Deflated CG Method for Modelling Groundwater Flow in a Layered Grid

Master's Thesis

L.A. Ros

Delft, Utrecht

September 2007 - July 2008

Delft University of Technology | Deltares

Thesis Committee: Prof.dr.ir. C. Vuik (Delft University of Technology)
Dr M. Genseberger (Deltares, Delft)
Ir. J. Verkaik (Deltares, Utrecht)
Dr H.M. Schuttelaars (Delft University of Technology)

Preface

This Master's thesis is written for the degree of Master of Science for the study Applied Mathematics, faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology. The graduation work is done in the chair of Numerical Mathematics of the department of Applied Mathematical Analysis. During ten months, the Master's project has been carried out at Deltares.

Deltares is a new Dutch institute for Delta Technology, starting on 1 January 2008. Deltares is formed from parts of Rijkswaterstaat /DWW, RIKZ and RIZA, WL | Delft Hydraulics, GeoDelft, and a part of TNO Built Environment and Geosciences. Together, they work on problems in the field of water, soil and the subsurface. Deltares works for and cooperates with Dutch government, provinces and water boards, international governments, knowledge institutes and market parties. The institute is located in two cities: Delft and Utrecht.

From September 2007 till July 2008 I have been working on a groundwater model for Limburg called IBRAHYM, developed by the former TNO. My goal was to improve the convergence behaviour of the solver near faults. During some reseach we observed that the faults has less effect on the convergence than we thought in advance. It turned out that the thick layers of clay in the subsoil were the largest causes of the bad convergence behaviour. By applying a deflation based preconditioner this can be solved.

Before I go on with the report I would like to thank some people. First of all I would like to thank my supervisor from the TU Delft, Prof.dr.ir. C. Vuik. He helped me through the whole project by always giving me new ideas about how to solve the problems. Also I would like thank my both supervisors from Deltares, Menno Genseberger and Jarno Verkaik for their excellent supervising during the graduation work. Menno always asked me to motivate my decisions which kept me really sharp. Jarno helped me during the frustrating moments with programming Fortran and almost always had time to check the code together with me when there was something wrong.

Furthermore, I would like to express my gratitude towards my colleagues at Deltares. Especially Peter Vermeulen who always was willing to help me if I had problems concerning IMOD and who also had some good ideas during my research and implementation. Also I would like to thank all my family and friends who supported me during my thesis. Last, but certainly not least, I would like to thank my girlfriend who always has supported me during my study and my Master's thesis. Especially the last couple of weeks during my thesis she motivated me and helped me to finish it.

Lennart Ros,

Delft, Utrecht, July 2008.

Summary

At Deltares a high resolution model, called IBRAHYM, is developed to calculate the groundwater heads in Limburg. The equations used are based on Darcy's Law and are discretized by using a finite volume method. The resulting system of equations is now solved by using a Modified Incomplete Cholesky Conjugate Gradient method (MICCG).

In Limburg there are many faults in the subsoil and thick packages of clay between aquifers, and therefore we have to deal with large contrasts in the medium parameters. For such contrasts the MICCG method seems to fail, resulting in bad convergence behaviour.

The goal of this Master's thesis was to improve the convergence behaviour and reduce wall-clock times. This has been done by using a deflation based preconditioner, that improves the condition number of the system matrix resulting in less iterations.

During the implementation we found some practical problems. The biggest problem was the memory usage. It turned out that there was not much memory left to apply the deflation preconditioner. So we had to think of a method which does not use too much memory. We decided to use a subdomain deflation based on the layers. Using this method we can do the calculation without using too much memory because we do not need to store matrix Z and we store only a part of AZ using only one vector. Indeed we observed that we needed less iterations, especially in the areas that showed bad convergence. Also wall-clock times were gained.

If we optimize the code for Fortran, we might be able to win even more wall-clock time. We also concluded that the original solver as it is implemented now, is far from optimal because they use much if-loops which are slowing things down. So a recommendation is to rewrite the solver without using if-loops for checking the ibounds. If we minimize the if-loops, then there is a big change the whole model is a lot faster. Another recommendation is to use more deflation vectors. We can split each layer in more blocks. In this way we might be able to split "the bad convergence areas" from the good ones and win more iterations.

It can be concluded that the main goal of the project, improving convergence, has been achieved. In areas where there was a bad convergence behaviour, we can win a lot of iterations. In the areas where the original solver already worked well we also can gain some iterations. Concerning wall-clock times it is recommended to further improve the code.

Contents

Chapter 1

Introduction

In densely populated areas, land use and planning are closely related to demands on water management of, for example, natural, agricultural and recreational areas. It is therefore important to base management of both groundwater and surfacewater systems on these demands.

Modelling groundwater flow can be done by the finite-volume code named MODFLOW [5], designed by the U.S. Geological Survey (USGS) [17], that is commonly used in the world. USGS calls it a finite-difference method, but we will refer to it as a finite-volume method since they use finite volumes to discretize the governing equations. Deltares uses their own finite-volume code IMODFLOW which is strongly based on MODFLOW. The equations used for solving groundwater flow are based on Darcy's Law. This code uses a conjugate gradient solver with an Incomplete Cholesky preconditioning (ICCG).

Deltares has participated in the development of a model for Limburg and its surroundings, called IBRAHYM. In Limburg we deal with faults in the subsoil, and we have thick packages of clay in the subsoil. Both the faults and the packages of clay result in a large contrast in the medium parameters. The preconditioned solver seems to fail for applications with large contrasts in medium parameters, resulting in poor convergence behavior. A possible technique to improve convergence is to use a deflation-based preconditioner. Such preconditioned solvers already proved their value for a variety of applications in the past. Since we want to calculate detailed solutions for large areas we deal with very large systems of equations. Therefore a lot of cells are required resulting in a large memory usage.

The main goal of this Master's thesis is to do something about the bad convergence behaviour of the MODFLOW solver for the IBRAHYM model. To achieve this we combine the ICCG method with a deflation based preconditioner, without using too much extra memory. Besides that we also hope to win some wall-clock time.

The structure of this report is as follows. In Chapter 2 we give a short introduction in the geohydrology. After that we derive the equations involving porous media flow. We describe how these equations are discretized in Chapter 3. These discretized equations are used in MODFLOW to solve the problem. The basics for solving such a discretized equation are given in Chapter 4. Here we introduce the preconditioned conjugate gradient method (PCG) to solve such an equation. Also we describe the Incomplete Cholesky preconditioner (ICCG), which is used by MODFLOW. In Chapter 5 we give an introduction into a new preconditioner technique, called deflation. We describe how deflation works and how we can use it to improve convergence of the ICCG solver. We end this chapter with a simple example in Matlab, to give some insight in the results of deflation. Then in Chapter 6 we finally come to the IBRAHYM model. We describe how we implemented the deflation technique in the existing solver and we give some

results of applications to the IBRAHYM model. Chapter 7 finishes this report with conclusions and some recommendations for future research.

Chapter 2

Modelling Porous Media Flow

Groundwater flow is three dimensional in space and it is also a function of time. The velocity of the flow is a function of x, y, z, and t.

The flow can be evaluated quantitively when the velocity, pressure, density, temperature, and viscosity of the water flowing through the ground are known. If these variables are only functions of space, then the flow is defined as being *steady*. If the variables are also functions of time, then the flow is defined as being unsteady or time-dependent.

Section 2.1 gives an overview about the geology and hydrology basics for modelling groundwater flow. In Section 2.2 we explain Darcy's Law, on which the final equations are based. Section 2.3 describes the basic steady-state flow equation and in Section 2.4 we describe how to handle with time-variant problems. So in this chapter we will derive the fundamental equation based on Darcy's Law. The derivation can also be found in Chapter 2 of [10].

2.1 Basics for Modelling Groundwater Flow

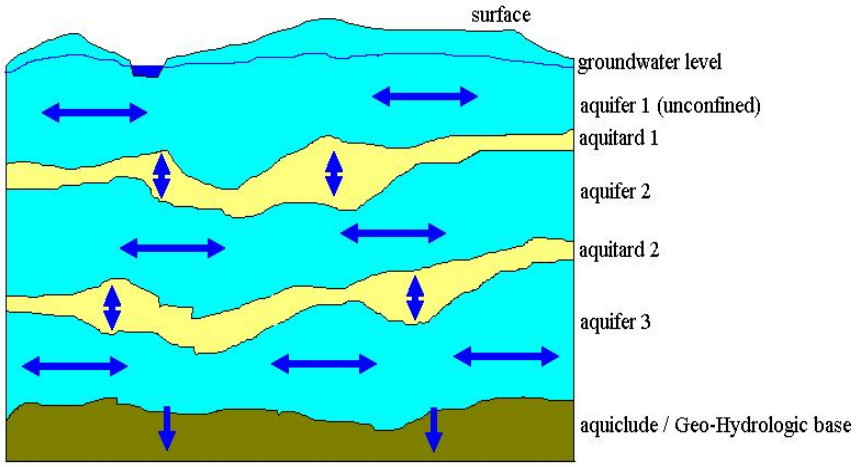

Figure 2.1: Schematization of the subsurface and the possible directions of the groundwater flow.

In this section a short overview is given about the geology and hydrology basics for modelling groundwater flow. The subsurface consists of different layers of alternating sand and clay. Directly under the surface we have an unconfined aquifer. An aquifer is a sediment that is sufficiently porous and permeable to store and transmit groundwater. In an aquifer water can move in all directions, but usually the flow is horizontal. After an aquifer we find an aquitard. An aquitard is a sediment that is not permeable enough to let water flow easily, but it can supply water to the underlying or overlaying aquifer. In an aquitard we assume water to flow vertically only. An example of an aquitard is a package of clay. After some of those aquifers and aquitards we find the aquiclude or the so called geo-hydrological base. This is an impermeable body of rock that may absorb water slowly, but does not transmit it. A simple overview is given in Figure 2.1.

Hydrology describes the movement of water in all it forms. The cycle is shown in Figure 2.2. It can be seen that a lot of processes deal with groundwater.

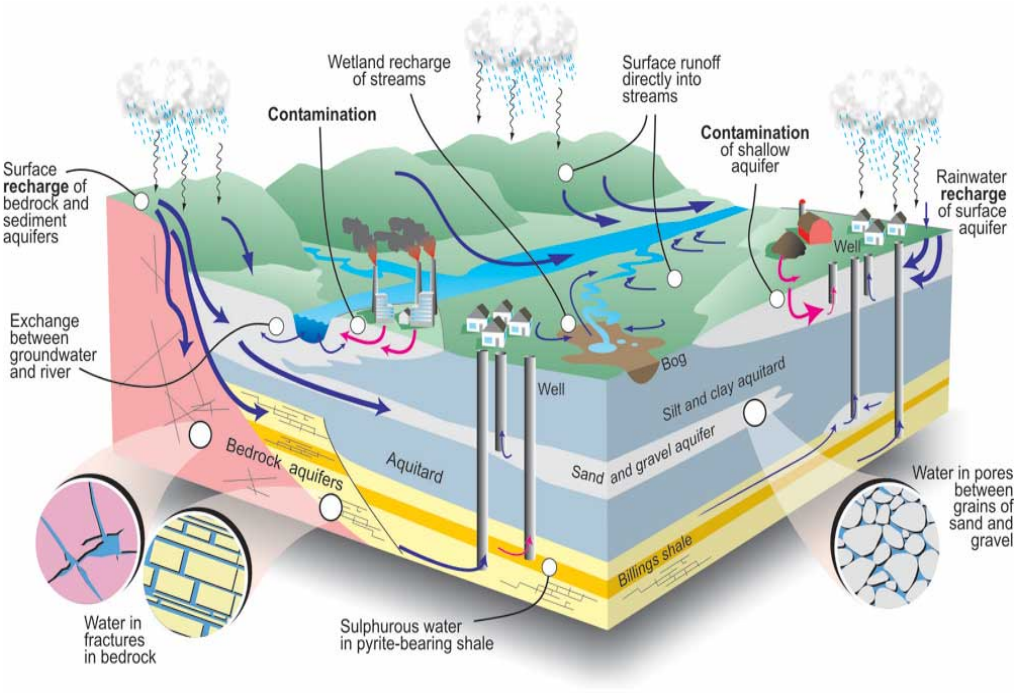

Figure 2.2: Overview of the hydrology cycle.

Geohydrology describes the groundwater flow. Almost everywhere in the Netherlands groundwater can be found up to a couple of meters beneath the surface. In the lower parts of the Netherlands it can even be found within a meter. Groundwater moves through the pores of the aquifers. Not every type of ground has the same porosity. Porosity is defined as the volume of the pores divided by the volume of the material. See also Figure 2.3. Conductivity K is a measure for the possibility of movement of a liquid through the pores of a sediment.

To separate the effect of the medium and the liquid we define the intrinsic permeability, k, as:

$$k = \frac{K\mu}{\gamma},$$

where:

μ dynamic viscosity [$ML^{-1}T^{-1}$],
γ specific weight ($\rho\, g$) [$ML^{-2}T^{-2}$],
ρ specific density [ML^{-3}]
g force of attraction [LT^{-2}]

The driving force for groundwater flow are the differences in height and pressure. To represent these differences we introduce the concept of hydraulic heads, h $[L]$. This groundwater potential equals the sum of the kinetic energy, the pressure and the potential energy heads:

$$h = \frac{v^2}{2g} + \frac{p}{\rho g} + z, \tag{2.1}$$

where ρg is the specific weight of the liquid. The term h is sometimes called the *groundwater head* or the *piezometric head*. The first term, the kinetic energy, is in practice negligible. So Equation (2.1) reduces to:

$$h = \frac{p}{\rho g} + z.$$

We can use a piezometer to determine the hydraulic head (see Figure 2.6).

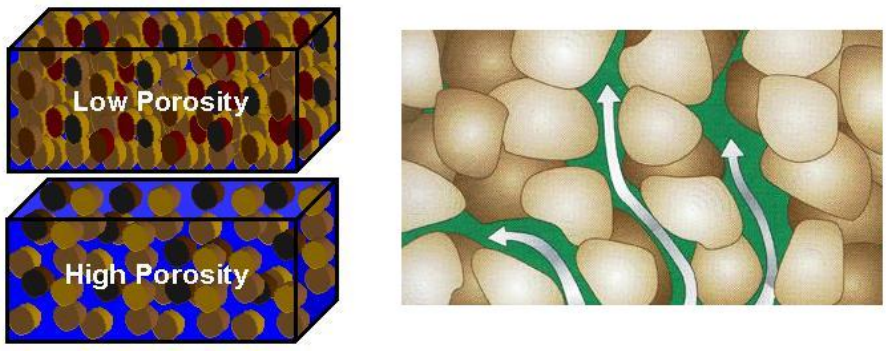

Figure 2.3: Left: Low and high porosity. Right: Connected pores give a rock permeability.

2.2 Darcy's Law

The flow through the porous media is a slow frictional flow governed by Darcy's law which we can write as:

$$v = K \cdot i. \tag{2.2}$$

Here:

v = Darcy's velocity $[LT^{-1}]$,
K = matrix with hydraulic conductivities of medium $[LT^{-1}]$,
i = hydraulic gradient.

The tensor K is given by:

$$K = \begin{bmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{bmatrix}. \tag{2.3}$$

Each element of K is defined as: $K = \frac{k\gamma}{\mu}$, where:

k = intrinsic permeability $[L^2]$,
γ = specific weight (ρg) $[ML^{-2}T^{-2}]$,
μ = dynamic viscosity $[ML^{-1}T^{-1}]$.

From a macroscopic point of view it can be shown that K is symmetric and positive definite. Symmetric means that we have $k_{xy} = k_{yx}, k_{zx} = k_{xz}, k_{yz} = k_{zy}$ and positive definite implies that $k_{xx}, k_{yy}, k_{zz} > 0$. This assures that energy is always dissipated during flow. MODFLOW assumes a quasi 3D-flow, i.e. the head gradient does not vary vertically within aquifers and does not vary horizontally within aquitards. These kind of flows are also know as *Dupuit-Forchheimer flows*. As a consequence of this, the components $k_{xz} = k_{zx} = k_{yz} = k_{zy} = 0$ and so will the corresponding K's [15]. In MODFLOW we also assume that $k_{xy} = k_{yx} = 0$, so we only model anisotropy along the main axes. If the material is isotropic we have $K_{xx} = K_{yy} = K_{zz}$, but if the material is anisotropic these values can differ.

The hydraulic gradient i results from a difference in groundpotential across an element of the medium. If Δh represents the total potential loss of the fluid over a distance Δs, then in the limit we have:

$$i = -\frac{dh}{ds}. \tag{2.4}$$

If we now substitute Equations (2.3) and (2.4) into Equation (2.2), then we end up with the following equations:

$$\begin{aligned} v_x &= -K_{xx}\tfrac{\partial h}{\partial x}, \\ v_y &= -K_{yy}\tfrac{\partial h}{\partial y}, \\ v_z &= -K_{zz}\tfrac{\partial h}{\partial z}, \end{aligned} \tag{2.5}$$

where K_{xx}, K_{yy}, K_{zz} are the values of hydraulic conductivity along x, y and z coordinate axes. We assume axes to be parallel to the major axes of hydraulic

conductivity. The lowercase x, y, and z at the v's mean that it is the velocity in that direction.

Note that these equations only hold when the x, y, and z axes coincide with principal permeability axes. Darcy's Law is not sufficient to solve the problem of groundwater flow. We now have only three equations, but we have four unknowns. These unknowns are the three components of the velocity and the groundwater potential. So we need a fourth equation, which we obtain by realizing that the flow has to satisfy the principle of conservation of mass. No matter what kind of flow we are looking at, we cannot gain or loose mass. This will lead to a continuity equation.

2.3 Steady-state Flow

Consider a small elementary volume (see Figure 2.4). Assume that the fluid filters in such a manner that the porous ground is incompressible. We can use the law of conservation of mass. This law states that the sum of mass flow entering the three faces is equal to the sum of the mass flow leaving by the opposite faces. Since we are working with water, which has a constant density, the conservation principle can be fulfilled by ensuring that the volume entering equals the volume that leaves the volume. We also assume the water to be incompressible.

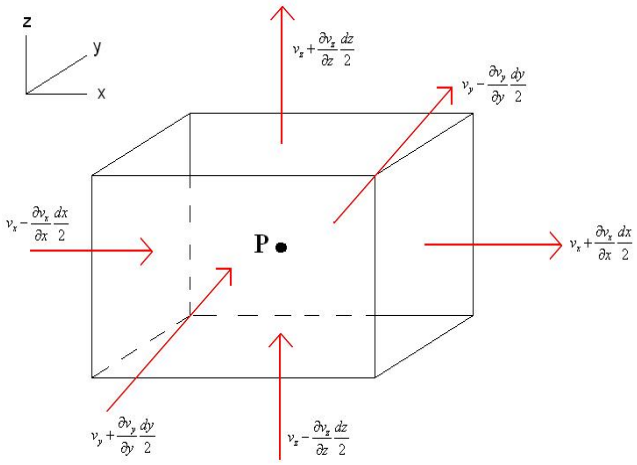

Figure 2.4: Flow for an elementary volume of fluid.

Now let P be the center point of such an element and let for the moment $x = y = z = 0$. At this point, the velocities are given by v_x, v_y and v_z. So on $x = -\frac{dx}{2}$, the flow in the x-direction will be $v_x\,(x - dx/2) \approx v_x - \frac{\partial v_x}{\partial x}\frac{dx}{2}$. We use here a first order Taylor approximation. If we multiply this by the area, which is $dy\,dz$, we end up with the volume entering the plane through this area. So in the x-direction the following volume will enter the element:

$$\left(v_x - \frac{\partial v_x}{\partial x}\frac{dx}{2}\right) dy\,dz.$$

For the other area in the x-direction fluid will leave the element. The volume leaving the element will be:

$$\left(v_x + \frac{\partial v_x}{\partial x}\frac{dx}{2}\right) dy\,dz.$$

So the net flow, which is outflow minus inflow, in the x-direction will be:

$$\left(\frac{\partial v_x}{\partial x}\right) dx\, dy\, dz.$$

Similarly we have a net flow in the y-direction of $\left(\frac{\partial v_y}{\partial y}\right) dx\, dy\, dz$ and in the z-direction of $\left(\frac{\partial v_z}{\partial z}\right) dx\, dy\, dz$. So the volume of the flow through the elementary volume will then be:

$$\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}\right) dx\, dy\, dz.$$

According to the conservation principle this must be equal to 0, so we must have:

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} = 0. \tag{2.6}$$

We call this the *steady-state continuity equation*.

If we now substitute Equation (2.5) into (2.6) we end up with:

$$\frac{\partial}{\partial x}\left(K_{xx}\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_{yy}\frac{\partial h}{\partial y}\right) + \frac{\partial}{\partial z}\left(K_{zz}\frac{\partial h}{\partial z}\right) = 0, \tag{2.7}$$

which describes basic steady-state flow without source and sink terms.

2.3.1 Boundary Conditions

Boundary conditions are needed to derive a unique solution for a partial differential equation. Various types of boundary conditions are possible in a steady state flow:

1. Known values of the groundwater potential are given on the boundary (Dirichlet condition).
2. If the magnitude of the flow crossing through the boundary is known we can set: $\frac{\partial h}{\partial n} = \phi$, where ϕ is the velocity normal to the boundary divided by the permeability normal to the boundary (inhomogeneous Neumann condition).
3. At an impermeable boundary we do not have a flow through this boundary. So we set: $\frac{\partial h}{\partial n} = 0$, where n is the direction normal to the boundary (homogeneous Neumann condition).)
4. Head-dependent boundary conditions (Cauchy condition). How we deal with these boundary conditions in MODFLOW is described in Section 3.4.1.

2.4 Time-variant Problems

Most groundwater problems are also time dependent. If we look also at the dependence of time, we must determine if we have a confined or an unconfined aquifer. We call groundwater flow confined when all the boundaries or bounding surfaces of the space through which the flow goes, are fixed in space for different states of the flow. See the left picture of Figure 2.5. If the groundwater potential goes a little bit up or down, our

aquifer is still completely filled with water. Groundwater flow is unconfined when it possesses a free surface, so if the position of the boundary varies with the state of the flow. See also the right picture in Figure 2.5. Since the groundwater potential lies within our aquifer we are facing a moving boundary here.

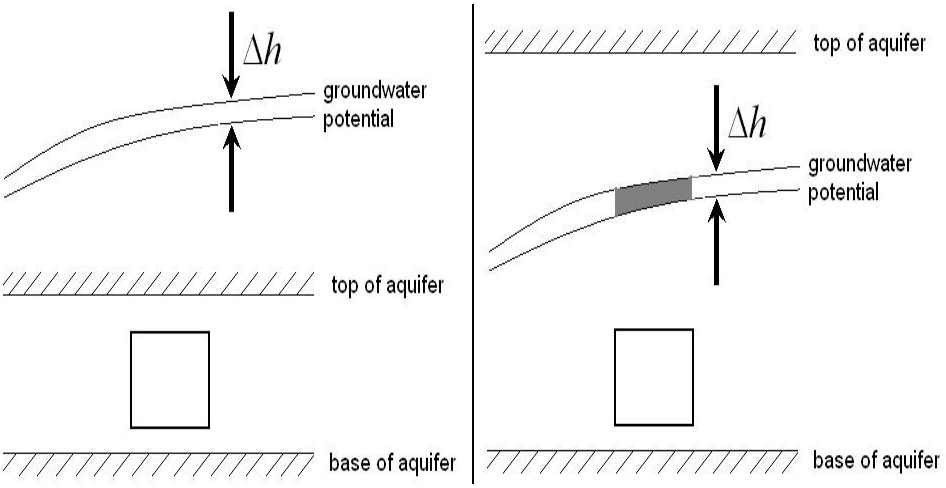

Figure 2.5: A confined (left) and an unconfined (right) aquifer.

For this project we restrict ourself to confined aquifers. A drop in the groundwater potential of Δh results in a reduction in pressure. The volume of water released *per unit volume* of aquifer due to a unit decrease in head will be named the *specific storage coefficient* S_s. If we look at the mass balance for a saturated element in a confined aquifer, we see that both the compressibility of the water and the change in pore volume due to the vertical compression of the aquifer contribute to the specific storage. So we conclude that the specific storage is a function of the density of water, the porosity, the pore volume compressibility and the compressibility of water. Although we can set up an expression for the specific storage, usually it is determined from field measurements. In general S_s is within the range of 10^{-5} to 10^{-7}. The dimension of the specific storage is $[L^{-1}]$.

We can extend the steady-state continuity equation (Equation 2.6) from Section 2.3 to obtain a differential equation including the effect of the specific storage. The effects of the compressibility and density are included in the specific storage, so we are allowed to work in terms of the conservation of volume.

If we do the same derivation as in Section 2.3 we see that the net volume leaving the element during a time δt due to changing velocities equals:

$$dx\,dy\,dz\left(\frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y}+\frac{\partial v_z}{\partial z}\right)\delta t.$$

During the time δt, the groundwater potential at point P increases by δh. So the volume of water taken into storage in groundwater potential due to this increase is

$$dx\,dy\,dz\,S_s\,\delta h.$$

Since we have the continuity principle these two equations must sum to zero, so we end up with:

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} = -S_s \frac{\partial h}{\partial t}.$$

If we now substitute Equation (2.5) again, we end up with the following differential equation, which holds for any element within the saturated aquifer:

$$\frac{\partial}{\partial x}\left(K_{xx}\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_{yy}\frac{\partial h}{\partial y}\right) + \frac{\partial}{\partial z}\left(K_{zz}\frac{\partial h}{\partial z}\right) = S_s \frac{\partial h}{\partial t}. \tag{2.8}$$

In the next chapter we are going to discretize this equation by using a finite volume method.

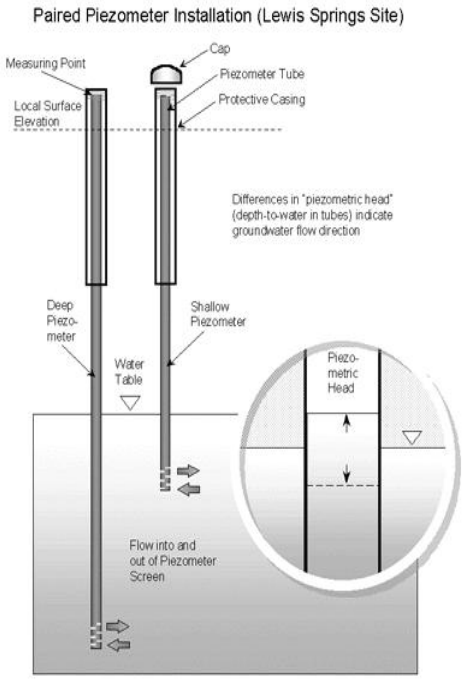

Figure 2.6: A piezometer (Figure taken from [20]).

Chapter 3

Finite Volume Method in MODFLOW

In general it is impossible to derive an analytical solution for the flow equation as given by Equation (2.8). Fortunately we are able to determine a solution by using numerical methods. To do this we use a software package called MODFLOW. MODFLOW is a software package which calculates hydraulic heads, developed by the U.S. Geological Survey. The model program uses a modular structure wherein similar program functions are grouped together, and specific computational and hydrologic options are constructed in such a manner that each option is independent of other options. Because of this structure it is possible to add new options without the necessity of changing existing subroutines. A complete manual of MODFLOW can be found on the internet [5].

The method used in MODFLOW is a finite volume method. In this chapter the main properties and details are described. In Section 3.1 we describe the grid as used by MODFLOW. Then in Section 3.2 we give some main and important properties of MODFLOW. Section 3.3 gives the derivation of the discretized equations and ends with the system of equation we need to solve. In Section 3.4 the simulation of the boundaries is discussed. The last section, Section 3.5, discusses how MODFLOW deals with faults in the subsoil and with the layers of clay.

3.1 Grid

In Figure 3.1 we see a three-dimensional spatial discretization of an aquifer system as used in MODFLOW. We have a grid of blocks, which we will call cells from now on. The location of such a cell is described in terms of rows, columns, and layers. We use an i, j, k indexing system, where $i = 1, 2, \ldots, NROW$ is the row index, $j = 1, 2, \ldots, NCOL$ is the column index and $k = 1, 2, \ldots, NLAY$ is the layer index. The convention followed in this model is to number the layers from top to bottom. So, an increase in k means a decrease in z. Rows are considered parallel to the x-axis, such that an increase in i corresponds with a decrease in y. Columns are considered to be parallel to the y-axis, such that an increase in j corresponds to an increase in x.

Within each cell there is a point in which head must be calculated. This point is called *node*. We will use a cell-centered formulation in which the nodes are in the center of the cells. Since we assume only horizontal flow in an aquifer, each layer contains only of one cell in vertical direction. So if we have 19 layers schematized, then $NLAY$ is equal to 19.

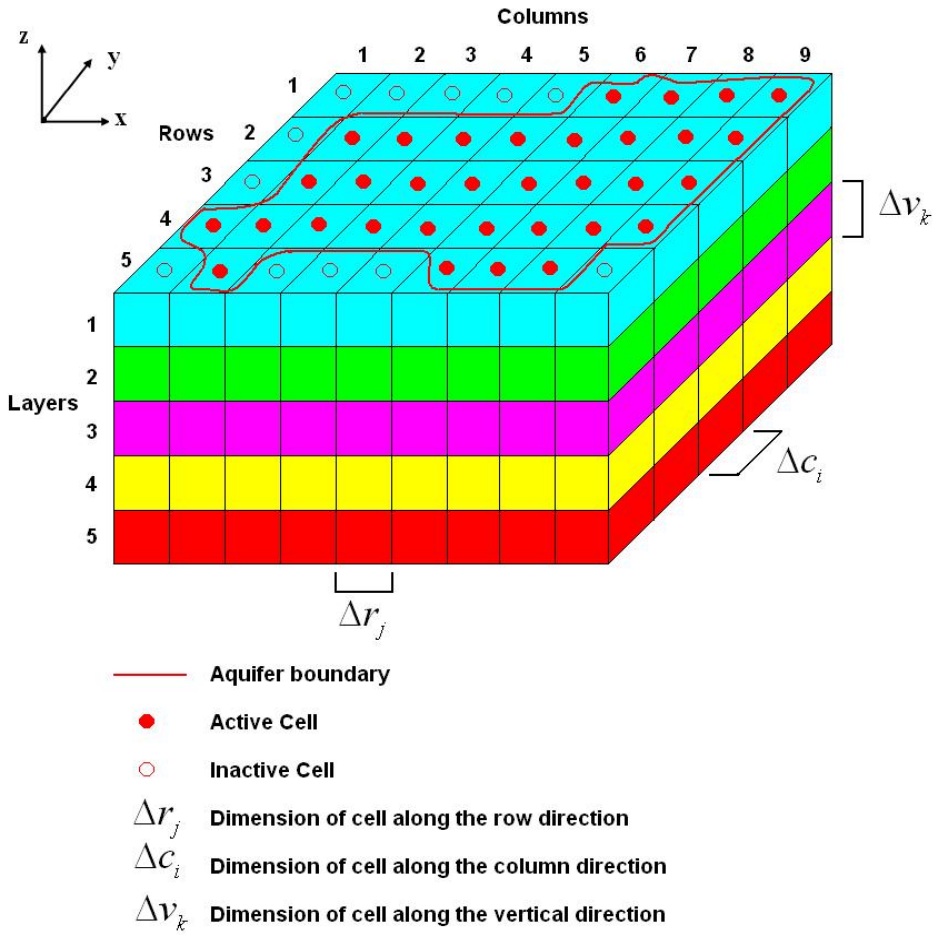

Figure 3.1: Three-dimensional grid.

If we use the conventions used in Figure 3.1 we see that the width in the row direction at a given column j is denoted by Δr_j. The width of a given row i in the column direction is denoted Δc_i. The thickness of cells in a given layer k is denoted Δv_k. So the cell with coordinates (i, j, k) has a volume of $\Delta r_j \, \Delta c_i \, \Delta v_k$. We assume that the grid is rectangular both in horizontal and vertical direction.

3.2 Main Properties of MODFLOW

MODFLOW works on a rectangular grid and uses cell-centered variables. In Section 3.1 is described how this grid is set up. The cells we use have three dimensions and they are numbered using an (i, j, k) grid. The i stands for the number of the column, the j is for the number of the row and k is the number of the layer. Since in real applications the groundstructure is not rectangular, you have to specify in every cell the transmissivity. Transmissivity is denoted by TR in the row direction and by TC in the column direction. Transmissivity is some measure for permeability, corrected for the real dimension represented by the cell. This value is also called kD-value, since you multiply the conductivity with the thickness (Dutch: dikte) of the layer. The conductance for a cell is now calculated as the transmissivity times the width of the cell divided by its length. The volumetric flow, Q, now is defined by:

$$Q = C(h_1 - h_2). \tag{3.1}$$

Conductance is defined for a particular prism of material and for a particular direction of flow. If a prism of porous material consists of two or more subprisms in series and for all the subprisms the conductance is known, then the conductance representing the entire prism can be calculated. We know that we have:

$$C = \frac{Q}{h_A - h_B}.$$

If we assume that the heads across each subprism are continuous, we get the identity:

$$\sum_{i=1}^{n} \Delta h_i = h_A - h_B.$$

Substituting Equation (3.1) gives:

$$\sum_{i=1}^{n} \frac{q_i}{c_i} = h_A - h_B.$$

where:

q_i is the flow across subprism i, and
c_i is the conductance of subprism i.

Since we consider flow in one direction and we assume no accumulation or depletion in storage, each q_i is equal to the total flow Q and therefore:

$$Q \sum_{i=1}^{n} \frac{1}{c_i} = h_A - h_B \text{ and } \frac{h_A - h_B}{Q} = \sum_{i=1}^{n} \frac{1}{c_i}. \tag{3.2}$$

Comparing Equation (3.2) and Equation (3.1), one can conclude that:

$$\frac{1}{C} = \sum_{i=1}^{n} \frac{1}{c_i},$$

which is called the harmonic mean. So if we only have two subprisms the equivalent conductance reduces to:

$$C = \frac{c_1 c_2}{c_1 + c_2}. \tag{3.3}$$

If we look into the row-direction , we get for the conductance between cell (i, j, k) and cell $(i, j + 1, k)$ (see also Figure 3.2):

$$CR_{i,j+1/2,k} = \frac{\frac{TR_{i,j,k}DELC_i}{(1/2)DELR_j} \times \frac{TR_{i,j+1,k}DELC_i}{(1/2)DELR_{j+1}}}{\frac{TR_{i,j,k}DELC_i}{(1/2)DELR_j} + \frac{TR_{i,j+1,k}DELC_i}{(1/2)DELR_{j+1}}}, \tag{3.4}$$

where the R stands for row-direction. Simplification of Equation (3.4) gives the final equation:

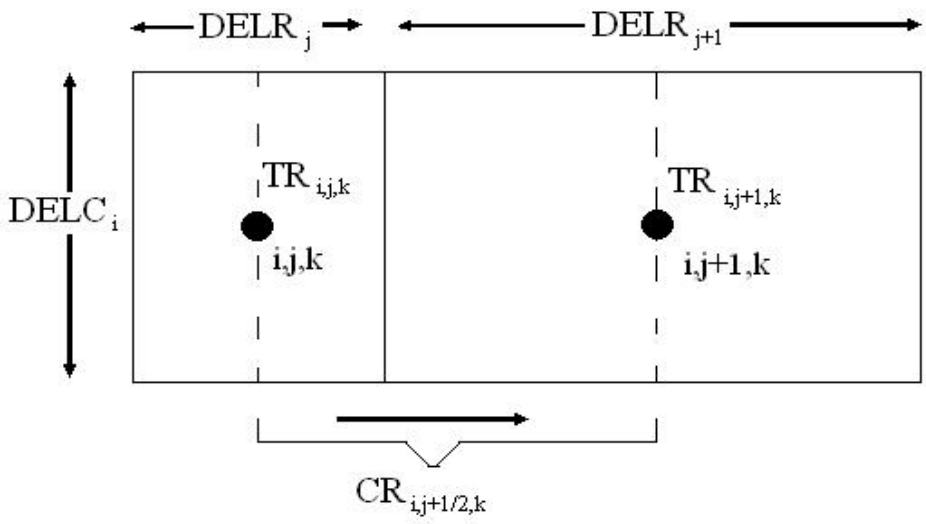

Figure 3.2: Calculation of conductance between two cells using transmissivities.

$$CR_{i,j+1/2,k} = 2DELC_i \frac{TR_{i,j,k}TR_{i,j+1,k}}{TR_{i,j,k}DELR_{j+1} + TR_{i,j+1,k}DELR_j}. \tag{3.5}$$

The same process can be applied to the calculation of $CC_{i+1/2,j,k}$ (C for column) giving:

$$CC_{i+1/2,j,k} = 2DELR_j \frac{TC_{i,j,k}TC_{i+1,j,k}}{TC_{i,j,k}DELC_{i+1} + TC_{i+1,j,k}DELC_i}. \tag{3.6}$$

Vertical conductance can also be calculated in the same way as the horizontal conductances. We only give the result:

$$CV_{i,j,k+1/2} = \frac{DELR_jDELC_i}{\frac{(1/2)DELV_k}{VK_{i,j,k}} + \frac{(1/2)DELV_{k+1}}{VK_{i,j,k+1}}}. \tag{3.7}$$

where VK is the vertical (hydraulic) conductivity between layers.

MODFLOW is a quasi-3D model since it models the aquitards as a vertical flux between layers. So it does not calculate anything inside an aquitard. Because of this we have to adapt Equation (3.7). An exact derivation can be found in [5]. We only give the result:

$$CV_{i,j,k+1/2} = \frac{DELR_jDELC_i}{\frac{(1/2)DELV_k}{VK_{i,j,k}} + \frac{\Delta v_{CB}}{VKCB_{i,j,k}} + \frac{(1/2)DELV_{k+1}}{VK_{i,j,k+1}}}, \tag{3.8}$$

where:

$VKCB_{i,j,k}$	the hydraulic conductivity of the aquitard between (i,j,k) and (i,j,k+1)
Δv_{CB}	the thickness of the aquitard.

The term $\frac{\Delta v_{CB}}{VKCB_{i,j,k}}$ compensates for the conductance in the aquitard between layer k and $k+1$.

MODFLOW uses an iterative method to solve a system of equations obtained from the finite-volume discretization as carried out in Section 3.3. To solve the derived system of equations a preconditioned conjugate gradient method is used to improve convergence. As a preconditioner, the modified incomplete Cholesky Factorization is used. More details on such solvers can be found in Chapter 4.

3.3 Discretization

In Chapter 2 we derived that the three-dimensional movement of groundwater of constant density through porous media can be described by partial-differential Equation (2.8). Since we also want to study the effects of sources and sinks of water, we add the term W to the equation. We then have the following equation:

$$\frac{\partial}{\partial x}\left(K_{xx}\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial y}\left(K_{yy}\frac{\partial h}{\partial y}\right)+\frac{\partial}{\partial z}\left(K_{zz}\frac{\partial h}{\partial z}\right)+W=S_s\frac{\partial h}{\partial t}, \tag{3.9}$$

where:

K_{xx}, K_{yy}, K_{zz}	the values of hydraulic conductivities along x,y, and z coordinate axes (we assume them to be parallel to the major axes of hydraulic conductivity) [LT^{-1}],
h	potentiometric head [L],
W	volumetric flux per unit volume representing sources and sinks of water [T^{-1}],
S_s	specific storage of porous material [L^{-1}],
t	time [T].

For W we define a flow out of the groundwater system as $W < 0$ and a flow into the system as $W > 0$.

In general S_s, K_{xx}, K_{yy}, and K_{zz} are functions of space where W may be a function of space and time. If we specify the flow and/or head conditions at the boundaries of an aquifer system and we specify the initial-head conditions, we have a mathematical representation of a groundwater flow system. Except for very simple systems it is impossible to determine analytic solutions for such a problem. So we must use numerical methods to obtain an approximate solution. For this purpose we use the finite volume method.

3.3.1 Finite-Volume Equation

We use the continuity equation to develop the finite-volume form of the groundwater flow equation. The sum of all flow into and out of the cell must equal the rate of storage within the cell. We multiply Equation (3.9) by the volume of a cell (ΔV). If we assume the density to be constant we have for a cell:

$$\sum Q_i = SS\frac{\Delta h}{\Delta t}\Delta V, \tag{3.10}$$

where:

Q_i	flow rate into the cell [L^3T^{-1}],
SS	specific storage, the volume of water that can be injected per unit of volume of aquifer material per unit of change in head [L^{-1}],
ΔV	volume of the cell [L^3],
Δh	change in head over a time interval of length Δt [L].

The right hand term is equivalent to the volume of the water taken into storage over a time interval Δt given a change in head of Δh. Outflows and loss are represented in Equation (3.10) by defining outflow as negative inflow and loss as negative gain.

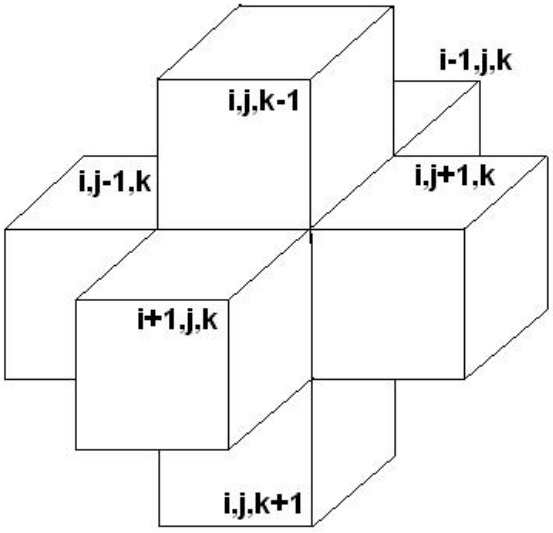

Figure 3.3: Indices for the six cells surrounding cell (i, j, k).

In Figure 3.3 we see the six aquifer cells adjacent to cell (i, j, k). For simplification we consider flow positive if they enter a cell. The negative sign usually incorporated in Darcy's Law has been dropped from all terms. If we now follow these conventions, the flow into cell (i, j, k) in the row direction from cell $(i, j-1, k)$ is given by Darcy's Law as (see also Figure 3.4):

$$q_{i,j-1/2,k} = KR_{i,j-1/2,k}\Delta c_i \Delta v_k \left(\frac{h_{i,j-1,k} - h_{i,j,k}}{\Delta r_{j-1/2}} \right), \tag{3.11}$$

where:

$h_{i,j,k}$	head at node i, j, k,
$q_{i,j-1/2,k}$	volumetric flow rate through the face between cells (i, j, k) and $(i, j-1, k)$ [L^3T^{-1}] (blue area in Figure 3.4),
$KR_{i,j-1/2,k}$	hydraulic conductivity along the row between nodes i, j, k and $i, j-1, k$ [LT^{-1}],
$\Delta c_i \Delta v_k$	area of the cell faces normal to the row direction,
$\Delta r_{j-1/2}$	distance between nodes i, j, k and $i, j-1, k$ [L].

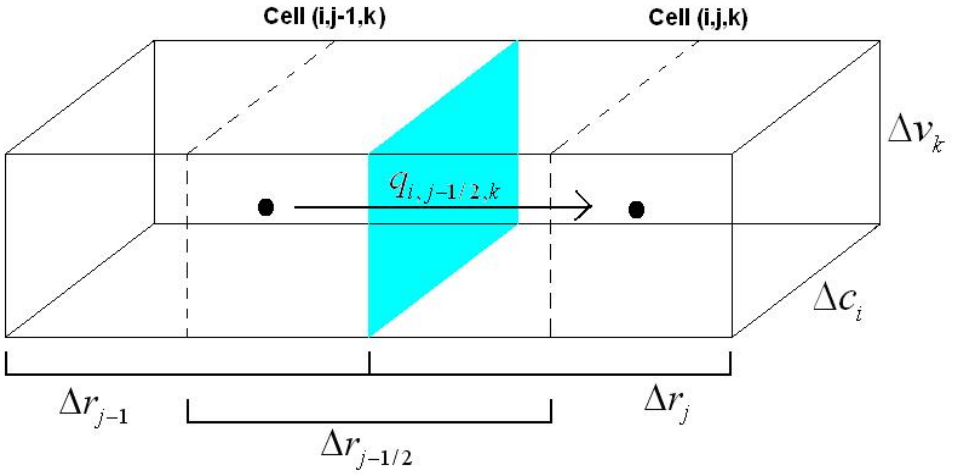

Figure 3.4: Flow into cell (i, j, k) from cell $(i, j-1, k)$.

Now Equation 3.11 gives the exact flow in one dimension for the steady-state case. In this equation it is the flow through a block of aquifer from node $i, j-1, k$ to node i, j, k, having a cross sectional area $\Delta c_i \Delta v_k$. The conductivity $KR_{i,j-1/2,k}$ of the material between nodes i, j, k and $i, j-1, k$ is the effective hydraulic conductivity for the entire region between the nodes. It is normally calculated as a harmonic mean. The term $1/2$ is used to indicate the region *between* nodes.

In the same way we can derive equations for the flow in the five other faces of the cell (i, j, k). So we have for the flow in the row direction through the face between cells (i, j, k) and $(i, j+1, k)$:

$$q_{i,j+1/2,k} = KR_{i,j+1/2,k} \Delta c_i \Delta v_k \frac{(h_{i,j+1,k} - h_{i,j,k})}{\Delta r_{j+1/2}}. \tag{3.12}$$

In the column direction we have for the flows through the front and rear face:

$$q_{i-1/2,j,k} = KC_{i-1/2,j,k} \Delta r_j \Delta v_k \frac{(h_{i-1,j,k} - h_{i,j,k})}{\Delta c_{i-1/2}}. \tag{3.13}$$

$$q_{i+1/2,j,k} = KC_{i+1/2,j,k} \Delta r_j \Delta v_k \frac{(h_{i+1,j,k} - h_{i,j,k})}{\Delta c_{i+1/2}}, \tag{3.14}$$

For the vertical direction we have for the inflow through the bottom and upper face:

$$q_{i,j,k-1/2} = KV_{i,j,k-1/2} \Delta r_j \Delta c_i \frac{(h_{i,j,k-1} - h_{i,j,k})}{\Delta v_{k-1/2}}. \tag{3.15}$$

$$q_{i,j,k+1/2} = KV_{i,j,k+1/2} \Delta r_j \Delta c_i \frac{(h_{i,j,k+1} - h_{i,j,k})}{\Delta v_{k+1/2}}, \tag{3.16}$$

The six equations, (3.11) - (3.16), express the flow through a boundary in terms of heads, grid dimensions, and hydraulic conductivity. We simplify the notation by combining hydraulic conductivity and grid dimensions into one single constant *hydraulic conductance*:

$$CR_{i,j-1/2,k} = \frac{KR_{i,j-1/2,k} \Delta c_i \Delta v_k}{\Delta r_{j-1/2}}$$

So, we define conductance as the product of hydraulic conductivity and cross-sectional area of flow divided by the length of the flow path, which is here the distance between two nodes. For the other equations we can do the same and we end up with the following six equations:

$$q_{i,j-1/2,k} = CR_{i,j-1/2,k}\left(h_{i,j-1,k} - h_{i,j,k}\right), \tag{3.17}$$

$$q_{i,j+1/2,k} = CR_{i,j+1/2,k}\left(h_{i,j+1,k} - h_{i,j,k}\right), \tag{3.18}$$

$$q_{i-1/2,j,k} = CC_{i-1/2,j,k}\left(h_{i-1,j,k} - h_{i,j,k}\right), \tag{3.19}$$

$$q_{i+1/2,j,k} = CC_{i+1/2,j,k}\left(h_{i+1,j,k} - h_{i,j,k}\right), \tag{3.20}$$

$$q_{i,j,k-1/2} = CV_{i,j,k-1/2}\left(h_{i,j,k-1} - h_{i,j,k}\right), \tag{3.21}$$

$$q_{i,j,k+1/2} = CV_{i,j,k+1/2}\left(h_{i,j,k+1} - h_{i,j,k}\right). \tag{3.22}$$

The values CR, CC and CV are the conductances along rows and columns and between layers, respectively. They are calculated the same way as in Equation (3.4)

3.3.2 External Sources

The flow into cell (i, j, k) from the six adjacent cells are governed by the equations (3.17)-(3.22). To account for boundary conditions extra terms need to be added. These flows can be dependent on the head in the cell, but independent of the head in the adjacent cells. It is also possible that they are completely independent of the head in the cell. We may represent the flow from outside the aquifer by the expession:

$$a_{i,j,k,n} = p_{i,j,k,n}h_{i,j,k} + q_{i,j,k,n},$$

where:

$a_{i,j,k,n}$	flow from the n^{th} external source into cell (i, j, k) $[L^3T^{-1}]$,
$p_{i,j,k,n}$ and $q_{i,j,k,n}$	constants ($[L^2T^{-1}]$ and $[L^3T^{-1}]$ respectively).

If a source is independent of the head, we can set $p_{i,j,k.n}$ equal to zero. If we have N external sources or stresses, then we can express the general external flow for cell (i, j, k) as:

$$\sum_{n=1}^{N} a_{i,j,k,n} = P_{i,j,k}h_{i,j,k} + Q_{i,j,k},$$

where:

$$P_{i,j,k} = \sum_{n=1}^{N} p_{i,j,k,n},$$

$$Q_{i,j,k} = \sum_{n=1}^{N} q_{i,j,k,n}.$$

If we now apply the continuity equation (3.10) to cell (i, j, k), we get:

$$\begin{aligned} & q_{i,j-1/2,k} + q_{i,j+1/2,k} + q_{i-1/2,j,k} + q_{i+1/2,j,k} \\ & + q_{i,j,k-1/2} + q_{i,j,k+1/2} + P_{i,j,k}h_{i,j,k} + Q_{i,j,k} \\ & = SS_{i,j,k}\left(\Delta r_j \Delta c_i \Delta v_k\right)\frac{\Delta h_{i,j,k}}{\Delta t}, \end{aligned} \tag{3.23}$$

where:

$\Delta h_{i,j,k}/\Delta t$	finite difference approximation for the derivative of head with respect to time [LT^{-1}],
$SS_{i,j,k}$	the specific storage of cell (i, j, k) [L^{-1}],
$\Delta r_j \Delta c_i \Delta v_k$	volume of cell (i, j, k) [L^3].

Now we can substitute Equations (3.17) - (3.22) into Equation (3.23) which gives the *finite volume approximation* for cell (i, j, k) as:

$$\begin{aligned} & CR_{i,j-1/2,k}\left(h_{i,j-1,k} - h_{i,j,k}\right) + CR_{i,j+1/2,k}\left(h_{i,j+1,k} - h_{i,j,k}\right) \\ & + CC_{i-1/2,j,k}\left(h_{i-1,j,k} - h_{i,j,k}\right) + CC_{i+1/2,j,k}\left(h_{i+1,j,k} - h_{i,j,k}\right) \\ & + CV_{i,j,k-1/2}\left(h_{i,j,k-1} - h_{i,j,k}\right) + CV_{i,j,k+1/2}\left(h_{i,j,k+1} - h_{i,j,k}\right) \\ & + P_{i,j,k}h_{i,j,k} + Q_{i,j,k} = SS_{i,j,k}\left(\Delta r_j \Delta c_i \Delta v_k\right)\frac{\Delta h_{i,j,k}}{\Delta t}. \end{aligned} \tag{3.24}$$

More on the external sources can be found in Section 3.4.1.

3.3.3 Discretization in Time

We still need to express the finite-difference approximation for the time derivative of head in terms of heads and times. Let t^m and t^{m-1} denote the current and past time, respectively. An approximation to the time derivative of head at time t^m is given by:

$$\frac{\Delta h_{i,j,k}}{\Delta t} \simeq \frac{h^m_{i,j,k} - h^{m-1}_{i,j,k}}{t^m - t^{m-1}}.$$

This method is called the *Euler backwards method*. The term $h^m_{i,j,k}$ denotes the head value at cell (i, j, k) at time t^m. We can approximate the time derivative of head in other ways, but the Euler backwards method is unconditionally numerically stable. Stability means that a small perturbation in the initial conditions does not effect the solution much. So we prefer the backward-difference approach even though this leads to large systems of equations that must be solved simultaneously for each time step. The advantage of a stable method is that we can take bigger time steps without caring about stability.

We can rewrite Equation (3.24) in a backwards-difference form. We must specify flow terms at t^m, the end of a time interval, and approximate the derivative of the head over the time interval from t^{m-1} to t^m. So we get:

$$
\begin{aligned}
&CR_{i,j-1/2,k}\left(h^m_{i,j-1,k}-h^m_{i,j,k}\right)+CR_{i,j+1/2,k}\left(h^m_{i,j+1,k}-h^m_{i,j,k}\right)\\
&+CC_{i-1/2,j,k}\left(h^m_{i-1,j,k}-h^m_{i,j,k}\right)+CC_{i+1/2,j,k}\left(h^m_{i+1,j,k}-h^m_{i,j,k}\right)\\
&+CV_{i,j,k-1/2}\left(h^m_{i,j,k-1}-h^m_{i,j,k}\right)+CV_{i,j,k+1/2}\left(h^m_{i,j,k+1}-h^m_{i,j,k}\right)\\
&+P_{i,j,k}h^m_{i,j,k}+Q_{i,j,k}=SS_{i,j,k}\left(\Delta r_j\Delta c_i\Delta v_k\right)\frac{h^m_{i,j,k}-h^{m-1}_{i,j,k}}{t^m-t^{m-1}}.
\end{aligned}
\tag{3.25}
$$

Since we can write down for each active cell an equation of this type, we are left with a system of "n" equations in "n" unknowns. Such a system can be solved simultaneously.

We start by taking $m = 1$ and calculate the values of $h^1_{i,j,k}$ for each active cell. Of course we need to specify the initial head distribution $h^0_{i,j,k}$, the boundary conditions, the hydraulic parameters and the external sources. If we found all the values for $h^1_{i,j,k}$ we can go on to calculate the values of $h^2_{i,j,k}$ by using $h^1_{i,j,k}$ as an "initial" condition. This process is continued for as many time steps as necessary to cover the time range of interest.

3.3.4 System of Equations

By reordering the terms of Equation (3.25) we can make a system of equations of the form $Ah = q$. All terms that are independent of head at the end of the current time step we put on the right-hand side (RHS). All terms of $h^m_{i,j,k}$ that are independent on the conductance between nodes we put into a single term ($HCOF$). We then write Equation (3.25) as:

$$
\begin{aligned}
&CR_{i,j-1/2,k}h^m_{i,j-1,k}+CR_{i,j+1/2,k}h^m_{i,j+1,k}+CC_{i-1/2,j,k}h^m_{i-1,j,k}\\
&+CC_{i+1/2,j,k}h^m_{i+1,j,k}+CV_{i,j,k-1/2}h^m_{i,j,k-1}+CV_{i,j,k+1/2}h^m_{i,j,k+1}\\
&+\Big(-CR_{i,j-1/2,k}-CR_{i,j+1/2,k}-CC_{i-1/2,j,k}h^m_{i-1,j,k}-CC_{i+1/2,j,k}-\\
&CV_{i,j,k-1/2}-CV_{i,j,k+1/2}+HCOF_{i,j,k}\Big)h^m_{i,j,k}=RHS_{i,j,k},
\end{aligned}
\tag{3.26}
$$

where:

$$
\begin{aligned}
HCOF_{i,j,k} &= P_{i,j,k}-\frac{SS_{i,j,k}\Delta r_j\Delta c_i\Delta v_k}{t^m-t^{m-1}},\\
RHS_{i,j,k} &= -Q_{i,j,k}-SS_{i,j,k}\Delta r_j\Delta c_i\Delta v_k\frac{h^{m-1}_{i,j,k}}{t^m-t^{m-1}}.
\end{aligned}
$$

Now this entire system of equations, which includes one equation for each variable head cell in the grid, may be written in matrix-vector form as

$$Ah = q,$$

where:

A A matrix of the coefficients of head, for all the active nodes in the grid,

h vector of head values at the end of time step m for all active nodes in the grid,

q vector of the constant terms, RHS, of all active nodes in the cell.

The matrix A now has the form as shown in Figure 3.5. The diagonal are the elements corresponding to $h^m_{i,j,k}$ in Equation (3.26). The first off-diagonal are exactly the CR-values, the second off diagonal are the CC-values and the last off-diagonal are the CV-values. When you only have one layer (2D-problem) then the third off-diagonal will not appear.

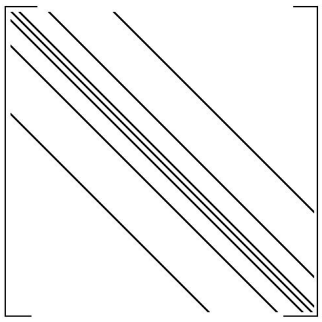

Figure 3.5: Structure of the system matrix A.

3.4 Simulation of Boundaries in MODFLOW

In MODFLOW we do not need to formulate an equation of the form of Equation (3.25) for every cell in a model grid. The status of certain cells is specified in advance to simulate the boundary conditions of the problem. We group the cells that simulate the boundary conditions into three categories:

- constant-head cells,
- no-flow cells,
- variable head cells.

A *constant-head cell* is used if for that cell the head is specified for each time. The head values do not change as a result of solving the flow equations. The *no-flow cells* are those for which no flow into or out of the cell is permitted. We will call the remaining cells *variable head cells*. These cells are characterized by heads that are unspecified and free to vary in time. For each variable-head cell in the grid we must formulate an equation of the form of Equation (3.25). The resulting system must be solved simultaneously for each time step in the simulation.

We can use constant-head cells and no-flow cells to represent conditions along various hydraulic boundaries inside the grid. To explain this, we can for example look at Figure 3.6. It shows a configuration of an aquifer boundary superimposed onto a grid of cells generated for the model.

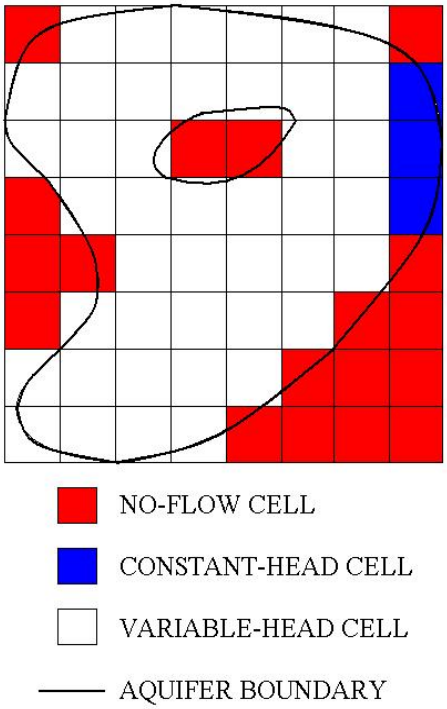

Figure 3.6: Discretized aquifer showing boundaries and cell designations.

Although the aquifer is of irregular shape, our model grid is always rectangular in outline. If the boundary of the aquifer coincides with the outside edge of the grid, we do not need special designations since MODFLOW does not compute inter-cell flow through the outside edges of the grid, including top and bottom. Within the grid we use no-flow cells to delete the part of the grid beyond the aquifer boundary. The constant-head cells can be used, for example, if at that place the aquifer is in direct contact with major surface water features.

Other boundary conditions can be simulated by using external source terms possible in combination with no-flow cells. In MODFLOW this is done by stress packages. These packages are called stress packages because in MODFLOW we define stress periods. These stress periods are time intervals in which the input data for all external sources are constant. These stress periods are subdivided into time steps. See also Figure 3.7. In the next subsection we will give more details on these stress packages.

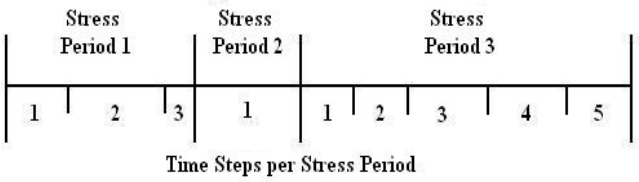

Figure 3.7: Division of simulation time into stress periods and time steps.

3.4.1 Stress Packages

The stress in MODFLOW adds terms to the flow equation representing inflow and outflow. Mathematically, these are boundary conditions. There are two types of important boundary conditions. First we have the constant flux conditions, or the so called Neumann conditions. The WEL and the RCH packages simulate such conditions. Furthermore we have the so called Cauchy boundary conditions. These are head-dependent boundary conditions. The GHB, RIV and DRN packages simulate such conditions.

The finite-volume flow equation for MODFLOW is Equation (3.26). The equation is formulated such that the inflows are added to the left side, with the outflows represented as negative inflows. Stresses are added to the equation by adding terms to $HCOF$ and RHS. A stress term that is a coefficient of head, $h_{i,j,k}$, is added to $HCOF$. A constant stress term is subtracted from RHS.

Well Package (WEL)

With the Well Package we can simulate features such as wells that withdraw water from or add water to the aquifer at a constant rate during a stress period. The rate is independent on both the cell area and the head in the cell. Mostly the package is used to simulate wells, either discharge or recharge. However, the package can also be used to simulate any features for which the recharge or discharge can be specified directly. The dimension of recharge or discharge is $[L^3T^{-1}]$.

The user has to specify the flow rate, Q, for a well and a fluid volume per unit time at which water is added to an aquifer. Negative values are used to indicate well discharge (pumping). The user can specify the location of the well by specifying the row, column and layer number of the well. The value of Q for each well is subtracted from the RHS of Equation (3.26) for the cell containing that well. Figure 3.10 shows the initial condition near a well in one of the layers.

Recharge Package (RCH)

The Recharge Package is designed to simulate areally distributed recharge to the groundwater system. Most commonly, recharges occurs as a result of precipitation that enters to the groundwater system and evaporation leaving the system. These processes occur normally at the surface, but also processes in other layers can be simulated. The recharge applied to the model is defined as:

$$QR_{i,j} = I_{i,j}\,\Delta r_j\,\Delta c_i,$$

where:

$QR_{i,j}$	the recharge flow rate applied to the model at horizontal cell location (i,j) expressed as a fluid volume per time $[L^3T^{-1}]$,
$I_{i,j}$	the recharge flux applicable to the map area of the cell $[LT^{-1}]$,
Δr_j	dimension of cell along the row direction,
Δc_i	dimension of cell along the column direction.

The recharge flow rate, $QR_{i,j}$, associated with a given horizontal cell location (i,j) and a vertical location, k, is subtracted from the value of $RHS_{i,j,k}$. Since recharge is independent of the aquifer head, nothing is added to the coefficient of head, $HCOF_{i,j,k}$.

General-Head Boundary Package (GHB)

The General-Head Boundary Package simulates flow into or out of a cell $(i, j, k,)$, from an external source in proportion to the difference between the head in the cell and head assigned to the external source. The constant of proportionality is called the *boundary conductance*. We have the following linear relation between flow into the cell and head in the cell:

$$QB_n = CB_n \left(HB_n - h_{i,j,k} \right), \tag{3.27}$$

where:

n	boundary number
QB_n	flow into cell (i, j, k) from the boundary $[L^3T^{-1}]$,
CB_n	the boundary conductance $[L^2T^{-1}]$,
HB_n	the head assigned to the external source $[L]$,
$h_{i,j,k}$	the head in cell (i, j, k) $[L]$.

A graph of the inflow from a general-head boundary and head in the cell containing the boundary as given by Equation (3.27) is shown in Figure 3.11.

River Package (RIV)

Rivers and streams contribute water to or drain water from the groundwater system. This depends on the head gradient between the river and the groundwater regime. The River Package simulates the effects of flow between surface water features and groundwater systems. The River Package does not simulate surface water flow in the river, but only the river/aquifer seepage.

By assumption, measurable head losses between the river and the aquifer are limited to those across the riverbed layer itself. Another assumption is that the water level does not drop below the bottom of the riverbed layer. Under these assumptions, flow between the river and the groundwater system for each n is given by:

$$\begin{cases} QRIV_n = CRIV_n \left(HRIV_n - h_{i,j,k} \right) & h_{i,j,k} > RBOT_n, \\ QRIV_n = CRIV_n \left(HRIV_n - RBOT_n \right) & h_{i,j,k} \leq RBOT_n, \end{cases} \tag{3.28}$$

where:

$QRIV_n$	flow between the river and the aquifer, taken positive if it is directed into the aquifer $[L^3T^{-1}]$,
$HRIV_n$	the water level (stage) in the river $[L]$,
$CRIV_n$	the hydraulic conductance of the river-aquifer interconnection $[L^2T^{-1}]$,
$RBOT_n$	bottom of riverbed layer $[L]$,
$h_{i,j,k}$	the head at the node in the cell underlying the river reach $[L]$.

The graph of the flow from a river as a function of the head in the corresponding cell as calculated in Equation (3.28) is shown in Figure 3.12. Figure 3.13 shows how the river "De Maas" is implemented in the IBRAHYM model.

Drain Package (DRN)

The effects of features as agricultural drains are simulated with the Drain Package. Drains remove water from the aquifer at a rate proportional to the difference between the head in the aquifer and some fixed head or elevation, called the drain elevation. The drain works as long as the head in the aquifer is above the elevation. If the aquifer head falls below the drain elevation, then the drain has no effect on the aquifer. The constant of proportionality is called the drain conductance. A mathematical statement of this situation is:

$$\begin{cases} QD_n = CD_n\,(HD_n - h_{i,j,k}) & h_{i,j,k} > HD_n, \\ QD_n = 0 & h_{i,j,k} \leq HD_n, \end{cases} \tag{3.29}$$

where:

QD_n	flow from aquifer into the drain [L^3T^{-1}],
CD_n	drain conductance [L^2T^{-1}],
HD_n	drain elevation [L],
$h_{i,j,k}$	the head in the cell containing the drain [L],
n	number of the drain.

Figure 3.14 shows a graph of flow from a drain and head in the cell containing the drain as defined by Equation (3.29). Figure 3.15 shows where drains are constructed in and around Eindhoven which are implemented in the IBRAHYM model.

3.5 Faults and Clay in MODFLOW

Faults

A fault in the subsoil effects the system matrix as defined in Section 3.3.4. In MODFLOW a fault is always translated to a fault on the cell boundaries as shown in Figure 3.8. Faults are modelled in MODFLOW as a change in the CR and the CC values between two grid-points. So a fault causes a change in the system matrix.

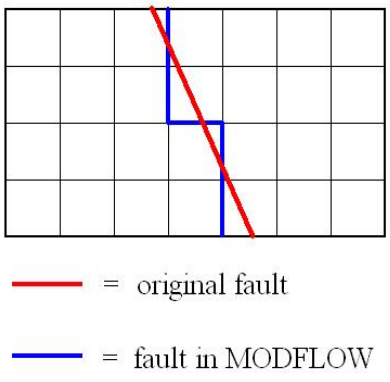

Figure 3.8: Fault as modelled by MODFLOW.

In IMODFLOW when a fault occurs it is modelled between two cells. So we have to adapt the conductance between two cells. The formula for the conductance between two

cells is given by Equation (3.4). In IMODFLOW we can describe the position of the faults (red line in Figure 3.8). A special package, called HFB, is used to adapt the conductance between predefined cells (blue line in Figure 3.8). Together with the position of the fault we also give a factor per fault. We simply multiply the original conductance between the two cells with this factor. Near the Peelrandbreuk Deltares now uses for example a factor of 0.001, but they want it to be closer to zero. Figure 3.9 is a picture of the solution in one layer near a fault. We see a jump in the calculated heads.

Layers of Clay

As we already mentioned in Section 2.1, between two aquifers there can be a package of clay schematized (aquitard). Since we assume vertical flow only in such an aquitard we do not explicitly solve for these aquitards. We simply define a vertical conductance (CV) between two layers which represents such an aquitard. If the resistance of clay is very large, we have a high value for CV and the other way around. In this way we do not calculate anything inside an aquitard, but we take them into account by calculating the heads in the aquifers.

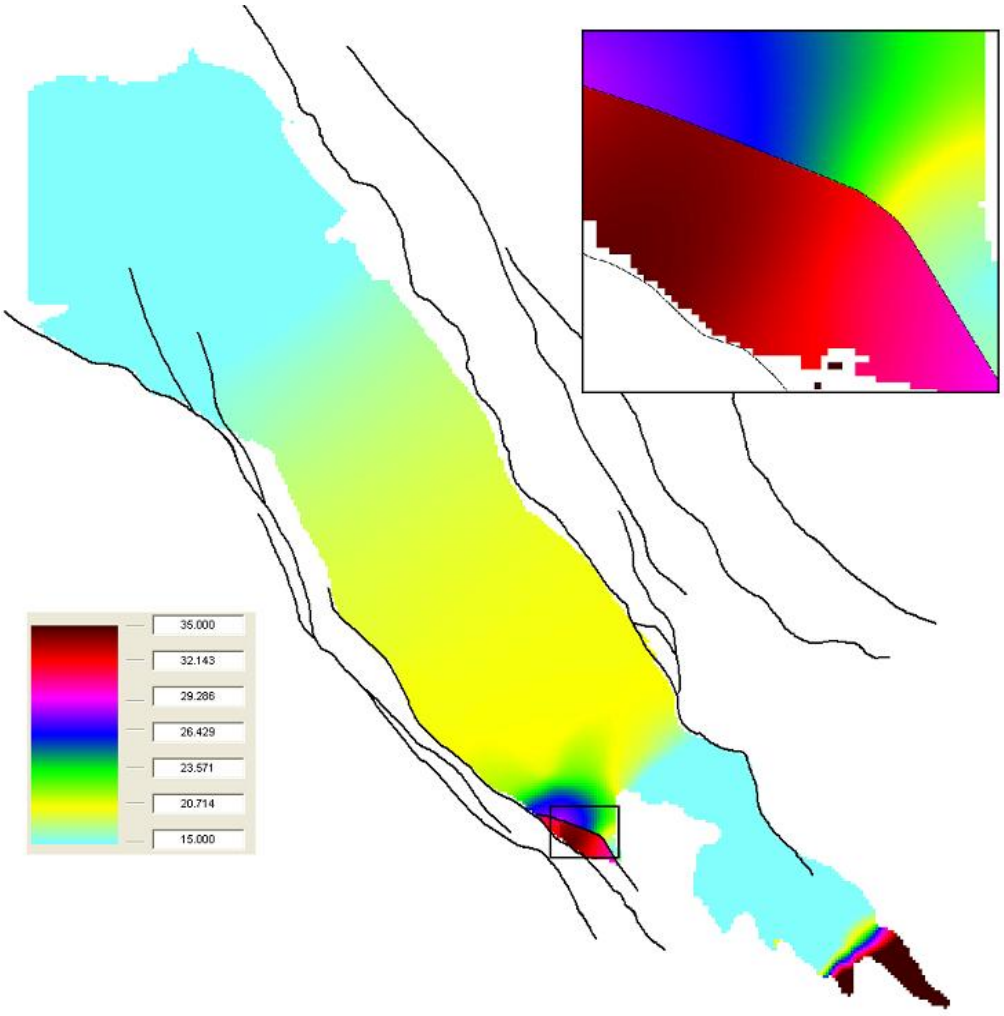

Figure 3.9: Heads in the 19th layer in the IBRAHYM model.

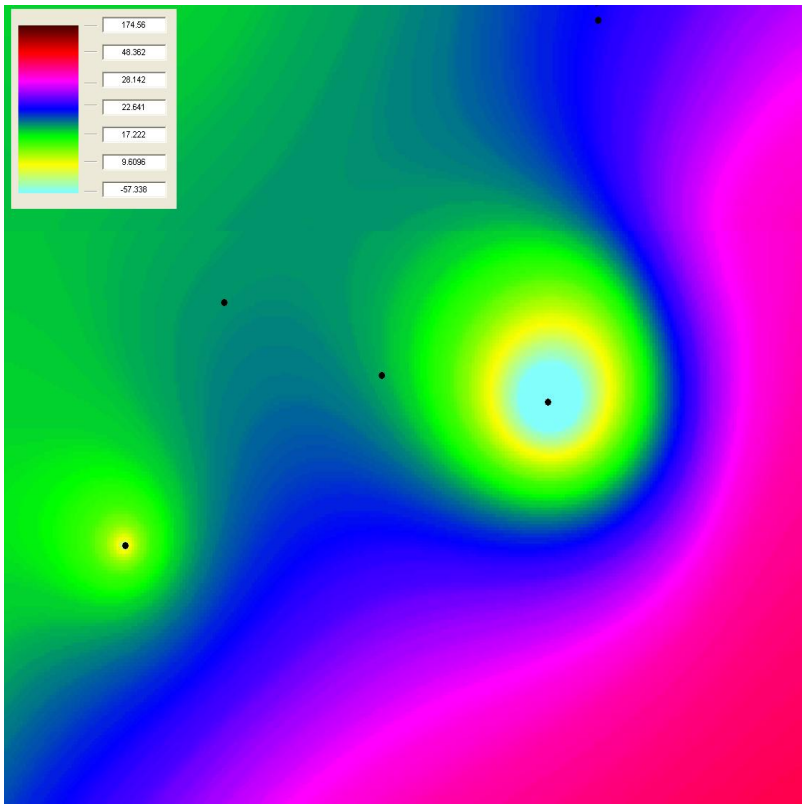

Figure 3.10: Example of the heads near a well.

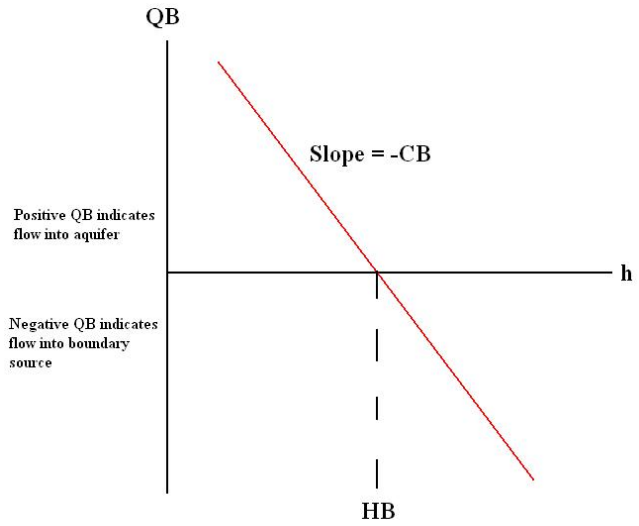

Figure 3.11: Plot of flow, QB, from a general-head boundary source into a cell as function of head.

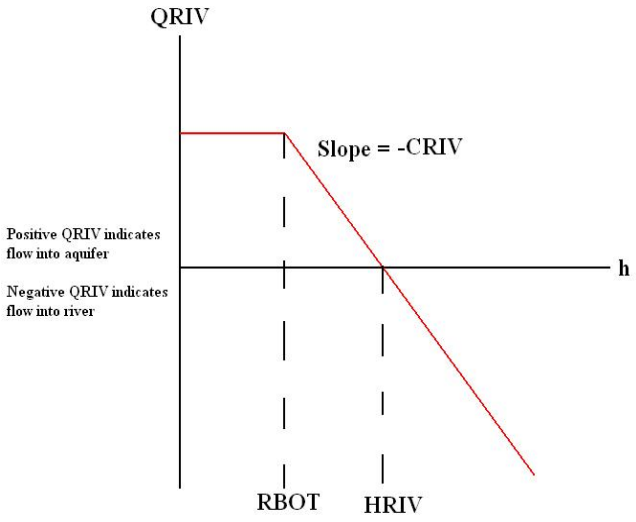

Figure 3.12: Plot of flow, QRIV, from a river into a cell as function of head.

Figure 3.13: Example of a river, de Maas, in MODFLOW.

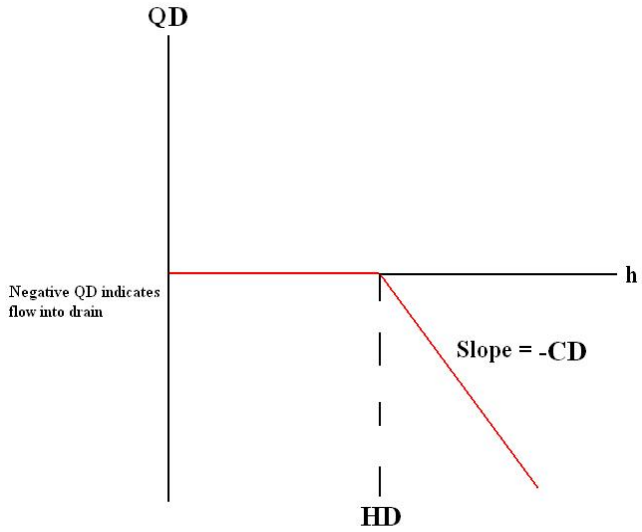

Figure 3.14: Plot of flow, QD, into a drain as a function of head.

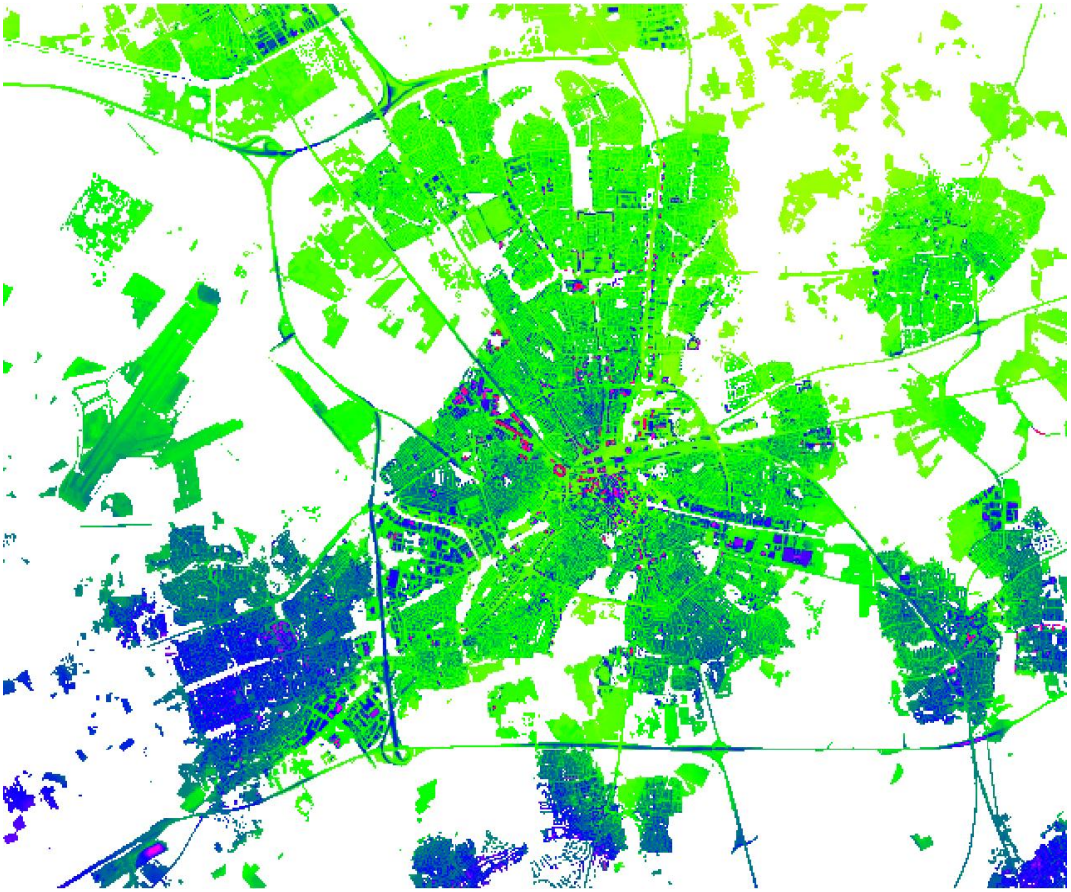

Figure 3.15: Overview of a drain system (Eindhoven) in MODFLOW.

Chapter 4

Solving the Equation

Iterative solvers are used to solve linear algebraic systems of equations [4, 8, 11]. Given a $n \times n$ real matrix and a real vector $b \in \mathbb{R}^n$, the problem considered is: find a vector $y \in \mathbb{R}^n$ such that:

$$Ay = b. \tag{4.1}$$

This equation is a *linear system*. We call the matrix A the *coefficient matrix*, b the *right-hand side vector* and y the *vector of unknowns*. We assume A to be invertible, i.e. $A^{-1}b$ can be computed.

Iterative methods for the solution of linear system of equations are useful in the following cases:

- if the number of iterations necessary is not too big,
- if A is large and sparse,
- if A has a special structure,
- if a good initial guess for y is available, as in time-stepping methods,
- if A is, for example, not known explicitly.

In other cases it is more advantageous to use direct methods.

4.1 Basic Iterative Methods

From equation (4.1), we construct a splitting $A = M - N$, such that M is easy invertible. We can now write:

$$My - Ny = b, \tag{4.2}$$

which is still exact. We now use the following iteration:

$$My^{(k+1)} - Ny^{(k)} = b. \tag{4.3}$$

Note that we have that $N = M - A$ and we can rewrite this equation as:

$$\begin{aligned} y^{(k+1)} &= M^{-1}Ny^{(k)} + M^{-1}b \\ &= M^{-1}\left(M - A\right)y^{(k)} + M^{-1}b \\ &= y^{(k)} - M^{-1}Ay^{(k)} + M^{-1}b \\ &= y^{(k)} + M^{-1}\left(b - Ay^{(k)}\right). \end{aligned} \tag{4.4}$$

In general we use the following iteration:

$$y^{(k+1)} = Gy^{(k)} + f,\ (k = 0, 1, 2, \ldots), \tag{4.5}$$

so that the system $y = Gy + f$ is equivalent to the original problem (4.1). G is called the *iteration matrix*.

If we look at equation (4.5) then we see that for a basic iterative method we have:

$$G = M^{-1}N = M^{-1}\left(M - A\right) = I - M^{-1}A,\ \ f = M^{-1}b.$$

This recurrence is also called a *linear fixed-point iteration*. If we take $M = D$, where D contains the diagonal elements of A on its diagonal and all other elements are zero, we have the Jacobi-method. If we take $M = D - E$, where $-E$ is the strict lower part of A, we have the forward Gauss-Seidel-method.

We can view the iteration $y^{(k+1)} = Gy^{(k)} + f$ as a technique to solve the system

$$(I - G)y = f$$

Since G has the form $G = I - M^{-1}A$, we can rewrite this system as

$$M^{-1}Ay = M^{-1}b$$

This system has the same solution as the original system and is called *left preconditioned system*, where M is the *preconditioning matrix* or *preconditioner*. There also exist right preconditioned systems. Preconditioning is used to speed up the convergence. The idea is that M is more or less similar to A, but easy to invert.

4.2 Projection Methods

The idea of *projection techniques* is to extract an approximate solution to the problem in Equation (4.1) from a subspace of $\mathbb{R}^n$. Let $\mathcal{K}_m \subset \mathbb{R}^n$ be this subspace of candidate approximants, or search subspace. Let m be the dimension of $\mathcal{K}_m$, then, in general, m constraints must be imposed to be able to extract such an approximation from $\mathcal{K}_m$. Sometimes one imposes m (independent) orthogonality conditions. Specifically, the residual vector $b - Ay$ is constrained to be orthogonal to m linearly independent vectors. These constraints define another subspace of dimension m. This *subspace of constraints* we denote with $\mathcal{L}_m$. In many different mathematical methods this simple framework is known as the Petrov-Galerkin condition.

Let A be a $n \times n$ real matrix and $\mathcal{K}_m$ and $\mathcal{L}_m$ two m-dimensional subspaces of $\mathbb{R}^n$. We want to have a projection technique onto the subspace $\mathcal{K}_m$ and orthogonal to $\mathcal{L}_m$. It should be a process which finds an approximate solution $\tilde{y}$ to (4.1) where we impose that $\tilde{y}$ belongs to $\mathcal{K}_m$ and that the new residual vector $b - A\tilde{y}$ is orthogonal to $\mathcal{L}_m$:

$$\text{find } \tilde{y} \in \mathcal{K}_m,\ \text{ such that }\ \ b - A\tilde{y} \perp \mathcal{L}_m.$$

If we have the knowledge of an initial guess y_0 and we want to exploit that, then we look for our approximation in the affine space $y_0 + \mathcal{K}_m$. So our problem then becomes:

$$\text{find } \tilde{y} \in y_0 + \mathcal{K}_m, \text{ such that } b - A\tilde{y} \perp \mathcal{L}_m. \tag{4.6}$$

If we write $\tilde{y}$ in the form $\tilde{y} = y_0 + \delta$ and we define the initial residual vector as:

$$r_0 := b - Ay_0,$$

then Equation (4.6) becomes $b - A\tilde{y} \perp \mathcal{L}_m$, or:

$$r_0 - A\delta \perp \mathcal{L}_m.$$

Now we can find the approximate solution $\tilde{y}$ by using that:

$$\begin{cases} \tilde{y} = y_0 + \delta & ,\delta \in \mathcal{K}_m \\ (r_{new}, w) = 0 & ,\forall w \in \mathcal{L}_m \end{cases} \tag{4.7}$$

where r_{new} is defined as $r_0 - A\delta$ (see also figure 4.1).

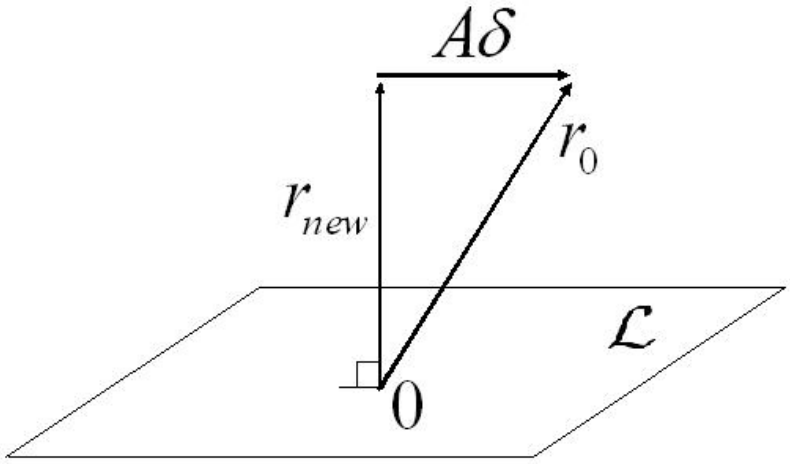

Figure 4.1: Interpretation of the orthogonality condition.

This is a projection method in its most general form. Typically, a new projection step uses a new pair of subspaces $\mathcal{K}_m$ and $\mathcal{L}_m$ and a new initial guess y_0 equal to the most recent approximation obtained from the previous step.

4.2.1 Matrix Representation

We define $V = [v_1 \cdots v_m]$ to be a $n \times m$ matrix whose column-vectors form a basis of $\mathcal{K}_m$ and, similarly, $W = [w_1 \cdots w_m]$ a $n \times m$ matrix whose column-vectors form a basis of $\mathcal{L}_m$. Now write the approximate solution as:

$$y = y_0 + Vz.$$

If z would be known, then we would have:

$$\begin{aligned} Ay &= Ay_0 + AVz, \\ Ay - Ay_0 &= AVz, \\ b - Ay_0 &= AVz, \\ r_0 &= AVz. \end{aligned}$$

Since we still have to determine z we use the orthogonality relation, which states:

$$\text{span}\{W\} \perp r_0 - AVz.$$

Therefore we have:

$$W^T r_0 = W^T AVz.$$

If we assume the matrix $W^T AV$ to be non-singular we can conclude that

$$z = \left(W^T AV\right)^{-1} W^T r_0,$$

and we end up with:

$$\tilde{y} = y_0 + V\left(W^T AV\right)^{-1} W^T r_0.$$

4.3 The Krylov Subspace

In Equation (4.4) we compute the iterates by the recursion:

$$y^{(k+1)} = y^{(k)} + M^{-1}\left(b - Ay^{(k)}\right) = y^{(k)} + M^{-1}r^{(k)}$$

If we write out the first terms of this process we obtain:

$$\begin{aligned}
&y^{(0)}, \\
y^{(1)} &= y^{(0)} + M^{-1}r^{(0)}, \\
y^{(2)} &= y^{(1)} + M^{-1}r^{(1)}, \\
&= y^{(1)} + M^{-1}\left(b - Ay^{(1)}\right), \\
&= y^{(1)} + M^{-1}\left(b - Ay^{(0)} - AM^{-1}r^{(0)}\right), \\
&= y^{(1)} + M^{-1}\left(r^{(0)} - AM^{-1}r^{(0)}\right), \\
&= y^{(1)} + M^{-1}r^{(0)} - M^{-1}AM^{-1}r^{(0)}, \\
&= y^{(0)} + M^{-1}r^{(0)} + M^{-1}r^{(0)} - M^{-1}AM^{-1}r^{(0)}, \\
&= y^{(0)} + 2M^{-1}r^{(0)} - M^{-1}AM^{-1}r^{(0)}, \\
y^{(3)} &= y^{(2)} + M^{-1}r^{(2)}, \\
&= y^{(2)} + M^{-1}\left(b - Ay^{(2)}\right), \\
&= y^{(2)} + M^{-1}\left(b - A\left(y^{(0)} + 2M^{-1}r^{(0)} - M^{-1}AM^{-1}r^{(0)}\right)\right), \\
&= y^{(2)} + M^{-1}r^{(0)} - 2\left(M^{-1}A\right)M^{-1}r^{(0)} + \left(M^{-1}A\right)^2 M^{-1}r^{(0)}, \\
&= y^{(0)} + 3M^{-1}r^{(0)} - 3\left(M^{-1}A\right)M^{-1}r^{(0)} + \left(M^{-1}A\right)^2 M^{-1}r^{(0)}, \\
&\vdots
\end{aligned}$$

which implies that:

$$y^{(k)} = y^{(0)} + \text{span}\left\{M^{-1}r^{(0)}, \left(M^{-1}A\right)M^{-1}r^{(0)}, \ldots, \left(M^{-1}A\right)^{k-1}M^{-1}r^{(0)}\right\}.$$

The subspace $\mathcal{K}_k\left(A; r^{(0)}\right) := \text{span}\left\{r^{(0)}, Ar^{(0)}, \ldots, A^{k-1}r^{(0)}\right\}$ is called the *Krylov subspace* of dimension k corresponding to the matrix A and the initial residual vector $r^{(0)}$. Now a $y^{(k)}$ calculated by the basic iterative method is an element of $y^{(0)} + \mathcal{K}_k\left(M^{-1}A; M^{-1}r^{(0)}\right)$

4.4 Preconditioned Conjugate Gradient Methods

4.4.1 The basic algorithm

To explain the basics of the Conjugate Gradient Method we first make some assumptions. We assume:

- $M = I$,
- $y^{(0)} = 0$,
- $r^{(0)} = b$,
- A is symmetric, i.e. $A^T = A$,
- A is positive definite, i.e. $x^T Ax > 0$, $\forall x \neq 0$.

The first three assumptions are only needed to facilitate the formula's, but are not necessary for the CG method itself. The last two assumptions are necessary for the CG method to work.

Our first idea would be to construct

$$y^{(i)} \in \mathcal{K}_i(A; r^{(0)}) \text{ such that } \|y - y^{(i)}\|_2 \text{ is minimal.}$$

If we look at the first iterate, $y^{(1)}$, we see that it can be written as $y^{(1)} = \alpha_0 \, r^{(0)}$, where α_0 is a constant which has to be chosen such that $\|y - y^{(i)}\|_2$ is minimal. Now:

$$\|y - y^{(i)}\|_2^2 = \left(y - \alpha_0 r^{(0)}\right)^T \left(y - \alpha_0 r^{(0)}\right) = y^T y - 2\alpha_0 (r^{(0)})^T y + \alpha_0^2 (r^{(0)})^T r^{(0)}.$$

This is minimal if:

$$\alpha_0 = \frac{(r^{(0)})^T y}{(r^0)^T r^{(0)}},$$

but since we do not know y, this choice cannot be determined and we have to look for another solution. We can define the *A-inner product* and the *A-norm* to be:

$$(y, z)_A = y^T A z,$$

$$\|y\|_A = \sqrt{(y, y)_A} = \sqrt{y^T A y},$$

respectively. If A is a Symmetric Positive Definite (SPD) matrix, then $(y, z)_A$ and $\|y\|_A$ satisfy the rules for inner product and norm respectively. We now want to construct:

$$y^{(i)} \in \mathcal{K}_i(A; r^{(0)}) \text{ such that } \|y - y^{(i)}\|_A \text{ is minimal.}$$

Now:

$$\|y - y^{(i)}\|_A^2 = y^T A y - 2\alpha_0 (r^{(0)})^T A y + \alpha_0^2 (r^{(0)})^T A r^{(0)},$$

which is minimal if:

$$\alpha_0 = \frac{(r^{(0)})^T A y}{(r^{(0)})^T A r^{(0)}} = \frac{(r^{(0)})^T b}{(r^{(0)})^T A r^{(0)}}.$$

We now have a new inner-product which leads to a minimization problem, which is easily solved. In the iterations we compute $y^{(i)}$ such that:

$$\|y - y^{(i)}\|_A = \min_{x \in \mathcal{K}_i(A; r^{(0)})} \|y - x\|_A$$

The solution of this minimization problem leads to the Conjugate Gradient Method. This algorithm can be found, for example, in [4] or in [8].

If we now also use a preconditioner, then we end up with the Preconditioned Conjugate Gradient Method, or the PCG method. The idea behind it is that we solve a transformed system $\tilde{A}\tilde{y} = \tilde{b}$, where:

$$\begin{aligned} \tilde{A} &= P^{-1} A P^{-T}, \\ \tilde{y} &= P^T y, \\ \tilde{b} &= P^{-1} b. \end{aligned}$$

We assume P to be a nonsingular matrix. We now define M to be $M := PP^T$ to be the preconditioner and we write down the algorithm such that the tildes do not appear. A complete derivation can be found, for example, in [4].

Algorithm 4.1 (PCG)

$y^{(0)}$ — initialization
$r^{(0)} = b - Ay^{(0)}$

for $k = 0, 1, \ldots,$ until convergence **do** — stop criterion
  $s^{(k)} = M^{-1} r^{(k)}$ — preconditioning

  if $(k = 0)$ **do**
    $p^{(0)} = s^{(0)}$
  else
    $\beta_k = \frac{(r^{(k)})^T s^{(k)}}{(r^{(k-1)})^T s^{(k-1)}}$ — *update of* $p^{(k)}$
    $p^{(k)} = r^{(k)} + \beta_k p^{(k-1)}$
  end if

  $\alpha_k = \frac{(r^{(k)})^T s^{(k)}}{(p^k)^T A p^{(k)}}$
  $y^{(k+1)} = y^{(k)} + \alpha_k p^{(k)}$ — *update iterate*
  $r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}$ — *update residual*
end for

For this algorithm we can guarantee that the denominator $(r^{(k-1)})^T s^{(k-1)} = (s^{(k-1)})^T M s^{(k-1)}$ never becomes zero while $r^{(k-1)} \neq 0$, because M is a symmetric positive definite matrix.

The PCG method solves the following minimization problem:

$$\|y - y^{(i)}\|_A = \min_{x \in \mathcal{K}_i(M^{-1}A; M^{-1} r^{(0)})} \|y - x\|_A.$$

4.4.2 Incomplete Cholesky Decomposition

To explain these kinds of preconditioners we assume $A \in \mathbb{R}^{n\times n}$ to be a matrix with at most 5 non-zero elements per row. Furthermore we assume A to be symmetric and positive definite. This matrix can, for example, represent a discretization of a partial differential equation on a 2-dimensional grid. If we assume that we have a 5-point stencil and m is the number of grid points in the x-direction, then A is of the form:

$$\mathbf{A} = \begin{bmatrix} a_1 & b_1 & & c_1 & & & & & \\ b_1 & a_2 & b_2 & & c_2 & & & & \emptyset \\ \vdots & \ddots & \ddots & & \emptyset & \ddots & & & \\ c_1 & \emptyset & b_m & a_{m+1} & b_{m+1} & & c_{m+1} & & \\ \emptyset & \ddots & & \ddots & \ddots & \ddots & & \ddots & \end{bmatrix}. \tag{4.8}$$

In order to the best convergence behaviour we take a lower triangular matrix L such that $A = L^T L$ and we take $P = L$. L is called the Cholesky factor. It is known that we have to deal with so called fill-in. If we observe the fill-in, we see a decrease in size if the 'distance' to the non zero element of A increases. This decrease motivates to discard the fill-in elements entirely. This will lead to an *incomplete Cholesky decomposition* of A.

Now we have to construct L first, before we can use it in the PCG algorithm (Algorithm 4.1). Inside the algorithm we need multiplications with L^{-1} and L^{-T}. Since it is expensive to calculate these matrices we do not calculate them. If we for example have to calculate $s = L^{-1}r$, we compute s by solving the linear system $Ls = r$. Since L is a lower triangular matrix this can be done with low effort using Forward Elimination.

We now want to compute the incomplete Cholesky decomposition: $A = LD^{-1}L^T - N$. The elements of L and D have to satisfy the following rules:

- $l_{ij} = 0$ for all (i, j) where $a_{ij} = 0 \;\; i > j$,
- $l_{ii} = d_{ii}$,
- $(LD^{-1}L^T)_{ij} = a_{ij}$ for all (i, j) where $a_{ij} \neq 0 \;\; i \geq j$.

If the elements of L are given as follows:

$$\mathbf{L} = \begin{bmatrix} \tilde{d}_1 & & & & \\ \tilde{b}_1 & \tilde{d}_2 & & & \\ & \ddots & \ddots & \emptyset & \\ \tilde{c}_1 & & \tilde{b}_m & \tilde{d}_{m+1} & \\ & \ddots & \emptyset & \ddots & \ddots \\ \emptyset & & & & \end{bmatrix}, \tag{4.9}$$

then we have:

$$\left.\begin{array}{rcl} \tilde{d}_i & = & a_i - \frac{b_{i-1}^2}{\tilde{d}_{i-1}} - \frac{c_{i-m}^2}{\tilde{d}_{i-m}} \\ \tilde{b}_i & = & b_i \\ \tilde{c}_i & = & c_i \end{array}\right\} i = 1, \ldots, N. \tag{4.10}$$

If we now define the preconditioner M to be $LD^{-1}L^T$ and we use this M in Algorithm 4.1, we have the *Incomplete Cholesky Conjungate Gradient Method* or ICCG. More information can be found in, for example, [8].

4.4.3 Modified ICCG

The modified incomplete Cholesky preconditioner is constructed by slightly adapted rules. We again split $A = LD^{-1}L^T - N$, but L and D must now satisfy:

- $l_{ij} = 0$ for all (i,j) where $a_{ij} = 0 \;\; i > j$,
- $l_{ii} = d_{ii}$,
- $rowsum(LD^{-1}L^T) = rowsum(A)$ for all rows and $(LD^{-1}L^T)_{ij}$ for all (i,j) where $a_{ij} \neq 0 \;\; i \geq j$.

The consequence of the third point is that we have:

$$LD^{-1}L^T \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = A \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \Rightarrow (LD^{-1}L^T)^{-1} A \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}. \tag{4.11}$$

We can conclude that if $Ay = b$ and x and/or b are slowly varying vectors, then this modified incomplete Cholesky decomposition is a very good approximation for the inverse of A with respect to x and/or b. We now end up with:

$$\left.\begin{array}{rcl} \tilde{d}_i & = & a_i - (b_{i-1} + c_{i-1})\frac{b_{i-1}}{\tilde{d}_{i-1}} - (b_{i-1} + c_{i-1})\frac{c_{i-m}}{\tilde{d}_{i-m}} \\ \tilde{b}_i & = & b_i \\ \tilde{c}_i & = & c_i \end{array}\right\} i = 1, \ldots, N. \tag{4.12}$$

MODFLOW uses a slightly different MICCG method, with relaxation. In appendix B you can read how this is implemented.

4.5 Convergence, Starting Vectors and Stopping Criteria

Convergence

The rate of convergence of the conjugate gradient method is bounded as a function of the condition number of the system matrix to which it is applied. We let $A \in \mathbb{R}^{n \times n}$ be a symmetric positive definite matrix and we assume the vector $b \in \mathbb{R}^n$ to represent a discrete function on a grid Ω. We are searching for the vector $y \in \mathbb{R}^n$ on Ω which solves the linear system

$$Ay = b.$$

If we use a finite volume method to discretize a partial differential equation, then we end up with such a system which we have to solve.

We denote the spectrum of A by $\sigma(A)$ and the ith eigenvalue in nondecreasing order by $\lambda_i(A)$, or simply λ_i if it is clear to which matrix we refer. It is known that after k iterations of the conjugate gradient method, the error is bounded by:

$$\|y - y^{(i)}\|_A \leq 2\|y - y^{(0)}\|_A \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k. \tag{4.13}$$

The proof can be found, for example, in [3] or [8]. Here, $\kappa = \kappa(A) = \lambda_n/\lambda_1$ is the (spectral) condition number of A and the A-norm of y is given by $\|y\|_A = \left(y^T A y\right)^{1/2}$. In [7] one can find the following bound for the preconditioned conjugate gradient method,

$$\|y - y^{(i)}\|_A \leq 2\|y - y^{(0)}\|_A \left(\frac{\sqrt{\kappa(MA)} - 1}{\sqrt{\kappa(MA)} + 1}\right)^k. \tag{4.14}$$

Starting Vectors

All iterative solution methods to solve $Ay = b$ start with a given vector $y^{(0)}$. Choosing a good starting vector can decrease the number of iterates that is needed. The choice you make is dependent on the problem and the information you have. If you do not have any information then one typically starts with $y^{(0)} = \mathbf{0}$. In case of a nonlinear problem, the solution is approximated by the solution of a number of linear systems. One can use the solution of a given outer iteration as a starting vector for the next iterative method used to solve the next linear system.

Sometimes starting vectors can also be obtained by the solution of a related problem. One can use the analytic solution of a simplified problem or a solution which is obtained by using a coarser grid.

The better the starting vector approximates the solution, the lesser iterations are needed.

Stop Criterion

It is also very important to define a good criterion to stop. If the criterion is too weak, then the approximate solution is useless. But if the criterion is too severe, then the iterative solution method might never stop or simply costs too much work.

An iterative method is linearly convergent if it satisfies:

$$\|y^{(k)} - y^{(k-1)}\|_2 \approx \bar{\rho}\|y^{(k-2)} - y^{(k-1)}\|_2, \;\; \bar{\rho} < 1,$$

and $y^{(k)} \to A^{-1}b$ for $k \to \infty$. This criterion is easily checked during the iteration. Initially, this relation will not hold, but after some iterations the quantity $\frac{\|y^{(k)} - y^{(k-1)}\|_2}{\|y^{(k-2)} - y^{(k-1)}\|_2}$ will converge to $\bar{\rho}$. Theorem 4.1 from [8] states that we have the following inequality:

$$\|y - y^{(k)}\|_2 \leq \frac{\bar{\rho}}{1 - \bar{\rho}}\|y^{(k)} - y^{(k-1)}\|_2.$$

This result can be used to define the following stopping criterion for linear convergent methods:

$$\text{Stop if:} \;\; \frac{\bar{\rho}}{1 - \bar{\rho}} \frac{\|y^{(k)} - y^{(k-1)}\|_2}{\|y^{(k)}\|_2} \leq \epsilon.$$

If this condition holds, then you can show that the relative error is smaller than ϵ:

$$\frac{\|y - y^{(k)}\|_2}{\|y\|_2} \cong \frac{\|y - y^{(k)}\|_2}{\|y^{(k)}\|_2} \leq \frac{\bar{\rho}}{1 - \bar{\rho}} \frac{\|y^{(k)} - y^{(k-1)}\|_2}{\|y^{(k)}\|_2} \leq \epsilon.$$

For iterative method that have a different convergence behaviour most stopping criteria are based on the norm of the residual. Now we list some possible criteria with some comments.

$$\text{Criterion 1} \quad \|b - Ay^{(k)}\|_2 \leq \epsilon$$

$$\text{Criterion 2} \quad \frac{\|b-Ay^{(k)}\|_2}{\|b-Ay^{(0)}\|_2} \leq \epsilon$$

$$\text{Criterion 3} \quad \frac{\|b-Ay^{(k)}\|_2}{\|b\|_2} \leq \epsilon$$

$$\text{Criterion 4} \quad \frac{\|b-Ay^{(k)}\|_2}{\|y^{(k)}\|_2} \leq \epsilon/\|A^{-1}\|_2$$

Criterion 1 is not scaling invariant. If $\|b - Ay^{(k)}\|_2 \leq \epsilon$, then it does not have to hold for $\|100(b-Ay^{(k)})\|_2$, while the accuracy of $y^{(k)}$ remains the same. In criterion 2 the number of iterates is now dependent on the initial estimate $y^{(0)}$. So a better initial estimate will not lead to a decrease in the number of iterations. If the initial estimate is very good, then it is possible that the method never stops due to round-off errors. The third criterion is a good one. The norm of the residual should be small with respect to the norm of the right-hand side. If we replace ϵ by $\epsilon/\kappa_2(A)$, then we can show that the error in y is less than ϵ, where $\kappa_2(A)$ is the condition number: $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$. In general $\|A\|_2$ and $\|A^{-1}\|_2$ are not known, but some iterative methods give approximations of these quantities. Criterion 4 is related to criterion 3 and in many cases this criterion implies that the relative error is smaller than ϵ.

It is also possible that a stopping criterion comes from physical relations. This can happen when the residuals have a physical meaning.

4.6 The MODFLOW solver

Since groundwater flow is modelled by a non-linear equation we need a special solver. MODFLOW uses a Picard outer iteration to linearize the problem. Once this is done it jumps to the PCG solver, which is also called the inner iteration process. If the solution converges inside the inner iteration or if the maximum number of inner iterations is reached, MODFLOW runs another outer iteration to linearize the problem again. It is possible that a solution obtained from an inner loop changes the system. Think for example about a drain which is activated because the groundwater rises on a particular place. After the new linearization the model runs again its inner loop. This is repeated until the solution converges in one step during an inner loop. If the solution converges in one step then the new solution will not effect the system anymore and we have the final solution.

MODFLOW uses two criteria to determine if we already have convergence. The first one checks between two successive solutions. If for every cell the head does not change more than a pre-specified number than we fulfill this criterion . The second criterion checks on the flux. If the total flux in the system is smaller then a pre-specified amount we fulfill this criterion. If both criteria are fulfilled, then we stop the inner loop. The stop criteria in MODFLOW are given by:

$$\|y_i^{(k)} - y_i^{(k-1)}\|_\infty \leq \textit{hclose},$$

and

$$\|r_i^{(k)} - r_i^{(k-1)}\|_\infty \leq \textit{rclose},$$

where *hclose* $= 0.001$ and *rclose* $= 10$.

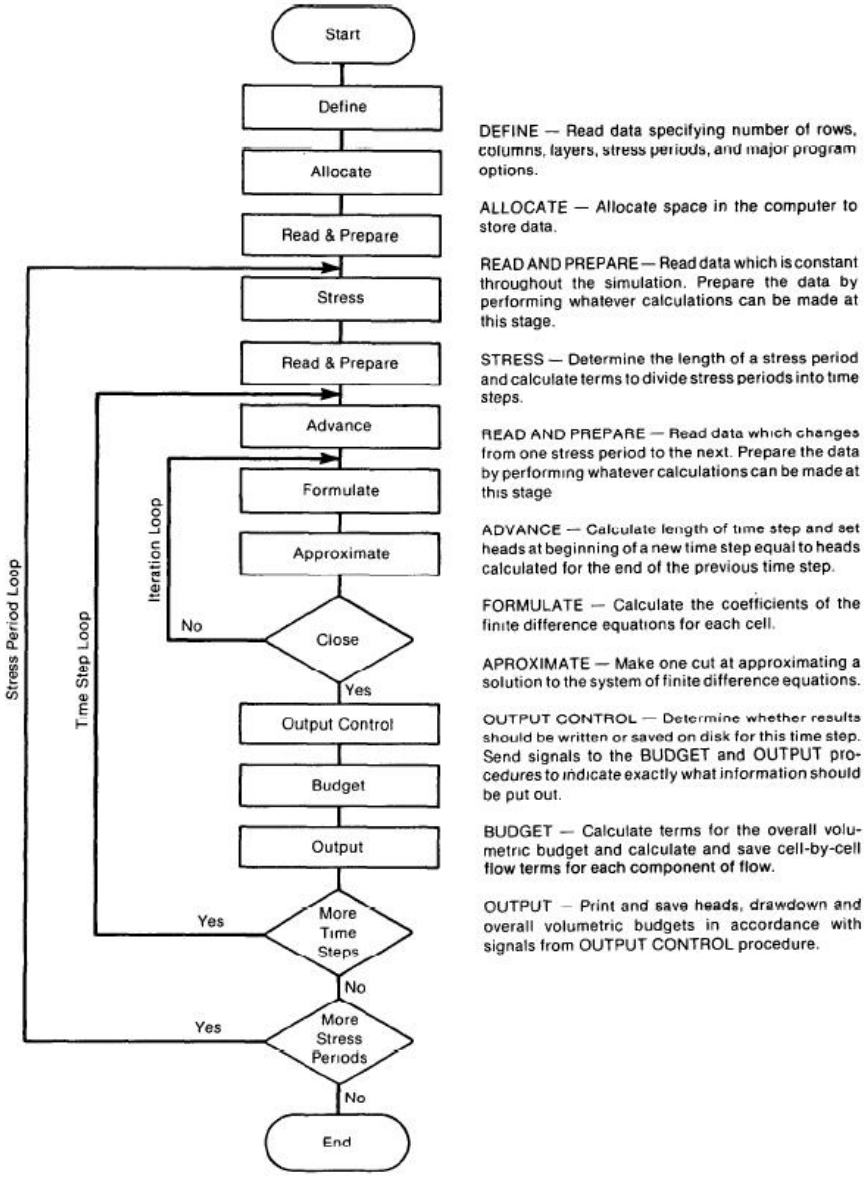

Figure 4.2: Flow chart of the MODFLOW solver [5]. See also Figure 3.7.

Chapter 5

Deflation Techniques

It is known that the rate of convergence is dependent on the condition number of A. The condition number is equal to λ_n/λ_1, where λ_n is the larges eigenvalue and λ_1 the smallest. In order to improve convergence we can try to increase the smallest eigenvalue or to decrease the biggest. In [16] it is proven that if we have large contrasts in our coefficients, for example due to a fault in the subsoil, that we have some extreme small eigenvalues. The idea of deflation now is to define a deflation subspace, which approximates the eigenspace belonging to those small eigenvalues. Then we can project the eigendirections of those small eigenvalues out of the residual. So we remove these extreme small eigenvalues. In practice the smallest eigenvalues become zero and do not have effect on the condition number anymore. The smallest eigenvalue, not equal to zero, left is now bigger than the original one. So the condition number decreases and convergence should be improved.

We start by explaining the basic idea of deflation in Section 5.1. In Section 5.2 we describe the idea of eigenvalue deflation. Section 5.3 gives an overview of some other possible deflation techniques. Finally, Section 5.4 visualises the idea when we look at a simple example. This chapter is mostly based on [3, 12, 14, 16].

5.1 Basic Idea of Deflation

Again we look at a general linear system of the form

$$Ay = b, \tag{5.1}$$

where we assume A to be symmetric positive definite. Let $Z \in \mathbb{R}^{n\times m}$ be a matrix where $m \leq n$. Furthermore we assume that the rank of Z is m, i.e. Z has linear independent columns. We call the columns of Z the *deflation vectors* and they span the deflation subspace. The deflation subspace is the space that approximates the eigenspace belonging to the smallest eigenvalues and which is to be projected out of the residual. For this purpose we define the projector

$$P \;=\; I - AZE^{-1}Z^T, \tag{5.2}$$

where we define the $m \times m$ matrix $E = Z^T AZ$. Note that we have the following expressions:

$$E^{-T} = (Z^T AZ)^{-T} = (Z^T A^T Z^{TT})^{-1} = (Z^T AZ)^{-1} = E^{-1},$$
$$P^T = (I - AZE^{-1}Z^T)^T = I^T - Z^{TT}E^{-T}Z^T A^T = I - ZE^{-1}Z^T A.$$

If we now want to solve system (5.1) using deflation, we can write $y = (I - P^T)y + P^T y$. Since we have:

$$(I - P^T)y = Z(Z^T AZ)^{-1}Z^T Ay = ZE^{-1}Z^T b, \tag{5.3}$$

which can be computed immediately, we only need to compute $P^T y$. Because we have that $AP^T = PA$ we can solve the system

$$PA\tilde{y} = Pb, \tag{5.4}$$

for $\tilde{y}$, and simply add $P^T\tilde{y}$ to Equation (5.3). The system (5.4) is called the *deflated system*.

Note that we have

$$\begin{aligned} PAZ &= (I - AZE^{-1}Z^T)AZ, \\ &= AZ - AZE^{-1}Z^T AZ, \\ &= AZ - AZE^{-1}E, \\ &= AZ - AZ, \\ &= 0, \end{aligned}$$

and therefore the system is singular, so there is no unique solution. However it can be shown that $P^T y$ is unique. A proof for the case that A is symmetric positive definite can be found in [16].

Since we want to use an iterative solution method we need an initial guess $\tilde{y}^{(0)}$. When $y^{(0)}$ is given we solve:

$$Au = f,$$

where $u \equiv y - y^{(0)}$ and $f \equiv b - Ay^{(0)}$. To do this we solve the deflated system

$$PA\tilde{u} = Pf,$$

for $\tilde{u}$ and we construct:

$$y = \left(I - P^T\right)u + P^T\tilde{u} + y^{(0)} \tag{5.5}$$

5.1.1 Preconditioning and Starting Vectors

It is also possible to combine deflation with preconditioning. If M is a suitable preconditioner of A, then Equation (5.4) can be replaced by:

$$M^{-1}PA\tilde{u} = M^{-1}Pb, \tag{5.6}$$

and we have to solve $\tilde{u}$ from it. After that we have to form $P^T\tilde{u}$ again. This is called left preconditioning. We can also use right preconditioning, then we must solve $\tilde{u}$ from

$$PAM^{-1}\tilde{u} = Pb$$

and form $P^T M^{-1}\tilde{u}$.

Both systems can be solved by a Krylov subspace method. Since A is a symmetric positive definite matrix we can use for example a CG method. If we use deflation we have to determine expressions as $f = E^{-1}g$. To do this efficiently we solve the system $Ef = g$ instead of calculating E^{-1} directly. To solve such a system we use an LL^T factorization which involves the following steps:

- compute L such that $E = LL^T$,
- solve $Lq = g$,
- solve $L^T f = q$.

The algorithms for evaluating these steps are given in appendix A, by respectively Algorithms A.1, A.2 and A.3.

We can use as a preconditioner, for example, a modified incomplete Cholesky decomposition as given in Algorithm 5.1. The algorithm is written such that it follows closely the implementation of [6]. The method is called the *Deflated Incomplete Cholesky Conjugate Gradient Method*, or simply DICCG. The success of the DICCG method is related to the choice of Z.

5.1.2 Deflated CG method

Equation (5.5) gives the solution for our original problem if we use deflation. If we rewrite this equation we get:

$$y = \left(I - P^T\right)\left(y - y^{(0)}\right) + P^T\tilde{y} + y^{(0)}.$$

Since we have:

$$P^T = \left(I - AZE^{-1}Z^T\right)^T = I - ZE^{-1}Z^T A,$$

we find that:

$$\begin{aligned} y &= ZE^{-1}Z^T Ay - ZE^{-1}Z^T Ay^{(0)} + \left(I - ZE^{-1}Z^T A\right)\tilde{y}^{(k+1)} + y^{(0)}, \\ &= ZE^{-1}Z^T b - ZE^{-1}Z^T Ay^{(0)} - ZE^{-1}Z^T A\tilde{y}^{(k+1)} + \tilde{y}^{(k+1)} + y^{(0)}, \\ &= ZE^{-1}Z^T r^{(0)} - ZE^{-1}Z^T A\tilde{y}^{(k+1)} + \tilde{y}^{(k+1)} + y^{(0)}, \\ &:= Zq_1 - Zq_2 + \tilde{y}^{(k+1)} + y^{(0)}, \\ &= Z\left(q_1 - q_2\right) + \tilde{y}^{(k+1)} + y^{(0)}, \end{aligned} \tag{5.7}$$

where we define:

$$\begin{aligned} q_1 &:= E^{-1}Z^T r^{(0)}, \\ q_2 &:= E^{-1}Z^T A\tilde{y}^{(k+1)}. \end{aligned}$$

Note that $y^{(0)}$ is the original initial value and that $\tilde{y}^{(k+1)}$ is the vector y which follows from the iteration process for the deflated system. Furthermore we have to construct q_1 and q_2 during the algorithm.

The Deflated CG method can be implemented such that it follows the PCG algorithm as given in Algorithm 4.1. When deflation is used, some extra work need to be done. This extra work is carried out in three phases during the algorithm. If we do not use deflation we still have the standard PCG algorithm.

The first piece of extra work is called the *deflation pre-processing phase* and is carried out before we actually start the iteration process. During this phase the original initial condition is stored in the computer and the new initial condition is set to zero. We compute q_1 and we update the residual. Since we must solve $PA\tilde{y} = Pf$ (Equation (5.6)) and we start with the zero vector as initial condition for deflation we can calculate the residual in case of deflation.

$$\begin{aligned}
\tilde{r}^{(0)} &= Pf - PA\tilde{y}^{(0)}, \\
&= P\left(b - Ay^{(0)}\right) - PA\underline{0}, \\
&= \left(I - AZE^{-1}Z^T\right) r^{(0)}, \\
&= r^{(0)} - AZE^{-1}Z^T r^{(0)}, \\
&= r^{(0)} - AZq_1.
\end{aligned}$$

During the iteration process we have to update the vector v for each iteration step. This is all done in the so-called *deflation run-time phase*.

After the iteration process we have to calculate the solution for our original problem. This is done in the *deflation post-processing phase*. We now construct the q_2 we need and we can calculate our solution using Equation (5.7).

The corresponding algorithm is now given by Algorithm 5.1.

Algorithm 5.1 (DICCG) Given a $n \times n$ symmetric positive definite matrix A, a vector b, the modified Cholesky preconditioner $LD^{-1}L^T$ where L is lower diagonal and D^{-1} the inverse diagonal of L, the projector P as in Equation(5.2), and an initial guess $y^{(0)}$. This algorithm solves the linear system $Ay = b$. If we need to solve a system involving E^{-1} we use a LU-factorization as described in Appendix A.

$r^{(0)} = b - Ay^{(0)}$
if deflation **do** (/* deflation pre-processing phase)
  $\tilde{y}^{(0)} = y^{(0)}$;
  $y^{(0)} = 0$;
  Decompose $Z^T AZ = \tilde{L}\tilde{U}$
  Solve $\tilde{L}\tilde{q}_1 = Z^T r^{(0)}; \tilde{U}q_1 = \tilde{q}_1$
  $r^{(0)} := r^{(0)} - AZq_1$
end if
for $k = 0, 1, \ldots,$ convergence **do**
  Solve $L\tilde{s} = r^{(k)}; D^{-1}L^T s^{(k)} = \tilde{s}$;
  if $k = 0$ **do**
    $p^{(0)} = s^{(0)}$;
  else
    $\beta = \frac{(s^{(k)})^T r^{(k)}}{(r^{(k-1)})^T s^{(k-1)}}$;
    $p^{(k)} = s^{(k)} + \beta p^{(k-1)}$;
  end if
  $v^{(k)} = Ap^{(k)}$;
  if deflation **do** (/* deflation run-time phase)
    Solve $\tilde{L}\tilde{q}_3 = Z^T v^{(k)}; \tilde{U}q_3 = \tilde{q}_3$;
    $v^{(k)} := v^{(k)} - AZq_3$;
  end if
  $\alpha = \frac{(r^{(k)})^T s^{(k)}}{(p^{(k)})^T v^{(k)}}$;

$y^{(k+1)} = y^{(k)} + \alpha p^{(k)}$;
$r^{(k+1)} = r^{(k)} - \alpha v^{(k)}$;
end for
if deflation **do** (/* deflation post-processing phase)
Solve $\tilde{L}\tilde{q}_2 = Z^T A y^{(k+1)}; \tilde{U} q_2 = \tilde{q}_2$;
$y^{(k+1)} := y^{(k+1)} + \tilde{y}^{(0)} + Z(q_1 - q_2)$;
end if

5.2 Eigenvalue Deflation

Eigenvalue deflation means that we take exact or perturbed eigenvectors, corresponding to the small eigenvalues, to build our deflation matrix Z. First we look at deflation with exact eigenvectors and after that we look at perturbed eigenvectors.

5.2.1 Deflation with Exact Eigenvectors

We order the eigenvalues of A such that $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ and we choose their corresponding eigenvectors v_j such that $v_j^T v_i = \delta_{ij}$, where δ_{ij} stands for the Kronecker delta. Furthermore we define $Z := [v_1 v_2 \ldots v_m]$. We will show that the spectrum of PA satisfies:

$$\sigma(PA) = \{0, \ldots, 0, \lambda_{m+1}, \ldots, \lambda_n\}$$

Note that we can denote AZ as:

$$AZ = [Av_1\ Av_2\ \cdots\ Av_n] = [\lambda_1 v_1\ \lambda_2 v_2\ \cdots\ \lambda_n v_n],$$

so that we have:

$$\begin{aligned} E = Z^T AZ &= \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_m^T \end{bmatrix} \begin{bmatrix} \lambda_1 v_1 & \lambda_2 v_2 & \cdots & \lambda_m v_m \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1 v_1^T v_1 & \lambda_2 v_1^T v_2 & \cdots & \lambda_m v_1^T v_m \\ \lambda_1 v_2^T v_1 & \lambda_2 v_2^T v_2 & \cdots & \lambda_m v_2^T v_m \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1 v_m^T v_1 & \lambda_2 v_m^T v_2 & \cdots & \lambda_m v_m^T v_m \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1 & & & \emptyset \\ & \lambda_2 & & \\ & & \ddots & \\ \emptyset & & & \lambda_m \end{bmatrix} = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m). \end{aligned}$$

Now look at the following:

$$E^{-1} Z^T = \begin{bmatrix} \frac{1}{\lambda_1} & & \emptyset \\ & \ddots & \\ \emptyset & & \frac{1}{\lambda_m} \end{bmatrix} \begin{bmatrix} v_1^T \\ \vdots \\ v_m^T \end{bmatrix} = \begin{bmatrix} \frac{1}{\lambda_1} v_1^T \\ \vdots \\ \frac{1}{\lambda_m} v_m^T \end{bmatrix},$$

$$AZE^{-1}Z^T = \begin{bmatrix} \lambda_1 v_1 & \cdots & \lambda_m v_m \end{bmatrix} \begin{bmatrix} \frac{1}{\lambda_1} v_1^T \\ \vdots \\ \frac{1}{\lambda_m} v_m^T \end{bmatrix} = v_1 v_1^T + \ldots + v_m v_m^T, \tag{5.8}$$

$$ZZ^T = \begin{bmatrix} v_1 & \cdots & v_m \end{bmatrix} \begin{bmatrix} v_1^T \\ \vdots \\ v_m^T \end{bmatrix} = v_1 v_1^T + \ldots + v_m v_m^T. \tag{5.9}$$

From Equation (5.8) and Equation (5.9) we can conclude that $AZE^{-1}Z^T = ZZ^T$ and therefore it holds that:

$$P = I - AZE^{-1}Z^T = I - ZZ^T, \tag{5.10}$$

and we can write:

$$PAv_j = \left(I - ZZ^T\right) \lambda_j v_j. \tag{5.11}$$

If we now denote Z as:

$$Z = \begin{bmatrix} v_1^1 & v_2^1 & \cdots & v_m^1 \\ v_1^2 & v_2^2 & \cdots & v_m^2 \\ \vdots & \vdots & \ddots & \vdots \\ v_1^n & v_2^n & \cdots & v_m^n \end{bmatrix},$$

then:

$$ZZ^T = \begin{bmatrix} (v_1^1 v_1^1 + \ldots + v_m^1 v_m^1) & (v_1^1 v_1^2 + \ldots + v_m^1 v_m^2) & \cdots & (v_1^1 v_1^n + \ldots + v_m^1 v_m^n) \\ (v_1^2 v_1^1 + \ldots + v_m^2 v_m^1) & (v_1^2 v_1^2 + \ldots + v_m^2 v_m^2) & \cdots & (v_1^2 v_1^n + \ldots + v_m^2 v_m^n) \\ \vdots & \vdots & \ddots & \vdots \\ (v_1^n v_1^1 + \ldots + v_m^n v_m^1) & (v_1^n v_1^2 + \ldots + v_m^n v_m^2) & \cdots & (v_1^n v_1^n + \ldots + v_m^n v_m^n) \end{bmatrix}.$$

Now we are going to calculate $ZZ^T v_j$ for $j = 1, \ldots, m, m+1, \ldots, n$. Herefore we look at the k^{th} column. That column is given by:

$$\left\{v_1^k v_1^1 v_j^1 + \ldots + v_m^k v_m^1 v_j^1\right\} + \left\{v_1^k v_1^2 v_j^2 + \ldots + v_m^k v_m^2 v_j^2\right\} + \ldots + \left\{v_1^k v_1^n v_j^n + \ldots + v_m^k v_m^n v_j^n\right\},$$

which can be rewritten as:

$$\begin{aligned} & \left\{v_1^k v_1^1 v_j^1 + v_1^k v_1^2 v_j^2 + \ldots + v_1^k v_1^n v_j^n\right\} + \left\{v_2^k v_2^1 v_j^1 + v_2^k v_2^2 v_j^2 + \ldots + v_2^k v_2^n v_j^n\right\} + \\ & \quad \cdots + \left\{v_m^k v_m^1 v_j^1 + v_m^k v_m^2 v_j^2 + \ldots + v_m^k v_m^n v_j^n\right\} \\ = \; & v_1^k \left(v_1^T v_j\right) + v_2^k \left(v_2^T v_j\right) + \cdots + v_m^k \left(v_m^T v_j\right). \\ = \; & \begin{cases} v_j^k & \text{if} \quad j = 1, 2, \ldots, m \\ 0 & \text{if} \quad j = m+1, \ldots, n \end{cases} \end{aligned}$$

where the last step is a consequence of $v_i^T v_j = \delta_{ij}$. So we can draw the following conclusion:

$$ZZ^T v_j = \begin{cases} v_j & \text{if} \quad j = 1, 2, \ldots, m \\ 0 & \text{if} \quad j = m+1, \ldots, n \end{cases} \tag{5.12}$$

If we now substitute expression (5.12) into Equation (5.11) we can conclude that:

$$PAv_j = \begin{cases} \mathbf{0} & \text{if} \quad j = 1, 2, \ldots, m \\ \lambda_j v_j & \text{if} \quad j = m+1, \ldots, n \end{cases}$$

This shows that eigenvector deflation cancels the smallest m eigenvalues of the spectrum of A, leaving the rest of the spectrum untouched. Although deflation with exact eigenvectors leads to fast convergence, such a choice will be inefficient in practice. This is because it is relatively expensive or impossible to compute the eigenvalues.

5.2.2 Deflation with Inexact Eigenvectors

It is also possible to use perturbed or approximate eigenvectors, such as Ritz vectors (see, for example, [4] or [8]). This means that we define $\bar{Z} := [\bar{v}_1 \bar{v}_2 \ldots \bar{v}_m]$, where each $\bar{v}_i$ is an approximation of the exact eigenvector $\mathbf{v}_i$. So:

$$\bar{v}_i := v_i + \delta_i, \quad \delta_i \in \mathbb{R}^n$$

It would be convenient that deflation with $\bar{V}$ has the same favorable features as deflation with V.

5.3 A Priori Deflation Vectors

Since it is expensive to calculate eigenvalues and eigenvectors of a (large) matrix we have to find other types of a priori deflation vectors. We list some possibilities.

5.3.1 Subdomain Deflation

In Section 5.2 it is mentioned that deflation of an eigenspace cancels the corresponding eigenvalues without affecting the rest of the spectrum. Calculating the eigenvectors and eigenvalues of a (large) system is expensive and therefore not feasible. This motivates to deflate with "nearly invariant" subspaces obtained during the iteration. We can make a specific choice for the subspace Z, based on the decomposition of the domain Ω into m disjunct sets Ω_j such that $\bigcup_{j=1}^{m} \Omega_j = \Omega$. The division in subdomains can be motivated, for example, by jumps in the coefficients.

Since MODFLOW uses a cell-centered grid we look at possible deflation techniques for this discretization. Since in a cell-centered grid the unknowns are located in the interior of a finite volume, domain decomposition is straightforward. The algebraic deflation vectors z_j are uniquely defined as:

$$z_j(x_i) = \begin{cases} 1, & \text{for} \quad x_i \in \Omega_j \\ 0, & \text{for} \quad x_i \in \Omega \backslash \Omega_j. \end{cases}$$

So now the question remains how we are going to choose the domains. If we do not have any information we can just cut the domain into squares (2D) or cubes (3D) and

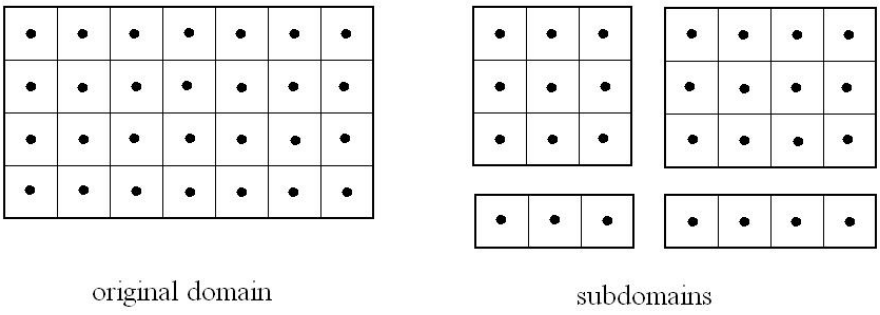

Figure 5.1: Random decomposition for a cell centered discretization.

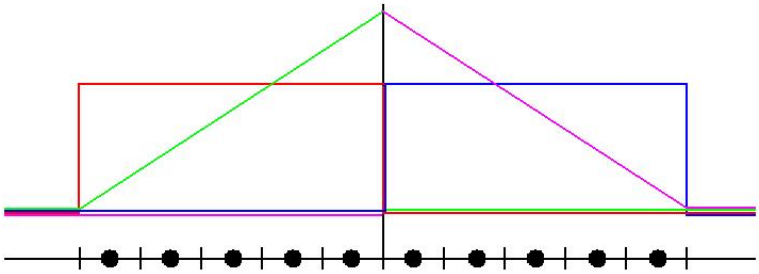

Figure 5.2: 4 possible deflation vectors in a 1-D case.

make the corresponding deflation vectors, see also Figure 5.1. This method is straight forward to implement on a computer.

Note that it is also possible to use linear vectors instead of constant vectors. See also Figure 5.2. The red and the blue one are constant vectors, where the green and the magenta ones are linear.

5.3.2 Deflation based on Physics

In our case it is possible to define the subdomain by using the geographical places of the faults. Since MODFLOW defines the fault to be exactly on the border of a cell, theoretically this is possible. For this kind of deflation we have two options.

The first option is to divide the domain into subdomains as in subsection 5.3.1, but now using the faults as boundaries. A second option could be to define the vectors such that an element next to the fault has value 1 and the rest value 0. We can make for each fault a vector. But we have to think about what to do with the elements that are next to two or more faults.

If we use a layered grid we have also another deflation type which is based on the physics. We can define a deflation vector per layer. In this case we divide the whole grid in subdomains defined by the layers. We will refere to this technique as *layer deflation*. The advantage of layer deflation is that it is relatively easy to implement. Especially in the way we number the cells in MODFLOW (see Section 3.1) we get a structured matrix Z.

5.4 A Simple Test Case in Matlab

We consider a simple case in Matlab to give more insight in the deflation technique. The problem consists of a rectangular area containing 15 rows, 15 columns and only one layer (2 dimensional). On the east side of the domain we defined constant head cells with $h = 0$. On the other boundaries we defined the so called no-flow cells. There also is a recharge on the whole area of $7 \cdot 10^{-4}$ meters per day. To make it a realistic problem we also add drains to the whole domain. The last thing we add is a horizontal flow barrier to simulate a fault. This barrier is located vertically on one thirth of the domain. So in this case it was located between the fifth and sixth column. The conductance of the barrier was set at 10^{-9}. If we rotate the whole problem by 90 degrees we can also think of this problem as being a layered grid with 15 rows, 1 column and 15 layers where 1 layer has a low permeability. This specific layer can be seen as a package of clay between two layers.

We implemented the PCG algorithm as given in Algorithm 4.1 using the Incomplete Cholesky preconditioner as described in Section 4.4.2. We also implemented the deflated PCG algorithm as given in Algorithm 5.1. For the DICCG algorithm we divided the domain in two parts. The first part is the right side of the fault and the second part is the left side of the fault. As a consequence we used two vectors for the matrix Z. The first vector contains the value 1 for each cell corresponding to the first domain and all other cells got value 0. The second vector is exactly the opposite of the first vector.

We now can compare both algorithms and we see already some improvement for this simple test case. In Figure 5.3 we see that the residuals for deflation are decreasing faster than the ICCG method. We look at the spectrum of the matrices $M^{-1}A$ (ICCG) and $M^{-1}PA$ (deflation), see Figure 5.4. We see that the eigenvalues are almost the same, but if we zoom in to the smallest eigenvalue (Figure 5.5), we see that by using deflation this one is zero. So the smallest eigenvalue has been canceled and therefore the condition of the deflated matrix is better.

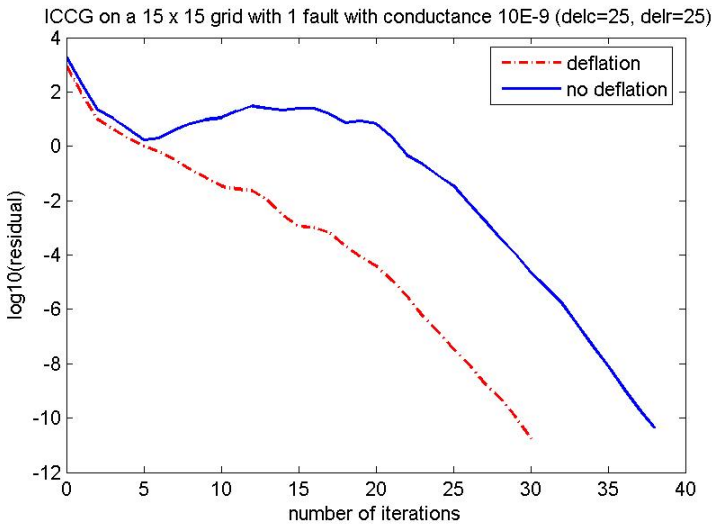

Figure 5.3: The residuals of the simple test case.

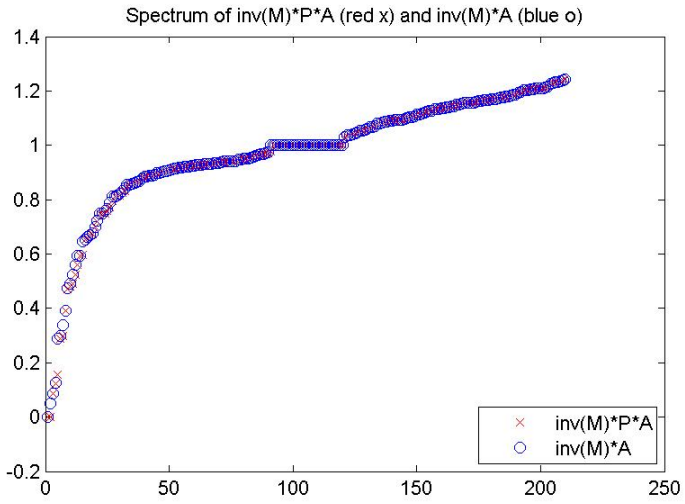

Figure 5.4: The eigenvalues of the simple test case.

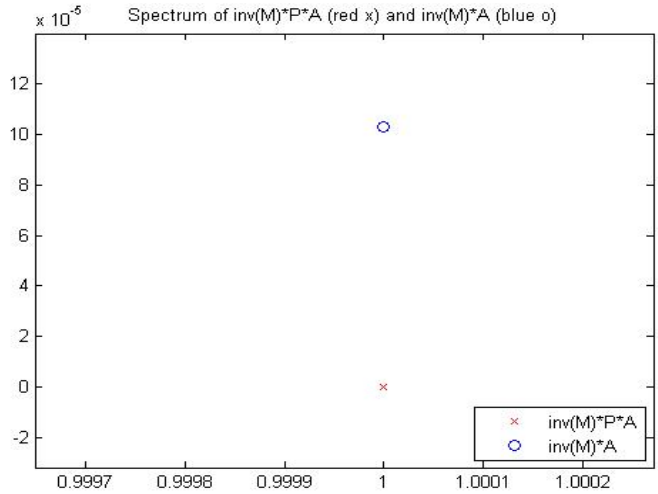

Figure 5.5: The smallest eigenvalue is set to zero by using deflation.

Chapter 6

Implementation and Results for IBRAHYM

IBRAHYM is a groundwater model that is developed for several water boards in Limburg. Limburg is a province in the Netherlands and has been schematized by 19 layers of sand which are separated by thick layers of clay. Also it has a large variety of faults in the subsoil. Such layers of clay and faults cause the model to suffer from bad convergence behaviour of the solver. IBRAHYM uses grid cells of 25 times 25 meter to get detailed information.

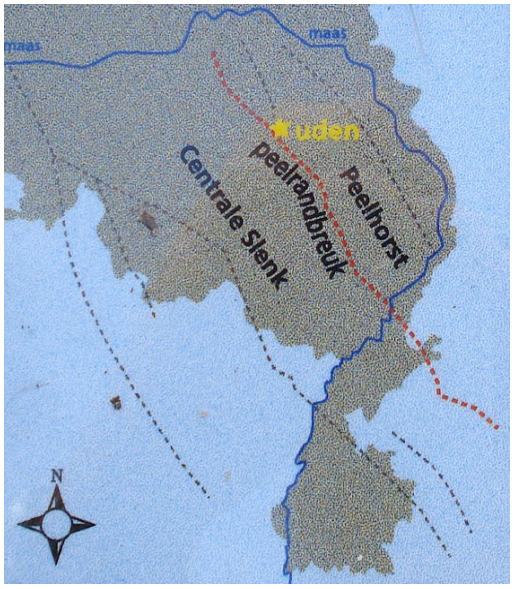

Figure 6.1: The most famous fault in Limburg is "De Peelrandbreuk" (Figure taken from [19]).

In this chapter we discuss the details about IBRAHYM. In Section 6.1 we introduce the problems Deltares has with the model. In Section 6.2 we discuss the implementation of the deflation technique. Section 6.3 goes in more detail on the optimization of the code. In Section 6.4 we discuss the results for some small areas inside the IBRAHYM model. Small areas do not use too much runtime, so we can compare a lot of variations that give us insight. Section 6.5 finally gives the results for a big area of interest. In the whole section we restrict ourselves to steady state runs.

6.1 Problem Description

IBRAHYM is a groundwater model for Limburg, which is developed by the former TNO and Alterra. The model is strongly based on MODFLOW, but when it comes to the solver of the model they now use exactly the same PCG solver as used in MODFLOW. In the IBRAHYM model bad convergence can be observed for certain areas. Deltares suspects that this is caused by the multiple faults inside the subsoil. When there is a fault one simply reduces the conductivity between two cells. The result is that we have to deal with a large contrast in the parameters. It is generally known that the standard PCG solver (like the ICCG) can not deal with these contrasts in the parameters easily. Large contrasts in the parameters cause very small eigenvalues which is a problem for the standard PCG solver.

There is also a problem with strong clay layers between the aquifers. In the model they cannot set the vertical conductances too small. If the resistance of the layer is getting larger, the conductance is getting smaller. They would like to put the conductance to zero, but than the method will not converge. This problem can also be explained by the large contrasts in the parameters. Again we have to deal with extreme small eigenvalues which causes the PCG solver to fail.

We tried to implement the deflation technique as explained in Chapter 5. We initially choose to implement deflation while storing the whole matrix Z. An algorithm was implemented for subdomain deflation (see Algorithm C.1). It turned out that memory was limiting for large areas so we had to think about some optimization steps. So we choose to implement layer deflation because this was the most simple case to implement since Z has a nice structure, which can be implemented efficiently.

The solutions of the heads could be made visual by using IMOD. IMOD is a program developed by TNO to visualize solutions. IMOD has a raster calculator which enables us to subtract two solutions and take a look at the differences. Also it it is possible to plot the faults in the figure to see where in your domain the faults are. Most figures in this chapter are made using IMOD.

6.2 Implementation in MODFLOW

As seen in Algorithm 5.1 one can implement the deflation technique by writing three blocks of code. These blocks we call the *deflation pre-processing phase*, the *deflation run-time phase* and the *deflation post-processing phase*. This structure makes it easier to implement the new code inside the original code. All three blocks are surrounded by an IF statement. We can initially set the flag 'DEFLATION' to 0 or 1. Putting it to 0 means we skip the deflation technique and setting it to 1 means we use deflation.

Almost all calculations are matrix-matrix products and matrix-vector products. During the implementation we tried to use the same methods that has already been used in the original code. So we will not discuss every detail here. Although it seems quite easy there are some difficulties. In this section we will describe how the deflation method is implemented within the existing PCG solver of MODFLOW.

6.2.1 Building Z for Subdomain Deflation

The matrix Z is the most important part in the deflation method. Choosing a suitable Z will improve convergence. For now we are going to store the matrix Z completely. In this way we can use the matrix further in the process for every possible Z we might

come up with. If we will find an optimal Z during some tests, we can think about another way to deal with Z such that we do not need the whole matrix.

First we start by building a matrix Z based on subdomain deflation, since it has proven many times that this already improves convergence considerably. Therefore we need to divide the grid into blocks. First we decide in how many blocks we divide the grid in each direction. In the row direction we take $NUMI$ blocks, in the column direction we take $NUMJ$ blocks and in the vertical direction we take $NUMK$ blocks. The number of deflation vectors we use now is $NUMJ \times NUMI \times NUMK$. Of course we cannot divide a direction in more blocks than we have cells in that direction, so we check if this is not the case. The user can specify the three values. It might be possible that the user specifies more blocks in a direction than there are cells. In this case the number of blocks is set to the number of cells in the corresponding direction ($NROW$, $NCOL$, and $NLAY$). Now an algorithm is needed to find out for each single cell in which block this cell lies.

The algorithm must step through every cell in a structured way. We use the same structured way as MODFLOW itself. We first loop over the layers ($K = 1, NLAY$), then over the rows ($I = 1, NROW$) and finally over the columns ($J = 1, NCOL$). This is the same way as in the original PCG code and the standard way in MODFLOW as described in Section 3.3. During the loops the algorithm calculates in which block the cell lies in each direction. These three values are stored in PK, PI, and PJ. So for example a cell can lie in the second block in the layer direction ($PK = 2$), in the third block in the row direction ($PI = 3$) and in fifth block in the column direction ($PJ = 5$).

Now the number of the corresponding cell is given by the expression $N = J + (I - 1) \times NCOL + (K - 1) \times NROW \times NCOL$, since we use the structure as discussed in Section 3.1. If we number the domain blocks in the same way we can calculate the corresponding number of the block by $L = PJ + (PI - 1) \times NUMJ + (PK - 1) \times NUMI \times NUMJ$. Then we only need to put a 1 in the matrix Z on the corresponding place, $Z(N, L)$.

The only thing we need to be aware of is that we sometimes deal with the so-called no-flow cells as discussed in Section 3.4. More on this will be discussed later in Section 6.2.4. The corresponding algorithm as implemented in MODFLOW using a fully stored matrix Z is given in Algorithm C.1 in Appendix C.

6.2.2 Building matrix $E = Z^T AZ$

Matrix $E = Z^T AZ$ is generally a small matrix which we store completely. If we use nz deflation vectors then $E \in \mathbb{R}^{nz \times nz}$. So it is a small matrix compared to the original system matrix. Now that we have chosen and build a matrix Z we can go on by calculating matrix E.

Using the matrix Z we need to calculate the matrix-matrix product AZ. For this we use a standard method for a matrix-vector product. We only need an extra loop over all the vectors inside matrix Z. The resulting matrix is also fully stored. If we have the matrix AZ, we calculate the matrix E by using a matrix-matrix product in the same way.

6.2.3 LU-Factorization and Singularity

During the algorithm we see we have to solve equations of the form $Ey = b$. So once we found the matrix E we use a LU-factorization as described in Appendix A in order to calculate q_1, q_2, and q_3, from Algorithm 5.1. If we can write matrix E as a product of a lower and an upper triangular matrix (L and U) we can solve the system efficiently. First we do a forward substitution which solves the system $Lx = b$. After

that we do a backward substitution which solves the system $Uy = x$. Since U and L are triangular matrices, these systems are easily to solve. In order to do this we first use Algorithm A.1 in which we overwrite the original matrix E to store a lower and upper triangular matrix for the forward and backward substitution. Since the diagonal of matrix L contains only ones we do not need to store them.

Before we do the forward and backward substitution (Algorithm A.2 and Algorithm A.3) we do a check for singularity. Because if E is singular we can not use a LU-factorization, since we cannot invert E then. We can check simply for singularity by looking at the 'new' matrix E which now contains the factors of the LU-decomposition. If one or more of its diagonal elements is equal to zero we have a singular system. In practice this generally never happens with the IBRAHYM model, but if it is the case we terminate the deflation process and continue with the original PCG solver. In this way we always get a solution. In [13, 14] it is described that deflation is also possible in the case that the system matrix is singular, but we will not discuss this here.

6.2.4 Ibounds

As we already mentioned in Section 3.4, MODFLOW deals with so-called no-flow cells and constant-head cells. Every cell has a corresponding ibound. The ibounds can take three different values:

- ibound < 0, constant-head cell
- ibound $= 0$, no-flow cell
- ibound > 0, variable-head cell

In the original PCG code used in MODFLOW if a cell has ibound equal to zero the corresponding conductances (CC, CV and CR) for that cell are set to zero. This is to make sure there is no flow in that particular cell. As an initialization of the PCG solver, the residuals are set to zero for cells that are constant-head or no-flow. This makes sense because the solution should not change in these cells during the process. To be sure that this will not happen, every time when a vector is updated, this is skipped if the ibound is less or equal to zero. In this way the solution in these cells is held constant.

When we implement the deflation technique, we should also be aware of the ibounds. In the matrix Z which we must define we should keep a value zero if the ibound is less or equal to zero, even if this cell is in a corresponding subdomain. This explains the IF statement in Algorithm C.1.

The initialization of the starting solution should be done for all cells. But when we construct the correct solution again during the deflation-post processing phase we should only do it for ibounds greater than zero. If the ibound is less or equal to zero we only take the original starting value there.

6.3 Optimization

Using our new program we can see that the concept of layer deflation really works for small areas in the IBRAHYM model. Since we want to use it for bigger areas we need to do some optimization. First of all we use too much memory since we store the matrix Z and AZ completely. In Section 6.3.1 we explain how we can save a lot of memory for layer deflation. The big disadvantage is that we need to calculate everything over and over again even if we can use some results if we would be able to store them. In Section

6.3.2 we will explain how we improve some calculations in such a way that we only need to store one extra vector, but that we can skip a matrix-matrix calculation during the iteration steps. In this way we also hope to win some runtime.

6.3.1 Saving Memory

First of all we want to rewrite our code in such a way that we do not need much extra memory. In order to do this we have to look at the structure of matrix Z and try to write some subroutines which can do calculations involving Z without really use Z itself.

Structure of Matrix Z

The first problem we ran into when we had the first version of the deflation code, was a lack of memory for large areas of interest. In the first version we stored the matrix Z and AZ completely. Since we decided to use layer-deflation we have a structured matrix Z. We can use this structure to do the calculation without using the matrix Z itself.

Layer deflation means that we have a matrix $Z \in \mathbb{R}^{n \times m}$ where n is the number of nodes used $(NROW \times NCOL \times NLAY)$ and m is the number of deflation vectors, which is in this case equal to the number of layers $(NLAY)$. Each vector contains values 1 for the nodes corresponding to a specific layer. Each layer has exactly $NRC = NCOL \times NROW$ nodes. So if

$$Z = [z_1 \;\; z_2 \cdots z_m], \tag{6.1}$$

than z_i has ones for the nodes that are in the interval

$$\Big[(i-1) \cdot NRC + 1, i \cdot NRC\Big].$$

So if we now write that $\underline{1}$ is a vector of length NRC containing ones and $\underline{0}$ is a vector of length NRC containing zeros than we can write that

$$Z = \begin{bmatrix} \underline{1} & \underline{0} & \cdots & \cdots & \underline{0} \\ \underline{0} & \underline{1} & \underline{0} & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \underline{0} & \underline{1} & \underline{0} \\ \underline{0} & \cdots & \cdots & \underline{0} & \underline{1} \end{bmatrix}, \tag{6.2}$$

Subroutines

If we look again at Algorithm 5.1 we see that we need a couple of times a multiplication with matrix Z. All these calculations can be done by implementing at least 4 subroutines. We need the following subroutines:

- Z^T times a vector of length $NODES$,
- Z times a vector of length $NLAY$,
- AZ times a vector of length $NLAY$,
- a subroutine which constructs matrix E itself.

For simplification we introduce a fifth subroutine which calculates A times a vector of length $NODES$ (BUILD_A_VEC). This subroutine is constructed in the same way as the original way to multiply A with a vector. We just defined it as a separate subroutine to avoid making mistakes. We shortly discuss some relevant details of the four subroutine named above.

The first subroutine (BUILD_ZT_VEC_RL and BUILD_ZT_VEC_DP) multiplies the matrix Z^T with a vector of length $NODES$, for example $Z^T r^{(0)}$. The result is a vector of length $NLAY$. The first element of this vector is exactly a summation of the first NRC elements of the original vector. The second element is the summation of the second NRC elements of the vector, and so on. So Z^T times a vector is simply doing a summation over the right elements of the original vector since Z only contains ones and zeros and we do not have any overlap.

The second subroutine (BUILD_Z_VEC) multiplies the matrix Z with a vector of length $NLAY$, for example $Z^T q_1$. The result is a vector of length $NODES$. The first NRC elements of this vector are equal to the first element of the original vector. The second NRC elements are equal to the second element of the original vector and so on. So Z times a vector is simply filling the new vector with the right values of the original vector.

The third subroutine (BUILD_AZQ) is just a combination of two other subroutines. We first calculate Z times the vector by using the second subroutine. The new vector is used in the extra subroutine which multiplies A with this vector. This gives the result we want without using the matrix AZ.

The last subroutine (BUILD_E) calculates the most important matrix, E. If Z is as given in Equation (6.1), than E can be written as:

$$E = E_{i,j} = \begin{cases} z_i^T A z_j & if \quad \|i-j\| \leq 1 \\ 0 & if \quad \|i-j\| > 2 \end{cases}, \tag{6.3}$$

where $i, j = 1, \ldots, NLAY$. This means that we first loop over index j and calculate Az_j by using the extra subroutine. If we have the vector Az_j we calculate $z_{j-1}Az_j$, $z_j Az_j$ and $z_{j+1}Az_j$ by summing the right element as in the subroutine which calculates Z times a vector. If $j = 1$ we do not calculate $z_{j-1}Az_j$ and if $j = NLAY$ we do not calculate $z_{j+1}Az_j$.

Using Less Vectors

If we take a good look at Algorithm 5.1 we can try to use the same vector to store multiple calculations. Sometimes we only calculate and/or store something in order to use it directly again. For example, the vectors $\tilde{q_1}$ and $\tilde{q_2}$ we use directly to calculate q_1 and q_2. So $\tilde{q_1}$ and $\tilde{q_2}$ can be stored in the same vector to save memory. Also the results of AZq_1, AZq_2, $Ay^{(k+1)}$ and $Z(q_1 - q_2)$ we never use at the same time. So we only need to store one extra vector for these four vectors. In this way we can save even more memory.

6.3.2 Gaining Wall-clock Time

Now that we have a code which uses not too much extra memory we observed that the new code was not really faster if we compared the runtime. So we tried to find out which part of the program took such a long time. By using timers we found out that calculating AZ was most time consuming and we did not win any runtime although we need less iterations. This is the a draw-down for minimizing memory usage. Since the matrix AZ does not change during an inner loop we tried to think of something such that we do not have to calculate this matrix-matrix product over and over again.

If we take a good look at matrix AZ we see that we could benefit from its structure for layer-deflation.

Storing a part of AZ using one vector

We will look at the structure of matrix AZ now for a really simple example. We take a $2 \times 2 \times 3$ grid (see Figure 6.2). This mean we only have 2 columns, 2 rows and 3 layers. For Z we again take layer deflation which gives us a Z containing of 3 vectors. If we use the same names for the conductances between cells we get the following matrices:

$$A = \begin{pmatrix} a1 & cr1 & cc1 & & cv1 & & & & & & & \\ cr1 & a2 & cr2 & cc2 & & cv2 & & & & & & \\ cc1 & cr2 & a3 & cr3 & cc3 & & cv3 & & & & & \\ & cc2 & cr3 & a4 & cr4 & cc4 & & cv4 & & & & \\ cv1 & & cc3 & cr4 & a5 & cr5 & cc5 & & cv5 & & & \\ & cv2 & & cc4 & cr5 & a6 & cr6 & cc6 & & cv6 & & \\ & & cv3 & & cc5 & cr6 & a7 & cr7 & cc7 & & cv7 & \\ & & & cv4 & & cc6 & cr7 & a8 & cr8 & cc8 & & cv8 \\ & & & & cv5 & & cc7 & cr8 & a9 & cr9 & cc9 & \\ & & & & & cv6 & & cc8 & cr9 & a10 & cr10 & cc10 \\ & & & & & & cv7 & & cc9 & cr10 & a11 & cr11 \\ & & & & & & & cv8 & & cc10 & cr11 & a12 \end{pmatrix}, \quad (6.4)$$

$$Z = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \quad (6.5)$$

where

$$a_i = \sum_{j=1, j \neq i}^{j=m} A(i,j) + HCOF(i), \quad (6.6)$$

which means that the diagonal elements of A is just a summation of the elements of the corresponding row and the corresponding $hcof$ term.

If we look at Figure 6.2, we conclude that $CR2$, $CR4$, $CR6$, $CR8$, and $CR10$ are all equal to zero, since these cells have no neighbour in the row direction. We also see that $CC3$, $CC4$, $CC7$, and $CC8$ are all equal to zero, since these cells have no neighbour in the column direction. So we end up with the following matrix A which can be divided into 9 blocks:

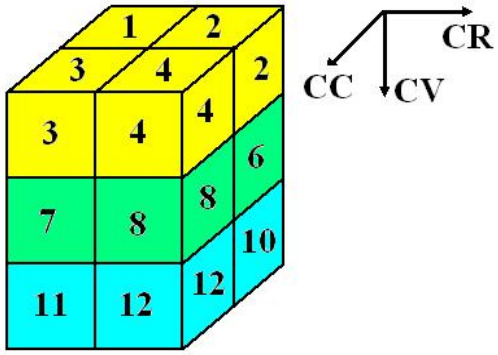

Figure 6.2: A simple example.

$$\mathbf{A} = \left(\begin{array}{cccc|cccc|cccc} a1 & cr1 & cc1 & & cv1 & & & & & & & \\ cr1 & a2 & & cc2 & & cv2 & & & & & & \\ cc1 & & a3 & cr3 & & & cv3 & & & & & \\ & cc2 & cr3 & a4 & & & & cv4 & & & & \\ \hline cv1 & & cc3 & & a5 & cr5 & cc5 & & cv5 & & & \\ & cv2 & & cc4 & cr5 & a6 & & cc6 & & cv6 & & \\ & & cv3 & & cc5 & & a7 & cr7 & & & cv7 & \\ & & & cv4 & & cc6 & cr7 & a8 & & & & cv8 \\ \hline & & & & cv5 & & cc7 & & a9 & cr9 & cc9 & \\ & & & & & cv6 & & cc8 & cr9 & a10 & & cc10 \\ & & & & & & cv7 & & cc9 & & a11 & cr11 \\ & & & & & & & cv8 & & cc10 & cr11 & a12 \end{array}\right), \quad (6.7)$$

If we now calculate AZ we see that we end up with:

$$\mathbf{AZ} = \left(\begin{array}{c|c|c} a1+cr1+cc1 & cv1 & \\ cr1+a2+cc2 & cv2 & \\ cc1+a3+cr3 & cv3 & \\ cc2+cr3+a4 & cv4 & \\ \hline cv1 & a5+cr5+cc5 & cv5 \\ cv2 & cr5+a6+cc6 & cv6 \\ cv3 & cc5+a7+cr7 & cv7 \\ cv4 & cc6+cr7+a8 & cv8 \\ \hline & cv5 & a9+cr9+cc9 \\ & cv6 & cr9+a10+cc10 \\ & cv7 & cc9+a11+cr11 \\ & cv8 & cc10+cr11+a12 \end{array}\right), \quad (6.8)$$

We see that the matrix AZ has a nice structure. We can take advantage of this structure to save some runtime. If we store the diagonal blocks in one vector we can use it later on to make calculations involving AZ. In this way we do not need to calculate matrix AZ over and over again and we can save time. The 'off-diagonal' block contains only the CV values, which are already stored in MODFLOW. So a calculation like AZq_2 now simply reduces to a summation of three products.

Using a pointer vector instead of if-loops

Now that we found an efficient way to calculate AZ we still did not see much gain in runtime. Calculating AZq took too much time comparing the total runtime of one iteration. It turned out that the IF statements, that are used for checking the ibounds in the computations of AZq, are time consuming. So we came up with a new idea.

We define an extra integer vector containing zeros and ones depending on the corresponding ibound. We now do an extra multiplication with this vector instead of checking the ibound. If the ibound is such that we do not need to calculate something for that cell, we now do a calculation but multiply it by zero. This turned out to be about 3 times faster than checking for ibounds.

6.4 Results for Three Small Areas

The resultant code is tested for three different (small) areas of the IBRAHYM model. In this section we will restrict ourselves to iterations only. Each area is 4×4 kilometer. If the results for the three small areas are good we can do a calculation for a bigger area. Some figures are put in Appendix D to keep this section conveniently arranged.

The three areas are chosen close to each other. If we look at Figure 6.3 we see that there is some overlap between the areas. Area 1 is drawed in yellow, area 2 is drawed in blue and area 3 in orange. We clearly see some overlap. The positions of the areas are defined by giving the x and y coordinates of the lower left corner and the upper right corner. These positions are as follows. Area 1: 219000, 321000, 223000, 325000. Areas 2: 225700, 317000, 229700, 321000. Area 3: 226000, 318000, 230000, 322000.

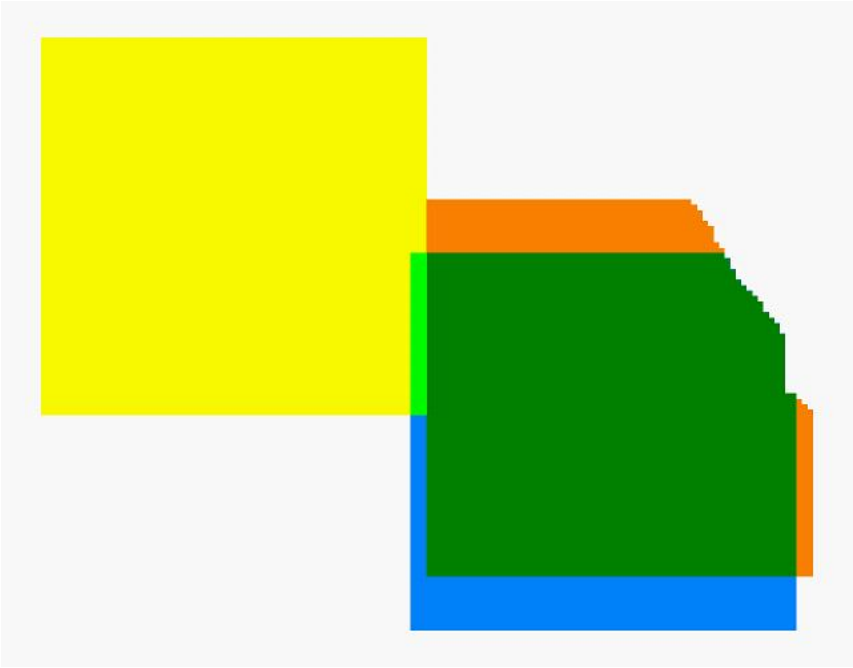

Figure 6.3: The 3 small areas we use to do some tests.

6.4.1 Total Number of Iterations for Each Area

First of all we look at the total number of iteration needed to calculate the heads for three different areas. For a MODFLOW run we can define the maximum number of inner iterations we want to use before the program does another outer loop. We set the maximum number of inner iterations equal to 50 that is common used as a default setting. The results are shown in Table 6.1 gives us the total number of iterations for each area.

Although the three areas lay close to each other and have some overlap, we see a large difference in the number of iterations. This can be explained when we take a look at Figure 6.6 at the end of this chapter. In this figure the packages of clay between the layers are showed. The first picture (upper left) is the package of clay between the first and second layer. The second picture (right of the first) is the package of clay between the second and third layer, et cetera. The picture in the lower right corner is the location of the three areas again. We can see in picture 1 to 9, 16 and 18 that the thickness of the clay in the three areas is roughly the same. In picture 10 to 14 we see that the first area does not have thick layers where areas 2 and 3 have thick packages of clay. Picture 15 and 17 show the opposite. These clay layers can have a high resistance as discussed in Section 3.5. The result is that we have to deal with small eigenvalues which could be an explanation for the bad convergence behaviour.

Area	number of iterations using original code	number of iterations using layer-deflation
1	62	53
2	164	121
3	287	168

Table 6.1: Number of iterations using the original code and deflation.

For the third area we will now also vary the maximal number of inner iterations. We choose to compare 5 different numbers: 20, 30, 40, 50 and 75. First we look at the result for the number of total iterations when we vary the maximum number of inner iterations. These results are shown in Table 6.2.

Maximum number of inner iterations	original code	deflation code
20	566	271
30	383	199
40	335	179
50	287	168
75	230	153

Table 6.2: Number of iterations using the original code and deflation when varying the maximal number of inner iterations.

We see that the effect of deflation is the biggest when we use a smaller number as a maximum number of inner iterations. We also see that we use less iterations when we use a bigger number.

6.4.2 Comparing the Heads

We still need to check if we get the same solutions with the deflation method as with the original code. In order to do this we subtract the both solutions. Because there are

a lot of uncertainties in the parameters, our convergence criterion is that the solution of two iteration steps differ less than a millimeter for each cell. In this way we hope that the results are accurate up to a centimeter. So we compare the differences of the two solutions with two different accuracies. We compare them with the accuracy of a centimeter and with the accuracy of a millimeter. If the solutions differs less than a centimeter we consider them as being correct. In the following results we always give two plots. The left one is with an accuracy of a centimeter and the right one is with an accuracy of a millimeter. In both cases we plotted the figures in such a way that a 'blue area' is okay but a 'red area' is not okay. The used legends can be found in Figure 6.4. We only give the results for the 15^{th} layer of the 3^{th} area, but for the other areas and layers the results are similar.

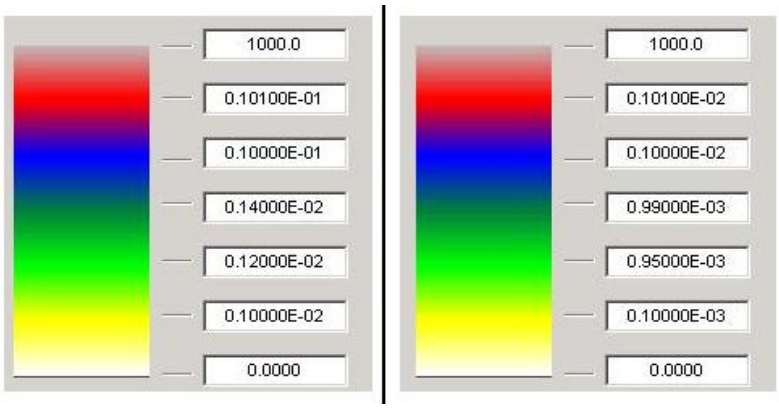

Figure 6.4: Legend used in the figures for comparison. Left: accuracy in cm. Right: accuracy in mm.

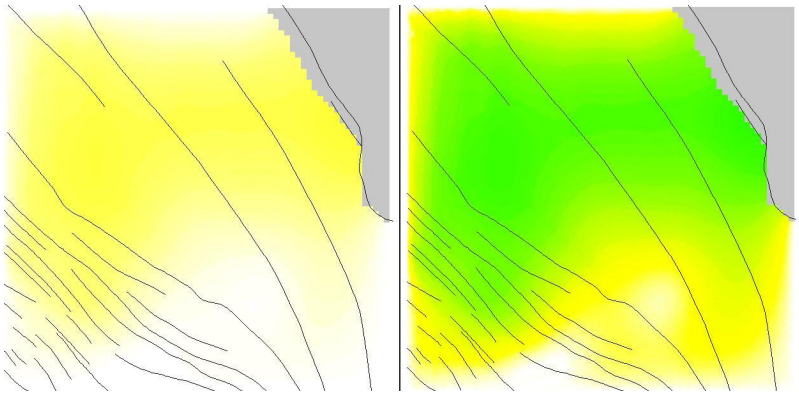

Figure 6.5: Absolute difference between the original and the deflation code using 50 inner iterations.

When we compare the solutions of the original code and the deflation code we see that some of them differs more than a millimeter. Using 40 or 50 as a maximum number of inner iterations gives a solution that differs less than a millimeter. For 50 inner iterations the result is given in Figure 6.5. The other results (20, 30, 40 and 75 inner iterations) can be found in Figures D.2 - D.5. This variation in results motivates us to do further investigations.

Fluctuation in the Original Code

We are going to look at the fluctuations in the solutions using the original code when we vary the maximal number of inner iterations. We compare the solutions with 20, 30, 40 and 75 as a maximum number of inner iterations with the solution when we use 50 as a maximum. We take 50 as a 'basis' because 50 is often used in practice. Again we only look at the third area and the 15^{th} layer. The 4 figures (D.6 - D.9) are given in Appendix D again.

We see that if we compare original(20) and original(50) (Figure D.6) that the solution differs more than a millimeter. But if we look at the difference using centimeters that it is okay. So the solution varies a little bit for using different numbers as a maximum number of inner iterations.

Fluctuation in the Deflation Code

We do the same test but now for using our deflation code. The results are given in Figures D.10 - D.13

If we now look at the results we see that the solution is more stable for different numbers of maximum inner iterations. In all cases the solutions differ less than a millimeter. So deflation gives less fluctuations in the solutions when varying the maximum number of inner iterations. We can conclude that when using deflation the solution is less dependent on the maximum number of inner iterations chosen. The deflation technique is more robust than the original code.

Comparing the Original Code using the Deflation Code as a Basis

Of course we also want to compare the solutions between the two different methods if we take for one method a given number of maximum inner iterations as a basis. First we take deflation with a maximum of 50 inner iterations as a base and compare the different solutions by using the original code.

When we compare the solutions of the original code with the solution of the deflation method using 50 as a maximum number of inner iterations, we get Figures D.14 - D.18. Again we see that if we use 20, 30 or 75 as a maximal number of inner iterations that the solution differs more than a millimeter, which again verifies that the solution varies much when using the original code.

Comparing the Deflation Code using the Original Code as a Basis

Now we switch to taking the original code with a maximum of 50 inner iterations as a base and compare it with the results from using deflation.

When we compare the solutions for deflation with the solution of the original method, using 50 as a maximum number of inner iterations, we get Figures D.19 - D.23. Again we see that all the differences are less then a millimeter, which again verifies that the solution of the deflated method differs less then the original code.

6.4.3 Effects of the Faults

We also did a simple test to see if the faults cause as much problems as we think they do. We did four runs for the 3^{th} area. We did two runs with the original code and two runs with the deflation code. With both codes we did one run where we had faults in

the subsoil and one run where we assumed there were no faults. In Table 6.3 we see how much iterations were needed for all cases when we use 50 as a maximum number of inner iterations. We again only did it for the third area.

If we look at the original code we see that there is a difference in the number of iterations needed. When we do not take the faults into account we need about 10 % less iterations. When using deflation we see that it does not matter much if we take faults into account. This motivates us again to use layer deflation only for now.

	with faults	without faults
deflation code	168	171
original code	287	254

Table 6.3: Number of iterations using the original code and deflation when we do and do not take faults into account.

6.5 Results for Large Area

As a final test we tried to calculate an area of 15 times 15 kilometers. (215000, 313000, 230000, 328000) We expanded the third area from the last section because this small area was a difficult area already. For this large area we also observe a gain in iterations as expected (see Table 6.4). We also looked at the total runtime of the program. The calculations had been carried out on a AMD Athlon 64 x2 Dual Core Processor 5000+, 2.60 GHz, which has 2047 MB RAM memory.

	number of iterations	runtime in seconds
original	1082	1660.40
deflation	757	1543.20

Table 6.4: Number of iterations and the runtime in seconds using the original code and deflation for a large area.

If we compare the solutions for the deflation code and the original code we see that the differences are all smaller than a centimeter. So deflation still gives a comparable answer. Unfortunately we see that we do not gain very much runtime yet. Probably this is due to the fact that the implementation is not optimal yet for Fortran. We might be able to rewrite some routines in such a way that Fortan can deal with them better.

	Q-in (m3/d)	Q-out(m3/d)	Error
original	0.66848E+06	-0.66851E+06	-0.31188+E02
deflation	0.66849E+06	-0.66848E+06	0.49375E+E01

Table 6.5: In- and outflow of a run and the error.

During a run the program keeps up the water balance. It keeps up how much total in- and outflow there is during a run. This difference is calculated in m^3/day. When using deflation we see that this error is a factor ten smaller (see Table 6.5). This means that the deflation technique is more accurate than the original code. This also motivated us to look at the solution for deflation when we put the stop criterion on a centimeter instead of a millimeter. Unfortunately this did not work, because the solutions differs too much from each other. Still we can say that the deflation technique is more accurate.

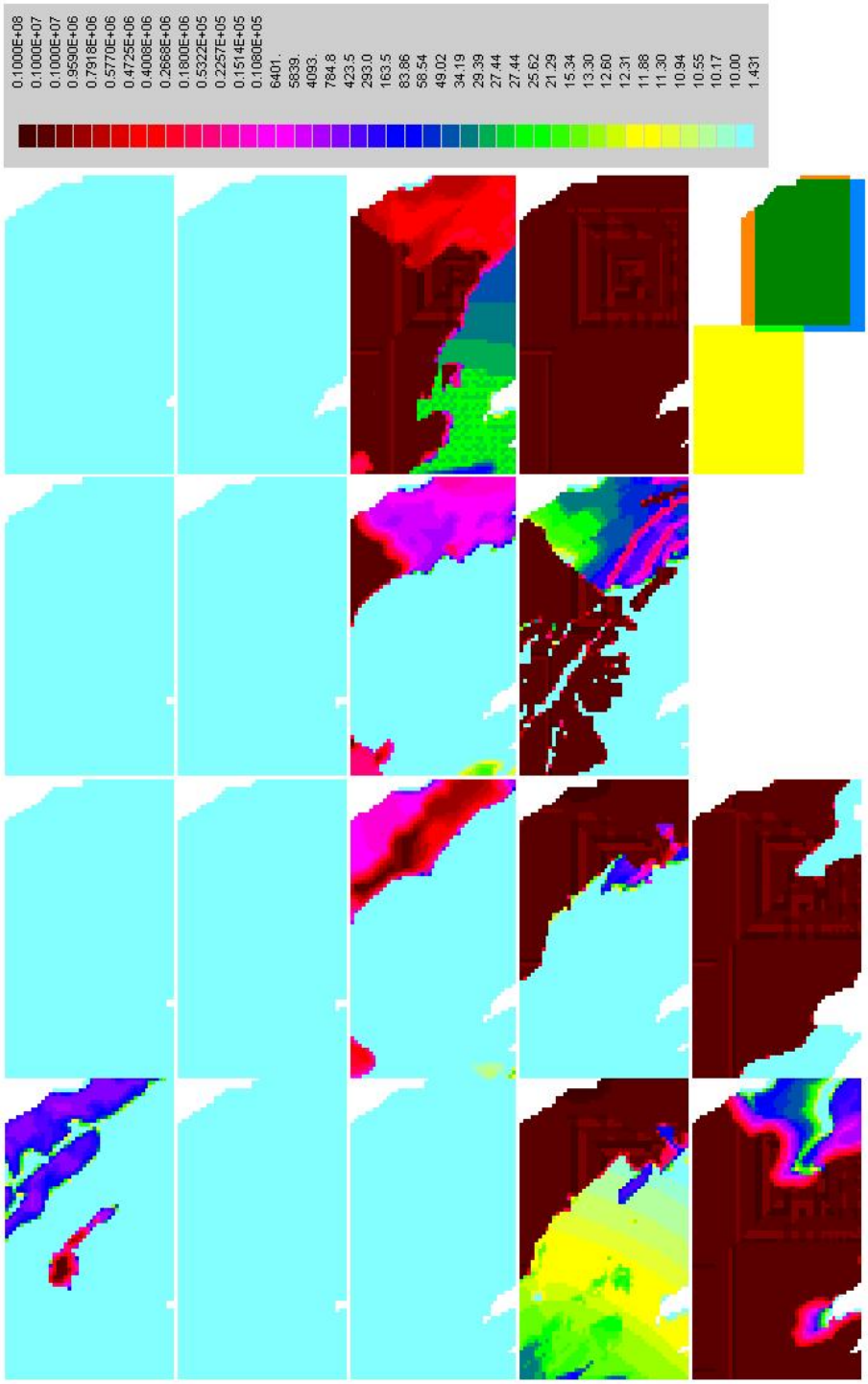

Figure 6.6: The C-values around the three small areas. The legend represents the number of days it takes for the water to flow through the clay.

Chapter 7

Conclusions and Recommendations

7.1 Conclusions

In this Master's project the deflation method is implemented in MODFLOW for simulating groundwater flow. The code is applied to the IBRAHYM groundwater model. The following conclusions can be drawn:

1. In general, the deflation method works for modelling groundwater flow.
2. Less iterations are required, especially for areas with large clay resistances (vertical contrast in parameter) and faults (horizontal contrast in parameter).
3. The deflation preconditioner makes the solution more robust.
4. Wall-clock times can be gained.
5. Gaining wall-clock times with the deflation method depends strongly on the code used.

7.2 Recommendations

Concerning the results, the following recommendations for future reseach can be made:

1. The code, with and without deflation, should be further optimized for Fortran. This involves minimizing memory usage, IF-statements and using smart mappings.
2. Deflation vectors in the horizontal directions (e.g. subdomian deflation) could possibly improve convergence.
3. it is recommended to compare the solution with an analytical solution to give insight in the error.
4. In case of a large number of layers deflation vectors could be clustered for several layers.

Appendix A

LU Factorization Algorithms

Algorithm A.1 (LU Decomposition) Given a $n \times n$ matrix A, and the $n \times n$ matrices L and U, containing only zero coefficients. Then this algorithm computes an LU factorization of A. Since L has one on its diagonal we do not need to store them so we can overwrite th original matrix A in order to save memory.

for $k = 1, \ldots, n-1$ **do**
  for $i = k+1, \ldots, n$ **do**
    $A(i,k) = \frac{A(i,k)}{A(k,k)}$;
  end for
  for $j = k+1, \ldots, n$ **do**
    for $i = k+1, \ldots, n$ **do**
      $A(i,j) = A(i,j) - A(i,k)A(k,j)$;
    end for
  end for
end for

Algorithm A.2 (Forward Substitution) Given a $n \times n$ matrix L which is unit lower triangular, and a vector $\mathbf{b}$. This algorithm solves the system $L\mathbf{x} = \mathbf{b}$.

for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, i-1$ **do**
    $x(i) = b(i) - L(i,j)x(j)$;
  end for
end for

Algorithm A.3 (Backward Substitution) Given a $n \times n$ matrix U which is upper triangular, and a vector $\mathbf{x}$. This algorithm solves the system $U\mathbf{y} = \mathbf{x}$.

for $i = n, \ldots, 1$ **do**
  for $j = i+1, \ldots, n$ **do**
    $y(i) = \frac{x(i) - U(i,j)y(j)}{U(i,i)}$;
  end for
end for

Appendix B

MICCG in MODFLOW

In MODFLOW a slightly different version of the MICCG method is used. They use a preconditioner $M = U^T DU$, where U is an upper triangular matrix. U has nonzero elements along the main diagonal and on the off-diagonal positions where A itself has nonzero values. D is a diagonal matrix with $d_{ii} = 1/u_{ii}$. For a problem with 2 rows, 3 columns and 2 layers, U should be of the following form:

$$\mathbf{U} = \begin{bmatrix} u_1 & -r_1 & 0 & -c_1 & 0 & 0 & -v_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & u_2 & -r_2 & 0 & -c_2 & 0 & 0 & -v_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & u_3 & -r_3 & 0 & -c_3 & 0 & 0 & -v_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_4 & -r_4 & 0 & -c_4 & 0 & 0 & -v_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & u_5 & -r_5 & 0 & -c_5 & 0 & 0 & -v_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & u_6 & -r_6 & 0 & -c_6 & 0 & 0 & -v_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & u_7 & -r_7 & 0 & -c_7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & u_8 & -r_8 & 0 & -c_8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & u_9 & -r_9 & 0 & -c_9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & u_{10} & -r_{10} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & u_{11} & -r_{11} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & u_{12} \end{bmatrix}, \tag{B.1}$$

Here, r_n, c_n and v_n are the conductances along rows and columns and between layers, respectively. They are the same as CR, CC and CV from section 3.3.1. They are exactly the same as in the matrix A.

The u_n are defined such that the sum of the elements along a row of M equals the sum of the elements along the row of A. If we do the calculations we find that:

$$\begin{aligned} u_1 &= a_{11}, \\ u_2 &= a_{22}\tfrac{r_1^2}{u_1} - \tfrac{r_1 c_1}{u_1} - \tfrac{r_1 v_1}{u_1}, \\ u_3 &= a_{33}\tfrac{r_2^2}{u_2} - \tfrac{r_2 c_2}{u_2} - \tfrac{r_2 v_2}{u_2}, \\ &\text{etc.} \end{aligned}$$

The general algorithm can now be expressed as:

$$u_{ii} = (1+\delta)\, a_{ii} - \sum_{k=1}^{i-1} \frac{u_{ki}^2}{u_{kk}} - \alpha \sum_{j=1}^{i-1} f_{ji} + \sum_{j=i+1}^{N} f_{ij} \tag{B.2}$$

where:

$$f_{ij} = \begin{cases} \sum_{k=1}^{i-1} u_{ki} u_{kj} u_{kk} & a_{ij} = 0 \\ 0 & a_{ij} \neq 0 \end{cases},$$

and

$$u_{ij} = 0 \text{ for } j < i.$$

The parameter α is a relaxation parameter, defined by the user. It is used to diminish the value of f_{ij} in equation B.2. Using a relaxation parameter value of 0.97, 0.98 or 0.99 instead of 1.00 sometimes improves convergence.

The parameter δ is a overcompensation parameter. It usually equals zero, but when a value of u_{ii} gets zero or negative, δ is increased.

More details can be found in [6]

Appendix C

Build Z Algorithm

Algorithm C.1 (Build Z) Subroutine to calculate the deflation matrix Z. We need to have $NUMJ$, $NUMI$ and $NUMK$ specified as the number of block in the column-, row- and layer direction respectively.
NOTE: this algorithm is written in a Fortran language and we call the matrix ZD.

```
if (numj.gt.ncol) numj = ncol
if (numi.gt.nrow) numi = nrow
if (numj.gt.nlay) numk = nlay

do 001 k = 1,nlay
    do telk = 1,numk
        help = (nlay/numk)*(telk-1)
        p1 = nint(help)
        help = (nlay/numk)*(telk)
        p2 = nint(help)
        if (k.gt.p1 .and. k.le.p2 ) pk = telk
    enddo

do 001 i = 1,nrow
    do teli = 1,numi
        help = (nrow/numi)*(teli-1)
        p1 = nint(help)
        help = (nrow/numi)*(teli)
        p2 = nint(help)
        if (i.gt.p1 .and. i.le.p2 ) pi = teli
    enddo

do 001 j = 1,ncol
    do telj = 1,numj
        help = (ncol/numj)*(telj-1)
        p1 = nint(help)
        help = (ncol/numj)*(telj)
        p2 = nint(help)
        if (j.gt.p1 .and. j.le.p2 ) pj = telj
    enddo

N = j+ (i-1)*ncol + (k-1)*ncol*nrow

if (ibound(N).gt.0) then
```

```
      L = pj + (pi-1)*numj + (pk-1)*numi*numj
      ZD(N,L) = 1
   endif

   001 continue
```

Appendix D

Figures for Section 6.4

This appendix gives some figures which are needed in Section 6.4. In order to keep this section conveniently arranged we place most figures in this appendix. The legends used for this figures can be found in Figure 6.4, but we print it again here.

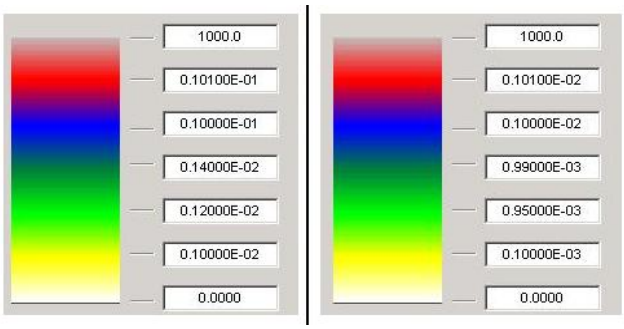

Figure D.1: Legend used in the figures for comparison. Left: accuracy in cm. Right: accuracy in mm.

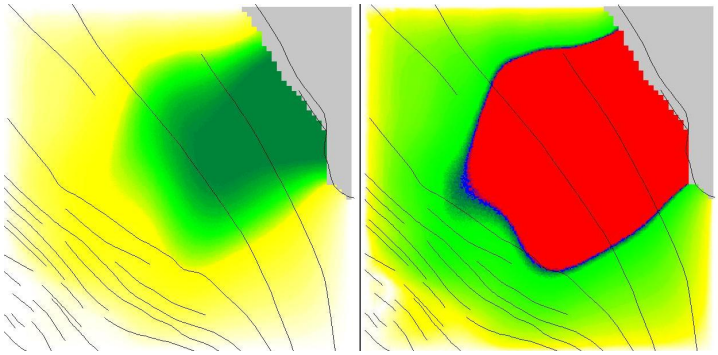

Figure D.2: Absolute difference between the original and the deflation code using 20 inner iterations as a maximum.

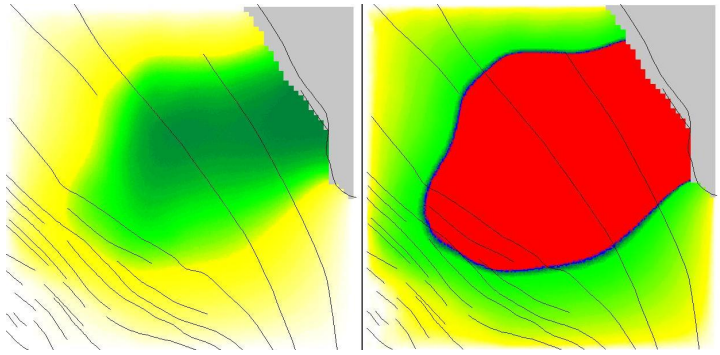

Figure D.3: Absolute difference between the original and the deflation code using 30 inner iterations as a maximum.

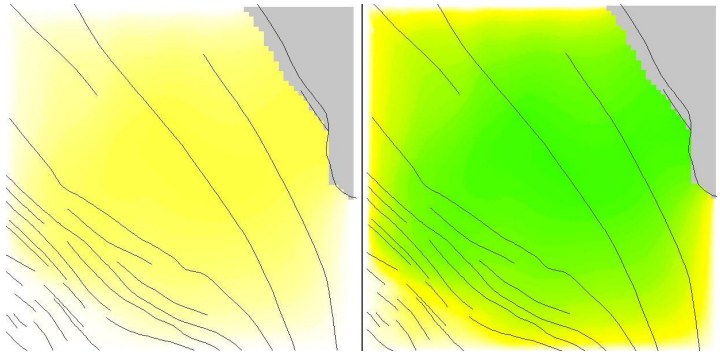

Figure D.4: Absolute difference between the original and the deflation code using 40 inner iterations as a maximum.

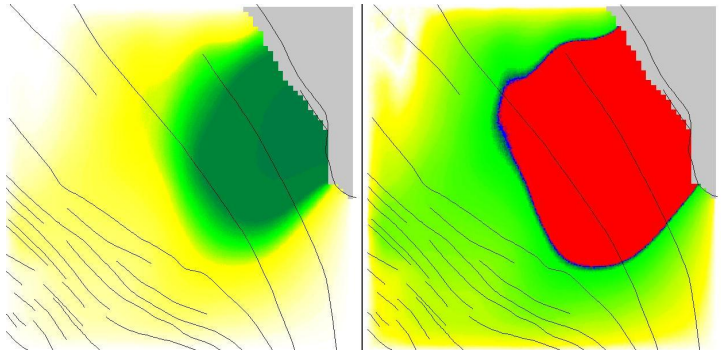

Figure D.5: Absolute difference between the original and the deflation code using 75 inner iterations as a maximum.

Fluctuation in the Original Code

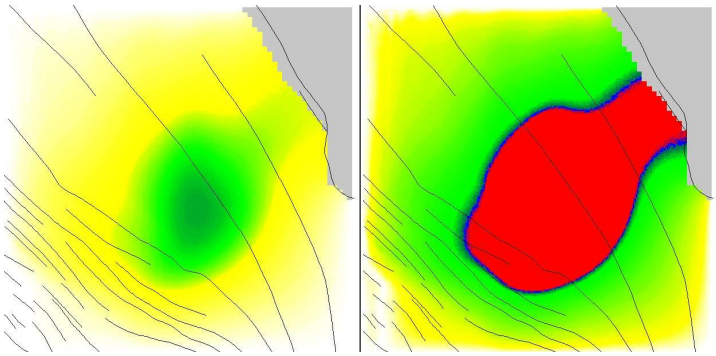

Figure D.6: Original code: absolute difference between 50 and 20 inner iterations as a maximum.

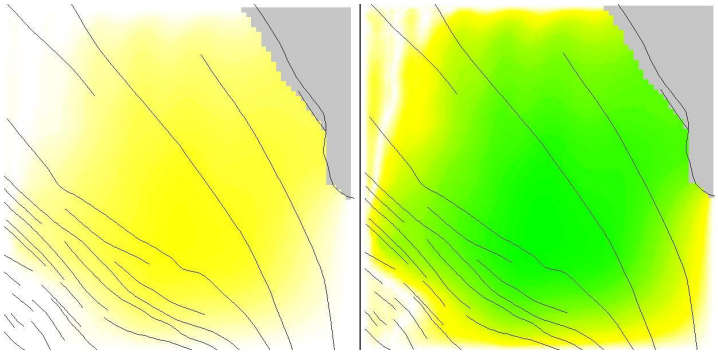

Figure D.7: Original code: absolute difference between 50 and 30 inner iterations as a maximum.

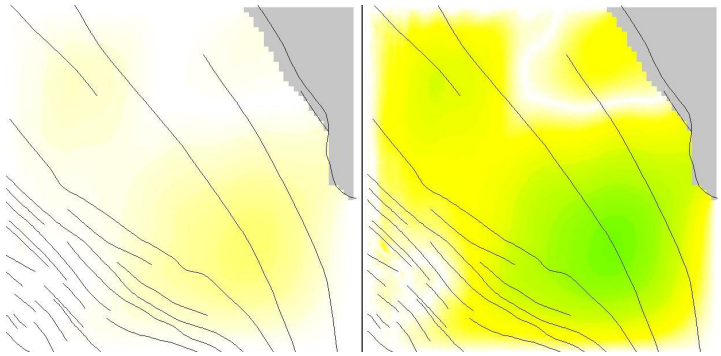

Figure D.8: Original code: absolute difference between 50 and 40 inner iterations as a maximum.

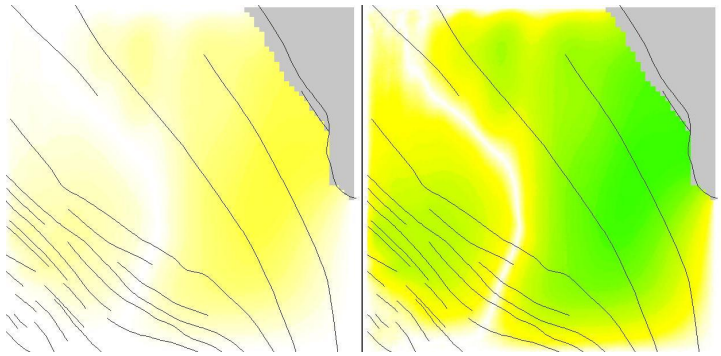

Figure D.9: Original code: absolute difference between 50 and 75 inner iterations as a maximum.

Fluctuation in the Deflation Code

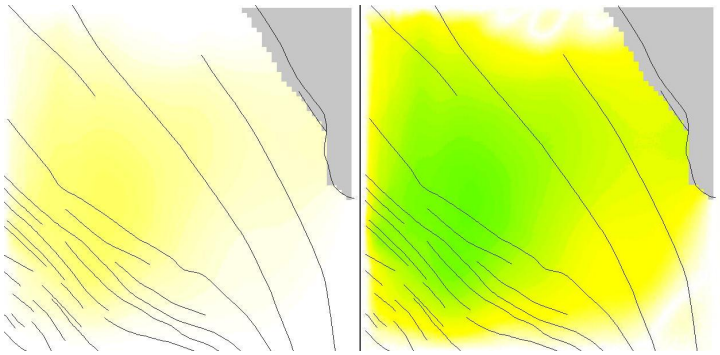

Figure D.10: Deflation code: absolute difference between 50 and 20 inner iterations as a maximum.

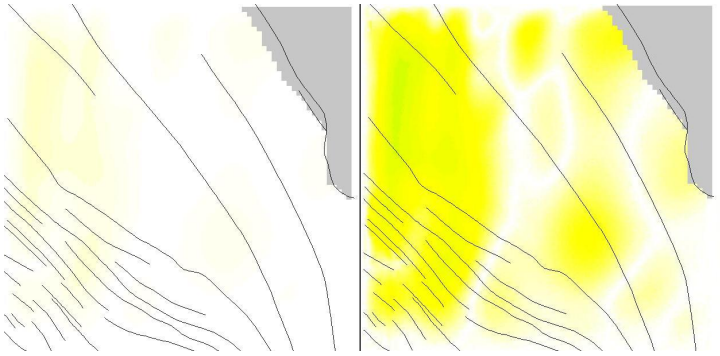

Figure D.11: Deflation code: absolute difference between 50 and 30 inner iterations as a maximum.

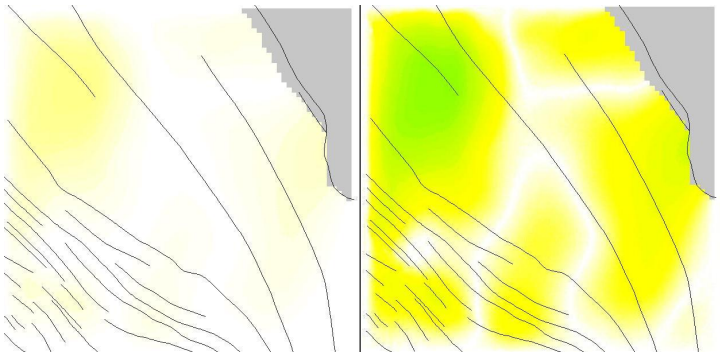

Figure D.12: Deflation code: absolute difference between 50 and 40 inner iterations as a maximum.

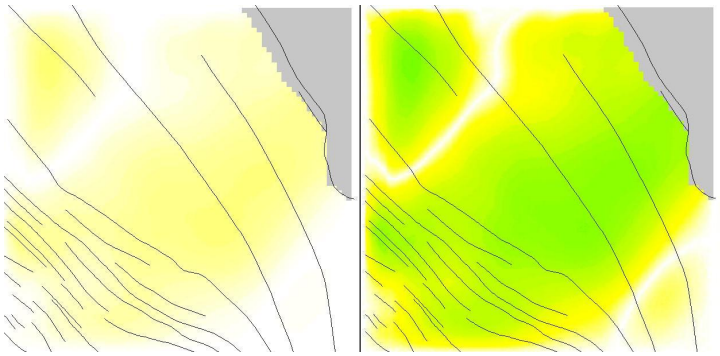

Figure D.13: Deflation code: absolute difference between 50 and 75 inner iterations as a maximum.

Comparing the Original Code using the Deflation Code as a Basis

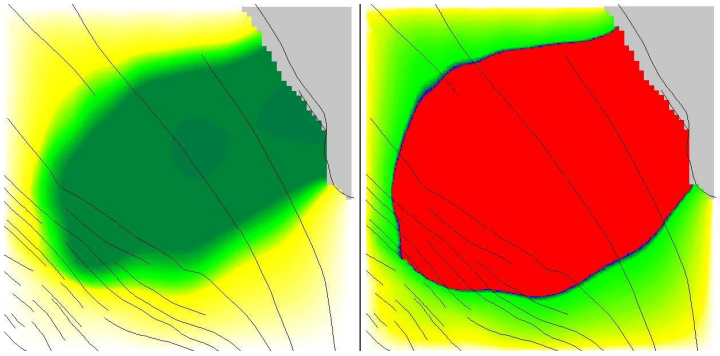

Figure D.14: Absolute difference between deflation(50) and original(20).

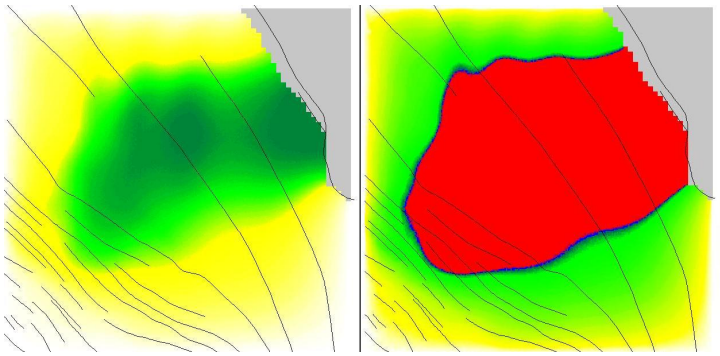

Figure D.15: Absolute difference between deflation(50) and original(30).

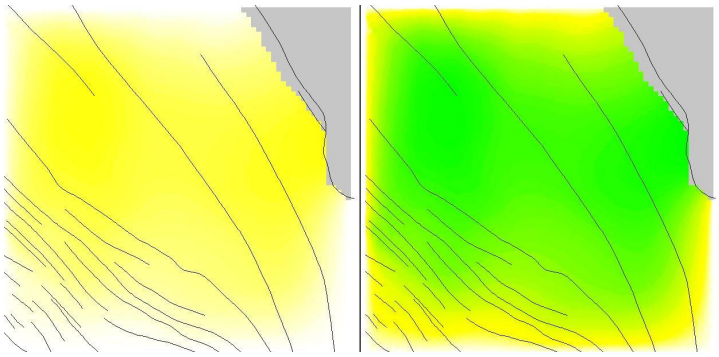

Figure D.16: Absolute difference between deflation(50) and original(40).

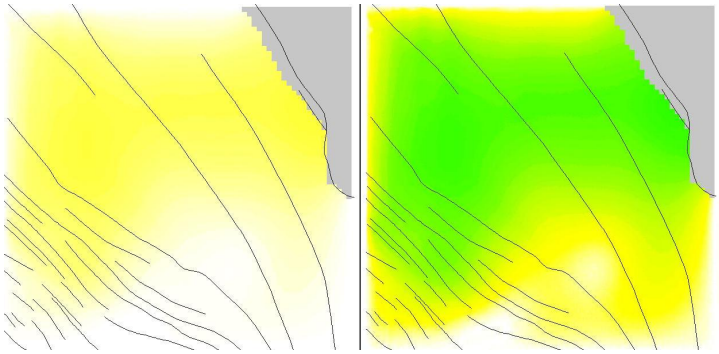

Figure D.17: Absolute difference between deflation(50) and original(50).

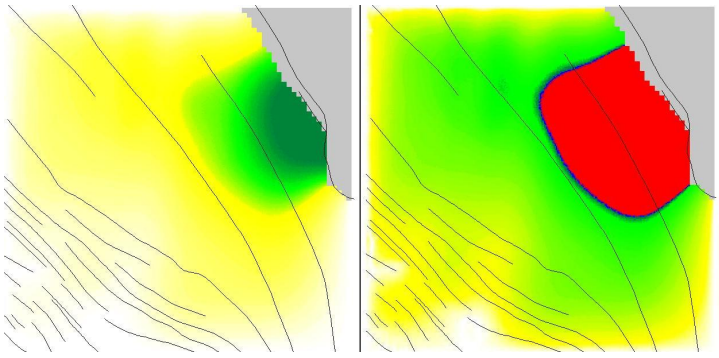

Figure D.18: Absolute difference between deflation(50) and original(75).

Comparing the Deflation Code using the Original Code as a Basis

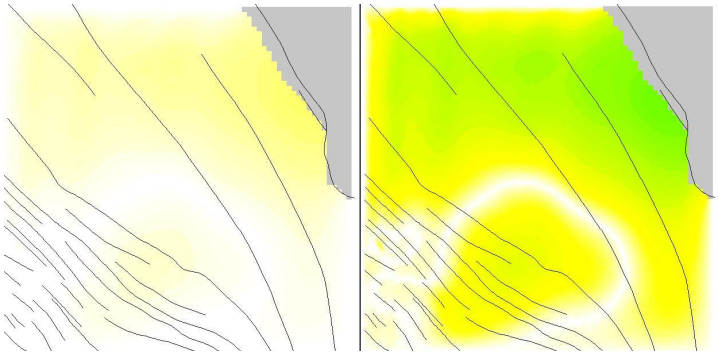

Figure D.19: Absolute difference between original(50) and deflation(20).

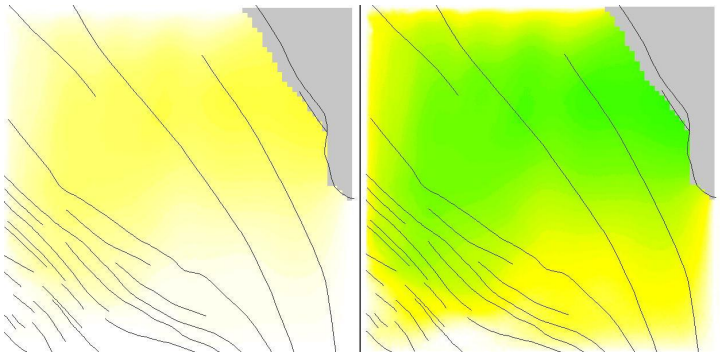

Figure D.20: Absolute difference between original(50) and deflation(30).

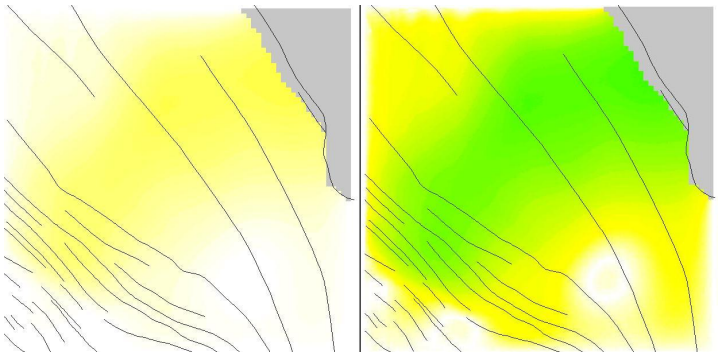

Figure D.21: Absolute difference between original(50) and deflation(40).

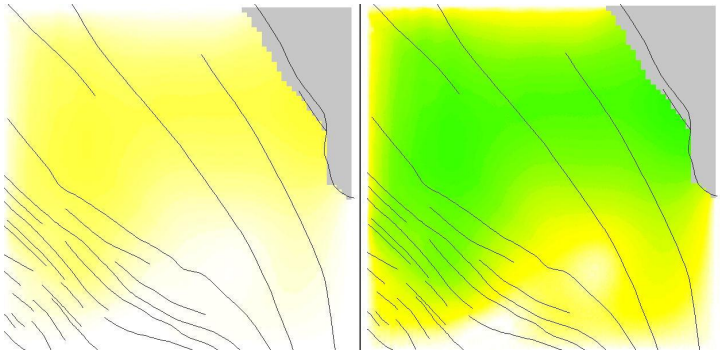

Figure D.22: Absolute difference between original(50) and deflation(50).

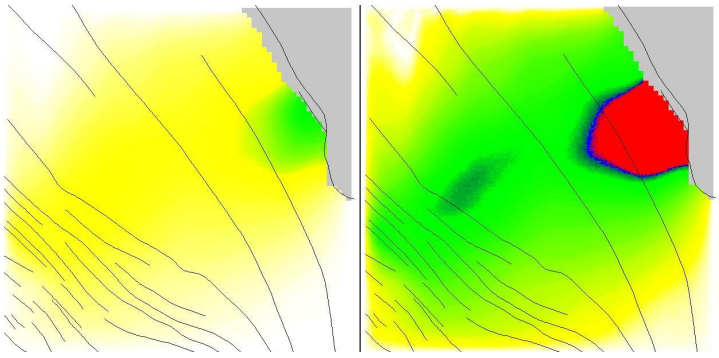

Figure D.23: Absolute difference between original(50) and deflation(75).

Nomenclature

Roman symbols

Symbol	*Description*	*Units*
$a_{i,j,k,n}$	flow from nth source into cell (i,j,k)	[L^3 T^{-1}]
A	system matrix	[—]
$\tilde{A}$	system matrix of transformed system	[—]
b	right-hand side vector	[—]
$\tilde{b}$	right-hand side vector of transformed system	[—]
$\tilde{b_i}$	off-diagonal elements in the (M)ICCG method	[—]
c_i	conductance in subprism i	[L^2 T^{-1}]
$\tilde{c_i}$	off-diagonal elements in the (M)ICCG method	[—]
C	conductance	[L^2 T^{-1}]
CB_n	boundary conductance	[L^2 T^{-1}]
$CC_{(i+1/2,j,k)}$	conductance between cell (i,j,k) and $(i+1,j,k)$	[L^2 T^{-1}]
CD_n	drain conductance	[L^2 T^{-1}]
$CR_{(i,j+1/2,k)}$	conductance between cell (i,j,k) and $(i,j+1,k)$	[L^2 T^{-1}]
$CRIV_n$	conductance of river-aquifer interconnection	[L^2 T^{-1}]
$CV_{(i,j,k+1/2)}$	conductance between cell (i,j,k) and $(i,j,k+1)$	[L^2 T^{-1}]
$\tilde{d_i}$	diagonal elements in the (M)ICCG method	[—]
D	matrix containing diagonal of A	[—]
$DELC_i$	grid width of row i	[L]
$DELR_j$	grid width of column j	[L]
$DELV_k$	grid width of layer k	[L]
E	matrix containing the lower elements of A multiplied by -1	[—]
E	matrix in deflation technique: $E = Z^T AZ$	[—]
f	part of BIM	[—]
f	right-hand side in system of equations	[—]
g	force of attraction	[L T^{-2}]
g	vector	[—]
G	part of BIM, iteration matrix	[—]
h	hydraulic heads	[L]
h_i	head in cell i	[L]
$h_{i,j,k}$	head in node (i,j,k)	[L]
HB_n	head assigned to external source	[L]
$HCOF$	terms independent of conductance between nodes	[—]
HD_n	drain elevation	[L]
$HRIV_n$	water level (stage) at river	[L]
i	hydraulic gradient	[—]
i, I	index of columns	
$I_{i,j}$	recharge flux	[L T^{-1}]
I	identity matrix	[—]

j, J	index of rows	
k, K	index of layers	
k	intrinsic permeability	[L^2]
k	dimension of a space	[—]
K	conductivity	[L T^{-1}]
K	matrix with hydraulic conductivities (tensor)	[L T^{-1}]
K_{xx}	hydraulic conductivity along x-axis	[L T^{-1}]
K_{yy}	hydraulic conductivity along y-axis	[L T^{-1}]
K_{zz}	hydraulic conductivity along z-axis	[L T^{-1}]
$\mathcal{K}_m$	m dimensional subspace of candidate approximants in projection methods ($\subset \mathbb{R}^n$)	[—]
$\mathcal{K}_k\left(A; r^{(0)}\right)$	Krylov subspace of dimension k	[—]
$KC_{i-1/2,j,k}$	conductivity along row between cels (i,j,k) and $(i-1,j,k)$	[L T^{-1}]
$KR_{i,j-1/2,k}$	conductivity along row between cels (i,j,k) and $(i,j-1,k)$	[L T^{-1}]
$KR_{i,j,k-1/2}$	conductivity along row between cels (i,j,k) and $(i,j,k-1)$	[L T^{-1}]
L	lower triangular matrix	[—]
$\mathcal{L}_m$	m dimensional subspace of constraints in projection methods ($\subset \mathbb{R}^n$)	[—]
m	dimension of a space	[—]
m	rank of Z	[—]
m	number of grid points in the x-direction	[—]
M	matrix used in iterative methods, preconditioner	[—]
n	boundary number	
n	number of the drain	
n	number of unknowns	[—]
n	dimension of a space	[—]
n	direction normal to boundary	[—]
N	number of external sources	[—]
N	matrix used in iterative methods	[—]
$NCOL$	number of cells in the column direction	[—]
$NLAY$	number of cells in the layer direction	[—]
$NODES$	number of cells inside your grid	[—]
NRC	$NROW \times NCOL$, number of nodes in one layer	[—]
$NROW$	number of cells in the row direction	[—]
$NUMI$	number of blocks in the rowdirection for subdomain deflation	[—]
$NUMJ$	number of blocks in the columndirection for subdomain deflation	[—]
$NUMK$	number of blocks in the layerdirection for subdomain deflation	[—]
p	pressure	[M L^{-1} T^{-1}]
P	center point of elementary volume	[—]
P	nonsingular matrix such that $M = PP^T$	[—]
P	projector in deflation tchnique (matrix)	[—]
$p_{i,j,k,n}$	constant concerning sources dependent of head	[L^2 T^{-1}]
$P_{i,j,k}$	sum of all the $p_{i,j,k,n}$	[L^2 T^{-1}]
q	vector containing the right-hand side (RHS) of a system	[—]
q_1	in deflation algorithm: $q_1 := E^{-1}Z^T r^{(0)}$	[—]
q_2	in deflation algorithm: $q_2 := E^{-1}Z^T A\tilde{y}^{(k+1)}$	[—]
q_2	in deflation algorithm: $q_3 := E^{-1}Z^T v^{(k)}$	[—]
q_i	flow across subprism i	[L^3]
$q_{i-1/2,j,k}$	flow rate through face between cells (i,j,k) and $(i-1,j,k)$	[L^3 T^{-1}]
$q_{i+1/2,j,k}$	flow rate through face between cells (i,j,k) and $(i+1,j,k)$	[L^3 T^{-1}]
$q_{i,j-1/2,k}$	flow rate through face between cells (i,j,k) and $(i,j-1,k)$	[L^3 T^{-1}]
$q_{i,j+1/2,k}$	flow rate through face between cells (i,j,k) and $(i,j+1,k)$	[L^3 T^{-1}]
$q_{i,j,k-1/2}$	flow rate through face between cells (i,j,k) and $(i,j,k-1)$	[L^3 T^{-1}]
$q_{i,j,k+1/2}$	flow rate through face between cells (i,j,k) and $(i,j,k+1)$	[L^3 T^{-1}]
$q_{i,j,k,n}$	constant concerning sources independent of head	[L^3 T^{-1}]

Q	volumetric flow	[L^3 T^{-2}]
$Q_{i,j,k}$	sum of al the $q_{i,j,k,n}$	[L^3 T^{-1}]
QB_n	flow into cell from boundary	[L^3 T^{-1}]
QD_n	flow from aquifer into drain	[L^3 T^{-1}]
$QR_{i,j}$	recharge flow rate	[L^3 T^{-1}]
$QRIV_n$	flow between river and aquifer	[L^3 T^{-1}]
r_0	initial residual	[—]
$\tilde{r_0}$	initial residual in case of deflation	[—]
r_{new}	projected residual	[—]
$RBOT_n$	bottom of riverbed layer	[L]
S_s	Specific storage coefficient	[L^{-1}]
SS_{ijk}	Specific storage coefficient in cell (i, j, k)	[L^{-1}]
t	time	[T]
t^m	current time	[T]
t^{m-1}	past time	[T]
$TC_{(i,j,k)}$	transmissivity in column direction at cell (i, j, k)	[L^2 T^{-1}]
$TR_{(i,j,k)}$	transmissivity in row direction at cell (i, j, k)	[L^2 T^{-1}]
u	vector of unknowns	[—]
$\tilde{u}$	vector of unknowns in deflated system	[—]
v	seepage velocity	[L T^{-1}]
v	Darcy's velocity	[L T^{-1}]
v	vector inside (P)CG algorithm	[—]
v_i	vector of length n from $\mathcal{K}_m$	[—]
v_i	deflation vectors	[—]
v_x	velocity in x-direction	[L T^{-1}]
v_y	velocity in y-direction	[L T^{-1}]
v_z	velocity in z-direction	[L T^{-1}]
$\bar{v_i}$	Ritz vectors	[—]
V	matrix ($\subset \mathbb{R}^{n\times m}$) containing $v_i's$	[—]
$\bar{V}$	matrix ($\subset \mathbb{R}^{n\times m}$) containing $\bar{v_i}'s$	[—]
VK	vertical conductivity between layers	[L T^{-1}]
$VKCB_{(i,j,k)}$	conductivity of aquifer between (i, j, k) and $(i, j, k + 1)$	[L T^{-1}]
w	element of $\mathcal{L}_m$	[—]
w_i	vector of length n from $\mathcal{L}_m$	[—]
W	Volumetric flux representing sources and sinks	[T^{-1}]
W	matrix ($\subset \mathbb{R}^{n\times m}$) containing $w_i's$	[—]
x	dimension in space	[L]
x	vector	[—]
x_i	point inside a (sub)domain	[—]
y	dimension in space	[L]
y	vector (of unknowns)	[—]
$\tilde{y}$	vector of approximate solution	[—]
$\tilde{y}$	vector of unknowns in transformed system	[—]
y_0, $y^{(0)}$	initial guess	[—]
$\tilde{y_0}$	initial guess	[—]
z	dimension in space	[L]
z	vector	[—]
z_i	algebraic deflation vector	[—]
Z	deflation matrix ($\subset \mathbb{R}^{n\times m}$), $m \leq n$	[—]
Z	deflation matrix containing Ritz vectors	[—]

Greek symbols

Symbol	*Description*	*Units*
α_0	constant for PCG method	[—]
γ	specific weight ($\rho\, g$)	[M L^{-2} T^{-2}]
δ	element of $\mathcal{K}_m$	[—]
δ_i	small perturbation in eigenvector (vector)	[—]
δ_{ij}	Kronecker delta function	[—]
Δ		
Δc_i	grid width of row i	[L]
Δr_j	grid width of column j	[L]
$\Delta r_{j-1/2}$	distance between nodes i, j, k and $i, j-1, k$	[L]
Δv_k	grid width of layer k	[L]
Δh	change in head	[L]
Δs	change in distance	[L]
Δt	change in time (time interval)	[T]
Δv_{CB}	thickness of aquitard	[L]
ΔV	volume of cell	[L^3]
ϵ	constant for stopping criterion	[—]
κ	spectral condition number of a matrix	[—]
κ_2	condition number $\|A\|_2\|A^{-1}\|_2$	[—]
λ_i	i^{th} eigenvalue in nondecreasing order of a matrix	[—]
μ	dynamic viscosity	[M L^{-1} T^{-1}]
Ω	discrete grid	[—]
Ω	domain	[—]
Ω	subdomain	[—]
Ω_j	subdomain of Ω	[—]
ρ	specific density	[M L^{-3}]
$\bar{\rho}$	constant	[—]
$\sigma(A)$	spectrum of A	[—]
ϕ	value of inhomogenious boundary condition	[—]

Abbreviations

Abbreviation	*Description*
CG	Conjugate Gradient
DICCG	Deflated Incomplete Cholesky Conjugate Gradient
ICCG	Incomplete Cholesky Conjugate Gradient
MICCG	Modified Incomplete Cholesky Conjugate Gradient
PCG	Preconditioned Conjugate Gradient
RHS	Right-hand side
SPD	Symmetric Positive Definite

List of Figures

List of Tables

Bibliography

[1] Akker, C. van der, en Savenije, H. (2006), *'Hydrologie'*, dictaat Civiele techniek TU Delft (CT1310).

[2] Dufour, F.C. (1998), *'Grondwater in Nederland'*, ISBN: 90 6743 536 8, Van de Rhee, Rotterdam.

[3] J. Frank and C. Vuik (2001), *On the Construction of Deflation-Based Preconditioners*. SIAM J. Sci. Comput., Vol. 23, No. 2, pp. 442-462.

[4] Golub, Gene H. & Van Loan, Charles F. (1996), *'Matrix Computations'*, ISBN: 0-8018-5414-8, Baltimore : Johns Hopkins University Press.

[5] Harbaugh, A.W., *'MODFLOW 2005, the U.S. Geological Survey Modular Ground-Water Model - The Ground Water Flow Process'*. Online document: `http://pubs.usgs.gov/tm/2005/tm6A16/PDF/TM6A16.pdf`.

[6] Hill, M.C., *Preconditioned Conjugate-Gradient 2 (PCG2), 'A Computer Program for Solving Ground-Water Flow Equations'*. United States Geological Survey, Report 90-4048. Online document: `http://water.usgs.gov/nrp/gwsoftware/modflow2000/PCG-usgs-wrir_90-4048-\second_printing.pdf`.

[7] Nabben, R. and Vuik, C. (2004), *'A comparison of abstract versions of deflation, balancing and additive coarse grid correction preconditioners'*, SIAM J. Numer. Anal., 42, pp. 1631-1647.

[8] Oosterlee, C.W. and Vuik, C. (2005), *'Scientific Computing'*, lecture notes Applied Mathematics TU Delft (wi4201).

[9] Ros, L.A. (2008), *'Deflated CG Method for Modelling Groundwater Flow Near Faults'*, Interim Report. Online: `http://ta.twi.tudelft.nl/nw/users/vuik/numanal/ros_scriptie.pdf`

[10] Rushton, K.R. (1979), *'Seepage and Groundwater Flow'*, ISBN: 0 471 99754 4, Wiley, Chichester.

[11] Saad, Y. (2000), *'Iterative Methods for Sparse linear Systems' (First edition)*. Online: `http://www-users.cs.umn.edu/~saad/books.html`.

[12] Tang, J.M. and Vuik, C. (2007), *New Variants of Deflation Techniques for Pressure Correction in Bubbly Flow Problems*, Journal of Numerical Analysis, Industrial and Applied Mathematics, volume 2, no. 3-4, pp 227-249. ISSN: 1790-8140.

[13] Tang, J.M. and Vuik, C. (2007), *On deflation and singular symmetric positive semi-definite matrices*, Journal of Numerical Analysis and Applied Mathematics, volume 206. Pages 603-614.

[14] Verkaik, J. (2003), *Deflated Krylov-Schwarz Domain Decomposition for the Incompressible Navier-Stokes Equations on a Colocated Grid*, Master's Thesis. Online: http://ta.twi.tudelft.nl/nw/users/vuik/numanal/verkaik.html

[15] Vermeulen, Peter (2006), *'Model-Reduced Inverse Modeling'*, Ph.D. thesis Delft University of Technology.

[16] Vermolen, F.J. and C. Vuik (2001), *The influence of deflation vectors at interfaces on the deflated conjugated gradient method*. Report of the Department of Applied Mathematical Analysis, ISSN 1389-6520, Delft University of Technology.

[17] http://water.usgs.gov/.

[18] http://www.gly.uga.edu/railsback/1121SimpleFaults.jpeg.

[19] http://upload.wikimedia.org/wikipedia/commons/thumb/6/68/Le_Peelhorst,_le_Peelrandbreuk_(faille)_et_le_Slenk_Central.jpg/250px-Le_Peelhorst,_le_Peelrandbreuk_(faille)_et_le_Slenk_Central.jpg.

[20] http://www.tucson.ars.ag.gov/salsa/research/research_1997/AMS_Posters/gw-sw_interactions/gw-sw_f1.html.