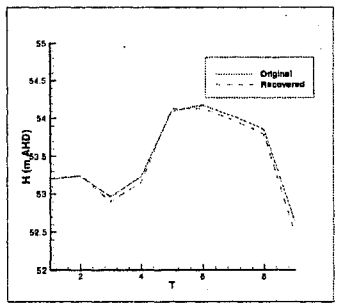
(b) Error



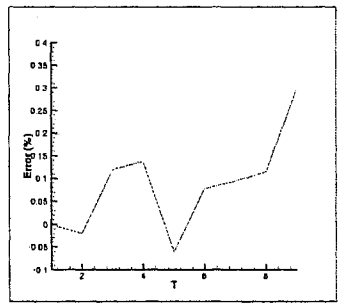
(c) WLG060



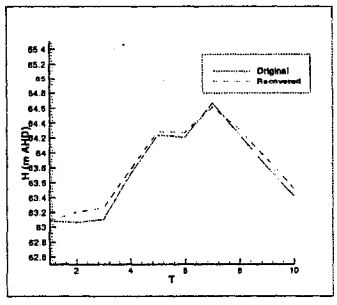
(d) Error



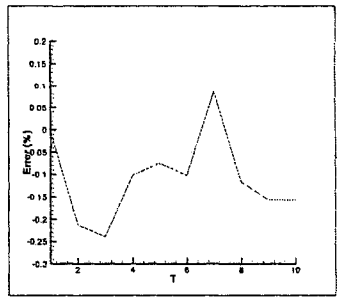
(e) WLG064



(f) Error

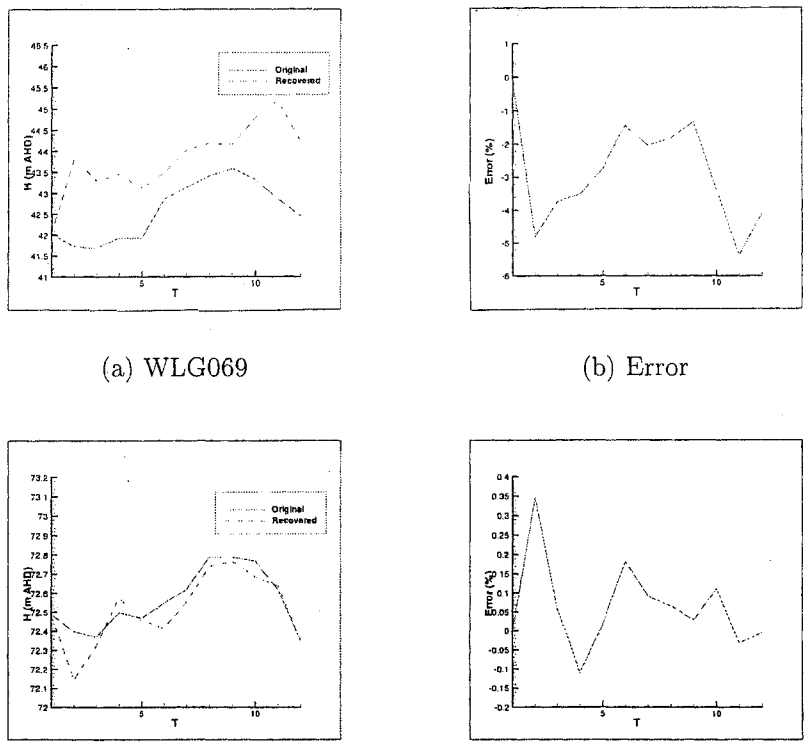


(g) WLG067



(h) Error

FIGURE 4.8. Accuracy of Recovery – 3



(a) WLG069 (b) Error

(c) WLG109 (d) Error

FIGURE 4.9. The Darcy flux in the test region at January 12, 1998

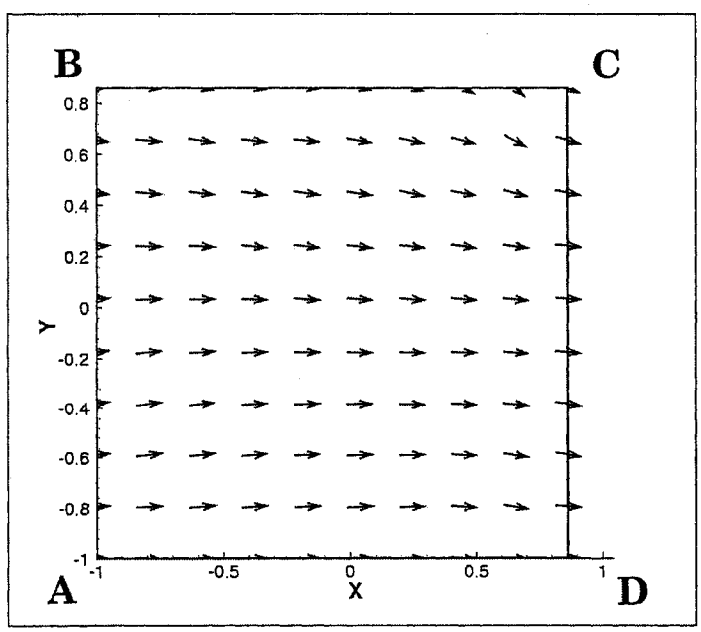
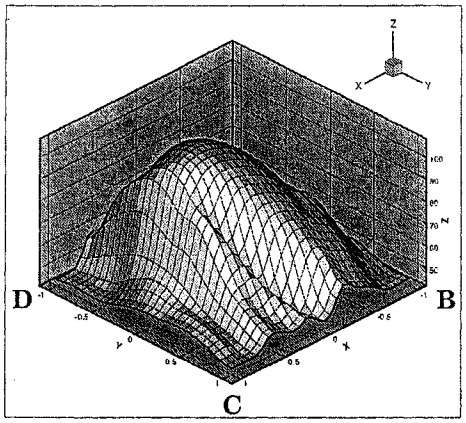
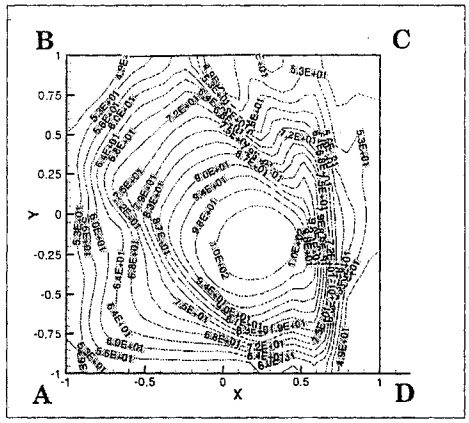


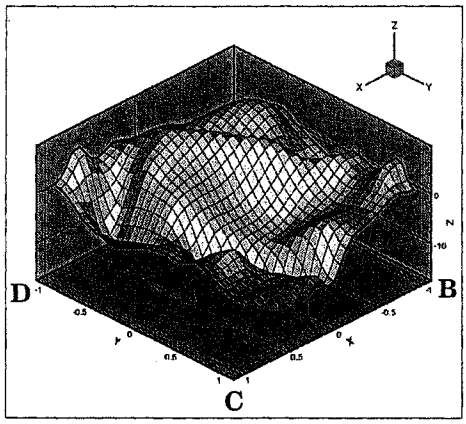
FIGURE 4.10. The recovered transmissivity T



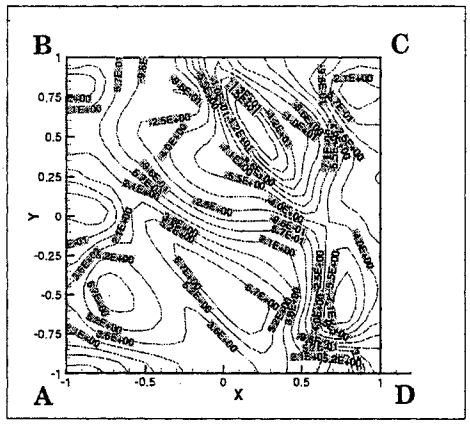
(a) T_{11}



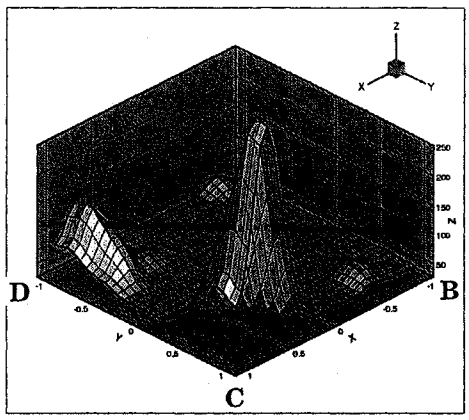
(b) Coutour plot of T_{11}



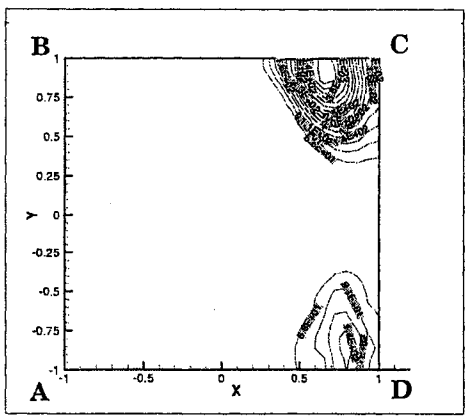
(c) T_{12}



(d) Coutour plot of T_{12}



(e) T_{22}

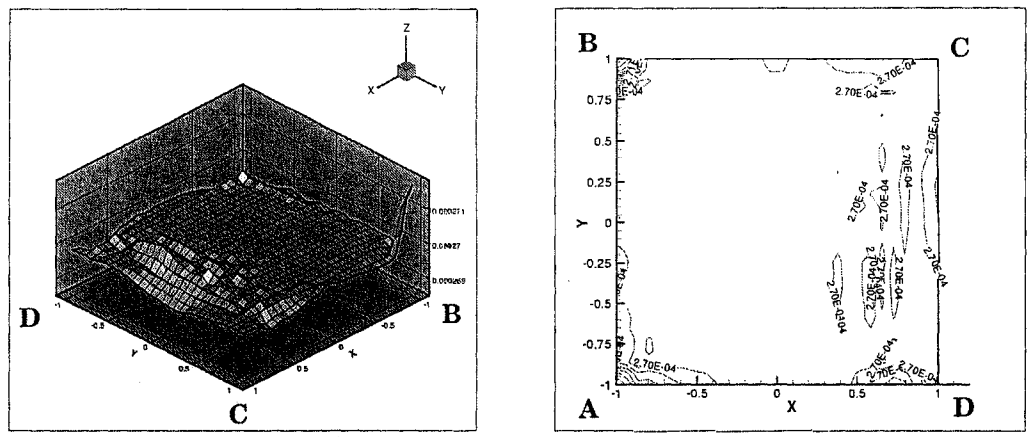


(f) Coutour plot of T_{22}

4.9. The storativity within the Port Willunga Formation Aquifer

The recovered storativity is shown in Figure 4.9. It can be seen that in most of the region, the storativity is small, around 2.7×10^{-4}.

FIGURE 4.11. The recovered storativity S



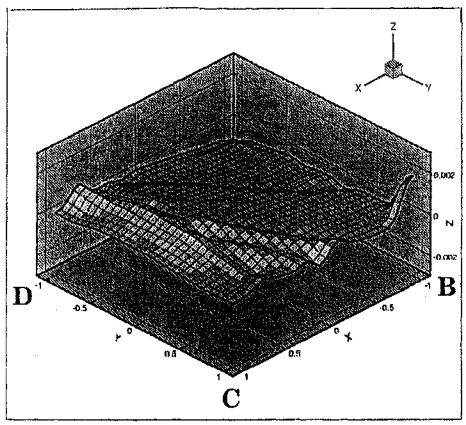
(a) S (b) Coutour plot of S

4.10. The recharge within the Port Willunga Formation Aquifer

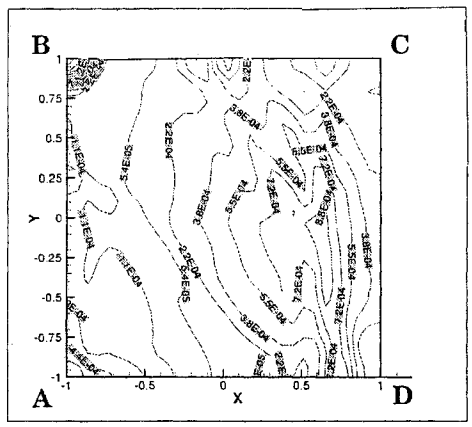
Figure 4.10, Figure 4.10, Figure 4.10, and Figure 4.10 show the recovered source/sink terms in the selected region. Positive values indicate inflow at those points, while negative values indicate outflow. We can see that the recharge, measured in units of $kL/m^2/day$, is changing gradually with respect to time. There is a big inflow in June and July and a big outflow in November and December.

From the figures we can see that near Point A, the area near McLaren Vale, which is the main rainfall recharge area of the Port Willunga Formation Aquifer, the inflow is high, especially in the winter. Near the line CD, the area closer to the coast, the outflow is high in the summer, which indicates that the artificial pumping in this region is high.

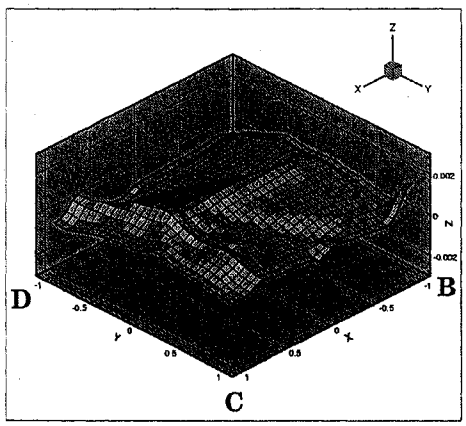
FIGURE 4.12. The recovered source term R, January – March, 1998



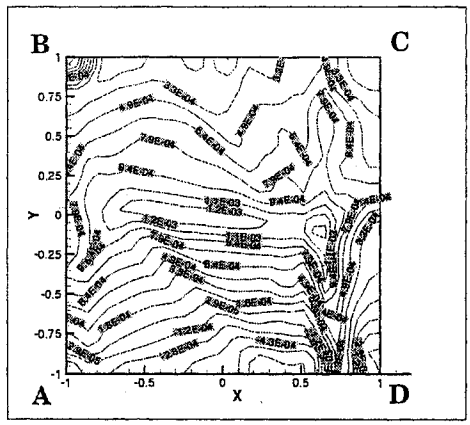
(a) January R_1



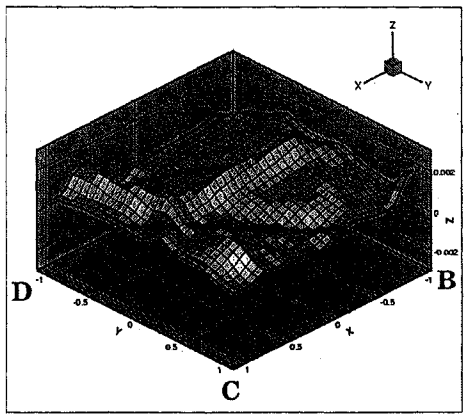
(b) Coutour plot of R_1



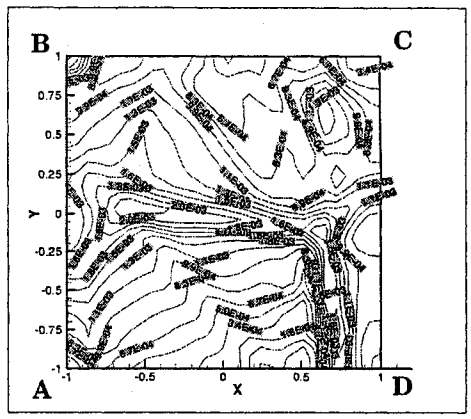
(c) February R_2



(d) Coutour plot of R_2

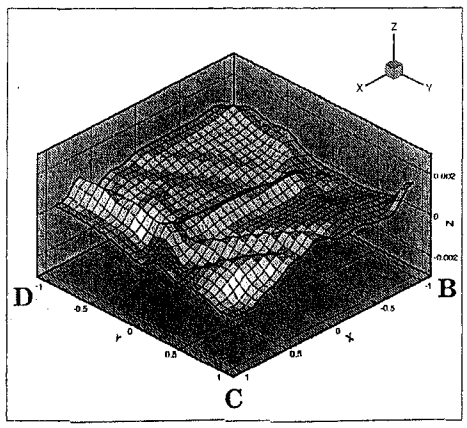


(e) March R_3

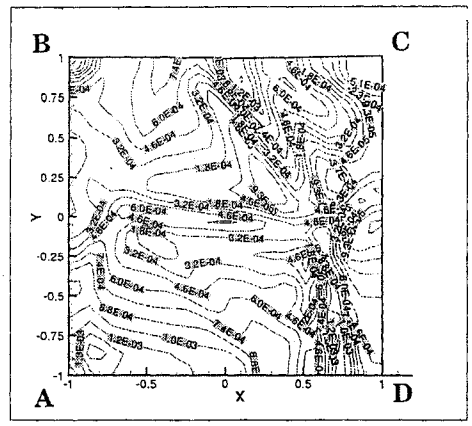


(f) Coutour plot of R_3

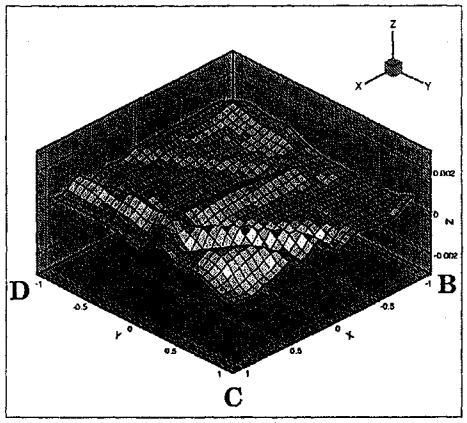
FIGURE 4.13. The recovered source term R, April – June, 1998



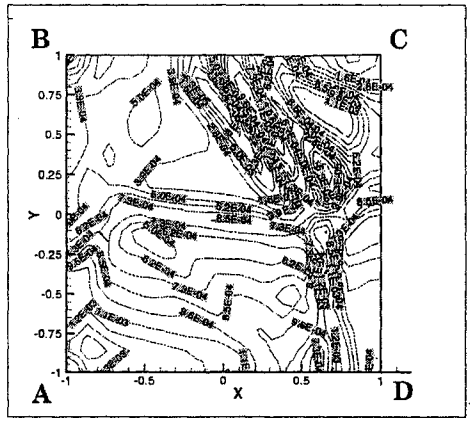
(a) April R_4



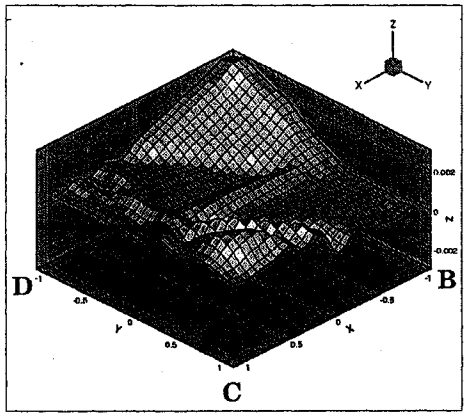
(b) Coutour plot of R_4



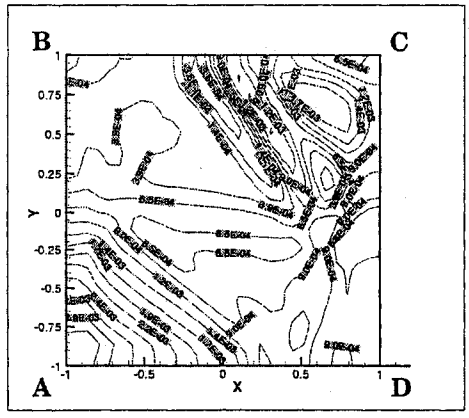
(c) May R_5



(d) Coutour plot of R_5

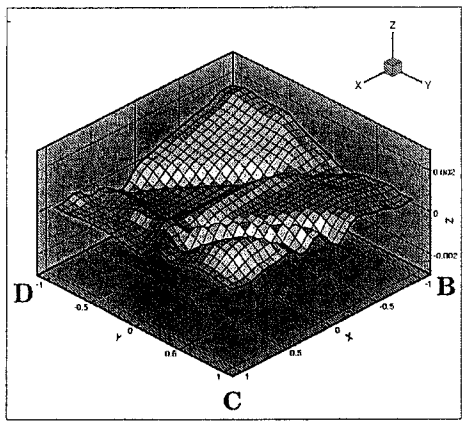


(e) June R_6

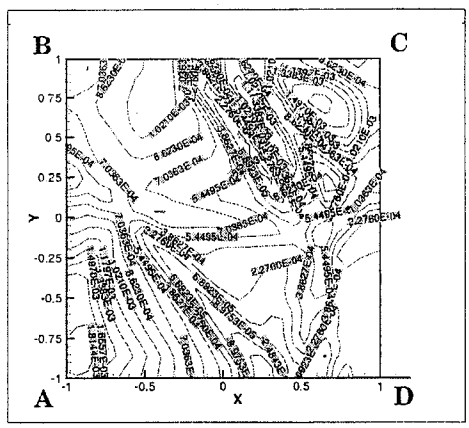


(f) Coutour plot of R_6

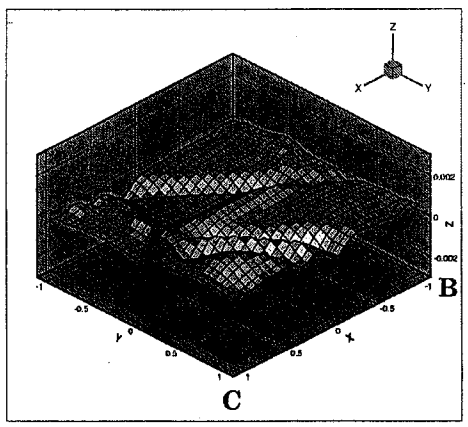
FIGURE 4.14. The recovered source term R, July - September, 1998



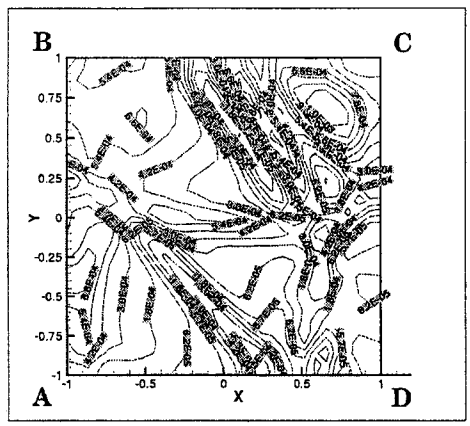
(a) July R_7



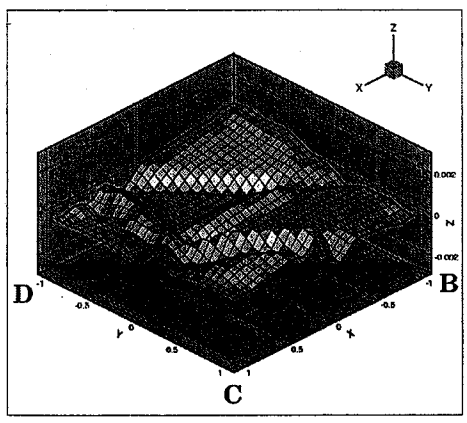
(b) Coutour plot of R_7



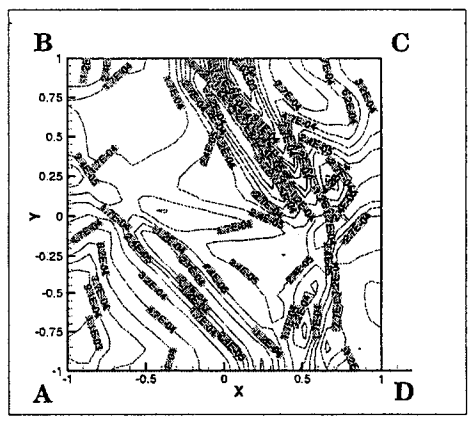
(c) August R_8



(d) Coutour plot of R_8

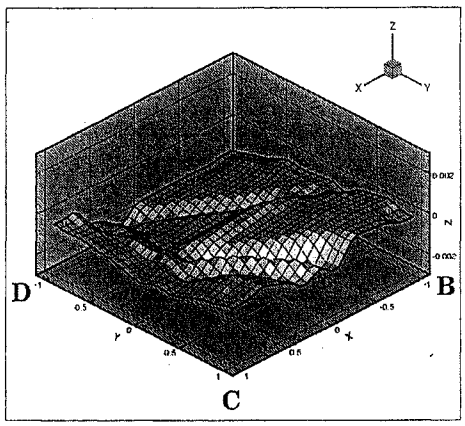


(e) September R_9

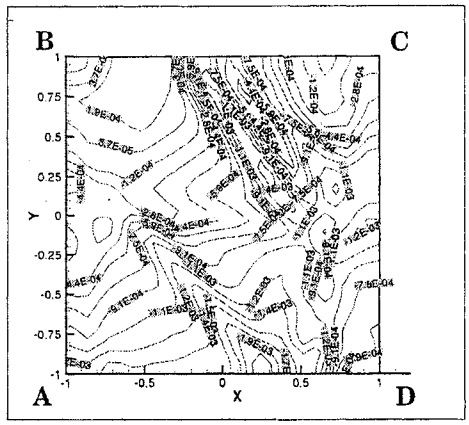


(f) Coutour plot of R_9

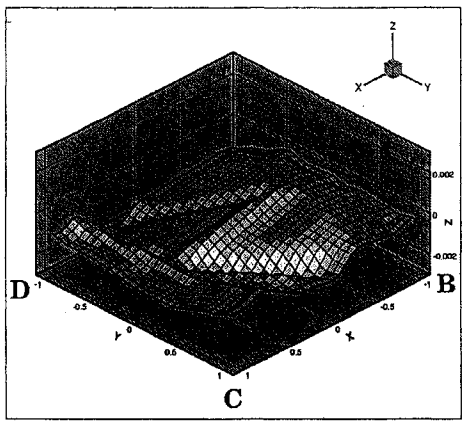
FIGURE 4.15. The recovered source term R, October – December, 1998



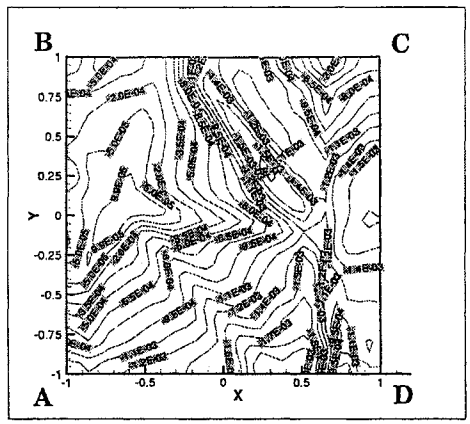
(a) October R_{10}



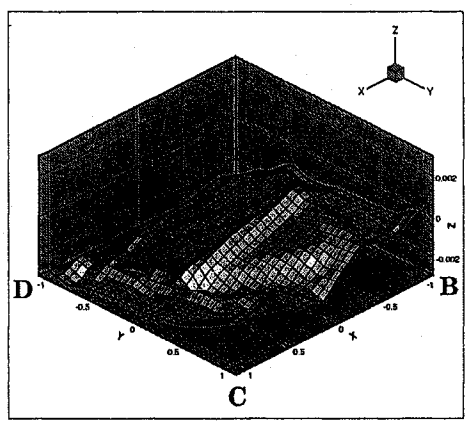
(b) Coutour plot of R_{10}



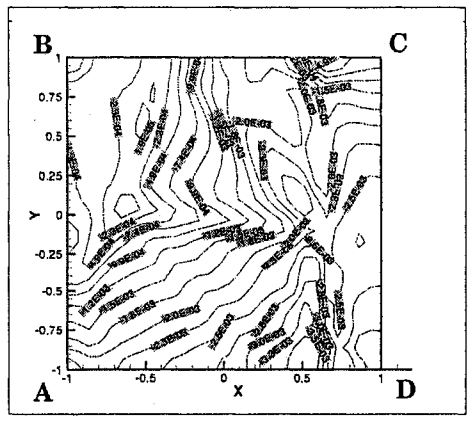
(c) November R_{11}



(d) Coutour plot of R_{11}



(e) December R_{12}



(f) Coutour plot of R_{12}

4.11. Sustainability

Gleick et al. [40] gave a useful definition of sustainable water use: "The use of water that supports the ability of human society to endure and flourish into the indefinite future without undermining the integrity of the hydrological cycle or the ecological systems that depend on it." Gleick et al. presented the following seven sustainability criteria.

1. A minimum water requirement will be guaranteed to all humans to maintain human health.
2. Sufficient water will be guaranteed to restore and maintain the health of ecosystems. Specific amounts will vary depending on climatic and other conditions. Setting these amounts will require flexible and dynamic management.
3. Water quality will be maintained to meet certain minimum standards. These standards will vary depending on location and how the water is to be used.
4. Human actions will not impair the long-term renewability of freshwater stocks and flows.
5. Data on water resources availability, use, and quality will be collected and made accessible to all parties.
6. Institutional mechanisms will be set up to prevent and resolve conflicts over water.
7. Water planning and decision-making will be democratic, ensuring representation of all affected parties and fostering direct participation of affected interests.

These criteria can provide the basis for alternative "visions" for future water management and can offer some guidance for legislative and non-govermental actions in the future [40].

In an area such as the Willunga Basin where groundwater is the main water resource, a relatively accurate record of recharge and discharge is especially important in determining the "safe yield." The main inflows include rainfall, underground streams, and the lateral inflow from the adjacent basement rocks. The outflows include evapotranspiration, the outflow to underground streams, and pumping. Since our test region is near McLaren Vale, which is the main recharge area of rainfall infiltration [**66**], rainfall is an important source to our test region. The other inflows of the Port Willunga Formation Aquifer are stream infiltration and the lateral inflow from adjacent aquifers. However, these inflows are difficult to measure. The discharge of the Port Willunga Formation Aquifer includes the outflow to the sea, the lateral flow to adjacent aquifers, and artificial well pumping. Also, heretofore it has been difficult to correctly estimate the overall discharge. Since $R(t,x) = R_i(t,x) - R_o(t,x)$, where $R_i(t,x)$ and $R_o(t,x)$ denotes the inflow and outflow respectively, it represents the difference of the inflow and outflow with respect to time t and position x. By integrating $R(t,x)$ over the region, we can get the total difference of the inflow and outflow.

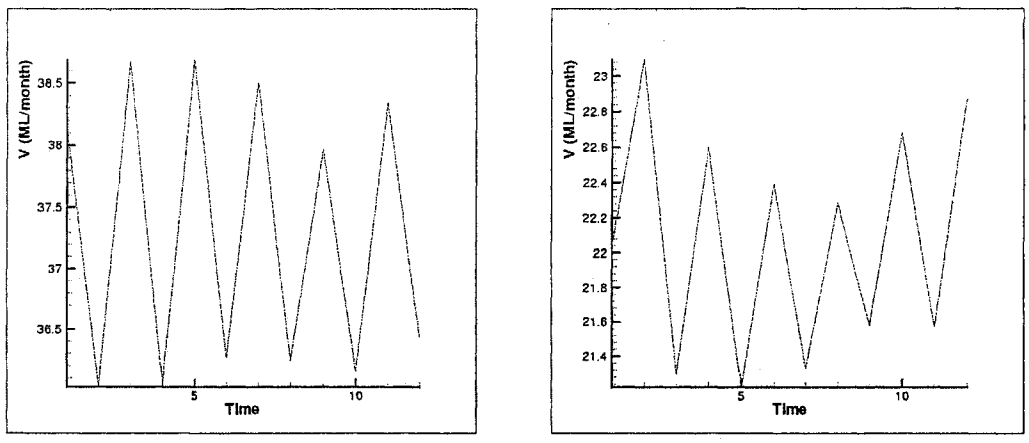
We note that $R_i(t,x)$ is the positive part of $R(t,x)$ and that $R_o(t,x)$ is the negative part of $R(t,x)$, and integrate them over the test rectangular region. We get the total inflow and outflow per month, as shown in Figure 4.11. We can see from the figure that the inflow is higher in the winter during the rainy season; the outflow is higher in the summer.

The total computed inflow and outflow are about 450 ML/year and 265 ML/year, respectively. Compared to the estimated total rainfall of 1050 ML/year [**66**] for the Port Willunga aquifer, this number is reasonable (the inflow includes underground recharges). The ratio of the inflow and outflow from January 12, 1998, to January 7, 1999, is approximately 1.70. In order to have a "safe yield," this ratio should be not less than 1; i.e., the inflow should be not less than the outflow. Since the ratio here is 1.70, we assert that the aquifer (in our test region) is sustainable. Note that

since our test region is near the main recharge area of the Port Willunga Formation Aquifer, the inflow is probably high compared to the rest of the aquifer. This result need not reflect the whole aquifer's sustainability.

To determine the sustainability of the whole basin, we need to calculate every aquifer's inflow and outflow, since there are lateral flows between the aquifers. A more complete study along these lines should provide a quantitatively effective model of this aquifer system.

FIGURE 4.16. The inflow and outflow in the test region



(a) inflow

(b) outflow

Bibliography

[1] Acar, R. Identification of the coefficient in elliptic equations. *SIAM J. Control and Optimization,* 31: 1221–1244, 1993.

[2] Ahmed, S. and de Marsilly, G. Cokriged estimation of aquifer transmissivity as an indirect solution of the inverse problem: A practical approach. *Water Resources Research,* 29(2): 521–530, 1993.

[3] Aldam, R. G. Willunga Basin hydrogeological investigations. Technical Report Bk. No. 89/22, South Australian Department of Mines and Energy, Adelaide, 1989.

[4] Alessandrini, G. An identification problem for an elliptic equation in two variables. *Ann. Mat. Pura Appl.* 145: 265–296, 1986.

[5] Anderson, M. P. and Woessner, W. W. *Applied Groundwater Modeling.* Academic Press, New York, 1992.

[6] Anderson, M. P. Aquifer heterogeneity-a geological perspective. In *Parameter Identification and Estimation for Aquifer and Reservoir Characterization.* 3-22, Columbus, Ohio, 1991. National Water Well Association.

[7] Banks, H. T. and Kunisch, K. *Estimation Techniques for Distributed Parameter Systems.* Birkhauser, New York, 1989.

[8] Bear, J. On the tensor form of dispersion. *J. Geophys. Res.* 66(4), 1185–1197, 1961.

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

[10] Bear, J. and Bachmat, Y. A generalized theory on hydrodynamic dispersion in porous media. *I.A.S.H. Symp. Artificial Recharge and Management of Aquifers, Haifa, Israel, IASH* 72, 7–16, 1967.

[11] Bear, J. and Bachmat, Y. Transport phenomena in porous media – Basic equations, in J.Bear and M.Y. Corapcioglu (eds.). *Fundamentals of Transport Phenomena in Porous Media.* Martinus Nijhoff, Dordrecht, 3–61, 1984.

[12] Bear, J. and Bachmat, Y. Macroscopic modelling of transport phenomena in porous media, 2. Applications to mass, momentum and energy transport. *Transport in Porous Media* 1, 241–269, 1986.

[13] Bear, J. and Verruijt, A. *Modeling Groundwater Flow and Pollution.* D. Reidel Publishing Company, 1987.

[14] Bear, J., Zaslavsky, D. and Irmay, S. *Physical Principles of Water Percolation and Seepage*, UNESCO, Paris, 1968.

[15] Beguelin, A., Dongarra, J., Geist, G. A., Manchek, R., and Sunderam, V. *A User's Guide to PVM: Parallel Virtual Machine.* Technical Report TM-11826, Oak Ridge National Laboratories, Oak Ridge, TN, 1991.

[16] Bolt, G. H. and Miller, R. D. Calculation of total and component potentials of water in soil. *Trans.. Am. Geophys. Union*, 39, No. 5, 917–928, 1958.

[17] Bouwer, H. *Groundwater Hydrology.* McGraw-Hill, New York, 1978.

[18] Bruckner, G., Handrock-Meyer, S., and Langmach, H. An inverse problem from 2D groundwater modelling. *Inverse Problems.* 13(4): 835–851, 1998.

[19] Carrera, J. State of the art of the inverse problem applied to the flow and solute equations. In E. Custodio, editor, *Groundwater Flow and Quality Modelling*, 549–583. D. Reidel Publ. Co., 1988.

[20] Carrera, J. and Neumann, S. Adjoint state finite element estimation of aquifer parameters under steady-state and transient conditions. *Proceedings of the 5th International Conference on Finite Elements in Water Resources.* Springer-Verlag, 1984.

[21] Carrera, J. and Neumann, S. Estimation of aquifer parameters under transient and steady state conditions: Maximum likelihood method incorporating prior information. *Water Resources Research*, 22(2) 199–210, 1986.

[22] Chavent, G. Identification of distributed parameter systems: about the output least square method, its implementation and identifiability, *Proceedings of the 5th IFAC Symposium on Identification and System Parameter Estimation*, R. Ismerman, editor, Pergamon Press, 1: 85–97, 1980.

[23] Cooley, R. L. and Naff, R. L. Regression modeling of groundwater flow. *Techniques of Water-Resources Investigations*, number 03-B4. USGS, 1990.

[24] Cooper, B. J. Eocene to Miocene stratigraphy of the Willunga Embayment. Technical Report of Investigations No. 50, South Australian Department of Mines and Energy, Adelaide, 1997.

[25] Courant, R. and Hilbert, D. *Methods of mathematical physics. vol. II: Partial Differential Equations.* Interscience Publishers, New York-London, 1962.

[26] Cresswell, D. Willunga Basin: Integrated water resource study. Technical report, Department of Environment and Natural Resources, Adelaide, 1994.

[27] Dagan, G. and Rubin, Y. Stochastic identification of recharge, transmissivity and storability in aquifer transient flow: A quasi-steady approach. *Water Resources Research*, 24(10): pp.1698, 1988.

[28] Darcy, H. *Les Fontaines Publiques de la Ville de Dijon*, Dalmont, Paris, 1856.

[29] Deimling, K. *Nonlinear Functional Analysis*. Springer-Verlag, Berlin, 1985.

[30] Desbarats, A. J. Macrodispersion in sand-shale sequences. *Water Resources Research*, 26(1): 153-164, 1990.

[31] Dietrich, C. R., Newsam, G. N., Anderssen, R. S., Ghassemi, F., and Jakeman, A. J. A practical account of instabilities in identification problems in groundwater systems. *BMR Journal of Australian Geology and Geophysics*, 11(2): 273–284, 1989.

[32] Domenico, P. A. and Schwartz, F. W. *Physical and Chemical Hydrogeology*. John Wiley & Sons, New York, 1990.

[33] Dupuit, J. *Etudes théoriques et pratiques sur le mouvement des eaux dans les canaux découverts et à travers les terrains perméables*, Dunod, Paris.

[34] Emsellem, Y. and de Marsily, G. An automatic solution for the inverse problem. *Water Resources Research*, 7(5): 1264–1283, 1971.

[35] Falk, R. S. Error estimates for the numerical identification of a variable coefficient, *Math. Comp.* 40(162): 537–546, 1983.

[36] Frind, E. O. and Pinder, G. F. Galerkin solution of the inverse problem for aquifer transmissivity, *Water Resources Research*, 9(5): 1397–1410, 1973.

[37] Galligani, I. Parameter identification using quasi-linearization. *Simulation*, 38(2): 55–60, 1982.

[38] Gilbarg, D. and Trudinger, N. S. *Elliptic Partial Differential Equations of Second Order*. Springer-Verlag, New York, 1977.

[39] Ginn, T. R., Cushman, J. H., and Houch, M. H. A continuous-time inverse operator for groundwater and contaminant transport modeling: deterministic case. *Water Resources Research*, 26: 241–252, 1990.

[40] Gleick, P. H. Human population and water: To the limits in the 21st Century." *American Association for the Advancement of Science Symposium: Human Population and Water, Fisheries, and Coastal Areas: Science and Policy Issues*. Washington, D. C.

[41] Hanke, M. A regularizing Levenberg-Marquardt scheme, with applications to inverse groundwater filtration problems. *Inverse Problems*, 13(1): 79–95, 1997.

[42] Hoeksema, R. J. and Kitandis, P. K. Comparison of Gaussian conditional mean and kriging estimation in the geostatistical solution of the inverse problem. *Water Resources Research*, 21(6): 825–836, 1985.

[43] Hoffman, K. H. and Sprekels, J. On the identification of coefficients of elliptic problems by asymptotic regularization. *Num. Funct. Anal. and Optimiz.*, 7: 157–177, 1984–85.

[44] Hughson, D. L. and Gutjahr, A. Effect of conditioning randomly heterogeneous transmissivity on temporal hydraulic head measurements in transient two-dimensional aquifer flow. *Stochastic Hydrol. Hydraul.*, 12: 155–170, 1998.

[45] Hubbert, M.K. The theory of ground-water motion. *Journal of Geology* 48, No. 8, Part 1, 785–944, 1940.

[46] Journel, A. G. and Huijbrgets, J. C. *Mining Geostatistics.* Academic Press, San Diego, California, 1978.

[47] Kärkkäinen, T. An equation error method to recover diffusion from the distributed observation. *Inverse Problems*, 13(4): 1033–1051, 1997.

[48] Keidser, A. and Rosbjerg, D. A comparison of four inverse approaches to groundwater flow and trans-parameter identification. *Water Resources Research*, 27(9): 2219–2232, 1991.

[49] Kleinecke, D. Use of linear programming for estimating geohydrologic parameters of groundwater basins. *Water Resources Research*, 7(2): 367–375, 1971.

[50] Knowlew, I. W. Parameter estimation in groundwater modelling. *Developments in Theoretical and Applied Mechanics* XXI, 415–421, 2002, Rivercross Publishing Inc., Orlando, ISBN 0-615-11944-1.

[51] Knowlew, I. W. Descent methods for inverse problems, *Nonlinear Analysis* 47: 3235–3245, 2001.

[52] Knowlew, I. W. Parameter identification for elliptic problems. *J. Comp. Appl. Math.* 131: 175–194, 2001.

[53] Knowlew, I. W. Coefficient identification in elliptic differential equations. *Direct and Inverse Problems of Mathematical Physics*, 149–160, Int. Soc. Anal. Appl. comput., 5, Kluwer Acad. Publ., Dordrecht, 2000.

[54] Knowlew, I. W. Uniqueness for an elliptic inverse problem. *SIAM J. Appl. Math.* 59(4): 1356–1370, 1999.

[55] Knowlew, I. W., Le, T. A., and Yan, A. On the recovery of multiple flow parameters from transient head data. *J. Comp. Appl. Math* (to appear)

[56] Knowlew, I. W., Teubner, M., Rasser, P., and Yan, A. Inverse Groundwater Modelling in the Willunga Basin of South Australia. Preprint.

[57] Knowlew, I. W. and Wallace, Robert. A variational method for numerical differentiation. *Numerische Mathematik,* 70, 91–110, 1995.

[58] Knowlew, I. W. and Wallace, Robert. A variational solution of the aquifer transmissivity problem. *Inverse Problems*, 12, 953–963, 1996.

[59] Knowlew, I. W. and Yan, A. The recovery of an anisotropic conductivity in groundwater modelling. *Applicable Analysis*, 81: 1347–1365, 2002.

[60] Knowlew, I. W. and Yan, A. On the recovery of transport parameters in groundwater modelling. *J. Comp. Appl. Math* (to appear)

[61] Knowlew, I. W. and Yan, A. The reconstruction of groundwater parameters from head data in an unconfined aquifer. Preprint.

[62] Kohn, R. V. and Lowe, B. D. A variational solution of the aquifer transmissivity problem. *Mathematical Modelling and Numerical Analysis*, 22(1): 119–158, 1988.

[63] Kravaris, C. and Seinfeld, J. H. Identification of parameters in distributed parameter systems by regularization. *SIAM J. Cont. Optim.*, 23: 217–241, 1985.

[64] Le, T. A. *An Inverse Problem in Groundwater Modeling.* PhD thesis, University of Alabama at Birmingham, 2000.

[65] Luce, R. and Perez, S. Parameter identification for an elliptic partial differential equation with distributed noisy data. *Inverse Problems*, 15(1): 291–307, 1999.

[66] Martin, Russel R. Willunga Basin – Status of Groundwater Resources 1998, Technical Report Book 98/28, *Department of Primary Industries and Resources SA, 1998.*

[67] Mavis, F. T. and Tsui, T. P. Percolation and capillary movement of water through sand prisms, *Bull. 18, Univ. of Iowa, Studies in Eng..*, Iowa City.

[68] Melgaard, D. and Sincovec, R. F. General software for two-dimensional non-linear partial differential equations. *ACM Transactions on Mathematical Software*, 7(1):106-125, 1981.

[69] Menke, W. *Geophysical Data Analysis: Discrete Inverse Theory.* Academic Press, New York, 1989.

[70] Morozov, V. A. *Methods for Solving Incorrectly Posed Problems.* Springer-Verlag, Berlin, 1984.

[71] Muskat, M., Wycoff, R. D., Botset, H. G., and Meres, M. W. Flow of gas liquid mixtures through sands, *Trans. A.I.M.E. Petrol.* 123, 69–96, 1937.

[72] Nelson, W. R. In-place determination of permeability distribution for heterogeneous porous media through analysis of energy dissipation. *Soc. Pet. Eng. J.*, 8(1): 32–42, 1968.

[73] Neuberger, J. W. *Sobolev Gradients in Differential Equations*, volume 1670 of *Lecture Notes in Mathematics.* Springer-Verlag, New York, 1997.

[74] Neuman, S. P. and Yakowitz, S. A statistical approach to the inverse problem of aquifer hydrology. *Water Resources Research*, 15(4): 845–860, 1979.

[75] Neumann, J. W. Perspective on "delayed yield". *Water Resources Research*, 15: 899–908, 1979.

[76] Nikolaevskii, V. N. Convective diffusion in porous media. *J. Appl. Math. Mech.* 23(6), 1042–1050, 1959.

[77] Payne, L. E. *Improperly Posed Problems in Partial Differential Equations.* SIAM, Philadelphia, 1975.

[78] Peck, A., Gorelic, S. M., de Marsily, G., Foster, S., and Kovalevsky, V. *Consequences of Spatial Variability in Aquifer Properties and Data Limitations for Groundwater Modeling Practice.* Number 175. International Association of Hydrologists, 1988.

[79] Perko, L. *Differential Equations and Dynamical Systems.* Springer-Verlag, New York, 1988.

[80] Piersol, R. J., Workman, L. E., and Watson, M. C. Porosity, total liquid saturation and permeability of Illinois oil sands III, *Geol. Survey*, Report No. 67, 1940.

[81] Ya, Polubarinova-Kochina, P. *Theory of Groundwater Movement*(in Russian), Gostekhizdat, Moscow. English trans. by Roger J. M. de Wiest, Princeton Univ. Press, Princeton, N.J., 1962.

[82] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes in Fortran 90, volume 2 of Fortran Numerical Recipes.* Cambridge University Press, Cambridge, second edition, 1996. The Art of Parallel Scientific Computing, With a foreword by Michael Metcalf; with separately available software.

[83] Rasser, Paul Edward. *Calibration of Numerical Models with Application to Groundwater Flow in the Willunga Basin, South Australia.* Thesis, June 1, 2001.

[84] Remson, I., Hornberger, G., and Molz, F. *Numerical Methods In Subsurface Hydrology.* John Wiley & Sons, Inc., 1971.

[85] Rice, J. H. and Boisvert, R. F. *Solving Elliptic Problems Using ELLPACK.* Springer-Verlag, Berlin, 1985.

[86] Richter, G. R. An inverse problem for the steady state diffusion equation. *SIAM J. Appl. Math.*, 41(2): 210–221, 1981.

[87] Richter, G. R. Numerical identification of a spacially varying diffusion coefficient. *Math. Comp.*, 36(154): 375–386, 1981.

[88] Rivière, B. and Jenkins, L. In pursuit of better models and simulations: Oil Industry looks to the math sciences. *SIAM News*, 35(1), 2002.

[89] Rizzo, D. M. and Dougherty, D. E. Characterization of aquifer properties using artificial neural networks: Neural kriging. *Water Resources Research*, 30(2): 483–497, 1994.

[90] Rose, H. E. An investigation into the laws of flow of fluids through beds of granular material. *Proc. Inst. Mech. Engrs.*, 153: 141–148, 1945.

[91] Sagar, B., Yakowitz, S., and Duckstein, L. A direct method for the identification of the parametrs of dynamics nonhomogeneous aquifers. *Water Resources Research*, 11(4): 563–570, 1975.

[92] Scheidegger, A. E. General theory of dispersion in porous media. *J. Geophys. Res.* 66, 3273–3278, 1961.

[93] Simmers, J. *Estimation of Natural Groundwater Recharge*, volume 222 of *NATO ASI Series C.* D. Reidel Publishing Co., 1988.

[94] Silin-Bekchurin. *Dynamics of Ground Water* (in Russian), Moscow Izdat., Moscow, 1958.

[95] Slichter, J. C. Field measurement of the rate of movement of underground waters, *USGS Water Supply Paper*, 140.

[96] Sun, N. and Yeh, W. A stochastic inverse solution for transient groundwater flow: Parameter identification and reliability analysis. *Water Resources Research*, 28(12): 3269, 1992.

[97] Surana, K. S. and Huels, C. R. A least squares finite element solution on the inverse problem of aquifer transmissivity. *Computers and Structures*, 31(2): 249, 1989.

[98] Tautenhahn, U. A new regularization method for parameter identification in elliptic problems. *Inverse Problems*, 6: 465–477, 1990.

[99] Thomas, H. E. Ground water regions of the United States–Their storage facilities, Vol. 3, Inter- and Insular Affairs Comm., House of Representatives, 5 U.S. Congress, Washington, D.C. 1952.

[100] Tikhonov, A. N. and Arsenin, V. Y. *Solutions of Ill-Posed Problems.* V. H. Winston & Sons, Washington D. C., 1977.

[101] Vázquez, C. R., Guidici, M., Parravicini, G., and Ponzini., G. The differential system method for the identification of transmissivity and storativity. *Transport in Porous Media*, 26: 339–371, 1997.

[102] Yakowitz, S. and Duckstein, L. Instability in aquifer identificatiion: theory and case studies. *Water Resources Research*, 16(6): 1045–1064, 1980.

[103] Yeh, W. W-G. Review of parameter identification procedures in groundwater hydrology: The inverse problem. *Water Resources Research*, 22(2): 95–108, 1986.

APPENDIX A

Fortran codes to recover the parameters

A.1. The master program

```
!#####################################################################
!# This program recovers the coefficients of the flow equation of  #
!# confined aquifer:                                                #
!#        Q(x) Du/Dt = -\nabla( P \nabla u ) + R(x,t)        (1)    #
!# where P is a symmetric, strictly positive definite 2X2 matrix, Q#
!# is a non-negative function, while R can be positive or negative #
!#                                                                  #
!# This program actually recover the parameters of the elliptic     #
!# equation:                                                        #
!#    -\nabla( P(x) \nabla u)                                       #
#         + (\lambda u + \alpha) Q(x) = \beta R(x)                  #
!# i.e., finite Laplace transformed equation of (1)                 #
!#                                                                  #
!# This is the server part program. It mainly doing the             #
!# minimization search. It must be run togather with the slave      #
!# program fl_slave.                                                #
!#                                                                  #
!# To run the program, the pvm enviroment should be set up          #
!# correctly.                                                       #
!#                                                                  #
!# When you start the program: fl_master, it will automatically     #
!# start the slave program: fl_slave.                               #
!#                                                                  #
!# Please refer the file: path.f90 for information about the name   #
!# of the source file and other files about the data                #
!#####################################################################
Program fl_master
  USE ntype
  USE para, ONLY : MGRID,NGRID,a,b,ha,hb,iterp=>fl_iterp,smstep, &
                   NUM=>FL_NUM,NSUBIN=>FL_NSUBIN,pc0=>FL_pc0, &
                   nplot=>FL_nplot,newSearch=>FL_newSearch,&
                   loops=>FL_loops
  USE path, ONLY : home=>fl_home,datapath=>fl_datapath,slave=>fl_slave
  USE para_func, ONLY : trueP,trueQ,trueF
  USE util, ONLY : Plotting
  USE PVM, ONLY : spawnTask, stopProgram,getdata,Grad,send_update
  IMPLICIT NONE

!C:*****************
!C: LOCAL VARIABLES
!C:*****************
  REAL(PREC),DIMENSION(NSUBIN,NUM+2) :: steps
  INTEGER :: i, ll, j, k,iter_step,ip
  LOGICAL :: success,fail,cut
```

```
  INTEGER :: nvars,ntstep,variable,var1,var2
  INTEGER, DIMENSION(100) :: vars
  REAL(PREC), DIMENSION(2) :: searchVal
  REAL(PREC) :: begVal, endVal,stepsize
  CHARACTER(LEN=5),DIMENSION(NUM+2) :: searchVar, successVar,failVar
  INTEGER :: nsuccess, nfail
  CHARACTER(LEN=1),DIMENSION(3) :: name=(/'P','Q','F'/)
  REAL(PREC), DIMENSION(NUM+1+NSUBIN,MGRID,NGRID) :: pqf_c

  INTERFACE
    SUBROUTINE descent_search(val,cut,step)
      USE ntype
      REAL(PREC),DIMENSION(:) :: val
      REAL(PREC) :: step
      LOGICAL :: cut
    END SUBROUTINE descent_search
  END INTERFACE

!c:*********************
!c: BEGIN PROGRAM
!c:
!c: INITIALIZATION
!c:*********************

    CALL spawnTask(slave)

    IF (nplot) THEN
      DO i=1, MGRID
      DO j=1, NGRID
        DO ll=1, NUM
          pqf_c(ll,i,j)=truep(ll,i,j)
        END DO
        pqf_c(NUM+1,i,j) = trueq(i,j)
        DO ll=1, NSUBIN
          pqf_c(NUM+1+ll,i,j) = truef(ll,i,j)
        END DO
      END DO
      END DO
      CALL PLOTTING(home,name,NUM,pqf_c, 'TRUE',a,b,ha,hb)
    END IF

!c:*******************************
!c: GET VARIABLES NEED TO SEARCH
!c:*******************************
    steps=1.0_prec
    nvars=0
```

```
      vars=0
      DO k=1, NUM+2
        IF (loops(k) /=0 ) THEN
          vars(nvars+1:nvars+loops(k))=k
          nvars=nvars+loops(k)
        END IF
      END DO
      nvars=SUM(LOOPS)

      IF (NUM==1) THEN
        searchVar(1)='P    '
        searchVar(2)='Q    '
        searchVar(3)='F    '
      ELSE
        searchVar(1)='P11  '
        searchVar(2)='P12  '
        searchVar(3)='P22  '
        searchVar(4)='Q    '
        searchVar(5)='F    '
      END IF

!c:*****************************
!c: INITIALIZE iter_step
!c:*****************************

      IF (newSearch) THEN
        iter_step=0
      ELSE
        OPEN(UNIT=4,FILE=datapath,STATUS='OLD', &
             ACCESS='SEQUENTIAL',ACTION='READ')
          READ(4,*) iter_step
        CLOSE(4,STATUS='KEEP')
      END IF


!c:ccccccccccccccccccccccc
!c:  MINIMIZE G
!c:
!c:  BEGIN DESCENT LOOP
!c:ccccccccccccccccccccccc

!C:******************
!C: BEGIN SEARCH LOOP
!C:******************
      DO ip=1, iterp
        iter_step=iter_step+1
```

```
      print *,''
      print *,' ***************************** '
      print *,' *  loop step = ', iter_step, ' * '
      print *,' ***************************** '
      fail=.TRUE.
      nsuccess=0
      nfail=0

      DO i=1, NSUBIN
        print *, ' time interval: ', i
      DO j=1, SUM(LOOPS)
        var1=i
        var2=vars(MOD(j+iter_step,nvars)+1)
        variable=var1*1000+var2
        print *, ' search variable : ', searchVar(var2)

!C:**********************
!C: COMPUTE THE GRADIENTS
!C:**********************
        CALL Grad(variable)

!C:**************
!C: DESCENT SEARCH
!C:**************
        stepsize=steps(var1,var2)
        CALL descent_search(searchVal,cut,stepsize)
        call send_update(stepsize)
        IF (stepsize > smstep)  steps(var1,var2)=stepsize*0.75_prec

        IF (i==1) begVal=searchVal(1)
        success=(searchVal(1)>searchVal(2))
        fail= fail .AND. (searchVal(1)<=searchVal(2))

        print *, ' cut = ', cut
        print *, ' stepsize = ', stepsize
        print *, ' At beginnig G  = ', searchVal(1)
        print *, ' At end      G  = ', searchVal(2)
        print *, ' **********************'

        IF (success) THEN
          endVal=searchVal(2)
          nsuccess=nsuccess+1
          successVar(nsuccess)=searchVar(var2)
        ELSE
          nfail=nfail+1
          failVar(nfail)=searchVar(var2)
```

```
        END IF
      END DO
      END DO
!C:************************
!C: print out search result
!C:************************
      print *, '*********************************'
      print *, ' In This Loop: '
      IF (nsuccess>0) THEN
          print *, ' success variables :'
          print *, successVar(1:nsuccess)
      END IF
      IF (nfail>0) THEN
          print *, '&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&'
          print *, ' failed  variables :'
          print *, failVar(1:nfail)
      END IF
      print *, '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
      print *, ' Beginning : G = ', begVal
      print *, ' End       : G = ', endVal
      print *, '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
      print *, '*********************************'

!C:*****************
!C: IF SEARCH FAILED, EXIT SEARCH
!C:*****************
      fail=.FALSE.
      IF (fail) THEN
        PRINT *, ' TOTAL ITERATION STEPS: ', iter_step
        CALL StopProgram()
      END IF
    END DO
END PROGRAM fl_master

FUNCTION getG(cut;dir)
  USE ntype
  USE pvm, ONLY : send_gstep, getData
  IMPLICIT NONE
  REAL(PREC), INTENT(IN) :: dir
  LOGICAL, INTENT(IN) :: cut
  REAL(PREC) :: getG

!C:*************************
!C: SEND OUT SEARCH STEPSIZE
!C:*************************
    CALL send_gstep(cut,dir)
```

```
!C:*************************
!C: GET G VALUE
!C:*************************
    CALL getData(getG)
END FUNCTION getG

!C***********************************************
```

A.2. The slave program

```
!######################################################################
!# This program recovers the coefficients of the flow equation of     #
!# confined aquifer:                                                  #
!#        Q(x) Dw/Dt = -\nabla( P \nabla u ) + R(x,t)                 #
!# where P is a symmetric, strictly positive definite 2X2 matrix      #
!#                                                                    #
!# This is the slave part program. It mainly doing the minimization   #
!# search. It must be run togather with the server program fl_server.  #
!#                                                                    #
!# Please refer the master program: fl_master.f90 for requirement     #
!# to run this program.                                               #
!#                                                                    #
!######################################################################


MODULE fl_slv_mod
  USE ntype
  USE para, ONLY : MGRID,NGRID,NUM=>FL_NUM,NSUBIN=>FL_NSUBIN, &
                   fNbg=>fl_nbgR,qNbg=>fl_nbgQ
  IMPLICIT NONE
  REAL(PREC),DIMENSION(MGRID,NGRID) :: pcc
  REAL(PREC),DIMENSION(:,:,:,:),POINTER :: alpha
  REAL(PREC),DIMENSION(:,:),POINTER :: beta
  REAL(PREC),DIMENSION(:),POINTER :: lmds
  REAL(PREC),DIMENSION(NSUBIN+1+NUM,MGRID,NGRID) :: spc

  REAL(PREC),DIMENSION(:,:,:,:),POINTER :: suu,sux,suy,sbndry
  INTEGER :: neq

  REAL(PREC),DIMENSION(3,MGRID,NGRID) :: pv
  REAL(PREC),DIMENSION(MGRID,NGRID) :: tmp,ginv,qv,fv
  REAL(PREC),DIMENSION(NUM+2,MGRID,NGRID) :: pct
  REAL(PREC),DIMENSION(:,:,:),POINTER :: &
                          uu,ux,uy,tu,tux,tuy,stu,stux,stuy
  INTEGER, PARAMETER :: lfactor=1
  integer count

END MODULE fl_slv_mod

!C:***********************************************************
!C: SLAVE PROGRAM : COMPUTING FUNCTIONAL G
!C:***********************************************************
PROGRAM flow_slave
  USE ntype
  USE para, ONLY : MGRID,NGRID,LMAX,NUM=>FL_NUM,NSUBIN=>FL_NSUBIN, &
```

```
                    lbd=>FL_lbd, ubd=>fl_ubd,nnplot=>fl_nnplot, &
                    nplotstep=>fl_nplotstep,loops=>fl_loops, &
                    no_bndry_value=>fl_no_bndry_value
  USE path, ONLY : home=>fl_home, datapath=>fl_datapath
  USE util, ONLY : pack,unpack,plotting
  USE PVM, ONLY : n_id,n_g,n_pc,n_pcc,n_u,n_para,n_ecd,&
                          n_hvec,n_update,n_stop
  USE fl_slv_mod, ONLY : spc,neq,pcc,lfactor
  IMPLICIT NONE
  INCLUDE 'fpvm3.h'

  INTERFACE
    SUBROUTINE init(lmdvec)
      USE ntype
      INTEGER, DIMENSION(:) :: lmdvec
    END SUBROUTINE init

    SUBROUTINE nbg(var1,var2)
      USE ntype
      INTEGER :: var1,var2
    END SUBROUTINE nbg

    SUBROUTINE compG(var1,var2,cut,step,G)
      USE ntype
      INTEGER :: var1,var2,cut
      REAL(PREC) :: step,G
    END SUBROUTINE compG
  END INTERFACE

!C: ****************
!C: LOCAL VARIABLES
!C: ****************
  INTEGER :: mytid, parentId
  REAL(PREC),DIMENSION(NSUBIN) :: funG

  REAL(PREC) :: gamma,g
  INTEGER :: nvar, ntvar,var,var1,var2,ipter,cut
  INTEGER,DIMENSION(1000) :: vars
  LOGICAL :: plotSite

  DOUBLE PRECISION, DIMENSION(MGRID*NGRID) :: vec
  INTEGER,DIMENSION(LMAX) :: lmdsvec
  INTEGER :: msgtype,ninit, bufid, tid, ierr,bytes,ibuf,k

!C:***************
!C: BEGIN PROGRAM
```

```
!C:***************
!C:
!C:**************************************
!C: ENROLL THIS PROGRAM, REQUIRED BY PVMD
!C:**************************************
    CALL PVMfmytid(mytid)
    CALL PVMfparent(parentid)

    CALL init1(nvar,vars,ipter)

!C:*******************************
!C: BEGIN LOOP OF RECEIVING DATA
!C:*******************************
    DO WHILE(.TRUE.)
      CALL PVMfrecv(parentID, -1, bufid)
      CALL PVMfbufinfo(bufid,bytes,msgtype,tid,ierr)

      SELECT CASE (msgtype)

!C*************************
!C RECEIVE INFORMATION:
!C NUMBER OF lambdas
!C*************************
        CASE (n_id)
          CALL PVMfunpack(INTEGER4,neq,1,1,ierr)
          CALL PVMfunpack(INTEGER4,lmdsvec(1),neq,1,ierr)
          IF (lmdsvec(1)==1) THEN
            plotSite=.TRUE.
          ELSE
            plotSite=.FALSE.
          END IF

!C************************************
!C INITIALIZE THE PARAMETERS:
!C READ IN SOURCE DATA, alpha AND beta
!C************************************
          CALL init(lmdsvec)


!C:***************************
!C: COMPUTE THE FUNCTIONAL G
!C: (AT INITIAL POINT)
!C:***************************
          cut=1
          gamma=0.0_prec
          var2=1
```

```
          DO var1=1,NSUBIN
            CALL compG(var1,var2,cut,gamma,g)
            funG(var1)=g
          END DO

!C:*****************************
!C: GET THE SEARCHING VARIABLE
!C:*****************************
          var1=0
          var2=0
          ntvar=1
          ipter=ipter+1
          var1=MOD(var1,NSUBIN)+1
          var2=vars(MOD(ntvar+ipter,nvar)+1)
          var=var1*1000+var2

!C:*******************************
!C: COMPUTE GRADIENT AT (p,q,f)_0
!C:*******************************
          CALL nbg(var1,var2)

!C:**************************
!C: SEND BACK THE GRADIENT:
!C:      SUM_{i=1,neq} (G_i)
!C:**************************
          k=SIZE(pcc,1)*SIZE(pcc,2)
          CALL pack(vec(1:k),pcc(:,:))

          CALL PVMfinitsend(n_ecd,ibuf)
          CALL PVMfpack(INTEGER4,var,1,1,ierr)
          CALL PVMfpack(INTEGER4,k,1,1,ierr)
          CALL PVMfpack(REAL8,vec(1),k,1,ierr)
          CALL PVMfsend(parentID, n_hvec, ierr)

!C:************************
!C: RECEIVE THE GRADIENT:
!C:      SUM_{i=1,N} (G_i)
!C:************************
        CASE (n_pcc)
          CALL PVMfunpack(INTEGER4,ninit,1,1,ierr)
          CALL PVMfunpack(REAL8,vec(1),ninit,1,ierr)

          CALL unpack(vec(1:ninit), pcc)

!C:*****************************************
!C: STOP THE PROGRAM DUE TO
```

```
!C: REQUEST BY THE SERVER PROGRAM
!C: THERE IS BUG DUE TO PVM, FOR THIS ROUTINE
!C:*****************************************
        CASE (n_stop)
          IF (plotSite) THEN
            CALL plot(ipter,1)
          END IF
          CALL PVMfexit(ierr)
          STOP
        CASE (n_pc)
          CALL plot(ipter,1)

!C:************************************************************
!C: UPDATE THE RECOVERED PARAMETERS
!C: RECEIVE THE gamma VALUE USED IN  H = H + gamma \nabla H
!C:************************************************************
        CASE (n_update)
          CALL PVMfunpack(REAL8,gamma,1,1,ierr)

!C:*********************************************
!C: UPDATE (p,q,f) TO : (p,q,f)_{i}
!C:*********************************************
          IF (gamma > 1.0e-20) THEN
            IF (var2<NUM+2) THEN
              k=var2
            ELSE
              k=NUM+1+var1
            END IF
            spc(k,:,:)=spc(k,:,:)+gamma*pcc

!C:*************************************
!C: IN CASE NO BOUNDARY VALUES ARE GIVEN
!C: THE INSIDE VALUES ARE PROPAGATE TO
!C: THE BOUNDARY:
!C:*************************************
            IF (no_bndry_value) THEN
              spc(k,1:lfactor,:)=spc(k,2:lfactor+1,:)
              spc(k,MGRID-lfactor+1:MGRID,:)= &
                            spc(k,MGRID-lfactor:MGRID-1,:)
              spc(k,:,1:lfactor)=spc(k,:,2:lfactor+1)
              spc(k,:,NGRID-lfactor+1:NGRID)= &
                            spc(k,:,NGRID-lfactor:NGRID-1)
            END IF

!C:*********************************
!C: CUTOFF THE VALUES EXCEEDED THE
```

```
!C: LOWER AND UPPER BOUNDS
!C:*********************************
            WHERE (spc(k,:,:)<lbd(var2)) spc(k,:,:)=lbd(var2)
            WHERE (spc(k,:,:)>ubd(var2)) spc(k,:,:)=ubd(var2)


            IF (var2<NUM+2) THEN
              DO k=1,NSUBIN
                CALL compG(k,var2,cut,0.0_prec,g)
                funG(k)=g
              END DO
            ELSE
              IF (cut==0) CALL compG(var1,var2,cut,0.0_prec,g)
              funG(var1)=g
            END IF
          END IF

!C:***********************************************
!C: PLOT THE RECOVERED PARAMETERS (IF REQUIRED)
!C: NOTE WE ONLY SAVE THE DATA TO A FORMAT THAT
!C: THE tecplot SOFTWARE CAN READ IT
!C:***********************************************
          IF (plotSite) THEN
            IF (MOD(ipter,nnplot)==0) CALL plot(ipter,2)
            IF (MOD(ipter,nplotstep)==0) CALL plot(ipter,1)
          END IF

!C:*********************************
!C: GET THE NEXT SEARCHING VARIABLE
!C:*********************************
          ntvar=MOD(ntvar,nvar)+1
          IF (ntvar==1) THEN
            var1=MOD(var1,NSUBIN)+1
            IF (var1==1) ipter=ipter+1
          END IF
          var2=vars(MOD(ntvar+ipter,nvar)+1)
          var=var1*1000+var2

!C:*********************************
!C: COMPUTE GRADIENT AT (p,q,f)_i
!C:*********************************
          CALL nbg(var1,var2)

!C:*************************
!C: SEND BACK THE GRADIENT:
!C:       SUM_{i=1,neq} (G_i)
```

```
!C:**************************
          k=SIZE(pcc,1)*SIZE(pcc,2)
          CALL pack(vec(1:k),pcc(:,:))

          CALL PVMfinitsend(n_ecd,ibuf)
          CALL PVMfpack(INTEGER4,var,1,1,ierr)
          CALL PVMfpack(INTEGER4,k,1,1,ierr)
          CALL PVMfpack(REAL8,vec(1),k,1,ierr)
          CALL PVMfsend(parentID, n_hvec, ierr)

!C:**********************************
!C: COMPUTE G, RECEIVE gamma
!C: THEN COMPUTE G(c+gamma \nabla g)
!C:**********************************
        CASE (n_g)
          CALL PVMfunpack(INTEGER4,cut,1,1,ierr)
          CALL PVMfunpack(REAL8,vec(1),1,1,ierr)
          gamma=REAL(vec(1),KIND=PREC)

!C:************************
!C: COMPUTE functional G
!C:************************
          IF (gamma == 0.0_prec) THEN
            g=funG(var1)
          ELSE
            IF (var2/=NUM+1) THEN
              CALL compG(var1,var2,cut,gamma,g)
            ELSE
              DO k=1,NSUBIN
                CALL compG(k,var2,cut,gamma,funG(k))
              END DO
              g=funG(var1)
            END IF
          END IF

          gamma=0.0_prec
          DO k=1,NSUBIN
            IF (k==var1) THEN
              gamma=gamma+g
            ELSE
              gamma=gamma+funG(k)
            END IF
          END DO

          CALL PVMfinitsend(n_ecd,ibuf)
          CALL PVMfpack(REAL8,dble(gamma),1,1,ierr)
```

```
          CALL PVMfsend(parentID,n_g,ierr)

        END SELECT
      END DO

!C:**************
!C: SUBROUTINES
!C:**************
    CONTAINS

!C:*********************************************************************
!C: THIS SUBROUTINE SAVE THE DATA P, Q, R TO THE FILES P11.dat,
!C: P12.dat,P22.dat AND Q.dat, R.dat respectively.
!C: THE FORMAT OF THE FILES ARE COMPATIBLE WITH tecplot PROGRAM SO
!C: THAT THE GRAPH CAN BE VIEWED WITH TECPLOT PROGRAM
!C:*********************************************************************
      SUBROUTINE plot(flag1,flag2)
        USE para, ONLY : NUM => FL_NUM,a,b,ha,hb
        INTEGER, INTENT(IN) :: flag1,flag2
        CHARACTER(LEN=1),DIMENSION(3) :: name=(/'P','Q','F'/)

        IF (flag2 == 1) THEN
          CALL PLOTTING(home,name,NUM,spc,ipter,a,b,ha,hb)
          OPEN(UNIT=4,FILE=datapath,STATUS='REPLACE', &
                  ACCESS='SEQUENTIAL',ACTION='WRITE')
            WRITE(4,*) flag1
            do k=1, nsubin+1+num
              WRITE(4,*) spc(k,:,:)
            end do
          CLOSE(4,STATUS='KEEP')
        ELSE IF (flag2==2) THEN
          CALL PLOTTING(home,name,NUM,spc,0,a,b,ha,hb)
        END IF
      END SUBROUTINE plot

!C:*****************************************
!C: THIS SUBROUTINE SET THE SEARCH VARIABLES
!C: AND INITIAL VALUES OF THE VARIABLES
!C:*****************************************
      SUBROUTINE init1(nvar,vars,ipter)
        USE ntype
        USE para, ONLY : MGRID,NGRID,NUM=>FL_NUM,NSUBIN=>FL_NSUBIN, &
                         loops=>FL_loops,newSearch=>fl_newSearch,&
                         pc0=>fl_pc0
        USE para_func, ONLY : trueP,trueQ,trueF
        USE fl_slv_mod, ONLY : spc
```

```
        IMPLICIT NONE
        INTEGER,INTENT(OUT) :: nvar,ipter
        INTEGER,DIMENSION(:),INTENT(OUT) :: vars

        INTEGER :: i,j,k

!C:**************************
!C: GET SEARCHING VARIABLES
!C:**************************
          nvar=0
          vars=0
          DO k=1, NUM+2
            IF (loops(k) /=0 ) THEN
              vars(nvar+1:nvar+loops(k))=k
              nvar=nvar+loops(k)
            END IF
          END DO
          nvar=SUM(LOOPS)

!C:********************
!C: INITIALIZE p,q,f
!C:********************
          IF (newSearch) THEN
            ipter=0
            DO k=1, NUM
              IF (loops(k) /=0 ) THEN
                spc(k,:,:)=pc0(k)
              ELSE
                DO i=1,MGRID
                DO j=1,NGRID
                  spc(k,i,j)=trueP(k,i,j)
                END DO
                END DO
              END IF
            END DO
            IF (loops(NUM+1)/=0) THEN
              spc(NUM+1,:,:)=pc0(NUM+1)
            ELSE
              DO i=1,MGRID
              DO j=1,NGRID
                !spc(NUM+1,i,j)=trueQ(i,j)
              END DO
              END DO
              spc(NUM+1,:,:)=pc0(NUM+1)
            END IF
```

```
            IF (loops(NUM+2)/=0) THEN
              spc(NUM+2:NUM+1+NSUBIN,:,:)=pc0(NUM+2)
            ELSE
              DO i=1,MGRID
              DO j=1,NGRID
              DO k=1,NSUBIN
                spc(NUM+1+k,i,j)=trueF(k,i,j)
              END DO
              END DO
              END DO
            END IF
          ELSE

            OPEN(UNIT=4,FILE=datapath,STATUS='OLD', &
                 ACCESS='SEQUENTIAL',ACTION='READ')
              READ(4,*) ipter
              do i=1, nsubin+1+num
                READ(4,*) spc(i,:,:)
              end do
            CLOSE(4,STATUS='KEEP')
          END IF
      END SUBROUTINE init1

  END PROGRAM flow_slave

!C:*********************************************************************
!C: THIS SUBROUTINE ALLOCATE THE VARIABLES, SET THE LAMBDA VALUES
!C: AND READ IN THE SOURCE DATA u AND alpha, beta
!:*********************************************************************
  SUBROUTINE init(lmdvec)
    USE ntype
    USE para, ONLY : MGRID,NGRID,a,b,ha,hb,lambda,TSIZE, &
                     refineData=>fl_refineData,NUM=>FL_NUM, &
                     NSUBIN=>FL_NSUBIN
    USE para_func, ONLY : trueP,trueQ,trueF,fb
    USE path, ONLY : source=>fl_source, fl_home
    USE util, ONLY : createArray
    USE dir, ONLY : dir
    USE fl_slv_mod, ONLY : suu,sux,suy,sbndry,alpha,beta,lmds, &
                           uu,ux,uy,tu,tux,tuy,stu,stux,stuy,pct,&
                           neq ,count
    IMPLICIT NONE
    INTEGER,DIMENSION(:), INTENT(IN) :: lmdvec

    REAL(PREC),DIMENSION(NSUBIN+1,MGRID,NGRID) :: tbU
```

```
      INTERFACE
        SUBROUTINE solve(pc,k,uu,ux,uy)
          USE ntype
          REAL(PREC),DIMENSION(:,:,:) :: pc,uu,ux,uy
          INTEGER :: k
        END SUBROUTINE solve
      END INTERFACE

      INTEGER :: i,j,k
      REAL(PREC) :: lmd
      REAL(PREC),DIMENSION(NSUBIN+1) :: tvec

!C:**********************
!C: ALLOCATE VARIABLES
!C**********************

        suu  => createArray(NSUBIN,neq,MGRID,NGRID,'slave')
        sux  => createArray(NSUBIN,neq,MGRID,NGRID,'slave')
        suy  => createArray(NSUBIN,neq,MGRID,NGRID,'slave')
        sbndry  => createArray(NSUBIN,neq,4,NGRID,'slave')

        alpha  => createArray(NSUBIN,neq,MGRID,NGRID,'slave')
        beta  => createArray(NSUBIN,neq,'slave')
        lmds  => createArray(neq,'slave')

        uu   => createArray(neq,MGRID,NGRID,'slave')
        ux   => createArray(neq,MGRID,NGRID,'slave')
        uy   => createArray(neq,MGRID,NGRID,'slave')
        tu   => createArray(neq,MGRID,NGRID,'slave')
        tux  => createArray(neq,MGRID,NGRID,'slave')
        tuy  => createArray(neq,MGRID,NGRID,'slave')
        stu  => createArray(neq,MGRID,NGRID,'slave')
        stux => createArray(neq,MGRID,NGRID,'slave')
        stuy => createArray(neq,MGRID,NGRID,'slave')


!C:**************************
!C: SET THE TIME INTERVALS
!**************************
        lmd=tsize/NSUBIN
        DO k=0, NSUBIN
          tvec(k+1)=k*lmd
        END DO

!C:************************
!C: SET THE LAMBDA VALUES
```

```
!C:************************
      DO k=1,neq
        lmds(k)=lmdvec(k)*lambda
      END DO

!C:***********************
!C: READ IN SOURCE DATA
!C:***********************
      OPEN(UNIT=4,FILE=source,STATUS='OLD', &
                  ACCESS='SEQUENTIAL',ACTION='READ')
        READ(4,*) tbU
        DO k=1, lmdvec(1)-1
          READ (4,*) suu(:,1,:,:)
        END DO
        DO k=1, neq
          READ (4,*) suu(:,k,:,:)
        END DO
      CLOSE(4,STATUS='KEEP')

      sbndry(:,1:neq,1,1:NGRID)=suu(:,1:neq,1,:)
      sbndry(:,1:neq,2,1:NGRID)=suu(:,1:neq,mgrid,:)
      sbndry(:,1:neq,3,1:MGRID)=suu(:,1:neq,:,1)
      sbndry(:,1:neq,4,1:MGRID)=suu(:,1:neq,:,ngrid)

!C:*****************************
!C: COMPUTE THE DERIVATIVES
!C:*****************************
      DO k=1,NSUBIN
        uu=suu(k,:,:,:)
        CALL dir(uu(:,:,:),ux(:,:,:),uy(:,:,:),a,b,ha,hb)
        sux(k,:,:,:)=ux
        suy(k,:,:,:)=uy
      END DO

!C:***********************
!C: COMPUTE alpha, beta
!C:***********************
      DO k=1,neq
        lmd=lmds(k)
        DO i=1,NSUBIN
          alpha(i,k,:,:)=tbU(i+1,:,:)*exp(-lmd*tvec(i+1)) &
                         - tbU(i,:,:)*exp(-lmd*tvec(i))

          beta(i,k)=(exp(-lmd*tvec(i))-exp(-lmd*tvec(i+1)))/lmd
        END DO
      END DO
```

```
!C:************************************
!C: RECOMPUTE SOURCE DATA, IF REQUIRED
!C:************************************
      IF (refinedata) THEN
        DO i=1,MGRID
        DO j=1,NGRID
          DO k=1,NUM
            pct(k,i,j)=trueP(k,i,j)
          END DO
          pct(NUM+1,i,j)=trueQ(i,j)
        END DO
        END DO

        DO k=1,NSUBIN
          DO i=1,MGRID
          DO j=1,NGRID
            pct(NUM+2,i,j)=trueF(k,i,j)
          END DO
          END DO
          CALL solve(pct,k,uu,ux,uy)
          suu(k,:,:,:)=uu
          sux(k,:,:,:)=ux
          suy(k,:,:,:)=uy
        END DO

      END IF
  END SUBROUTINE init

!C:*********************************************************************
!C: THIS SUBROUTINE COMPUTE THE (NEUBERGER) GRADIENT OR L^1 GRADIENT
!C: FOR THE DESCENT DIRECTION OF THE SEARCHED VARIABLES
!C:*********************************************************************
  SUBROUTINE nbg(var1,var2)
    USE ntype
    USE para, ONLY : MGRID,NGRID,a,b,ha,hb,NUM=>FL_NUM,NSUBIN=>FL_NSUBIN
    USE ellsov, ONLY : Elliptic_Solver
    USE util, ONLY : pack
    USE simpson, ONLY : quad2d
    USE fl_slv_mod, ONLY : spc,suu,sux,suy,neq,uu,ux,uy,tu,tux,tuy,&
                           qNbg,fNbg,stu,stux,stuy,pct,tmp,ginv,alpha,
                           beta,lmds,pcc,count
    USE path, ONLY : home=>fl_home

    IMPLICIT NONE
    INTEGER,INTENT(IN) :: var1,var2
```

```
    INTERFACE
      SUBROUTINE solve(pc,k,uu,ux,uy)
        USE ntype
        REAL(PREC),DIMENSION(:,:,:) :: pc,uu,ux,uy
        INTEGER :: k
      END SUBROUTINE solve
    END INTERFACE

!C:*******************
!C: LOCAL VARIABLES
!C:*******************
    INTEGER :: k,ll

!C:************************************************************
!C: THERE IS A BUG HERE, WE HAVE TO SET THE PROGRAM TO PRINT
!C: SOMETHING, OTHERWISE THE PROGRAM WILL STOP
!C:************************************************************
      print *, ' Hi'
      pct(1:NUM+1,:,:)=spc(1:NUM+1,:,:)
      pct(NUM+2,:,:)=spc(NUM+1+var1,:,:)

!C:***************************************************
!C: GET THE SOLUTION OF
!C:  -\nabla p \nabla u + (\lambda u + alpha) q = f
!C: CORRESPONDING TO VARIABLES var1 AND var2
!C:***************************************************
      CALL solve(pct,var1,tu,tux,tuy)
      uu=suu(var1,:,:,:)
      ux=sux(var1,:,:,:)
      uy=suy(var1,:,:,:)

      ginv=0.0_prec

!C:*****************************
!C: COMPUTE THE L^1 GRADIENT
!C:*****************************
      IF (NUM==1) THEN
        SELECT CASE (var2)
          CASE (1)
            DO k=1, neq
              ginv=ginv+tux(k,:,:)*tux(k,:,:)+tuy(k,:,:)*tuy(k,:,:) &
                       -ux(k,:,:)*ux(k,:,:)-uy(k,:,:)*uy(k,:,:)
            END DO

          CASE (2)
```

```
            DO k=1, neq
              ginv=ginv &
                  +lmds(k)*(tu(k,:,:)*tu(k,:,:)-uu(k,:,:)*uu(k,:,:)) &
                  +2.0_prec*alpha(var1,k,:,:)*(tu(k,:,:)-uu(k,:,:))
            END DO

          CASE DEFAULT
            DO k=1, neq
              ginv=ginv+beta(var1,k)*(uu(k,:,:)-tu(k,:,:))
            END DO
        END SELECT
      ELSE
        SELECT CASE (var2)
          CASE (1)
            DO k=1, neq
              ginv=ginv+tux(k,:,:)*tux(k,:,:)-ux(k,:,:)*ux(k,:,:)
            END DO

          CASE (2)
            DO k=1, neq
              ginv=ginv+tux(k,:,:)*tuy(k,:,:)-ux(k,:,:)*uy(k,:,:)
            END DO

          CASE (3)
            DO k=1, neq
              ginv=ginv+tuy(k,:,:)*tuy(k,:,:)-uy(k,:,:)*uy(k,:,:)
            END DO

          CASE (4)
            DO k=1, neq
              ginv=ginv &
                  +lmds(k)*(tu(k,:,:)*tu(k,:,:)-uu(k,:,:)*uu(k,:,:)) &
                  +2.0_prec*alpha(var1,k,:,:)*(tu(k,:,:)-uu(k,:,:))
            END DO

          CASE DEFAULT
            DO k=1, neq
              ginv=ginv+beta(var1,k)*(uu(k,:,:)-tu(k,:,:))
            END DO
        END SELECT
      END IF

!C:*********************************************
!C: COMPUTE THE NEUBERGER GRADIENT (IF REQUIRED)
!C:*********************************************
      IF ((var2<NUM+1) .OR. (var2==NUM+1 .AND. qNbg) &
```

```
          .OR. (var2==NUM+2 .AND. fNbg) ) THEN
        pct(1,:,:)=1.0_prec
        pct(2,:,:)=0.0_prec
        pct(3,:,:)=1.0_prec
        CALL Elliptic_Solver(pct(1:3,:,:),pct(1,:,:), &
                             ginv,pct(2,1:4,:),ha,hb,pcc)
      ELSE
        pcc=ginv
      END IF
  END SUBROUTINE nbg

!C:*********************************************************************
!C: THE SUBROUTINE CALLS THE ELLIPTIC SOLVER TO SOLVE THE EQUATIONS AND
!C: THE NUMERICAL DERIVATIVES OF THE CORRESPONDING SOLUTION FUNCTIONS
!C:*********************************************************************
  SUBROUTINE solve(pc,n,uu,ux,uy)
    USE ntype
    USE para, ONLY : a,b,ha,hb,lambda,NUM=>FL_NUM
    USE ellsov, ONLY : Elliptic_Solver
    USE dir, ONLY : dir
    USE fl_slv_mod, ONLY : sbndry,alpha,beta,pv,qv,fv,neq,lmds,count
    use path, only : home=>fl_home
    IMPLICIT NONE
    INTEGER,INTENT(IN) :: n
    REAL(PREC),DIMENSION(:,:,:),INTENT(IN) :: pc
    REAL(PREC),DIMENSION(:,:,:),INTENT(OUT) :: uu,ux,uy

    INTEGER :: k ,i,j
      DO k=1,NUM
        pv(k,:,:)=pc(k,:,:)
      END DO
      IF (NUM==1) THEN
        pv(2,:,:)=0.0_prec
        pv(3,:,:)=pv(1,:,:)
      END IF

      DO k=1,neq
        qv=lmds(k)*pc(NUM+1,:,:)
        fv=beta(n,k)*pc(NUM+2,:,:)-alpha(n,k,:,:)*pc(NUM+1,:,:)

        CALL Elliptic_Solver(pv,qv,fv,sbndry(n,k,:,:),ha,hb,uu(k,:,:))
      END DO
      CALL dir(uu(1:neq,:,:),ux(1:neq,:,:),uy(1:neq,:,:),a,b,ha,hb)
  END SUBROUTINE solve

!C:*********************************************
```

```
!C: THIS SUBROUTINE COMPUTE THE FUNCTIONALS G_i
!C:*********************************************
  SUBROUTINE compG(var1,var2,cut,step,G)
    USE ntype
    USE para, ONLY : MGRID,NGRID,a,b,ha,hb,NUM=>FL_NUM,
                     NSUBIN=>FL_NSUBIN,lbd=>fl_lbd,ubd=>fl_ubd
    USE simpson, ONLY : quad2d
    USE util, ONLY : positive
    USE fl_slv_mod, ONLY : neq,lmds,suu,sux,suy,spc,pcc,pct, &
                        uu,ux,uy,tu,tux,tuy,stu,stux,stuy,tmp ,count
    use path, only : home=>fl_home

    IMPLICIT NONE
    INTEGER, INTENT(IN) :: var1,var2,cut
    REAL(PREC), INTENT(IN) :: step
    REAL(PREC),INTENT(OUT) :: G

    INTERFACE
      SUBROUTINE solve(pc,n,uu,ux,uy)
        USE ntype
        REAL(PREC),DIMENSION(:,:,:) :: pc,uu,ux,uy
        INTEGER :: n
      END SUBROUTINE
    END INTERFACE

!C:******************
!C: LOCAL VARIABLES
!C:******************
    INTEGER :: k

!C:*****************************
!C: COMPUTE tu,tux,tuy at p+h
!C:*****************************
      pct(1:NUM+1,:,:)=spc(1:NUM+1,:,:)
      pct(NUM+2,:,:)=spc(NUM+1+var1,:,:)
      pct(var2,:,:)=pct(var2,:,:)+step*pcc

!C:*********************************************************************
!C:   IF WE NEED TO CHECK THE UPPER AND LOWER BOUND, THEN cut==1.
!C:   THIS SET WILL USUALLY MAKE THE SEARCH EFFICIENT, BUT WILL STUCK
!C:   AT SOME SEARCHING STEP. SO WE SET THE VARIABLE cut TO BE 1 AS
!C:   LONG AS THE SEARCH IS SUCCESSFUL. IF AT SOME STEP THE SEARCH IS
!C:   FAILED THEN WE RESET cut TO BE 0 SO THAT WE CAN MAKE FURTHER
!C:   SEARCH. THE CONTROL OF THE VARIABLE IS BY THE SERVER PART OF
!C:   THE PROGRAM.
!C:*********************************************************************
```

```
      IF (cut==1) THEN
        WHERE (pct(var2,:,:)<lbd(var2)) pct(var2,:,:)=lbd(var2)
        WHERE (pct(var2,:,:)>ubd(var2)) pct(var2,:,:)=ubd(var2)
      END IF

!C:*********************************************************************
!C: IF p IS NOT POSITIVE, EXIT BY ASSING G A VERY BIG VALUE TO TELL
!C: THE SERVER PART THAT SEARCHING IS FAILED. OTHERWISE, WE COMPUTE
!C: THE SOLUTION OF EQUATION
!C:*********************************************************************
      IF (.NOT. positive(pct(1:NUM,:,:))) THEN
        G=1000.0_prec
        RETURN
      END IF

      CALL solve(pct,var1,tu,tux,tuy)
      uu=suu(var1,:,:,:)
      ux=sux(var1,:,:,:)
      uy=suy(var1,:,:,:)

!C:*****************************************
!C: COMPUTE FUNCTIONAL G AT: (p,q,f)_{i}+a*h
!C:*****************************************
      tmp=0.0_prec
      DO k=1, neq
        IF (NUM==1) THEN
          tmp(:,:)=tmp(:,:)+pct(1,:,:) &
            *((tux(k,:,:)-ux(k,:,:))*(tux(k,:,:)-ux(k,:,:)) &
             +(tuy(k,:,:)-uy(k,:,:))*(tuy(k,:,:)-uy(k,:,:)))
        ELSE
          tmp(:,:)=tmp(:,:) &
            +pct(1,:,:)*(tux(k,:,:)-ux(k,:,:))*(tux(k,:,:)-ux(k,:,:)) &
            +2*pct(2,:,:)*(tux(k,:,:)-ux(k,:,:))*(tuy(k,:,:)-uy(k,:,:))&
            +pct(3,:,:)*(tuy(k,:,:)-uy(k,:,:))*(tuy(k,:,:)-uy(k,:,:))
        END IF
      END DO
      tmp=tmp
      DO k=1,neq
        tmp=tmp+pct(NUM+1,:,:)*lmds(k)*(tu(k,:,:)-uu(k,:,:)) &
                                       *(tu(k,:,:)-uu(k,:,:))
      END DO
      CALL quad2d(tmp,a,b,ha,hb,G)
  END SUBROUTINE compG
```

APPENDIX B

Fortran code: Finite Laplace transformation

```
MODULE fl_laplace_mod
  USE ntype
  USE para, ONLY : MGRID,NGRID,NSUBIN=>FL_NSUBIN,NUM=>FL_NUM
  REAL(PREC),DIMENSION(NUM+1+NSUBIN,MGRID,NGRID) :: spc
  REAL(PREC),DIMENSION(MGRID,NGRID) :: vec
  REAL(PREC),DIMENSION(2) :: r0
  REAL(PREC) :: const=0.4665123934
  LOGICAL :: smooth=.FALSE.
END MODULE fl_laplace_mod


PROGRAM laplace_transform
  USE ntype
  USE para, ONLY : LMAX,MGRID,NGRID,NSTEPS,TSIZE,lambda,ha,hb,a,b,h,&
                   NSUBIN=>FL_NSUBIN,compare=>FL_compare
  USE path, ONLY : source=>fl_source,origphi=>fl_origphi, &
                            compphi=>fl_compphi
  USE util, ONLY : qsimp,plotting
  USE fl_laplace_mod
  USE quad2d, ONLY : quad2d_qgaus

  IMPLICIT NONE
  REAL(PREC), DIMENSION(NSUBIN+1,MGRID,NGRID) :: tbU
  REAL(PREC), DIMENSION(NSUBIN,LMAX,MGRID,NGRID) :: u
  REAL(PREC), DIMENSION(NSUBIN,NSTEPS+1,MGRID,NGRID) :: data
  REAL(PREC), DIMENSION(NSUBIN*NSTEPS+1,MGRID,NGRID) :: phi
  REAL(PREC), DIMENSION(NSTEPS+1) :: func
  REAL(PREC) :: t,tt,t0,t1,lmd,hx,hy, dump
  INTEGER :: k,ll,i,j,it
  CHARACTER(LEN=100) :: filephi

!C:************************
!C: READ IN SOURCE DATA
!C:************************
    print *, ' Read in data'
    OPEN(UNIT=4,FILE=ORIGPHI,STATUS='OLD', &
            ACCESS='SEQUENTIAL',ACTION='READ')
        DO k=1, 5
          READ(4,*) dump
        END DO
        DO k=1, NSUBIN*NSTEPS+1
          READ (4,*) dump
          DO i=1, MGRID
          DO j=1, NGRID
            READ (4,*) phi(k,i,j)
          END DO
```

```
          END DO
        END DO
    CLOSE(4,STATUS='KEEP')
!C:*************************************
!C: PLOT THE DATA AT TIME=10, 20, 30, ...
!C:*************************************
    DO k=1, NSUBIN*NSTEPS, 30
!       CALL PLOTTING(phi(k,:,:),'phi','',a,b,ha,hb)
!       pause 'Check the graph result'
    END DO

!C:*******************************
!C: SMOOTH THE DATA (IF REQUIRED)
!C:*******************************
    print *, ' smooth data'
    IF (smooth) THEN
      hx=ha/(MGRID-1)
      hy=hb/(NGRID-1)
      DO k=1,SIZE(phi,1)
        print *, ' k = ', k
        vec=phi(k,:,:)
        DO i=1,MGRID
          r0(1)=a+hx*(i-1)
          DO j=1,NGRID
            r0(2)=b+hy*(j-1)
            CALL quad2d_qgaus(r0(1)-h,r0(1)+h,phi(k,i,j))
          END DO
        END DO
      END DO

!C:*************************************
!C: PLOT THE DATA AT TIME=10, 20, 30, ...
!C:*************************************
      DO k=10, 100, 10
!         CALL PLOTTING(phi(k,:,:),'phi','',a,b,ha,hb)
      END DO
    END IF

!C:*********************************
! COPY THE DATA TO data VARIABLE
!C:*********************************
    DO k=1,NSUBIN
      data(k,1:NSTEPS+1,:,:)=phi((k-1)*NSTEPS+1:k*NSTEPS+1,:,:)
      tbU(k,:,:)=data(k,1,:,:)
    END DO
    tbU(NSUBIN+1,:,:)=data(NSUBIN,NSTEPS+1,:,:)
```

```
!C:*************************
!C:  Finite Laplace transform
!C:*************************
      print *, ' Begin Laplace Transformation'
      tt=TSIZE/NSUBIN
      DO k=1,NSUBIN
        print *, ' k = ', k
        t0=(k-1)*tt
        t1=k*tt

        DO ll=1,LMAX
          DO i=1,MGRID
          DO j=1,NGRID

!C:**********************
!C:  load simpson vector
!C:**********************
            DO it=1,NSTEPS+1
              t=t0+(it-1)*(t1-t0)/NSTEPS
              lmd=real(lambda*ll)
              func(it)=data(k,it,i,j)*exp(-lmd*t)
            END DO
            call qsimp(func,t0,t1,u(k,ll,i,j))
          END DO
          END DO
        END DO
      END DO


!C:******************
!C: SAVE THE RESULTS
!C:******************
      OPEN(UNIT=4,FILE=source,STATUS='REPLACE', &
           ACCESS='SEQUENTIAL',ACTION='WRITE')
        WRITE(4,*) tbU
        DO k=1,LMAX
          WRITE(4,*) u(:,k,:,:)
        END DO
      CLOSE(4,STATUS='KEEP')

END PROGRAM laplace_transform


!C:*****************************************************
```

```
!C: BOUNDARY AND INTEGRAL FUNCTIONS : y1_2d,y2_2d,func
!C:***************************************************
  FUNCTION y1_2d(x)
    USE ntype
    USE para, ONLY : h
    USE fl_laplace_mod, ONLY : r0
    REAL(PREC), INTENT(IN) :: x
    REAL(PREC) :: y1_2d
      y1_2d=r0(2)-sqrt(h*h-(x-r0(1))*(x-r0(1)))
  END FUNCTION y1_2d

  FUNCTION y2_2d(x)
    USE ntype
    USE para, ONLY : h
    USE fl_laplace_mod, ONLY : r0
    REAL(PREC), INTENT(IN) :: x
    REAL(PREC) :: y2_2d
      y2_2d=r0(2)+sqrt(h*h-(x-r0(1))*(x-r0(1)))
  END FUNCTION y2_2d

  FUNCTION func_2d(x,y)
    USE ntype
    USE para, ONLY : h,TSIZE,a,b,ha,hb,NUM=>FL_NUM,NSUBIN=>FL_NSUBIN
    USE util, ONLY : blitp
    USE fl_laplace_mod
    IMPLICIT NONE
    REAL(PREC), INTENT(IN) :: x
    REAL(PREC), DIMENSION(:), INTENT(IN) :: y
    REAL(PREC), DIMENSION(size(y)) :: func_2d

    INTEGER :: k,n
    REAL(PREC) :: rho
    REAL(PREC) :: r
      DO k=1,SIZE(y)
        r=((x-r0(1))*(x-r0(1))+(y(k)-r0(2))*(y(k)-r0(2)))/(h*h)
        rho=exp(1.0_prec/(r-1))
        func_2d(k)=rho*blitp(x,y(k),vec,a,b,ha,hb)/(h*h*const)
      END DO
  END FUNCTION func_2d
```

APPENDIX C

Fortran code: Compute the errors between the recovered and the original data

```
MODULE fl_compErr_mod
  USE ntype
  USE para, ONLY : MGRID,NGRID,NSUBIN=>FL_NSUBIN,NUM=>FL_NUM,NSTEPS
  REAL(PREC),DIMENSION(NUM+1+NSUBIN,MGRID,NGRID) :: spc
  REAL(PREC),DIMENSION(MGRID,NGRID) :: vec
  REAL(PREC),DIMENSION(2) :: r0
  REAL(PREC) :: const=0.4665123934
  REAL,DIMENSION(NSTEPS*NSUBIN+1,MGRID,NGRID) :: phi
  REAL(PREC) :: dt,dx,dy
END MODULE fl_compErr_mod


PROGRAM compError
  USE ntype
  USE para, ONLY : MGRID,NGRID,NSTEPS,TSIZE,ha,hb,a,b,h,&
                   NSUBIN=>FL_NSUBIN
  USE path, ONLY : origphi=>fl_origphi,compphi=>fl_compphi, &
                   datapath=>fl_datapath,home=>fl_home
  USE util, ONLY : plotting
  USE fl_compErr_mod
  USE quad2d, ONLY : quad2d_qgaus

  IMPLICIT NONE
  REAL(PREC), DIMENSION(NSUBIN,NSTEPS+1,MGRID,NGRID) :: data
  REAL(PREC), DIMENSION(NSUBIN*NSTEPS+1,MGRID,NGRID) :: &
                                          newphi,error
  REAL(PREC), DIMENSION(NSTEPS+1) :: func
  INTEGER :: k,i,j
  REAL(PREC) :: err,norm,t,ht,hx,hy,dump
  CHARACTER(LEN=2),DIMENSION(13) :: &
              char=(/'00','01','02','03','04','05','06',&
                     '07','08','09','10','11','12'/)
  CHARACTER(LEN=1),DIMENSION(3) :: name=(/'P', 'Q', 'F'/)
  REAL(PREC), DIMENSION(MGRID,NGRID) :: vvec

  INTERFACE
    SUBROUTINE getData(data, bndryfunc)
      USE ntype
      REAL(PREC),DIMENSION(:,:,:,:) :: data

      INTERFACE
        FUNCTION bndryfunc(x,y,t)
          REAL :: x,y,t,bndryfunc
        END FUNCTION
      END INTERFACE
    END SUBROUTINE
```

```
    FUNCTION trueU(x,y,t)
      REAL :: x,y,t, trueU
    END FUNCTION
  END INTERFACE

!C:**************
!C: READ IN DATA
!C:**************
    dt=TSIZE/(NSUBIN*NSTEPS)
    dx=ha/(MGRID-1)
    dy=hb/(NGRID-1)
    OPEN(UNIT=4,FILE=datapath,STATUS='OLD', &
            ACCESS='SEQUENTIAL',ACTION='READ')
      READ(4,*) k
      DO k=1, size(spc,1)
        READ(4,*) spc(k,:,:)
      END DO
    CLOSE(4,STATUS='KEEP')

    print *, ' Read in data'
    OPEN(UNIT=4,FILE=ORIGPHI,STATUS='OLD', &
            ACCESS='SEQUENTIAL',ACTION='READ')
        DO k=1, 5
          READ(4,*) dump
        END DO
        DO k=1, NSUBIN*NSTEPS+1
          READ (4,*) dump
          DO i=1, MGRID
          DO j=1, NGRID
            READ (4,*) phi(k,i,j)
          END DO
          END DO
        END DO
    CLOSE(4,STATUS='KEEP')


!C:************************
!C: SMOOTH THE COMPUTED DATA
!C:************************

    hx=ha/(MGRID-1)
    hy=hb/(NGRID-1)
    DO k=1,SIZE(spc,1)
      vec=spc(k,:,:)
      DO i=1,MGRID
```

```
        r0(1)=a+hx*(i-1)
        DO j=1,NGRID
          r0(2)=b+hy*(j-1)
          CALL quad2d_qgaus(r0(1)-h,r0(1)+h,spc(k,i,j))
        END DO
      END DO
    END DO

!C:**************
!C: COMPUTE DATA
!C:**************
    CALL getData(data,trueU)

    DO k=1,NSUBIN
      newphi((k-1)*NSTEPS+1:k*NSTEPS,:,:)=data(k,1:NSTEPS,:,:)
    END DO
    newphi(NSUBIN*NSTEPS+1,:,:)=data(NSUBIN,NSTEPS+1,:,:)

!C:**************
!C: SAVE DATA
!C:**************
    OPEN(UNIT=4,FILE=compphi,STATUS='REPLACE', &
           ACCESS='SEQUENTIAL',ACTION='WRITE')
      DO k=1, size(newphi,1)
      DO i=1, MGRID
      DO j=1, NGRID
        WRITE(4,*) newphi(k,i,j)
      END DO
      END DO
      END DO
    CLOSE(4,STATUS='KEEP')

!C:***************
!C: COMPUTE ERROR
!C:***************
    DO k=1, NSUBIN*NSTEPS+1
      err=0.0_prec
      norm=0.0_prec
      norm=0.0_prec
      DO i=1,MGRID
      DO j=1,NGRID
        IF (err<abs(phi(k,i,j)-newphi(k,i,j))) &
                    err=abs((phi(k,i,j)-newphi(k,i,j)))
        IF (norm<abs(phi(k,i,j))) norm=abs(phi(k,i,j))
      END DO
      END DO
```

```
      error(k)=err/norm
      print *, ' error(k) =', error(k)

!C:*******************************
!C: plot the error at k=10,20,...
!C:*******************************
      IF (MOD(K-1,10)==0) THEN
        CALL PLOTTING(phi(k,:,:)-newphi(k,:,:), &
                  home//'data/error'//char((k-1)/13+1),'',a,b,ha,hb)
      END IF
    END DO

!C:*********************************
!C: PLOT THE ERROR AS FUNCTION OF T
!C:*********************************
    ht=TSIZE/NSTEPS
    OPEN (4, file=home//'data/error.dat')
      WRITE(4,*) 'TITLE=error:'
      WRITE(4,*) 'VARIABLES="T" "Error"'
      WRITE(4,*) 'ZONE I=',NSTEPS+1, ', C=BLUE'
      DO i=1, NSTEPS+1
        t = (i-1)*ht
        WRITE(4,*) t, error(i)
      END DO
    CLOSE(4, STATUS='keep')
END PROGRAM compError

!C:*********************************************
!C:
!C: test solution functions
!C:

  REAL FUNCTION trueU(X,Y,T)
    USE para, ONLY : a,b,ha,hb
    USE fl_compErr_mod, ONLY : phi, dt
    USE util, ONLY : blitp
    implicit none
    REAL :: T, X, Y, U
    REAL, PARAMETER :: pi=3.14159
    INTEGER :: nh1,nh2
    REAL :: h1,h2,val1,val2,val3,val4
      nh1=CEILING(t/dt)
      nh2=FLOOR(t/dt)
      h1=(dt*nh1-t)/dt
      h2=1.0-h1
```

```
      trueU=blitp(x,y,phi(nh2+1,:,:),a,b,ha,hb)*h1 &
           +blitp(x,y,phi(nh1+1,:,:),a,b,ha,hb)*h2
  END FUNCTION trueU

!C:***********************************
!C: SUBROUTINES USED BY PDETWO
!C:***********************************
  SUBROUTINE bndryv (t,x,y,u,av,bv,cv,npde)
    IMPLICIT NONE
    REAL  t,u,x,y,bv,av,cv
    INTEGER npde
    DIMENSION u(npde),av(npde),bv(npde),cv(npde)
    REAL, PARAMETER :: pi=3.14159

    INTERFACE
      FUNCTION trueU(x,y,t)
        REAL x,y,t, trueU
      END FUNCTION trueU
    END INTERFACE
      av(1) = 1.0
      bv(1) = 0.0
      cv(1)=trueU(x,y,t)
  END SUBROUTINE bndryv

  SUBROUTINE bndryh (t,x,y,u,ah,bh,ch,npde)
    IMPLICIT NONE
    REAL  t,u,x,y,bh,ah,ch
    INTEGER npde
    DIMENSION u(npde),ah(npde),bh(npde),ch(npde)
    REAL, PARAMETER :: pi=3.14159

    INTERFACE
      FUNCTION trueU(x,y,t)
        REAL x,y,t, trueU
      END FUNCTION trueU
    END INTERFACE

      ah(1) = 1.0
      bh(1) = 0.0
      ch(1)=trueU(x,y,t)
  END SUBROUTINE bndryh

  SUBROUTINE diffh (t,x,y,u,dh,npde)
    USE ntype
    USE para, ONLY : a,b,ha,hb
    USE fl_compErr_mod, ONLY : spc
```

```
  USE util, ONLY : blitp
  IMPLICIT NONE
  REAL, INTENT(IN) ::  t,x,y
  INTEGER npde
  REAL, DIMENSION(npde),INTENT(IN) :: u
  REAL, DIMENSION(npde,npde), INTENT(OUT) :: dh

    dh(1,1)=blitp(x,y,spc(1,:,:),a,b,ha,hb)
END SUBROUTINE diffh

SUBROUTINE diffv (t,x,y,u,dv,npde)
  USE ntype
  USE para, ONLY : a,b,ha,hb,NUM=>FL_NUM
  USE fl_compErr_mod, ONLY : spc
  USE util, ONLY : blitp
  IMPLICIT NONE
  REAL, INTENT(IN) ::  t,x,y
  INTEGER npde
  REAL, DIMENSION(npde),INTENT(IN) :: u
  REAL, DIMENSION(npde,npde), INTENT(OUT) :: dv

    IF (NUM==1) THEN
      dv(1,1)=blitp(x,y,spc(1,:,:),a,b,ha,hb)
    ELSE
      dv(1,1)=blitp(x,y,spc(3,:,:),a,b,ha,hb)
    END IF
END SUBROUTINE diffv

SUBROUTINE diffch (t,x,y,u,dch,npde)
  USE ntype
  USE para, ONLY : a,b,ha,hb,NUM=>FL_NUM
  USE fl_compErr_mod, ONLY : spc
  USE util, ONLY : blitp
  IMPLICIT NONE
  REAL, INTENT(IN) ::  t,x,y
  INTEGER npde
  REAL, DIMENSION(npde),INTENT(IN) :: u
  REAL, DIMENSION(npde,npde), INTENT(OUT) :: dch

    IF (NUM==1) THEN
      dch(1,1)=0.0_prec
    ELSE
      dch(1,1)=blitp(x,y,spc(2,:,:),a,b,ha,hb)
    END IF
END SUBROUTINE diffch
```

```
  SUBROUTINE diffcv (t,x,y,u,dcv,npde)
    USE ntype
    USE para, ONLY : a,b,ha,hb, NUM=>FL_NUM
    USE fl_compErr_mod, ONLY : spc
    USE util, ONLY : blitp
    IMPLICIT NONE
    REAL, INTENT(IN) ::  t,x,y
    INTEGER npde
    REAL, DIMENSION(npde),INTENT(IN) :: u
    REAL, DIMENSION(npde,npde), INTENT(OUT) :: dcv

      IF (NUM==1) THEN
        dcv(1,1)=0.0_prec
      ELSE
        dcv(1,1)=blitp(x,y,spc(2,:,:),a,b,ha,hb)
      END IF
  END SUBROUTINE diffcv

  SUBROUTINE f(t,x,y,u,ux,uy,duxx,duyy,duxy,duyx,dudt,npde)
    USE ntype
    USE para, ONLY : a,b,ha,hb,TSIZE,NUM=>FL_NUM
    USE fl_compErr_mod, ONLY : spc,NSUBIN
    USE util, ONLY : blitp
    IMPLICIT NONE
    REAL, INTENT(IN) ::  t,x,y
    INTEGER npde
    REAL, DIMENSION(npde),INTENT(IN) :: u,ux,uy
    REAL, DIMENSION(npde,npde), INTENT(IN) :: duxx,duxy,duyy,duyx
    REAL, DIMENSION(npde,npde), INTENT(OUT) :: dudt

    REAL :: tq,tf
    REAL(PREC) :: tt
    INTEGER :: k,n
      tq=blitp(x,y,spc(NUM+1,:,:),a,b,ha,hb)
      tt=TSIZE/NSUBIN
      DO k=1,NSUBIN+1
        IF (t>=tt*(k-1)) n=k
      END DO
      n=MIN(n,NSUBIN)
      tf=blitp(x,y,spc(NUM+1+n,:,:),a,b,ha,hb)
      dudt(1) = (duxx(1,1)+duxy(1,1)+duyx(1,1)+duyy(1,1)+tf)/tq
  END SUBROUTINE f

!C:*********************************************
!C: BOUNDARY AND INTEGRAL FUNCTIONS : y1_2d,y2_2d,func
!C:*********************************************
```

```
FUNCTION y1_2d(x)
  USE ntype
  USE para, ONLY : h
  USE fl_compErr_mod, ONLY : r0
  REAL(PREC), INTENT(IN) :: x
  REAL(PREC) :: y1_2d
    y1_2d=r0(2)-sqrt(h*h-(x-r0(1))*(x-r0(1)))
END FUNCTION y1_2d

FUNCTION y2_2d(x)
  USE ntype
  USE para, ONLY : h
  USE fl_compErr_mod, ONLY : r0
  REAL(PREC), INTENT(IN) :: x
  REAL(PREC) :: y2_2d
    y2_2d=r0(2)+sqrt(h*h-(x-r0(1))*(x-r0(1)))
END FUNCTION y2_2d

FUNCTION func_2d(x,y)
  USE ntype
  USE para, ONLY : h,TSIZE,a,b,ha,hb,NUM=>FL_NUM,NSUBIN=>FL_NSUBIN
  USE util, ONLY : blitp
  USE fl_compErr_mod
  IMPLICIT NONE
  REAL(PREC), INTENT(IN) :: x
  REAL(PREC), DIMENSION(:), INTENT(IN) :: y
  REAL(PREC), DIMENSION(size(y)) :: func_2d

  INTEGER :: k,n
  REAL(PREC) :: rho
  REAL(PREC) :: r
    DO k=1,SIZE(y)
      r=((x-r0(1))*(x-r0(1))+(y(k)-r0(2))*(y(k)-r0(2)))/(h*h)
      rho=exp(1.0_prec/(r-1))
      func_2d(k)=rho*blitp(x,y(k),vec,a,b,ha,hb)/(h*h*const)
    END DO
END FUNCTION func_2d
```

APPENDIX D

Fortran code: Compute the inflow and outflow

```
PROGRAM sustain
  USE ntype
  USE para, ONLY : MGRID,NGRID,NSUBIN=>FL_NSUBIN, &
                   NUM=>FL_NUM,a,b,ha,hb,dx,dy,conv
  USE path, ONLY : datapath=>fl_datapath, home=>fl_home
  USE simpson, ONLY : quad2d

  IMPLICIT NONE
  REAL(PREC), DIMENSION(NUM+1+NSUBIN,MGRID,NGRID) :: spc
  REAL(PREC), DIMENSION(MGRID,NGRID) :: vec
  REAL(PREC) :: dd,dd2,a1,b1,ha1,hb1
  REAL(PREC), DIMENSION(NSUBIN) :: inflow, outflow
  INTEGER :: k

!C:*********************************
!C: READ IN DATA
!c:*********************************
    OPEN(UNIT=4,FILE=datapath,STATUS='OLD', &
            ACCESS='SEQUENTIAL',ACTION='READ')
      READ(4,*) k
      READ(4,*) spc
    CLOSE(4,STATUS='KEEP')

    a1=0.0
    b1=0.0
    ha1=dx*conv
    hb1=hy*conv
    spc(NUM+1:NUM+1+NSUBIN,:,:)=spc(NUM+1:NUM+1+NSUBIN,:,:)/(conv*conv)

    DO k=1, NSUBIN
      vec=max(spc(NUM+1+k,:,:),0.0_prec)
      CALL quad2d(vec,a1,b1,ha1,hb1,dd)
      PRINT *, ' INFLOW = ', dd*30
      inflow(k)=dd*30

      vec=min(spc(NUM+1+k,:,:),0.0_prec)
      CALL quad2d(vec,a1,b1,ha1,hb1,dd)
      PRINT *, ' OUTFLOW = ', -dd*30
      outflow(k)=-dd*30
    END DO

    dd=0.0_prec
    dd2=0.0_prec
    DO k=1, NSUBIN
      dd=dd+inflow(k)
      dd2=dd2+outflow(k)
```

```
    END DO

!C:***************
!C: plot the data
!C:***************
    OPEN (4, file=home//'flow/in_flow.dat')
      WRITE(4,*) 'TITLE=inflow:'
      WRITE(4,*) 'VARIABLES="T" "V"'
      WRITE(4,*) 'ZONE I=', NSUBIN, ', C=BLUE'
      DO k=1, NSUBIN
        WRITE(4,*) k, inflow(k)
      END DO
    CLOSE(4,STATUS='keep')

    OPEN (4, file=home//'flow/out_flow.dat')
      WRITE(4,*) 'TITLE=outflow:'
      WRITE(4,*) 'VARIABLES="T" "V"'
      WRITE(4,*) 'ZONE I=', NSUBIN, ', C=BLUE'
      DO k=1, NSUBIN
        WRITE(4,*) k, outflow(k)
      END DO
    CLOSE (4, STATUS='keep')
END PROGRAM sustain
```

APPENDIX E

Fortran code: subroutines

E.1. Fortran code: parameters

```
!######################################################################
!#                                                                    #
!# THIS MODULE DEFINES THE PARAMETERS USED IN THE flow AND solute     #
!# PROGRAMS. IT HAS THE FOLLOWING FORMAT:                             #
!#   IF THE PARAMETER IS USED ONLY BY THE flow PROGRAM, THE           #
!#   PARAMETER BEGINS WITH fl                                         #
!#                                                                    #
!#   IF THE PARAMETER IS USED ONLY BY THE solute EQUATION THE         #
!#   PARAMETER BEGINS WITH sl                                         #
!#                                                                    #
!#   OTHER WISE IT IS USED BY BOTH flow and solute equations          #
!#                                                                    #
!######################################################################



MODULE para
  USE ntype
!######################################################################
!#                                                                    #
!#  PARAMETERS CONTROLLING THE NUMBER OF SLAVE PROCESSES AND NUMBER   #
!#  OF lambdas                                                        #
!#                                                                    #
!######################################################################
  INTEGER, PARAMETER :: LMAX=20, N_PROC=20

!######################################################################
!#                                                                    #
!#  PARAMETERS OF GRID NUMBER, ITERATION LIMIT (NGRID>=MGRID). NOTE   #
!#  WE ONLY TEST THE SITUATION WHERE NGRID=MGRID. IT IS EXPECTED TO   #
!#  WORK FOR NGRID>MGRID IF THERE IS NO BUG THERE.                    #
!#                                                                    #
!######################################################################
  INTEGER, PARAMETER :: NGRID=30,MGRID=30, FL_ITERP=1520, SL_ITERP=100

!######################################################################
!#                                                                    #
!# PARAMETERS DEFINING THE DOMAIN \Omega                              #
!#                                                                    #
!######################################################################
  REAL(PREC), PARAMETER :: a=-1.0,b=-1.0, ha=2.0, hb=2.0

!######################################################################
!#                                                                    #
!# SMOOTHING PARAMETER, THIS PARAMETER IS USED FOR SMOOTHIFY THE      #
```

```
!# DATA USING THE MOLIFIER:                                              #
!#                                                                       #
!#   u_p=\int_{|x-y|<h} \rho(|x-y|^n)u(y)dy                              #
!#                                                                       #
!# WHERE THE MOLIFIER \rho is defined by                                 #
!#                                                                       #
!#   \rho(x) = C \int_{|x-r|^n<1} exp( 1/|x-r|^n )dr                     #
!# AND                                                                   #
!#########################################################################
  REAL(PREC), PARAMETER :: h=1.0e-1

!#########################################################################
!#                                                                       #
!#  PARAMETER DEFINING THE lambda value                                  #
!#                                                                       #
!#########################################################################
  REAL(PREC), PARAMETER :: lambda=1.0/LMAX

!#########################################################################
!#                                                                       #
!#  PARAMETER SETTING THE NUMBER OF SUB INTERVALS USED FOR COMPUTE       #
!#  SOURCE DATA IN PDE2 AND LAPLACE TRANSFORM                            #
!#                                                                       #
!#########################################################################
  INTEGER, PARAMETER :: NSTEPS=30

!#########################################################################
!#                                                                       #
!#  TOTAL LENGTH OF TIME ([0,1])                                         #
!#                                                                       #
!#########################################################################
  REAL(PREC), PARAMETER :: TSIZE=1.0

!#########################################################################
!#                                                                       #
!#  A SMALL NUMBER USED TO TEST IF THE GIVEN DIRECTION IS SEARCHABLE#
!#                                                                       #
!#########################################################################
  REAL(PREC),PARAMETER :: smstep=1.0e-7


!#########################################################################
!#                                                                       #
!#  PARAMETERS USED FOR FLOW EQUATION OR TRANSPORT EQUATION              #
!#                                                                       #
!#########################################################################
```

```
!#####################################################################
!#                                                                   #
!#  BOOLEAN PARAMETERS:                                              #
!#                                                                   #
!#  newsearch = .TRUE. : SEARCH WITH NEW DATA                        #
!#              .FALSE.: MAKE FURTHERE SEARCH USING                  #
!#                       USING THE SEARCHED DATA BEFORE              #
!#  realData = .TRUE. :  SEARCH WITH REAL DATA                       #
!#             .FALSE. : SEARCH WITH SYNTHETIC DATA                  #
!#  nplot = .TRUE.    :  PLOT THE TRUE PARAMETERS P Q R              #
!#                       (realData, SHOULD BE .FALSE.)               #
!#  compare = .TRUE. :   COMPARE THE DATA SOLVED USING               #
!#                       THE RECOVERED PARAMETERS AND                #
!#                       THE ORIGINAL DATA                           #
!#                                                                   #
!#####################################################################

  LOGICAL, PARAMETER :: FL_newSearch=.FALSE., FL_realData=.TRUE., &
                        FL_nplot=.FALSE., FL_compare=.FALSE., &
                        SL_newSearch=.TRUE., SL_realData=.TRUE., &
                        SL_nplot=.TRUE., SL_compare=.FALSE., &
                        FL_refinedata=.FALSE.,SL_refineData=.TRUE.

!#####################################################################
!#                                                                   #
!#  INTGER PARAMETERS:                                               #
!#                                                                   #
!#  nplotstep : AFTER nplotsteps INTERATION SAVE THE DATA IN CASE    #
!#              PROGRAMS WAS DONE ACCIDENTLY AND PLOT THE SEARCH     #
!#              DATA P, Q, R                                         #
!#  dim = 1   : ISOTROPIC P / OR D                                   #
!#        2   : ANISOTROPIC P / OR D                                 #
!#                                                                   #
!#####################################################################
  INTEGER, PARAMETER :: FL_nplotstep=100, SL_nplotstep=1000, &
                        FL_dim=2, FL_num=FL_dim*(FL_dim+1)/2, &
                        SL_dim=2, SL_num=SL_dim*(SL_dim+1)/2, &
                        FL_nnplot=10, SL_nnplot=10

!#####################################################################
!#                                                                   #
!#  INTEGER PARAMETERS:                                              #
!#                                                                   #
!#   FL_NSUBIN : NUMBER OF SUBINTERVALS                              #
!#               IN R DECOMPOSITION                                  #
```

```
!#   SL_NSUBIN : NUMBER OF SUBINTERVALS                                  #
!#               IN D DECOMPOSITION                                      #
!#   LOOPS     : SIZE IS DEPEND ON DIM. IN EACH ITERATE STEP,            #
!#               THE i'S PARAMETER WILL BE SEEARCHED LOOPS(I) TIMES      #
!#               NOTE: IN REAL SITUATION, THE SIZE OF THE ARRAY          #
!#               MUST BE SET CORRECTLY                                   #
!#                                                                       #
!#########################################################################
  INTEGER, PARAMETER :: FL_NSUBIN=12, SL_NSUBIN=20
  INTEGER, DIMENSION(FL_NUM+2), PARAMETER :: &
!# IF FL_DIM=2
   FL_LOOPS=(/1,1,1,1,1/)

!# IF FL_DIM=1
!    FL_LOOPS=(/1,1,1/)

  INTEGER, DIMENSION(SL_NUM+3), PARAMETER :: &
!# IF SL_DIM=2
   SL_LOOPS=(/1,0,0,0,0,0/)

!# IF FL_DIM=1
!   SL_LOOPS=(/1,0,0,0/)

!#########################################################################
!#                                                                       #
!#  PARAMETERS THAT SET THE INITIAL VALUE, THE LOWER AND UPPER           #
!#  BOUND, UPPER BOUND OF THE RECOVERING PARAMETERS.                     #
!#  NOTE: IN REAL SITUATIONS THE SIZE OF THE ARRAY MUST BE SET           #
!#  CORRECTLY                                                            #
!#                                                                       #
!#########################################################################
  LOGICAL, PARAMETER :: FL_NbgQ=.FALSE., FL_NbgR=.FALSE.

  REAL(PREC), PARAMETER :: dx=2.0120, dy=1.4510, conv=1069.344, &
  REAL(PREC), DIMENSION(FL_NUM+2), PARAMETER :: &
! IF FL_DIM=2
    FL_lbd=(/45.0/(dx*dx)*4.0, -44.9/(dx*dy)*4.0,  45.0/(dy*dy)*4.0,
             2.7d-4*conv*conv, -1.0e-2*conv*conv/), &
    FL_pc0=(/0.45, 0.0,  45.0, 2.7d-4*conv*conv, -1.0e-2*conv*conv/), &
    FL_ubd=(/5560.0/(dx*dx)*4.0, 44.9/(dx*dy)*4.0, 5560.0/(dy*dy)*4.0, &
             1.1e-2*conv*conv,   1.0e-2*conv*conv/)

! IF FL_DIM=1
!    FL_pc0=(/0.5,0.0005,0.5/), &
!    FL_lbd=(/0.5,0.0005,0.5/), &
!    FL_ubd=(/5.0,0.0015,5.0/)
```

```
  REAL(PREC), DIMENSION(SL_NUM+3), PARAMETER :: &
! IF SL_DIM=2
    SL_pc0=(/0.5,0.0,0.5,0.02,0.5,0.5/), &
    SL_lbd=(/0.5,0.0,0.5,0.02,0.5,0.5/), &
    SL_ubd=(/5.0,0.49,5.0,1.0,5.0,5.0/)

! IF SL_DIM=1
!    SL_pc0=(/0.5,0.02,0.5,0.5/), &
!    SL_lbd=(/0.5,0.02,0.5,0.5/), &
!    SL_ubd=(/5.0,1.0,5.0,5.0/)


!#########################################################################
!# PARAMETERS THAT IS USED TO SMOOTH THE                                 #
!# SOURCE DATA phi                                                       #
!#                                                                       #
!# SL_SMOOTH = .TRUE. SMOOTH DATA                                        #
!#             .FALSE. DO NOT SMOOTH                                     #
!# SL_PHI_DERV = .TRUE. COMPUTE THE DERIVATIVES OF PHI                   #
!#               .FALSE. DO NOT COMPUTE THE DERIVATIVE                   #
!# SL_RAW = .TRUE. USE THE RAW SOURCE (UNSMOOTHED)                       #
!#                 DATA PHI TO GENERATE DATA C                           #
!#          .FALSE. USE THE SMOOTHED PHI TO GENERATE C                   #
!#                                                                       #
!#########################################################################
  LOGICAL, PARAMETER :: SL_SMOOTH = .FALSE., SL_PHI_DERV = .TRUE., &
                        SL_RAW = .FALSE.

!#########################################################################
!# PARAMETER CONTROL THE BOUNDARY CONDITION                              #
!# IF fl_no_bndry_value=.TRUE.,                                          #
!#                   THERE ARE NO BOUNDARY VALUES, WE THEN               #
!#                   PROPAGATE THE INTERIOR VALUES TO THE                #
!#                   BOUNDARY IN EVERY STEP OF DESCENT SEARCH            #
!#                                                                       #
!#########################################################################
  LOGICAL, PARAMETER :: FL_NO_BNDRY_VALUE=.TEUE.


END MODULE para
```

E.2. Fortran code: elliptic PDE solver

```
MODULE ellsov
  USE ntype
  IMPLICIT NONE
  PRIVATE
  PUBLIC Elliptic_Solver

  REAL(PREC), DIMENSION(:,:,:), POINTER :: ptr_p
  REAL(PREC), DIMENSION(:,:), POINTER :: ptr_q, ptr_f,ptr_bndry
  REAL(PREC) :: hx,hy
  INTEGER :: mgrid,ngrid
  INTEGER, PARAMETER :: ROW=10, COL=20
  INTERFACE Elliptic_Solver
    MODULE PROCEDURE Elliptic_Solver_0
  END INTERFACE
  INTERFACE getRow
    MODULE PROCEDURE getRow_1, getRow_2
  END INTERFACE
  INTERFACE getCol
    MODULE PROCEDURE getCol_1, getCol_2
  END INTERFACE

  CONTAINS
    SUBROUTINE Elliptic_Solver_0(pvec,qvec,fvec,bndryvec,ha,hb,u)
      USE ntype
      USE util
      IMPLICIT NONE
      REAL(PREC), DIMENSION(:,:,:),TARGET,INTENT(IN) :: pvec
      REAL(PREC), DIMENSION(:,:),TARGET,INTENT(IN) :: qvec,fvec,&
                                                      bndryvec
      REAL(PREC), INTENT(IN) :: ha,hb
      REAL(PREC), DIMENSION(:,:), INTENT(OUT) :: u

!C: LOCAL VARIABLES
      REAL(PREC), DIMENSION(:,:), POINTER :: m_a, m_b
      REAL(PREC), DIMENSION(:), POINTER :: m_u
      INTEGER, DIMENSION((SIZE(u,1)-2)*(SIZE(u,2)-2)):: indx
      REAL(PREC) :: dump, tmp,delta
      INTEGER :: j,m,n,i,k
        ptr_p=>pvec
        ptr_q=>qvec
        ptr_f=>fvec
        ptr_bndry=>bndryvec
        mgrid=SIZE(u,1)
        ngrid=SIZE(u,2)
        m=mgrid-2
```

```
        n=ngrid-2
        hx = ha/(m+1)
        hy = hb/(n+1)
        delta=4.0_prec*hx*hy

!C:*****************************************
!C: COMPUTE the parameters
!C: OF THE DIFFERENCE EQUATION
!C:
!C:  lower_left*U(i-1,j-1) + lower_diag*U(i,j-1) &
!C:  + lower_right*U(i+1,j-1) + middle_left*U(i-1,j) &
!C:  + middle_diag*U(i,j) + middle_right*U(i+1,j) &
!C:  + upper_left*U(i-1,j+1) + upper_diag*U(i,j+1)  &
!C:  + upper_right*U(i+1,j+1)  = rhs
!C: AND PACKED THEM TO THE MATRIX m_a
!C:*****************************************
!C: COMPUTE rhs, SAVE IT IN u
!C:**********************************
        u=0.0_prec
        u(1,1:n+2) = getRow(pb, 1, 1, n+2)
        u(2:m+1,1) = getCol(pb, 1, 2, m+1)
        DO j=2, n+1
          u(2:m+1,j) = getCol(pf, j, 2, m+1)
        END DO
        u(2:m+1, n+2) = getCol(pb, n+2, 2,m+1)
        u(m+2,1:n+2) = getRow(pb,m+2, 1,n+2)

        u(2,2:n+1) = u(2,2:n+1)-lower_left(ROW,2,2,n+1)*u(1,1:n) &
            -middle_left(ROW,2,2,n+1)*u(1,2:n+1) &
            -upper_left(ROW,2,2,n+1)*u(1,3:n+2)
        u(m+1,2:n+1)= &
            u(m+1,2:n+1)-lower_right(ROW,m+1,2,n+1)*u(m+2,1:n) &
            -middle_right(ROW,m+1,2,n+1)*u(m+2,2:n+1) &
            -upper_right(ROW,m+1,2,n+1)*u(m+2,3:n+2)
        u(2:m+1,2)=u(2:m+1,2)-lower_left(COL,2,2,m+1)*u(1:m,1) &
          -lower_diag(COL,2,2,m+1)*u(2:m+1,1) &
          -lower_right(COL,2,2,m+1)*u(3:m+2,1)
        u(2:m+1,n+1)=&
          u(2:m+1,n+1)-upper_left(COL,n+1,2,m+1)*u(1:m,n+2) &
          -upper_diag(COL,n+1,2,m+1)*u(2:m+1,n+2) &
          -upper_right(COL,n+1,2,m+1)*u(3:m+2,n+2)
        tmp = - (( pp(2,2,1)+pp(2,1,2))*0.5_prec + pp(2,2,2))/delta
        u(2,2) = u(2,2) +tmp*u(1,1)
        tmp = ((pp(2,2,n+2)+pp(2,1,n+1))*0.5_prec + pp(2,2,n+1))/delta
        u(2,n+1)=u(2,n+1)+tmp*u(1,n+2)
        tmp = ((pp(2,m+1,1)+pp(2,m+2,2))*0.5_prec + pp(2,m+1,2))/delta
```

```
        u(m+1,2)=u(m+1,2)+tmp*u(m+2,1)
        tmp =- ((pp(2,m+1,n+2) + pp(2,m+2,n+1))*0.5_prec &
            + pp(2,m+1,n+1)) / delta
        u(m+1,n+1)=u(m+1,n+1)+tmp*u(m+2,n+2)

!C:***********************************************
!C: PACK the parameters IN m_a
!C:***********************************************
        m_a => createArray(m*n, 2*(m+1)+1, 'Elliptic_Solver -- m_a')
        m_a=0.0_prec
        DO j=2, n
          m_a((j-1)*m+2:j*m,1)=lower_left(COL,j+1,3,m+1)
          m_a((j-1)*m+1:j*m,2)=lower_diag(COL,j+1,2,m+1)
          m_a((j-1)*m+1:j*m-1,3)=lower_right(COL,j+1,2,m)
        END DO
        DO j=1, n
          m_a((j-1)*m+2:j*m,m+1)=middle_left(COL,j+1,3,m+1)
          m_a((j-1)*m+1:j*m,m+2)=middle_diag(COL,j+1,2,m+1)
          m_a((j-1)*m+1:j*m-1,m+3)=middle_right(COL,j+1,2,m)
        END DO
        DO j=1, n-1
          m_a((j-1)*m+2:j*m,2*m+1)=upper_left(COL,j+1,3,m+1)
          m_a((j-1)*m+1:j*m,2*m+2)=upper_diag(COL,j+1,2,m+1)
          m_a((j-1)*m+1:j*m-1,2*m+3)=upper_right(COL,j+1,2,m)
        END DO
!C:*******************************************
!C: CALL SUBROUTINE bandec FOR LU DECOMPOSITION
!C:*******************************************
        m_b => createArray(m*n, m+1, 'Elliptic_Solver -- m_b')
        call bandec(m_a,m+1,m+1,m_b,indx,dump)

!C:***********************************************
!C: ADJUST u FOR BACKWARD AND FORWARD SUBSTITUTION
!C:***********************************************
        m_u => createArray(m*n, 'Elliptic_Solver -- m_u')
        DO j=1, n
          m_u((j-1)*m+1:j*m)=u(2:m+1,j+1)
        END DO
        call banbks(m_a,m+1,m+1,m_b,indx,m_u)

!C:************************
!C: ADJUSTBACK THE SOLUTION
!C:************************
        DO j=1, n
          u(2:m+1,j+1)=m_u((j-1)*m+1:j*m)
        END DO
```

```
        CALL RELEASE_MEMORY(m_a, 'Elliptic_Solver -- m_a')
        CALL RELEASE_MEMORY(m_b, 'Elliptic_Solver -- m_b')
        CALL RELEASE_MEMORY(m_u, 'Elliptic_Solver -- m_u')

    END SUBROUTINE Elliptic_Solver_0
!c:*********************************************
    SUBROUTINE bandec(a,m1,m2,al,indx,d)
      USE ntype
      use util
      IMPLICIT NONE
      REAL(PREC),DIMENSION(:,:),INTENT(INOUT)::a
      INTEGER,INTENT(IN)::m1,m2
      REAL(PREC),DIMENSION(:,:),INTENT(OUT)::al
      INTEGER,DIMENSION(:),INTENT(OUT)::indx
      REAL(PREC),INTENT(OUT)::d
      REAL(PREC),PARAMETER ::TINY=1.0e-20_prec

      INTEGER::i,k,l,mdum,mm,n,ii
      REAL(PREC)::dum
      REAL(PREC), DIMENSION(m1+m2+1) :: temp

        n=SIZE(a,1)
        mm=m1+m2+1
        mdum=m1

        a(1:m1,:)=eoshift(a(1:m1,:),shift=arth(m1,-1,m1),dim=2)
        d=1.0
        do k=1,n
          l=min(m1+k,n)
          i=imaxloc(abs(a(k:l,1)))+k-1
          dum=a(i,1)
          if (dum ==0.0)a(k,1)=TINY

          indx(k)=i
          if (i /=k)then
            d=-d
            temp(1:mm)=a(k,1:mm)
            a(k,1:mm)=a(i,1:mm)
            a(i,1:mm)=temp(1:mm)
          end if
          do i=k+1,l
            dum=a(i,1)/a(k,1)
            al(k,i-k)=dum
            a(i,1:mm-1)=a(i,2:mm)-dum*a(k,2:mm)
            a(i,mm)=0.0
          end do
```

```
        end do
    END SUBROUTINE bandec
!C:*****************************************************
    SUBROUTINE banbks(a,m1,m2,al,indx,b)
      USE ntype
      USE util
      IMPLICIT NONE
      REAL(PREC),DIMENSION(:,:),INTENT(IN)::a,al
      INTEGER,INTENT(IN)::m1,m2
      INTEGER,DIMENSION(:),INTENT(IN)::indx
      REAL(PREC),DIMENSION(:),INTENT(INOUT)::b

      INTEGER::i,k,l,mdum,mm,n,ii
      REAL(PREC) :: temp
        n=SIZE(a,1)
        mm=m1+m2+1
        mdum=m1
        do k=1,n
          l=min(n,m1+k)
          i=indx(k)
          if (i /=k) then
            temp=b(i)
            b(i)=b(k)
            b(k)=temp
          end if
          b(k+1:l)=b(k+1:l)-al(k,1:l-k)*b(k)
        end do
        do i=n,1,-1
          l=min(mm,n-i+1)
          b(i)=(b(i)-dot_product(a(i,2:l),b(1+i:i+l-1)))/a(i,1)
        end do
    END SUBROUTINE banbks
!C:*****************************************************
    FUNCTION imaxloc(array)
      USE ntype
      IMPLICIT NONE
      REAL(PREC), DIMENSION(:), INTENT(IN) :: array

      INTEGER :: imaxloc
      INTEGER, DIMENSION(1) :: imax
        imax=maxloc(array(:))
        imaxloc=imax(1)
    END FUNCTION
!C:*********************************
    FUNCTION getRow_1(func, i, bg, ed)
      USE ntype
```

```
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: i, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: getRow_1
      INTERFACE
        FUNCTION func(a,b)
        USE ntype
        IMPLICIT NONE
        INTEGER, INTENT(IN) :: a, b
        REAL(PREC) :: func
        END FUNCTION
      END INTERFACE

      INTEGER :: k
        if (ed <= bg) call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'getRow_1')
        DO k=bg, ed
        getRow_1(k)=func(i,k)
        END DO
    END FUNCTION getRow_1

    FUNCTION getRow_2(func, k, i, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: k, i, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: getRow_2
      INTERFACE
        FUNCTION func(a,b,c)
        USE ntype
        IMPLICIT NONE
        INTEGER, INTENT(IN) :: a, b, c
        REAL(PREC) :: func
        END FUNCTION
      END INTERFACE

      INTEGER :: m
        if (ed <= bg) call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'getRow_2')
        getRow_2 = (/(func(k,i,m), m=bg,ed)/)
    END FUNCTION getRow_2
!C:************************************
    FUNCTION getCol_1(func, j, bg, ed)
      USE ntype
```

```
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: j, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: getCol_1
      INTERFACE
        FUNCTION func(a,b)
        USE ntype
        IMPLICIT NONE
        INTEGER, INTENT(IN) :: a, b
        REAL(PREC) :: func
        END FUNCTION
      END INTERFACE

      INTEGER :: k
        if (ed <= bg) call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'getCol_1')
        getCol_1 = (/(func(k,j), k=bg,ed)/)
    END FUNCTION getCol_1

    FUNCTION getCol_2(func, k, i, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: k, i, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: getCol_2
      INTERFACE
        FUNCTION func(a,b,c)
        USE ntype
        IMPLICIT NONE
        INTEGER, INTENT(IN) :: a, b, c
        REAL(PREC) :: func
        END FUNCTION
      END INTERFACE

      INTEGER :: m
        if (ed <= bg) call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'getCol_2')
        getCol_2 = (/(func(k,m,i), m=bg,ed)/)
    END FUNCTION getCol_2
!C:***********************************
    FUNCTION lower_left(flag, k, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
```

```
      INTEGER, INTENT(IN) :: flag, k, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: lower_left
        if(ed <= bg)call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'lower_left')
        SELECT CASE (flag)
        CASE (ROW)
          lower_left = - (getRow(pp,2,k,bg-1,ed-1)*0.5_prec &
                + getRow(pp,2,k-1,bg,ed)*0.5_prec &
                + getRow(pp,2,k,bg,ed))/(4.0_prec*hx*hy)
        CASE (COL)
          lower_left = - (getCol(pp,2,k-1,bg,ed)*0.5_prec &
                + getCol(pp,2,k,bg-1,ed-1)*0.5_prec &
                + getCol(pp,2,k,bg,ed))/(4.0_prec*hx*hy)
        END SELECT
    END FUNCTION lower_left
!C:************************************
    FUNCTION lower_diag(flag, k, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: flag, k, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: lower_diag
        if(ed <= bg)call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'lower_diag')
        SELECT CASE (flag)
        CASE (ROW)
          lower_diag = (getRow(pp,2,k+1,bg,ed) &
                - getRow(pp,2,k-1,bg,ed))/(4.0_prec*hx*hy) &
              - (getRow(pp,3,k,bg,ed) &
                + getRow(pp,3,k,bg-1,ed-1))/hy**2
        CASE (COL)
          lower_diag = (getCol(pp,2,k,bg+1,ed+1) &
              - getCol(pp,2,k,bg-1,ed-1))/(4.0_prec*hx*hy) &
              - (getCol(pp,3,k,bg,ed) &
              + getCol(pp,3,k-1,bg,ed))/hy**2
        END SELECT
        lower_diag=lower_diag*0.5_prec
    END FUNCTION lower_diag
!C:************************************
    FUNCTION lower_right(flag, k, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: flag, k, bg, ed
```

```
      REAL(PREC), DIMENSION(bg:ed) :: lower_right
        if(ed <= bg)call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'lower_right')
        SELECT CASE (flag)
        CASE (ROW)
          lower_right = (getRow(pp,2,k,bg-1,ed-1)*0.5_prec &
                + getRow(pp,2,k+1,bg,ed)*0.5_prec &
                + getRow(pp,2,k,bg,ed))/(4.0_prec*hx*hy)
        CASE (COL)
          lower_right = ( getCol(pp,2,k-1,bg,ed)*0.5_prec &
                + getCol(pp,2,k,bg+1,ed+1)*0.5_prec &
                + getCol(pp,2,k,bg,ed))/(4.0_prec*hx*hy)
        END SELECT
    END FUNCTION lower_right
!C:************************************
    FUNCTION upper_left(flag, k, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: flag, k, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: upper_left
        if(ed <= bg)call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'upper_left')
        SELECT CASE (flag)
        CASE (ROW)
          upper_left = ( getRow(pp,2,k,bg+1,ed+1)*0.5_prec &
                + getRow(pp,2,k-1,bg,ed)*0.5_prec &
                + getRow(pp,2,k,bg,ed))/(4.0_prec*hx*hy)
        CASE (COL)
          upper_left = ( getCol(pp,2,k+1,bg,ed)*0.5_prec &
              + getCol(pp,2,k,bg-1,ed-1)*0.5_prec &
              + getCol(pp,2,k,bg,ed))/(4.0_prec*hx*hy)
        END SELECT
    END FUNCTION upper_left
!C:************************************
    FUNCTION upper_diag(flag, k, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: flag, k, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: upper_diag
        if(ed <= bg)call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'upper_diag')
```

```
        SELECT CASE (flag)
        CASE (ROW)
          upper_diag = (getRow(pp,2,k-1,bg,ed) &
                - getRow(pp,2,k+1,bg,ed))/(4.0_prec*hx*hy) &
                - ( getRow(pp,3,k,bg,ed) &
                +getRow(pp,3,k,bg+1,ed+1))/hy**2
        CASE (COL)
          upper_diag = ( getCol(pp,2,k,bg-1,ed-1) &
                - getCol(pp,2,k,bg+1,ed+1))/(4.0_prec*hx*hy) &
                - ( getCol(pp,3,k,bg,ed) &
                + getCol(pp,3,k+1,bg,ed))/hy**2
        END SELECT
        upper_diag=upper_diag*0.5_prec
    END FUNCTION upper_diag
!C:************************************
    FUNCTION upper_right(flag, k, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: flag, k, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: upper_right
        if(ed <= bg)call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'upper_right')
        SELECT CASE (flag)
        CASE (ROW)
          upper_right = - ( getRow(pp,2,k,bg+1,ed+1)*0.5_prec &
                + getRow(pp,2,k+1,bg,ed)*0.5_prec &
                + getRow(pp,2,k,bg,ed))/(4.0_prec*hx*hy)
        CASE (COL)
          upper_right = - ( getCol(pp,2,k+1,bg,ed)*0.5_prec &
                + getCol(pp,2,k,bg+1,ed+1)*0.5_prec &
                + getCol(pp,2,k,bg,ed))/(4.0_prec*hx*hy)
        END SELECT
    END FUNCTION upper_right
!C:************************************
    FUNCTION middle_left(flag, k, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: flag, k, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: middle_left
        if(ed <= bg)call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'middle_left')
        SELECT CASE (flag)
```

```
          CASE (ROW)
            middle_left = ( getRow(pp,2,k,bg+1,ed+1) &
                  - getRow(pp,2,k,bg-1,ed-1))/(4.0_prec*hx*hy) &
                  - ( getRow(pp,1,k,bg,ed) &
                  + getRow(pp,1,k-1,bg,ed))/hx**2
          CASE (COL)
            middle_left = ( getCol(pp,2,k+1,bg,ed) &
                  - getCol(pp,2,k-1,bg,ed))/(4.0_prec*hx*hy) &
                - ( getCol(pp,1,k,bg,ed) &
                  + getCol(pp,1,k,bg-1,ed-1))/hx**2
        END SELECT
        middle_left=middle_left*0.5_prec
    END FUNCTION middle_left
!C:************************************
    FUNCTION middle_diag(flag, k, bg, ed)
      USE ntype
      USE util
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: flag, k, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: middle_diag
        if(ed <= bg)call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'middle_diag')
        SELECT CASE (flag)
          CASE (ROW)
            middle_diag = getRow(pq,k,bg,ed) &
                + ( getRow(pp,1,k+1,bg,ed)*0.5_prec &
                  + getRow(pp,1, k-1,bg,ed)*0.5_prec &
                  + getRow(pp,1,k,bg,ed))/hx**2 &
                + ( getRow(pp,3,k,bg+1,ed+1)*0.5_prec &
                  + getRow(pp,3,k,bg-1,ed-1)*0.5_prec &
                  + getRow(pp,3,k,bg,ed))/hy**2
          CASE (COL)
            middle_diag = getCol(pq,k,bg,ed) &
                + ( getCol(pp,1,k,bg+1,ed+1)*0.5_prec &
                  + getCol(pp,1,k,bg-1,ed-1)*0.5_prec &
                  + getCol(pp,1,k,bg,ed))/ hx**2 &
                + ( getCol(pp,3,k+1,bg,ed)*0.5_prec &
                  + getCol(pp,3,k-1,bg,ed)*0.5_prec &
                  + getCol(pp,3,k,bg,ed))/hy**2
        END SELECT
    END FUNCTION middle_diag
!C:************************************
    FUNCTION middle_right(flag, k, bg, ed)
      USE ntype
      USE util
```

```
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: flag, k, bg, ed
      REAL(PREC), DIMENSION(bg:ed) :: middle_right
        if(ed <= bg)call Error &
         ('The ending point should be bigger than &
           the beginning point in :', 'middle_right')
        SELECT CASE (flag)
          CASE (ROW)
            middle_right = ( getRow(pp,2,k,bg-1,ed-1) &
                  -getRow(pp,2,k,bg+1,ed+1))/(4.0_prec*hx*hy) &
                - (getRow(pp,1,k,bg,ed) &
                  + getRow(pp,1,k+1,bg,ed))/hx**2
          CASE (COL)
            middle_right = ( getCol(pp,2,k-1,bg,ed) &
                  - getCol(pp,2,k+1,bg,ed))/(4.0_prec*hx*hy) &
                  - ( getCol(pp,1,k,bg,ed) &
                  + getCol(pp,1,k,bg+1,ed+1))/hx**2
        END SELECT
        middle_right=middle_right*0.5_prec
    END FUNCTION middle_right
!C:************************************
    FUNCTION pp(k,i,j)
      USE ntype
      IMPLICIT NONE
        INTEGER, INTENT(IN) :: k,i,j
        REAL(PREC) :: pp
          pp=ptr_p(k,i,j)
    END FUNCTION pp
!C:************************************
    FUNCTION pq(i,j)
      USE ntype
      IMPLICIT NONE
        INTEGER, INTENT(IN) :: i,j
        REAL(PREC) :: pq
          pq=ptr_q(i,j)
    END FUNCTION pq
!C:************************************
    FUNCTION pf(i,j)
      USE ntype
      IMPLICIT NONE
        INTEGER, INTENT(IN) :: i,j
        REAL(PREC) :: pf
          pf=ptr_f(i,j)
    END FUNCTION pf
!C:************************************
    FUNCTION pb(i,j)
```

```
      USE ntype
      USE util
      IMPLICIT NONE
        INTEGER, INTENT(IN) :: i,j
        REAL(PREC) :: pb
          IF (i == 1) THEN
            pb=ptr_bndry(1,j)
          ELSE IF (i == mgrid) THEN
            pb=ptr_bndry(2,j)
          ELSE IF (j == 1) THEN
            pb=ptr_bndry(3,i)
          ELSE IF (j == NGRID) THEN
            pb=ptr_bndry(4,i)
          ELSE
            CALL Error('In function pb', 'Wrong parameter i or j')
          END IF
    END FUNCTION pb
!C:************************************
END MODULE ellsov
```

E.3. Fortran code: subroutine of quadratic interpolation

```
MODULE quad2d
  USE ntype
  IMPLICIT NONE

  PRIVATE
  PUBLIC quad2d_qgaus
  REAL(PREC) :: xsav,ysav
  INTERFACE
    FUNCTION func_2d(x,y)
      USE ntype
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) :: x
      REAL(PREC), DIMENSION(:), INTENT(IN) :: y
      REAL(PREC), DIMENSION(size(y)) :: func_2d
    END FUNCTION func_2d

    FUNCTION y1_2d(x)
      USE ntype
      REAL(PREC), INTENT(IN) :: x
      REAL(PREC) :: y1_2d
    END FUNCTION y1_2d

    FUNCTION y2_2d(x)
      USE ntype
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) :: x
      REAL(PREC) :: y2_2d
    END FUNCTION y2_2d
  END INTERFACE

  CONTAINS
    FUNCTION h(x)
      IMPLICIT NONE
      REAL(PREC), DIMENSION(:), INTENT(IN) :: x
      REAL(PREC), DIMENSION(size(x)) :: h
      INTEGER :: i
        do i=1,size(x)
          xsav=x(i)
          h(i)=qgaus(f,y1_2d(xsav),y2_2d(xsav))
        end do
    END FUNCTION h

    FUNCTION f(y)
      IMPLICIT NONE
```

```
      REAL(PREC), DIMENSION(:), INTENT(IN) :: y
      REAL(PREC), DIMENSION(size(y)) :: f
      integer :: k
        do k=1,size(y)
        end do
        f=func_2d(xsav,y)
    END FUNCTION f

    RECURSIVE FUNCTION qgaus(func,a,b)
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) :: a,b
      REAL(PREC) :: qgaus

      INTERFACE
        FUNCTION func(x)
          USE ntype
          IMPLICIT NONE
          REAL(PREC), DIMENSION(:), INTENT(IN) :: x
          REAL(PREC), DIMENSION(size(x)) :: func
        END FUNCTION func
      END INTERFACE
      REAL(PREC) :: xm,xr
      REAL(PREC), DIMENSION(5) :: dx, &
            w = (/ 0.2955242247_prec,0.2692667193_prec,&
                   0.2190863625_prec,0.1494513491_prec,&
                   0.0666713443_prec /),&
            x = (/ 0.1488743389_prec,0.4333953941_prec,&
                   0.6794095682_prec,0.8650633666_prec,&
                   0.9739065285_prec /)
        xm=0.5_prec*(b+a)
        xr=0.5_prec*(b-a)
        dx(:)=xr*x(:)
        qgaus=xr*sum(w(:)*(func(xm+dx)+func(xm-dx)))
    END FUNCTION qgaus

    SUBROUTINE quad2d_qgaus(x1,x2,ss)
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) :: x1,x2
      REAL(PREC), INTENT(OUT) :: ss
        ss=qgaus(h,x1,x2)
    END SUBROUTINE quad2d_qgaus
END MODULE quad2d
```

E.4. Fortran code: subroutine of Simpson's rule

```
MODULE simpson
  USE ntype
  IMPLICIT NONE
  PRIVATE
  PUBLIC quad2d
  INTEGER :: mgrid, ngrid
  REAL(PREC) :: xsav,ysav
  REAL(PREC) :: xmin, xmax, ymin, ymax
  REAL(PREC), DIMENSION(:,:), POINTER :: pf

  CONTAINS
!C:*************************
    FUNCTION h(x)
      REAL(PREC), INTENT(IN)::x
      REAL(PREC)::h

      REAL(PREC) :: sum
      INTEGER :: mstep
        mstep=mgrid-1
        xsav=x
        CALL qsimpy(g, mstep, y1(xsav), y2(xsav), sum)
        h=sum
    END FUNCTION h

!C:*************************

    FUNCTION y1(x)
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) :: x
      REAL(PREC) :: y1
        y1=ymin
    END FUNCTION y1
!c:***********************************

    FUNCTION y2(x)
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) :: x
      REAL(PREC) :: y2
        y2=ymax
    END FUNCTION y2
!c:***********************************
    FUNCTION g(y)
      USE util, ONLY : blitp
      IMPLICIT NONE
      REAL(PREC),INTENT(IN) :: y
```

```
      REAL(PREC) :: g

        ysav=y
        g=blitp(xsav,ysav,pf,xmin,ymin,xmax-xmin,ymax-ymin)
    END FUNCTION g

!c:**********************************
    SUBROUTINE qsimpx(func,nstep,a,b,sum)
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) :: a, b
      REAL(PREC), INTENT(OUT) :: sum
      INTEGER, INTENT(INOUT) :: nstep
      INTERFACE
      FUNCTION func(x)
        USE ntype
        IMPLICIT NONE
        REAL(PREC), INTENT(IN) :: x
        REAL(PREC) :: func
      END FUNCTION
      END INTERFACE

      INTEGER :: i
      REAL(PREC) :: h

        if (MOD(nstep,2)/=0) nstep = nstep+1
        h=(b-a)/nstep
        sum=func(a)+func(b)
        DO i=2,nstep,2
          sum=sum+4.0_prec*func(a+h*(i-1))
        END DO
        DO i=3,nstep-1,2
          sum=sum+2.0_prec*func(a+h*(i-1))
        END DO
        sum=sum*h/3.0_prec
    END SUBROUTINE qsimpx

!C:*************************
    SUBROUTINE qsimpy(func,nstep,a,b,sum)
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) :: a, b
      REAL(PREC), INTENT(OUT) :: sum
      INTEGER, INTENT(INOUT) :: nstep
      INTERFACE
      FUNCTION func(x)
        USE ntype
        IMPLICIT NONE
```

```
        REAL(PREC), INTENT(IN) :: x
        REAL(PREC) :: func
      END FUNCTION
      END INTERFACE

      INTEGER :: i
      REAL(PREC) :: h

        if(MOD(nstep,2)/=0) nstep=nstep+1
        h=(b-a)/nstep
        sum=func(a)+func(b)
        DO i=2,nstep,2
          sum=sum+4.0_prec*func(a+h*(i-1))
        END DO
        DO i=3,nstep-1,2
          sum=sum+2.0_prec*func(a+h*(i-1))
        END DO
        sum=sum*h/3.0_prec
    END SUBROUTINE qsimpy

!c::**********************************
    SUBROUTINE quad2d(func,a,b,ha, hb, sum)
      IMPLICIT NONE
      REAL(PREC) :: a,b,ha,hb,sum
      REAL(PREC), DIMENSION(:,:), INTENT(IN), TARGET :: func
      INTEGER :: nstep
        mgrid=SIZE(func,1)
        ngrid=SIZE(func,2)
        xmin=a
        xmax=a+ha
        ymin=b
        ymax=b+hb
        pf=>func
        nstep=ngrid-1
        CALL qsimpx(h,nstep,xmin,xmax,sum)

    END SUBROUTINE quad2d
!c::**********************************
END MODULE simpson
```

E.5. Fortran code: subroutine of utility functions

```
MODULE util
  USE ntype
  INTERFACE error
    MODULE PROCEDURE Error1
  END INTERFACE
  INTERFACE ASSERT_EQ
    MODULE PROCEDURE ASSERT_EQ_2,ASSERT_EQ_3,ASSERT_EQ_4,ASSERT_EQ_5,&
        ASSERT_EQ_6,ASSERT_EQ_7
  END INTERFACE
  INTERFACE createArray
    MODULE PROCEDURE createArray_1,createArray_2,createArray_3, &
                     createArray_4, createArray_5, createArray_6
  END INTERFACE
  INTERFACE RELEASE_MEMORY
    MODULE PROCEDURE RELEASE_MEMORY_1,RELEASE_MEMORY_2,RELEASE_MEMORY_3
  END INTERFACE
  INTERFACE outer_prod
    MODULE PROCEDURE outer_prod1
  END INTERFACE
  INTERFACE diagnal_assign
    MODULE PROCEDURE diagnal_assign1
  END INTERFACE

  INTERFACE show
    MODULE PROCEDURE show_1,show_2,show_3,show_4,show_5,show_6,show_7
  END INTERFACE

  INTERFACE arth
    MODULE PROCEDURE arth1,arth_d
  END INTERFACE

  INTERFACE locate
    MODULE PROCEDURE locate1,locate2
  END INTERFACE

  INTERFACE toString
    MODULE PROCEDURE toString1
  END INTERFACE

  INTERFACE blitp
    MODULE PROCEDURE blitp1, blitp2, blitp3,blitp4,blitp5,blitp6,blitp7
  END INTERFACE

  INTERFACE plotting
```

```
  MODULE PROCEDURE plot_vec1, plot_vec2, plot_func, plot_real, &
                   plot1,plot2,plot3,plot4,plot_1d
END INTERFACE

INTERFACE containing
  MODULE PROCEDURE contains1
END INTERFACE

INTERFACE computeError
  MODULE PROCEDURE computeError_1, computeError_2,  computeError_0
END INTERFACE


INTERFACE laplaceTransform
  MODULE PROCEDURE laplaceTransform1
END INTERFACE

INTERFACE put
  MODULE PROCEDURE put1
END INTERFACE

INTERFACE qsimp
  MODULE PROCEDURE qsimp1, qsimp2
END INTERFACE

INTERFACE pack
  MODULE PROCEDURE pack1, pack2, pack3
END INTERFACE

INTERFACE unpack
  MODULE PROCEDURE unpack1, unpack2, unpack3
END INTERFACE

INTERFACE positive
  MODULE PROCEDURE positive1
END INTERFACE

INTERFACE smooth
  MODULE PROCEDURE smooth1
END INTERFACE

INTERFACE split
  MODULE PROCEDURE split2,split3,split4
END INTERFACE

INTERFACE polint
```

```
    MODULE PROCEDURE polint1,polint2
  END INTERFACE

  INTERFACE iminloc
    MODULE PROCEDURE iminloc1,iminloc2
  END INTERFACE

  INTERFACE geop
    MODULE PROCEDURE geop1
  END INTERFACE


  CONTAINS
!C: ************************************
    SUBROUTINE Error1(string1, string2)
      CHARACTER(LEN=*), INTENT(IN) :: string1, string2
        WRITE(*,*) string1
        WRITE(*,*) '***', string2, '***'
        !CALL EXIT(1)
        STOP 'PROGRAM TERMINATED BY AN ERROR'
    END SUBROUTINE Error1
!C: ************************************

    FUNCTION createArray_1(n, string)
      INTEGER, INTENT(IN) :: n
      REAL(PREC), DIMENSION(:), POINTER :: createArray_1
      CHARACTER(LEN=*), INTENT(IN) :: string

      REAL(PREC), DIMENSION(:), TARGET, ALLOCATABLE :: array
      INTEGER :: ierr
        Allocate(array(n), STAT=ierr)
        IF (ierr/=0) THEN
          CALL Error('ALLOCATION REQUEST IS DENIED IN:', string)
        END IF
        createArray_1 => array
    END FUNCTION createArray_1
    FUNCTION createArray_2(m, n, string)
      INTEGER, INTENT(IN) :: m, n
      REAL(PREC), DIMENSION(:,:), POINTER :: createArray_2
      CHARACTER(LEN=*), INTENT(IN) :: string

      REAL(PREC), DIMENSION(:,:), TARGET, ALLOCATABLE :: array
      INTEGER :: ierr
        Allocate(array(m,n), STAT=ierr)
        IF (ierr/=0) THEN
          CALL Error('ALLOCATION REQUEST IS DENIED IN:', string)
```

```
    END IF
    createArray_2 => array
END FUNCTION createArray_2
FUNCTION createArray_3(m, n, r, string)
  INTEGER, INTENT(IN) :: m, n, r
  REAL(PREC), DIMENSION(:,:,:), POINTER :: createArray_3
  CHARACTER(LEN=*), INTENT(IN) :: string

  REAL(PREC), DIMENSION(:,:,:), TARGET, ALLOCATABLE :: array
  INTEGER :: ierr
    Allocate(array(m,n,r), STAT=ierr)
    IF (ierr/=0) THEN
      CALL Error('ALLOCATION REQUEST IS DENIED IN:', string)
    END IF
    createArray_3 => array
END FUNCTION createArray_3
FUNCTION createArray_4(l,m, n, r, string)
  INTEGER, INTENT(IN) :: l,m, n, r
  REAL(PREC), DIMENSION(:,:,:,:), POINTER :: createArray_4
  CHARACTER(LEN=*), INTENT(IN) :: string

  REAL(PREC), DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: array
  INTEGER :: ierr
    Allocate(array(l,m,n,r), STAT=ierr)
    IF (ierr/=0) THEN
      CALL Error('ALLOCATION REQUEST IS DENIED IN:', string)
    END IF
    createArray_4 => array
END FUNCTION createArray_4
FUNCTION createArray_5(l,m,n,r,s, string)
  INTEGER, INTENT(IN) :: l,m,n,r,s
  REAL(PREC), DIMENSION(:,:,:,:,:), POINTER :: createArray_5
  CHARACTER(LEN=*), INTENT(IN) :: string

  REAL(PREC), DIMENSION(:,:,:,:,:), TARGET, ALLOCATABLE :: array
  INTEGER :: ierr
    Allocate(array(l,m,n,r,s),STAT=ierr)
    IF (ierr/=0) THEN
      CALL Error('ALLOCATION REQUEST IS DENIED IN:', string)
    END IF
    createArray_5 => ARRAY
END FUNCTION createArray_5
FUNCTION createArray_6(m,str1,str2)
  INTEGER, INTENT(IN) :: m
  INTEGER, DIMENSION(:), POINTER :: createArray_6
  CHARACTER(LEN=*), INTENT(IN) :: str1,str2
```

```
      INTEGER, DIMENSION(:), TARGET, ALLOCATABLE :: array
      INTEGER :: ierr
        Allocate(array(m), STAT=ierr)
        IF (ierr/=0) THEN
          CALL Error('ALLOCATION REQUEST IS DENIED IN:', str2)
        END IF
        createArray_6 => array
    END FUNCTION createArray_6
!C: ************************************

    SUBROUTINE ASSERT_EQ_2(n1,n2,string)
      INTEGER, INTENT(IN) :: n1, n2
      CHARACTER(LEN=*), INTENT(IN) :: string
        IF(n1/=n2) THEN
          CALL Error('THE INPUT ARRAYS ARE NOT CONFORMAL IN:', string)
        END IF
    END SUBROUTINE
    SUBROUTINE ASSERT_EQ_3(n1,n2,n3,string)
      INTEGER, INTENT(IN) :: n1, n2, n3
      CHARACTER(LEN=*), INTENT(IN) :: string
        CALL ASSERT_EQ_2(n1,n2,string)
        CALL ASSERT_EQ_2(n1,n3,string)
    END SUBROUTINE
    SUBROUTINE ASSERT_EQ_4(n1,n2,n3,n4,string)
      INTEGER, INTENT(IN) :: n1, n2, n3, n4
      CHARACTER(LEN=*), INTENT(IN) :: string
        CALL ASSERT_EQ_3(n1,n2,n3,string)
        CALL ASSERT_EQ_2(n1,n4,string)
    END SUBROUTINE
    SUBROUTINE ASSERT_EQ_5(n1,n2,n3,n4,n5,string)
      INTEGER, INTENT(IN) :: n1, n2, n3, n4,n5
      CHARACTER(LEN=*), INTENT(IN) :: string
        CALL ASSERT_EQ_4(n1,n2,n3,n4,string)
        CALL ASSERT_EQ_2(n1,n5,string)
    END SUBROUTINE
    SUBROUTINE ASSERT_EQ_6(n1,n2,n3,n4,n5,n6,string)
      INTEGER, INTENT(IN) :: n1, n2, n3, n4,n5,n6
      CHARACTER(LEN=*), INTENT(IN) :: string
        CALL ASSERT_EQ_5(n1,n2,n3,n4,n5,string)
        CALL ASSERT_EQ_2(n1,n6,string)
    END SUBROUTINE
    SUBROUTINE ASSERT_EQ_7(n1,n2,n3,n4,n5,n6,n7,string)
      INTEGER, INTENT(IN) :: n1, n2, n3, n4,n5,n6,n7
      CHARACTER(LEN=*), INTENT(IN) :: string
        CALL ASSERT_EQ_6(n1,n2,n3,n4,n5,n6,string)
```

```
        CALL ASSERT_EQ_2(n1,n6,string)
    END SUBROUTINE

!C:*****************************************
    SUBROUTINE RELEASE_MEMORY_1(p, string)
      REAL(PREC), DIMENSION(:), POINTER :: P
      CHARACTER(LEN=*), INTENT(IN) :: string

      INTEGER :: ierr
        IF(ASSOCIATED(p)) DEALLOCATE(p, STAT=ierr)
        !IF (ierr/=0) THEN
          !  WRITE(*,*) 'DEALLOCATION REQUEST IS DENIED IN'
          ! WRITE(*,*) string
        !END IF
    END SUBROUTINE RELEASE_MEMORY_1
    SUBROUTINE RELEASE_MEMORY_2(p, string)
      REAL(PREC), DIMENSION(:,:), POINTER :: p
      CHARACTER(LEN=*), INTENT(IN) :: string

      INTEGER :: ierr
        IF(ASSOCIATED(p)) DEALLOCATE(p, STAT=ierr)
        !IF (ierr/=0) THEN
          !  WRITE(*,*) 'DEALLOCATION REQUEST IS DENIED IN'
          ! WRITE(*,*) string
        !END IF
    END SUBROUTINE RELEASE_MEMORY_2
    SUBROUTINE RELEASE_MEMORY_3(p, string)
      REAL(PREC), DIMENSION(:,:,:), POINTER :: p
      CHARACTER(LEN=*), INTENT(IN) :: string
      INTEGER :: ierr
        IF(ASSOCIATED(p)) DEALLOCATE(p, STAT=ierr)
        !IF (ierr/=0) THEN
          !  WRITE(*,*) 'DEALLOCATION REQUEST IS DENIED IN'
          ! WRITE(*,*) string
        !END IF
    END SUBROUTINE RELEASE_MEMORY_3
!C: ************************************
    SUBROUTINE show_1(string)
      CHARACTER(LEN=*), INTENT(IN) :: string
        WRITE(*,*) '*********************************'
        WRITE(*,*) '*', string
        WRITE(*,*) '*********************************'
    END SUBROUTINE show_1

    SUBROUTINE show_2(string1, string2)
      CHARACTER(LEN=*), INTENT(IN) :: string1, string2
```

```
      WRITE(*,*) string2, '=', string1
    END SUBROUTINE show_2

    SUBROUTINE show_3(x, string)
      REAL(PREC), INTENT(IN) :: x
      CHARACTER(LEN=*), INTENT(IN) :: string
        WRITE (*,*) string, ' = ', x
    END SUBROUTINE show_3

    SUBROUTINE show_4(x, string)
      REAL(PREC), DIMENSION(:), INTENT(IN) :: x
      CHARACTER(LEN=*), INTENT(IN) :: string
      INTEGER k
        DO k=1, SIZE(x)
          call show(x(k), string)
        END DO
    END SUBROUTINE show_4

    SUBROUTINE show_5(x, string)
      REAL(PREC), DIMENSION(:,:), INTENT(IN) :: x
      CHARACTER(LEN=*), INTENT(IN) :: string
      INTEGER :: j
        DO j=1, SIZE(x,2)
          WRITE(*,*) 'J = ',j
          CALL SHOW(x(:,j), string)
        END DO
    END SUBROUTINE show_5

    SUBROUTINE show_6(x, string)
      REAL(PREC), DIMENSION(:,:,:), INTENT(IN) :: x
      CHARACTER(LEN=*), INTENT(IN) :: string
      INTEGER :: k
        DO k=1, SIZE(x,3)
          WRITE(*,*) 'K = ', k
          call show(x(:,:,k),string)
        END DO
    END SUBROUTINE show_6

    SUBROUTINE show_7(n, string)
      CHARACTER(LEN=*), INTENT(IN) :: string
      INTEGER :: n
        WRITE(*,*) string, ' = ', n
    END SUBROUTINE show_7
!C:*****************************************
    FUNCTION outer_prod1(a,b)
      REAL(PREC), DIMENSION(:), INTENT(IN) :: a,b
```

```
      REAL(PREC), DIMENSION(size(a), size(b)) :: outer_prod1
        outer_prod1 = spread(a,dim=2,ncopies=size(b)) * &
          spread(b,dim=1,ncopies=size(a))
    END FUNCTION outer_prod1

!C:*****************************************
    SUBROUTINE diagnal_assign1(mat, vec)
      REAL(PREC), DIMENSION(:,:), INTENT(OUT) :: mat
      REAL(PREC), DIMENSION(:), INTENT(IN) :: vec
        CALL ASSERT_EQ(SIZE(mat,1), SIZE(mat, 2), SIZE(vec), &
              'diagnal_assign1')
        mat=0.0_prec
        DO k=0, size(vec)
          mat(k,k)=vec(k)
        END DO
    END SUBROUTINE diagnal_assign1

!C:*****************************************

  FUNCTION arth_d(first,increment,n)
    REAL(PREC), INTENT(IN) :: first,increment
    INTEGER, INTENT(IN) :: n
    REAL(PREC), DIMENSION(n) :: arth_d
    INTEGER :: k,k2
    REAL(PREC) :: temp
    INTEGER, PARAMETER :: NPAR_ARTH=16,NPAR2_ARTH=8
      if (n > 0) arth_d(1)=first
      if (n <= NPAR_ARTH) then
        do k=2,n
                arth_d(k)=arth_d(k-1)+increment
        end do
      else
        do k=2,NPAR2_ARTH
                arth_d(k)=arth_d(k-1)+increment
        end do
        temp=increment*NPAR2_ARTH
        k=NPAR2_ARTH
        do
                if (k >= n) exit
                k2=k+k
                arth_d(k+1:min(k2,n))=temp+arth_d(1:min(k,n-k))
                temp=temp+temp
                k=k2
        end do
      end if
  END FUNCTION arth_d
```

```
!C:*****************************************
    FUNCTION arth1(first, increment, n)
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: first, increment, n
      INTEGER, DIMENSION(n) :: arth1

      INTEGER :: k
        if (n<=0) call Error('INVALID DIMENSION IN: ', 'arth1')
        arth1(1)=first
        DO k=2, n
          arth1(k)=arth1(k-1)+increment
        END DO
    END FUNCTION arth1
!C:*****************************************
    FUNCTION toString1(int)
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: int
      CHARACTER(LEN=8) :: toString1
      CHARACTER(LEN=1), DIMENSION(10), PARAMETER :: &
          char=(/'0','1','2','3','4','5','6','7','8','9'/)
      INTEGER :: n, k
        IF( int>99999999.OR. int<0) THEN
          CALL Error('argument out of bound','toString1')
        END IF
        n=int
        k=1
        DO while(n >0)
          toString1=char(mod(n,10)+1)//toString1
          n=n/10
          k=k+1
        END DO
        DO n=k, 8
          toString1=char(1)//toString1
        END DO
    END FUNCTION toString1
!C:*****************************************
    FUNCTION locate1(xx,x)
      IMPLICIT NONE
      REAL(PREC), DIMENSION(:), INTENT(IN) :: xx
      REAL(PREC), INTENT(IN) :: x
      INTEGER :: locate1

      INTEGER :: n,jl,jm,ju
      LOGICAL :: ascnd
        n=size(xx)
```

```
      ascnd = (xx(n) >= xx(1))
      jl=0
      ju=n+1
      DO WHILE (ju-jl > 1)
        jm=(ju+jl)/2
        IF (ascnd .eqv. (x >= xx(jm))) THEN
          jl=jm
        ELSE
          ju=jm
        END IF
      END DO
      IF (x == xx(1)) THEN
        locate1=1
      ELSE IF (x == xx(n)) THEN
        locate1=n-1
      ELSE
        locate1=jl
      END IF
    END FUNCTION locate1

!C:*****************************************
    FUNCTION locate2(xx,x)
      IMPLICIT NONE
      REAL, DIMENSION(:), INTENT(IN) :: xx
      REAL, INTENT(IN) :: x
      INTEGER :: locate2

      INTEGER :: n,jl,jm,ju
      LOGICAL :: ascnd
        n=size(xx)
        ascnd = (xx(n) >= xx(1))
        jl=0
        ju=n+1
        DO WHILE (ju-jl > 1)
          jm=(ju+jl)/2
          IF (ascnd .eqv. (x >= xx(jm))) THEN
            jl=jm
          ELSE
            ju=jm
          END IF
        END DO
        IF (x == xx(1)) THEN
          locate2=1
        ELSE IF (x == xx(n)) THEN
          locate2=n-1
        ELSE
```

```
      locate2=jl
    END IF
  END FUNCTION locate2

!C:*****************************************

  FUNCTION blitp1(x1,x2,vec,a,b,ha,hb)
    IMPLICIT NONE
    REAL(PREC), INTENT(IN) :: x1,x2,a,b,ha,hb
    REAL(PREC),DIMENSION(:,:), INTENT(IN):: vec
    REAL(PREC) :: blitp1

    REAL(PREC) :: y1,y2,y3,y4,t,u,hx,hy
    INTEGER :: m,n, j, k

      m=SIZE(vec,1)-1
      n=SIZE(vec,2)-1
      hx=ha/m
      hy=hb/n
      j=min(max(int(m*(x1-a)/ha)+1,1),m)
      k=min(max(int(n*(x2-b)/hb)+1,1),n)

      y1=vec(j,k)
      y2=vec(j+1,k)
      y3=vec(j+1,k+1)
      y4=vec(j,k+1)
      t=( x1 - (a + hx*(j-1)) )/hx
      u=( x2 - (b + hy*(k-1)) )/hy

      blitp1=(1.0_prec-t)*(1.0_prec-u)*y1+t*(1.0_prec-u)*y2+t*u*y3 &
          +(1.0_prec-t)*u*y4
  END FUNCTION blitp1

  FUNCTION blitp2(x1,x2,vec,a,b,ha,hb)
    IMPLICIT NONE
    REAL, INTENT(IN) :: x1,x2,a,b,ha,hb
    REAL,DIMENSION(:,:), INTENT(IN):: vec
    REAL :: blitp2

    REAL :: y1,y2,y3,y4,t,u,hx,hy
    INTEGER :: m,n, j, k

      m=SIZE(vec,1)-1
      n=SIZE(vec,2)-1
      hx=ha/m
      hy=hb/n
```

```
    j=min(max(int(m*(x1-a)/ha)+1,1),m)
    k=min(max(int(n*(x2-b)/hb)+1,1),n)

    y1=vec(j,k)
    y2=vec(j+1,k)
    y3=vec(j+1,k+1)
    y4=vec(j,k+1)
    t=( x1 - (a + hx*(j-1)) )/hx
    u=( x2 - (b + hy*(k-1)) )/hy

    blitp2=(1.0-t)*(1.0-u)*y1+t*(1.0-u)*y2+t*u*y3 &
        +(1.0-t)*u*y4
END FUNCTION blitp2

FUNCTION blitp3(x1,x2,vec,a,b,ha,hb)
  IMPLICIT NONE
  REAL, INTENT(IN) :: x1,x2
  REAL(PREC), INTENT(IN) :: a,b,ha,hb
  REAL,DIMENSION(:,:), INTENT(IN):: vec
  REAL :: blitp3

    blitp3=blitp2(x1,x2,vec,real(a),real(b),real(ha),real(hb))
END FUNCTION blitp3

FUNCTION blitp4(x1,x2,vec,a,b,ha,hb)
  IMPLICIT NONE
  REAL, INTENT(IN) :: x1,x2
  REAL(PREC), INTENT(IN) :: a,b,ha,hb
  REAL(PREC),DIMENSION(:,:), INTENT(IN):: vec
  REAL :: blitp4

    blitp4=blitp1(REAL(x1,KIND=PREC),REAL(x2,KIND=PREC), &
                  vec,a,b,ha,hb)
END FUNCTION blitp4

FUNCTION blitp5(x1,x2,x3,vec,a,b,c,ha,hb,hc)
  IMPLICIT NONE
  REAL, INTENT(IN) :: x1,x2,x3
  REAL, INTENT(IN) :: a,b,c,ha,hb,hc
  REAL,DIMENSION(:,:,:), INTENT(IN):: vec
  REAL :: blitp5

  REAL,DIMENSION(8) :: y
  REAL :: t,u,w,hx,hy,hz
  INTEGER :: l,m,n, i,j,k
    l=SIZE(vec,1)-1
```

```
    m=SIZE(vec,2)-1
    n=SIZE(vec,3)-1
    hx=ha/l
    hy=hb/m
    hz=hc/n
    i=min(max(int(l*(x1-a)/ha)+1,1),l)
    j=min(max(int(m*(x2-b)/hb)+1,1),m)
    k=min(max(int(n*(x3-c)/hc)+1,1),n)
    y(1)=vec(i,j,k)
    y(2)=vec(i+1,j,k)
    y(3)=vec(i+1,j+1,k)
    y(4)=vec(i,j+1,k)
    y(5)=vec(i,j,k+1)
    y(6)=vec(i+1,j,k+1)
    y(7)=vec(i+1,j+1,k+1)
    y(8)=vec(i,j+1,k+1)
    t=( x1 - (a + hx*(i-1)) )/hx
    u=( x2 - (b + hy*(j-1)) )/hy
    w=( x3 - (c + hz*(k-1)) )/hz

    blitp5=(1-w)*((1-u)*((1-t)*y(1)+t*y(2))+u*(t*y(3)+(1-t)*y(4)))&
           +w *((1-u)*((1-t)*y(5)+t*y(6))+u*(t*y(7)+(1-t)*y(8)))
END FUNCTION blitp5

FUNCTION blitp6(x1,x2,x3,vec,a,b,c,ha,hb,hc)
  IMPLICIT NONE
  REAL, INTENT(IN) :: x1,x2,x3
  REAL(PREC), INTENT(IN) :: a,b,c,ha,hb,hc
  REAL(PREC),DIMENSION(:,:,:), INTENT(IN):: vec
  REAL :: blitp6

  REAL :: a1,b1,c1,ha1,hb1,hc1
    a1=REAL(a)
    b1=REAL(b)
    c1=REAL(c)
    ha1=REAL(ha)
    hb1=REAL(hb)
    hc1=REAL(hc)
    blitp6=blitp5(x1,x2,x3,REAL(vec),a1,b1,c1,ha1,hb1,hc1)
END FUNCTION blitp6

FUNCTION blitp7(x1,x2,x3,vec,a,b,c,ha,hb,hc)
  IMPLICIT NONE
  REAL(PREC), INTENT(IN) :: x1,x2,x3
  REAL(PREC), INTENT(IN) :: a,b,c,ha,hb,hc
  REAL(PREC),DIMENSION(:,:,:), INTENT(IN):: vec
```

```
      REAL(PREC) :: blitp7
        blitp7=REAL(blitp6(REAL(x1),REAL(x2),REAL(x3),&
                    vec,a,b,c,ha,hb,hc),KIND=PREC)
    END FUNCTION blitp7



!C***********************************************
    SUBROUTINE plot_func(func, fileName,a,b,ha,hb, MGRID,NGRID)
      IMPLICIT NONE
      CHARACTER(LEN=*) :: fileName
      REAL(PREC), INTENT(IN) :: a,b,ha,hb
      INTEGER, INTENT(IN) :: MGRID,NGRID
      INTERFACE
        FUNCTION func(x,y)
          USE ntype
          REAL(PREC), INTENT(IN) :: x, y
          REAL(PREC) func
        END FUNCTION
      END INTERFACE

      REAL(PREC) :: x, y, hx,hy
      INTEGER :: i, j
        hx=ha/(MGRID-1)
        hy=hb/(NGRID-1)

        OPEN (4, file=fileName//'.dat')
        WRITE(4,*) 'TITLE='//fileName
        WRITE(4,*) 'VARIABLES="X" "Y" "Z"'
        WRITE(4,*) 'ZONE I=',mgrid,', J=',ngrid,', C=BLUE'
        DO i=1, mgrid
          x = a + (i-1)*hx
          DO j=1, ngrid
            y = b + (j-1)*hy
            WRITE(4,*) x, y, REAL(func(x,y),KIND=PREC)
          END DO
        END DO
        CLOSE(4, STATUS='keep')
    END SUBROUTINE plot_func

    SUBROUTINE plot1(home,name,NUM,vec,int,a,b,ha,hb)
      IMPLICIT NONE

      CHARACTER(LEN=*),INTENT(IN) :: home
      CHARACTER(LEN=*),DIMENSION(:),INTENT(IN) :: name
      INTEGER, INTENT(IN) :: NUM,int
```

```
  REAL(PREC), DIMENSION(:,:,:), INTENT(IN) :: vec
  REAL(PREC), INTENT(IN) :: a,b,ha,hb

  CHARACTER(LEN=8) :: str
    str=toString(int)
    IF (int<10) THEN
      CALL PLOTTING(home,name,NUM,vec,str(8:8),a,b,ha,hb)
    ELSE IF (int<100) THEN
      CALL PLOTTING(home,name,NUM,vec,str(7:8),a,b,ha,hb)
    ELSE IF (int<1000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(6:8),a,b,ha,hb)
    ELSE IF (int<10000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(5:8),a,b,ha,hb)
    ELSE IF (int<100000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(4:8),a,b,ha,hb)
    ELSE IF (int<1000000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(3:8),a,b,ha,hb)
    ELSE IF (int<10000000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(2:8),a,b,ha,hb)
    ELSE IF (int<100000000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(1:8),a,b,ha,hb)
    END IF
END SUBROUTINE plot1

SUBROUTINE plot2(home,name,NUM,vec,int,a,b,ha,hb)
  IMPLICIT NONE

  CHARACTER(LEN=*),INTENT(IN) :: home
  CHARACTER(LEN=*),DIMENSION(:),INTENT(IN) :: name
  INTEGER, INTENT(IN) :: NUM,int
  REAL(PREC), DIMENSION(:,:,:,:), INTENT(IN) :: vec
  REAL(PREC), INTENT(IN) :: a,b,ha,hb

  CHARACTER(LEN=8) :: str
    str=toString(int)
    IF (int<10) THEN
      CALL PLOTTING(home,name,NUM,vec,str(8:8),a,b,ha,hb)
    ELSE IF (int<100) THEN
      CALL PLOTTING(home,name,NUM,vec,str(7:8),a,b,ha,hb)
    ELSE IF (int<1000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(6:8),a,b,ha,hb)
    ELSE IF (int<10000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(5:8),a,b,ha,hb)
    ELSE IF (int<100000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(4:8),a,b,ha,hb)
    ELSE IF (int<1000000) THEN
```

```
      CALL PLOTTING(home,name,NUM,vec,str(3:8),a,b,ha,hb)
    ELSE IF (int<10000000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(2:8),a,b,ha,hb)
    ELSE IF (int<100000000) THEN
      CALL PLOTTING(home,name,NUM,vec,str(1:8),a,b,ha,hb)
    END IF
END SUBROUTINE plot2

SUBROUTINE plot3(home,name,NUM,vec,str,a,b,ha,hb)
  IMPLICIT NONE

  INTEGER, INTENT(IN) :: NUM
  CHARACTER(LEN=*),INTENT(IN) :: home
  CHARACTER(LEN=*),DIMENSION(:),INTENT(IN) :: name
  CHARACTER(LEN=*), INTENT(IN) :: str
  REAL(PREC), DIMENSION(:,:,:), INTENT(IN) :: vec
  REAL(PREC), INTENT(IN) :: a,b,ha,hb
  CHARACTER(LEN=1),DIMENSION(10),PARAMETER :: &
      ch1=(/'0','1','2','3','4','5','6','7','8','9'/)
  CHARACTER(LEN=3), DIMENSION(100), PARAMETER :: &
     ch2=(/'001','002','003','004','005','006','007','008','009',&
           '010','011','012','013','014','015','016','017','018',&
           '019','020','021','022','023','024','025','026','027',&
           '028','029','030','031','032','033','034','035','036',&
           '037','038','039','040','041','042','043','044','045',&
           '046','047','048','049','050','051','052','053','054',&
           '055','056','057','058','059','060','061','062','063',&
           '064','065','066','067','068','069','070','071','072',&
           '073','074','075','076','077','078','079','080','081',&
           '082','083','084','085','086','087','088','089','090',&
           '091','092','093','094','095','096','097','098','099',&
           '100'/)

  INTEGER :: k

    DO k=1,NUM
      CALL PLOT_VEC1(vec(k,:,:), home//'PC/'//str//'_'//&
                                 name(1)//ch1(k+1),'',a,b,ha,hb)
    END DO
    CALL PLOT_VEC1(vec(NUM+1,:,:), home//'PC/'//str//'_'//&
                                 name(2),'',a,b,ha,hb)
    DO k=1,SIZE(vec,1)-NUM-1
      CALL PLOT_VEC1(vec(NUM+1+k,:,:), home//'PC/'//str//'_'// &
                                 name(3)//ch2(k),'',a,b,ha,hb)
    END DO
END SUBROUTINE plot3
```

```
SUBROUTINE plot4(home,name,NUM,vec,str,a,b,ha,hb)
  IMPLICIT NONE

  CHARACTER(LEN=*),INTENT(IN) :: home
  CHARACTER(LEN=*),DIMENSION(:),INTENT(IN) :: name
  INTEGER, INTENT(IN) :: NUM
  CHARACTER(LEN=*), INTENT(IN) :: str
  REAL(PREC), DIMENSION(:,:,:,:), INTENT(IN) :: vec
  REAL(PREC), INTENT(IN) :: a,b,ha,hb
  CHARACTER(LEN=1), DIMENSION(20), PARAMETER :: &
      ch1=(/'0','1','2','3','4','5','6','7','8','9'/)
  CHARACTER(LEN=3), DIMENSION(100), PARAMETER :: &
     ch2=(/'001','002','003','004','005','006','007','008','009',&
           '010','011','012','013','014','015','016','017','018',&
           '019','020','021','022','023','024','025','026','027',&
           '028','029','030','031','032','033','034','035','036',&
           '037','038','039','040','041','042','043','044','045',&
           '046','047','048','049','050','051','052','053','054',&
           '055','056','057','058','059','060','061','062','063',&
           '064','065','066','067','068','069','070','071','072',&
           '073','074','075','076','077','078','079','080','081',&
           '082','083','084','085','086','087','088','089','090',&
           '091','092','093','094','095','096','097','098','099',&
           '100'/)

  INTEGER :: k1,k2
    DO k1=1, size(vec,1)
      DO k2=1,NUM
        CALL PLOT_VEC1(vec(k1,k2,:,:),home//'PC/'//str//'_' &
             //name(1)//ch1(k2+1)//'_'//ch2(k1),'',a,b,ha,hb)
      END DO
      CALL PLOT_VEC1(vec(k1,NUM+1,:,:),home//'PC/'//str//'_' &
             //name(2)//'_'//ch2(k1),'',a,b,ha,hb)
      CALL PLOT_VEC1(vec(k1,NUM+2,:,:),home//'PC/'//str//'_' &
             //name(3)//'_'//ch2(k1),'',a,b,ha,hb)
      CALL PLOT_VEC1(vec(k1,NUM+3,:,:),home//'PC/'//str//'_' &
             //name(4)//'_'//ch2(k1),'',a,b,ha,hb)
    END DO
END SUBROUTINE plot4

SUBROUTINE plot_vec1(vec, fileName,STR,a,b,ha,hb)
  IMPLICIT NONE
  CHARACTER(LEN=*) :: fileName,STR
  REAL(PREC), DIMENSION(:,:), INTENT(IN) :: vec
  REAL(PREC), INTENT(IN) :: a,b,ha,hb
```

```
  REAL(PREC) :: x, y, hx,hy
  INTEGER :: i, j, m, n
    m=SIZE(vec,1)
    n=SIZE(vec,2)
    hx=ha/(m-1)
    hy=hb/(n-1)

    OPEN (4, file=fileName//'.dat')
    WRITE(4,*) 'TITLE='//fileName//STR
    WRITE(4,*) 'VARIABLES="X" "Y" "Z"'
    WRITE(4,*) 'ZONE I=',m,', J=',n,', C=BLUE'
    DO i=1, m
      x = a + (i-1)*hx
      DO j=1, n
        y = b + (j-1)*hy
        WRITE(4,*) x, y, REAL(vec(i,j),KIND=PREC)
      END DO
    END DO
    CLOSE(4, STATUS='keep')
END SUBROUTINE plot_vec1

SUBROUTINE plot_vec2(vec, fileName,STR,a,b,ha,hb)
  IMPLICIT NONE
  CHARACTER(LEN=*) :: fileName,STR
  REAL(PREC), DIMENSION(:,:,:), INTENT(IN) :: vec
  REAL(PREC), INTENT(IN) :: a,b,ha,hb

  INTEGER :: n,i
  CHARACTER(LEN=8) :: sn
    n=SIZE(vec,1);
    IF (n==1) THEN
      CALL PLOTTING(VEC(1,:,:), fileName,STR,a,b,ha,hb)
    ELSE
      call plotting(vec(1,:,:),fileName//'_11',STR,a,b,ha,hb)
      call plotting(vec(2,:,:),fileName//'_12',STR,a,b,ha,hb)
      call plotting(vec(3,:,:),fileName//'_22',STR,a,b,ha,hb)
    END IF
END SUBROUTINE plot_vec2

SUBROUTINE plot_real(vec, fileName,STR,a,b,ha,hb)
  IMPLICIT NONE
  CHARACTER(LEN=*) :: fileName,STR
  REAL, DIMENSION(:,:), INTENT(IN) :: vec
  REAL(PREC), INTENT(IN) :: a,b,ha,hb
```

```
      REAL :: x, y, hx,hy
      INTEGER :: i, j, m, n
        m=SIZE(vec,1)
        n=SIZE(vec,2)
        hx=ha/(m-1)
        hy=hb/(n-1)

        OPEN (4, file=fileName//'.dat')
        WRITE(4,*) 'TITLE='//fileName//STR
        WRITE(4,*) 'VARIABLES="X" "Y" "Z"'
        WRITE(4,*) 'ZONE I=',m,', J=',n,', C=BLUE'
        DO i=1, m
          x = a + (i-1)*hx
          DO j=1, n
            y = b + (j-1)*hy
            WRITE(4,*) x, y, REAL(vec(i,j),KIND=PREC)
          END DO
        END DO
        CLOSE(4, STATUS='keep')
    END SUBROUTINE plot_real

    SUBROUTINE plot_1d(vec, home,name,a,ha)
      IMPLICIT NONE
      CHARACTER(LEN=*) :: home,name
      REAL(PREC), DIMENSION(:), INTENT(IN) :: vec
      REAL(PREC), INTENT(IN) :: a,ha

      REAL :: x, y, hx,hy
      INTEGER :: i, m
        m=SIZE(vec)
        hx=ha/(m-1)

        OPEN (4, file=home//name//'.dat')
        WRITE(4,*) 'TITLE='//name
        WRITE(4,*) 'VARIABLES="X" "Y" '
        WRITE(4,*) 'ZONE I=',m,', C=BLUE'
        DO i=1, m
          x = a + (i-1)*hx
          WRITE(4,*) x, y, vec(i)
        END DO
        CLOSE(4, STATUS='keep')
    END SUBROUTINE plot_1d

!C***********************************************
    FUNCTION contains1 (i, vec)
      IMPLICIT NONE
```

```
      INTEGER, INTENT(IN) :: i
      INTEGER, DIMENSION(:) :: vec
      LOGICAL :: contains1

      INTEGER :: k
        DO k=1, SIZE(vec,1)
          IF (vec(k)==i) THEN
            contains1=.TRUE.
            RETURN
          END IF
        END DO
        contains1=.FALSE.
        RETURN
    END FUNCTION contains1
!C:***************************************
    SUBROUTINE computeError_0(vec,a,b,ha,hb,t1,t2,func)
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) :: a,b,ha,hb,t1,t2
      REAL(PREC), DIMENSION(:,:,:), INTENT(IN) :: vec
      INTERFACE
        FUNCTION func(x,y,t)
          USE ntype
          REAL(PREC) :: x,y,t
          REAL(PREC) :: func
        END FUNCTION
      END INTERFACE

      INTEGER :: nstep,n1,n2,ll,i,j, ierr,jerr
      REAL(PREC), DIMENSION(SIZE(vec,2),SIZE(vec,3)) :: vec1
      REAL(PREC) :: x, y, hx, hy,err,t
      CHARACTER(LEN=12) :: fileName
      CHARACTER(LEN=10) :: title
        nstep = SIZE(vec,1)
        n1=SIZE(vec,2)
        n2=SIZE(vec,3)
        hx=ha/(n1-1)
        hy=hb/(n2-1)

        DO ll=1, nstep
          err=0.0_prec
          t=t1+(t2-t1)*(ll-1)/(nstep-1)
          DO i=1,n1
            x=a+hx*(i-1)
            DO j=1,n2
              y=b+hy*(j-1)
              vec1(i,j)=func(x,y,t)
```

```
            IF(abs(vec1(i,j)-vec(ll,i,j)) > err) THEN
              err=abs(vec1(i,j)-vec(ll,i,j))
              ierr=i
              jerr=j
            END IF
          END DO
        END DO
        fileName='PC/error.dat'
        title='difference'
        CALL plotting(vec1(:,:)-vec(ll,:,:),'PC/error.dat','error',&
                      a,b,ha,hb)
        CALL plotting(vec1(:,:),'PC/vec1.dat','vec1',a,b,ha,hb)
        CALL plotting(vec(:,:),'PC/vec.dat','vec',a,b,ha,hb)
        PRINT *, ' THE ERROR IS :'
        PRINT *, ' I = ', IERR, ' J = ', JERR, ' ERR = ', err
        pause
      END DO
      STOP
END SUBROUTINE computeError_0

SUBROUTINE computeError_1(u,ux,uy, u1, u1x,u1y,a,b,ha,hb)
  IMPLICIT NONE
  REAL(PREC), DIMENSION(:,:,:), INTENT(IN):: u,ux,uy,u1,u1x,u1y
  REAL(PREC), INTENT(IN) :: a,b,ha,hb

  INTEGER :: nstep,n1,n2,ll
    CALL assert_eq(SIZE(u,1),SIZE(ux,1),SIZE(uy,1), &
                SIZE(u1,1),SIZE(u1x,1),SIZE(u1y,1), 'computeError')
    nstep=SIZE(u,1)
    n1=SIZE(u,2)
    n2=SIZE(u,3)
    DO ll=1,nstep
      CALL plotting(u(ll,:,:)-u1(ll,:,:), &
                    'PC/error.dat','error',a,b,ha,hb)
      CALL plotting(u(ll,:,:), 'PC/vec.dat','vec',a,b,ha,hb)
      CALL plotting(u1(ll,:,:),'PC/vec1.dat','vec1',a,b,ha,hb)

      CALL plotting(ux(ll,:,:)-u1x(ll,:,:), &
                    'PC/errorx.dat','errorx',a,b,ha,hb)
      CALL plotting(ux(ll,:,:), 'PC/vecx.dat','vecx',a,b,ha,hb)
      CALL plotting(u1x(ll,:,:),'PC/vecx1.dat','vecx1',a,b,ha,hb)

      CALL plotting(uy(ll,:,:)-u1y(ll,:,:), &
                    'PC/error.dat','error',a,b,ha,hb)
      CALL plotting(uy(ll,:,:), 'PC/vecy.dat','vecy',a,b,ha,hb)
      CALL plotting(u1y(ll,:,:),'PC/vecy1.dat','vecy1',a,b,ha,hb)
```

```
      PAUSE
    END DO
    STOP
  END SUBROUTINE computeError_1

SUBROUTINE computeError_2(vec,a,b,ha,hb,t1,t2,func)
  IMPLICIT NONE
  REAL(PREC), INTENT(IN) :: a,b,ha,hb,t1,t2
  REAL, DIMENSION(:,:,:), INTENT(IN) :: vec
  INTERFACE
    FUNCTION func(x,y,t)
      USE ntype
      REAL :: x,y,t
      REAL :: func
    END FUNCTION
  END INTERFACE

  INTEGER :: nstep,n1,n2,ll,i,j, ierr,jerr
  REAL(PREC), DIMENSION(SIZE(vec,2),SIZE(vec,3)) :: vec1
  REAL(PREC) :: x, y, hx, hy,err,t
  CHARACTER(LEN=12) :: fileName
  CHARACTER(LEN=10) :: title
    nstep = SIZE(vec,1)
    n1=SIZE(vec,2)
    n2=SIZE(vec,3)
    hx=ha/(n1-1)
    hy=hb/(n2-1)

    DO ll=1, nstep
      err=0.0_prec
      t=t1+(t2-t1)*(ll-1)/(nstep-1)
      DO i=1,n1
        x=a+hx*(i-1)
        DO j=1,n2
          y=b+hy*(j-1)
          vec1(i,j)=REAL(func(real(x),real(y),real(t)),KIND=PREC)
          IF(abs(vec1(i,j)-vec(ll,i,j)) > err) THEN
            err=abs(vec1(i,j)-vec(ll,i,j))
            ierr=i
            jerr=j
          END IF
        END DO
      END DO
      fileName='PC/error.dat'
      title='difference'
      CALL plotting(vec1(:,:)-vec(ll,:,:),'PC/error.dat','error',&
```

```
                        a,b,ha,hb)
          CALL plotting(vec1(:,:),'PC/vec1.dat','vec1',a,b,ha,hb)
          CALL plotting(vec(:,:),'PC/vec.dat','vec',a,b,ha,hb)
          PRINT *, ' THE ERROR IS :'
          PRINT *, ' I = ', IERR, ' J = ', JERR, ' ERR = ', err
          pause
        END DO
        STOP
    END SUBROUTINE computeError_2

!C:*****************************************
    SUBROUTINE laplaceTransform1(func,t1,t2,lmd,u)
      IMPLICIT NONE
      REAL(PREC), DIMENSION(:,:,:), INTENT(IN) :: func
      REAL(PREC), DIMENSION(:), INTENT(IN) :: lmd
      REAL(PREC), INTENT(IN) :: t1,t2
      REAL(PREC), DIMENSION(:,:,:), INTENT(OUT) :: u

      INTEGER :: nstep, m, n, ll,i,j, it, nlmd
      REAL(PREC) :: lambda, t
      REAL(PREC), DIMENSION(SIZE(func,1)) :: func1

        CALL assert_eq(SIZE(u,1), SIZE(lmd,1),'laplaceTransform')
        CALL assert_eq(SIZE(u,2), SIZE(func,2),'laplaceTransform')
        CALL assert_eq(SIZE(u,3), SIZE(func,3),'laplaceTransform')

        nstep=SIZE(func,1)-1
        m=SIZE(func,2)
        n=SIZE(func,3)
        nlmd=SIZE(lmd)
        DO ll=1,nlmd
          lambda=lmd(ll)
          DO i=1,m
          DO j=1,n
            DO it=1,nstep+1
              t=t1+(t2-t1)*(it-1)/nstep
              func1(it)=func(it,i,j)*exp(-lambda*t)
            END DO
            call qsimp(func1,t1,t2,u(ll,i,j))
          END DO
          END DO
        END DO
    END SUBROUTINE laplaceTransform1

!C:*********************************
    SUBROUTINE qsimp1(func,a,b,ss)
```

```
      IMPLICIT NONE
      REAL(PREC), INTENT(IN) ::a,b
      REAL(PREC), DIMENSION(:) :: func
      REAL(PREC), INTENT(OUT) :: ss

      INTEGER nstep, i
      REAL(PREC) h
        nstep=SIZE(func)-1
        h=(b-a)/nstep
        ss=func(1)+func(nstep+1)
        do i=2,nstep,2
          ss=ss+4.0_prec*func(i)
        end do
        do i=3,nstep-1,2
          ss=ss+2.0_prec*func(i)
        end do
        ss=ss*h/3.0_prec
    END SUBROUTINE qsimp1
!C:***********************************
    SUBROUTINE qsimp2(func,a,b,ss)
      IMPLICIT NONE
      REAL, INTENT(IN) ::a,b
      REAL, DIMENSION(:) :: func
      REAL, INTENT(OUT) :: ss

      INTEGER nstep, i
      REAL h
        nstep=SIZE(func)-1
        h=(b-a)/nstep
        ss=func(1)+func(nstep+1)
        do i=2,nstep,2
          ss=ss+4.0*func(i)
        end do
        do i=3,nstep-1,2
          ss=ss+2.0*func(i)
        end do
        ss=ss*h/3.0
    END SUBROUTINE qsimp2
!C:***********************************
    SUBROUTINE put1 (k, vec)
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: k
      INTEGER, DIMENSION(:), INTENT(INOUT) :: vec

      INTEGER :: i
        DO i=SIZE(vec)-1, 1, -1
```

```
        vec(i+1)=vec(i)
      END DO
      vec(1)=k
  END SUBROUTINE put1

!C:**********************************
SUBROUTINE pack1(vec, mat)
    IMPLICIT NONE
    DOUBLE PRECISION, DIMENSION(:), INTENT(OUT) :: vec
    REAL(PREC), DIMENSION(:), INTENT(IN) :: mat
    INTEGER :: m
      CALL assert_eq(SIZE(vec), SIZE(mat), 'pack1')
      m=SIZE(mat)
      vec(1:m)=dble(mat(1:m))
  END SUBROUTINE pack1
!C:**********************************

  SUBROUTINE pack2(vec, mat)
    IMPLICIT NONE
    DOUBLE PRECISION, DIMENSION(:), INTENT(OUT) :: vec
    REAL(PREC), DIMENSION(:,:), INTENT(IN) :: mat
    INTEGER :: k, m, n
      CALL assert_eq(SIZE(vec), SIZE(mat,1)*SIZE(mat,2), 'pack2')
      m=SIZE(mat,1)
      n=SIZE(mat,2)
      DO k=1, m
        vec((k-1)*n+1:k*n)=dble(mat(k,1:n))
      END DO
  END SUBROUTINE pack2
!C:**********************************

  SUBROUTINE pack3(vec, mat)
    IMPLICIT NONE
    DOUBLE PRECISION, DIMENSION(:), INTENT(OUT) :: vec
    REAL(PREC), DIMENSION(:,:,:), INTENT(IN) :: mat
    INTEGER :: k, i, m, n, r
      CALL assert_eq(SIZE(vec), &
          SIZE(mat,1)*SIZE(mat,2)*SIZE(mat,3), 'pack3')
      m=SIZE(mat,2)
      n=SIZE(mat,3)
      DO k=1, SIZE(mat,1)
      DO i=1, m
        r=(k-1)*m*n+(i-1)*n
        vec(r+1:r+n)=dble(mat(k,i,1:n))
      END DO
      END DO
```

```
    END SUBROUTINE pack3
!C:***********************************

    SUBROUTINE unpack1(vec, mat)
      IMPLICIT NONE
      DOUBLE PRECISION, DIMENSION(:), INTENT(IN) :: vec
      REAL(PREC), DIMENSION(:), INTENT(OUT) :: mat
      INTEGER :: m
        CALL assert_eq(SIZE(vec), SIZE(mat), 'unpack1')
        m=SIZE(mat)
        mat(1:m)=REAL(vec(1:m),KIND=PREC)
    END SUBROUTINE unpack1
!C:***********************************

    SUBROUTINE unpack2(vec, mat)
      IMPLICIT NONE
      DOUBLE PRECISION, DIMENSION(:), INTENT(IN) :: vec
      REAL(PREC), DIMENSION(:,:), INTENT(OUT) :: mat
      INTEGER :: k, m, n

        CALL assert_eq(SIZE(vec), SIZE(mat,1)*SIZE(mat,2), 'unpack2')
        m=SIZE(mat,1)
        n=SIZE(mat,2)
        DO k=1, m
          mat(k,1:n)=REAL(vec((k-1)*n+1:k*n),KIND=PREC)
        END DO
    END SUBROUTINE unpack2
!C:***********************************

    SUBROUTINE unpack3(vec, mat)
      IMPLICIT NONE
      DOUBLE PRECISION, DIMENSION(:), INTENT(IN) :: vec
      REAL(PREC), DIMENSION(:,:,:), INTENT(OUT) :: mat
      INTEGER :: k, i, m, n, r
        CALL assert_eq(SIZE(vec), &
                SIZE(mat,1)*SIZE(mat,2)*SIZE(mat,3), 'unpack3')
        m=SIZE(mat,2)
        n=SIZE(mat,3)
        DO k=1, SIZE(mat,1)
        DO i=1, m
          r=(k-1)*m*n+(i-1)*n
          mat(k,i,1:n)=REAL(vec(r+1:r+n),KIND=PREC)
        END DO
        END DO
    END SUBROUTINE unpack3
```

```
!C:***********************************
  FUNCTION positive1(vec)
    IMPLICIT NONE
    REAL(PREC), DIMENSION(:,:,:), INTENT(IN) :: vec
    LOGICAL :: positive1

!C:*********************
!C: LOCAL VARIABLES
!C:*********************
      IF(SIZE(vec,1)==1) THEN
       IF (ANY(vec(1,:,:)<=0.0_prec)) THEN
         positive1=.FALSE.
         RETURN
       END IF
      ELSE
       IF (ANY(vec(1,:,:)<=0.0_prec ).OR. ANY(vec(3,:,:)<=0.0_prec) &
           .OR. ANY(vec(1,:,:)*vec(3,:,:)
                    -vec(2,:,:)*vec(2,:,:)<=0.0_prec)) THEN
         positive1 = .FALSE.
         RETURN
       END IF
      END IF
      positive1 = .TRUE.
  END FUNCTION positive1

!c ********************************************
    SUBROUTINE smooth1(u)
      IMPLICIT NONE
      REAL(PREC), DIMENSION(:,:), INTENT(INOUT) :: u

      REAL(PREC), DIMENSION(SIZE(u,1),SIZE(u,2)) :: tu
      INTEGER :: m, n, k
        m=SIZE(u,1)
        n=SIZE(u,2)
        tu(1,1:n)=(u(1,1:n)+u(2,1:n))*0.5_prec
        tu(m,1:n)=(u(m-1,1:n)+u(m,1:n))*0.5_prec
        DO k=2,m-1
          tu(k,1:n)=(u(k-1,1:n)+2*u(k,1:n)+u(k+1,1:n))*0.25_prec
        END DO
        u(1:m,1)=(tu(1:m,1)+tu(1:m,2))*0.5_prec
        u(1:m,n)=(tu(1:m,n-1)+tu(1:m,n))*0.5_prec
        DO k=2,n-1
          u(1:m,k)=(tu(1:m,k-1)+2*tu(1:m,k)+tu(1:m,k+1))*0.25_prec
        END DO
    END SUBROUTINE smooth1
```

```
!c ******************************************
    SUBROUTINE split2(num,n,i,j)
      IMPLICIT NONE
      INTEGER,INTENT(IN) :: num,n
      INTEGER,INTENT(OUT) :: i,j

        j=MOD(num-1,n)+1
        i=(num-j)/n+1
    END SUBROUTINE split2

    SUBROUTINE split3(num,m,n,i,j,k)
      IMPLICIT NONE
      INTEGER,INTENT(IN) :: num,m,n
      INTEGER,INTENT(OUT) :: i,j,k

      INTEGER :: it
        it=num
        k=MOD(it-1,n)+1
        it=(it-k)/n+1
        CALL split2(it,m,i,j)
    END SUBROUTINE split3

    SUBROUTINE split4(num,m,n,r,i,j,k,l)
      IMPLICIT NONE
      INTEGER,INTENT(IN) :: num,m,n,r
      INTEGER,INTENT(OUT) :: i,j,k,l

      INTEGER :: it
        it=num
        l=MOD(it-1,r)+1
        it=(it-l)/r+1
        CALL split3(it,m,n,i,j,k)
    END SUBROUTINE split4


!*****************************************

    SUBROUTINE polint1(xa,ya,x,y,dy)
      IMPLICIT NONE
      REAL(PREC),DIMENSION(:),INTENT(IN)::xa,ya
      REAL(PREC),INTENT(IN)::x
      REAL(PREC),INTENT(OUT)::y,dy

      INTEGER::m,n,ns
      REAL(PREC),DIMENSION(size(xa))::c,d,den,ho
        CALL assert_eq(size(xa),size(ya),'polint')
```

```
        n=size(xa)
        c=ya
        d=ya
        ho=xa-x
        ns=iminloc(abs(x-xa))
        y=ya(ns)
        ns=ns-1
        do m=1,n-1
          den(1:n-m)=ho(1:n-m)-ho(1+m:n)
          if (any(den(1:n-m)==0.0_prec))&
            call Error('Calculation failure', 'Polint')
          den(1:n-m)=(c(2:n-m+1)-d(1:n-m))/den(1:n-m)
          d(1:n-m)=ho(1+m:n)*den(1:n-m)
          c(1:n-m)=ho(1:n-m)*den(1:n-m)
          if (2*ns <n-m)then
          dy=c(ns+1)
          else
          dy=d(ns)
          ns=ns-1
          end if
          y=y+dy
        end do
    END SUBROUTINE polint1


!*****************************************

    FUNCTION iminloc1(arr)
      REAL(PREC), DIMENSION(:), INTENT(IN) :: arr
      INTEGER, DIMENSION(1) :: imin
      INTEGER :: iminloc1
        imin=minloc(arr(:))
        iminloc1=imin(1)
    END FUNCTION iminloc1
!c *****************************************

    FUNCTION iminloc2(arr)
      REAL, DIMENSION(:), INTENT(IN) :: arr
      INTEGER, DIMENSION(1) :: imin
      INTEGER :: iminloc2
        imin=minloc(arr(:))
        iminloc2=imin(1)
    END FUNCTION iminloc2
!c *****************************************

    SUBROUTINE polint2(xa,ya,x,y,dy)
```

```
      IMPLICIT NONE
      REAL,DIMENSION(:),INTENT(IN)::xa,ya
      REAL,INTENT(IN)::x
      REAL,INTENT(OUT)::y,dy

      INTEGER::m,n,ns
      REAL,DIMENSION(size(xa))::c,d,den,ho
        CALL assert_eq(size(xa),size(ya),'polint')
        n=size(xa)
        c=ya
        d=ya
        ho=xa-x
        ns=iminloc(abs(x-xa))
        y=ya(ns)
        ns=ns-1
        do m=1,n-1
          den(1:n-m)=ho(1:n-m)-ho(1+m:n)
          if (any(den(1:n-m)==0.0))&
            call Error('Calculation failure', 'Polint')
          den(1:n-m)=(c(2:n-m+1)-d(1:n-m))/den(1:n-m)
          d(1:n-m)=ho(1+m:n)*den(1:n-m)
          c(1:n-m)=ho(1:n-m)*den(1:n-m)
          if (2*ns <n-m)then
          dy=c(ns+1)
          else
          dy=d(ns)
          ns=ns-1
          end if
          y=y+dy
        end do
    END SUBROUTINE polint2
!c ***********************************************

    FUNCTION geop1(first, factor, n)
      REAL(PREC), INTENT(IN) :: first, factor
      INTEGER, INTENT(IN) :: n
      REAL(PREC), DIMENSION(n) :: geop1

      INTEGER :: k, k2
      REAL(PREC) :: temp
        IF(n>0) geop1(1)=first
        DO k=2, n
          geop1(k)=geop1(k-1)*factor
        END DO
    END FUNCTION geop1
```

```
!C:*****************************************
 END MODULE util
```

GRADUATE SCHOOL
UNIVERSITY OF ALABAMA AT BIRMINGHAM
DISSERTATION APPROVAL FORM
DOCTOR OF PHILOSOPHY

Name of Candidate Aimin Yan

Graduate Program Applied Mathematics

Title of Dissertation An Inverse Groundwater Model

I certify that I have read this document and examined the student regarding its content. In my opinion, this dissertation conforms to acceptable standards of scholarly presentation and is adequate in scope and quality, and the attainments of this student are such that he may be recommended for the degree of Doctor of Philosophy.

Dissertation Committee:

Name	Signature
Ian W. Knowles, **Chair**	Ian Knowles
Robert M. Hyatt	Robert M Hyatt
Tsun-Zee Mai	
S. S. Ravindran	S. S. R
Yanni Zeng	Yanni Zeng

Director of Graduate Program Ruth Williams

Dean, UAB Graduate School

Date