ENG6090-S7 – Advanced Numerical Methods for Engineers

Term Project – 2-D Groundwater Flow using MATLAB

Brayden McNeill (student ID: 1156780)

June 23rd, 2020

1.0 Introduction

This report describes the laws and principles which form the basis for groundwater modeling, and the process undertaken to develop a simple 2-dimensional groundwater modeling application using MATLAB ('the MATLAB code').

Groundwater modeling is the process of describing or predicting the movement of water through the ground using a mathematical analog to the real-world. Hydrogeologists have used groundwater modeling as a tool in a variety of applications ever since the late 1970s. Any scenario in which a hydrogeologist or engineer is required to manage groundwater can benefit from the use of a groundwater model as a way of validating engineered designs. From a water quantity standpoint, groundwater models may be used to determine if a city/province/nation will have enough water for residential or industrial supply purposes based on projected increases in demand, or groundwater models may be used to predict impacts on natural hydrologic features due to any increases in extraction (e.g. if a water supply well is pumped at larger than normal capacity will it have a negative impact on a nearby stream). Groundwater models are also important tools in the characterization and assessment of groundwater contaminant issues. For example, if a chemical spill occurs you may use groundwater modeling to estimate the time it will take for the chemical plume to intercept some target location such as a stream or water supply well. A groundwater model may also be used to design and evaluate the effectiveness of groundwater remediation systems.

Numerical models are particularly helpful in the field of hydrogeology since there is always significant uncertainty regarding the subsurface environment. Aquifers (i.e. the soil units through which water flows) are usually highly heterogeneous and don't lend themselves particularly well to analytical solutions. There is considerable uncertainty regarding the character of the subsurface environment (i.e. distribution of soil types/parameters), the physical processes which are at play in the subsurface (i.e. is flow occurring through macropores in the soil), and the temporal nature of the system (e.g. how to characterize precipitation as a long-term average or discrete events). Since the inputs/processes within a groundwater flow system are typically heterogeneous and difficult to characterize (since the system is not directly observable) they are difficult systems to model. Even if the groundwater flow system were perfectly characterized, the heterogeneity of these systems means that most analytical solutions – which typically assume that the system is homogeneous – are inherently flawed. The use of numerical models allows hydrogeologists to test these systems under many different conditions and determine the most suitable parameterization of the system based on observational data (i.e. physical measurements of groundwater elevations, stream fluxes, etc.).

Commercial groundwater models today allow the user to construct highly elaborate and complex groundwater models with irregularly shaped model domains, several different boundary condition types to emulate dozens of real-world hydrologic features, and highly parameterized model domains with dozens of different parameters and levels of heterogeneity. Commercial groundwater applications also offer several different solution techniques, including solutions

based on the finite difference methodology, finite volume methodology and the finite element analysis. The code described in this report is much simpler than today's commercial groundwater modeling applications. Nevertheless, the MATLAB code presented here is based on the same fundamentals principles and is based on the finite different solution technique.

This report includes a brief introduction to the fundamental principles of groundwater modeling in the 'Problem Statement' and 'Governing Equations' sections. The finite difference solution technique used to solve the governing equations for groundwater flow is presented in the 'Solution Using Finite Difference Method' section. Since the MATLAB code is relatively simple and is only applicable to a narrow range of situations, assumptions and limitations inherent to the MATLAB code are discussed in the 'Assumptions' section. The 'Building the MATLAB Code' section describes the overall structure of the MATLAB code and describes how the finite difference solution was applied in MATLAB. Finally, the overall accuracy of the MATLAB code is evaluated in the 'Benchmark Testing and Sources of Error' section. And of course, the 'Conclusion' section reviews the major findings associated with this project and highlights potential improvements to the code that could make it suitable in different settings.

2.0 Problem Statement

The primary problem statement when solving a groundwater modeling problem is always the same: to calculate or solve for the groundwater elevation (or head) throughout the model domain. Once the groundwater elevation is known several other helpful pieces of information can then be calculated. For example, knowing groundwater elevations between two points would allow a hydrogeologist to calculate the groundwater velocity and flux between the two points. A well-defined groundwater flow regime is also required to solve subsequent contaminant transport problems, and as such all contaminant transport models begin with the primary problem statement of solving the groundwater elevations through the model domain.

The problem statement for the MATLAB code is also to calculate the heads or groundwater elevations throughout the model domain. However, unlike commercial groundwater models the MATLAB code presented here has a fixed conceptual model domain, with fixed boundary condition types and locations.

The conceptual model for the MATLAB code is displayed in Figure 1, which is a plan view of the model domain. From the image it is seen that the conceptual model of the code is based on a rectangular model domain. The user of the MATLAB code can specify the minimum and maximum coordinates of the model domain in both the X and Y direction, allowing the code to calculate the total model length and width. The user can also specify the total number of rows and columns, which allows the code to determine the size of each model cell in the X- and Y-direction (i.e. Δx and Δy respectively).

The conceptual model for the MATLAB code also incorporates two distinct boundary condition types which are applied in three ways. Firstly, the upper and lower model boundaries (highlighted

with a thick red line in Figure 1) are represented by Type 2/Neumann boundary conditions, or specified flux. These Type 2 boundaries are specifically meant to represent 'zero-flow' boundaries and are hard-coded into the MATLAB code. In other words, the user is not able to alter the magnitude of the boundaries represented by the thick red lines. Type 2 boundary conditions may also be applied to all 'inner' cells in the model domain, or in other words the cells indicated with a green dot in Figure 1. These green dots indicate a user-specified flux which can conceptualized as precipitation which is infiltrating into the ground and recharging the groundwater. Finally, the cells in the first and last column of the model domain are represented by Type 1/Dirichlet boundary conditions where a groundwater elevation or head is specified by the user. Since water flows from areas of high energy potential to low energy potential (with the total energy potential being a sum of gravitational/elevation and pressure energy), the relative difference in elevation between these two boundary locations will be the primary driver of groundwater flow in the model domain.

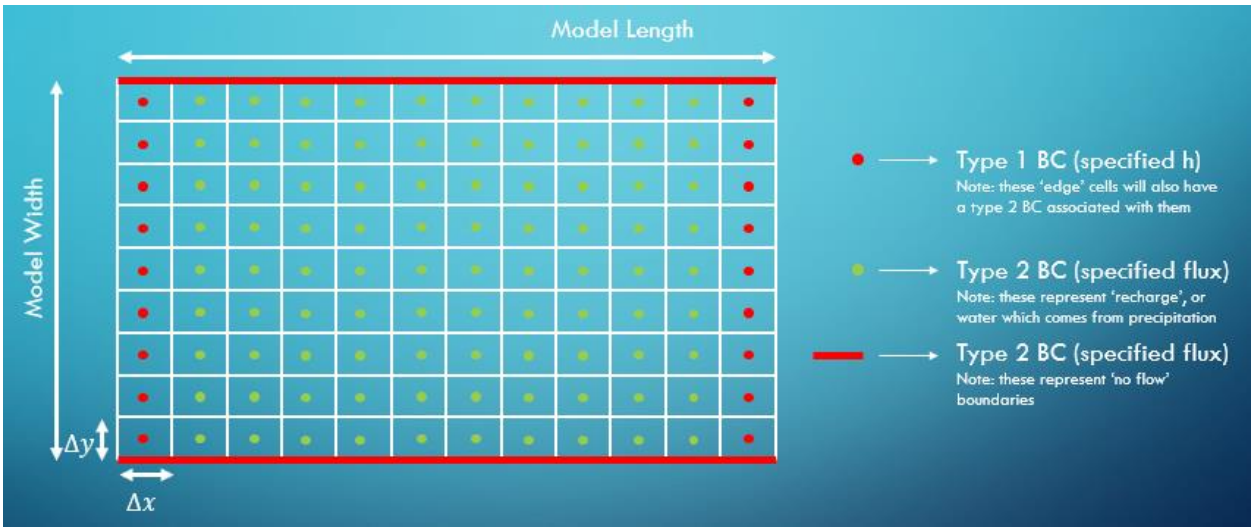

Figure 1: Conceptual Model for the MATLAB Code

Now that the general nature of the problem and the conceptualization of the model domain are understood the governing equations for groundwater flow must be considered. These governing equations are relatively simple and are discussed in the next section.

3.0 Governing Equations

The governing equation for 2-dimensional groundwater flow is a simple combination of the principle of conservation of mass and the foundational law of hydrogeology – Darcy's Law (Anderson & Woessner, 1991); . First let's consider Darcy's Law, which is shown in equation 1 below:

$$Q = K * i * A \qquad \textbf{(1)}$$

where **Q [L^3/T]** represents the volumetric flux of water through a cross-sectional area

K [L/T] is the hydraulic conductivity a parameter which described a porous medias ability to transmit a fluid, or in the case of groundwater modeling the ability of a soil to transmit water

i [unitless] represents the hydraulic gradient (can be rewritten as $\frac{\partial h}{\partial x}$)

A [L^2] represents the cross-sectional area of flow

Darcy's law calculates the flow of a fluid (typically water) through a porous medium (e.g. soil). From equation 1 the overall flow of water is a product of the hydraulic gradient, hydraulic conductivity and the cross-sectional area of flow. The hydraulic conductivity is a parameter of the soil itself and represents the soils ability to transmit water. The other important component of Darcy's law is the hydraulic gradient. As with electricity, water will always flow from an area of high potential energy to an area of low potential energy. In the case of water, the potential energy is known as the potentiometric surface, and in unconfined aquifers (i.e. a water-bearing soil unit which is in contact with atmospheric pressure) is simply represented by the elevation of the water table. Or in other words, the hydraulic gradient can be thought of as the change in groundwater elevation (∂h) over some distance (∂x)

The other fundamental equation for groundwater modeling is the principle of mass conservation. If we consider a control volume, the principle of mass conservation states that the difference between the mass entering and leaving the control volume is equal to the change in mass stored within the control volume. In mathematical terms, the concept of mass conservation is expressed as shown below in equation 2:

$$(Mass\ In) - (Mass\ Out) = (Change\ in\ mass\ stored) \qquad \textbf{(2)}$$

To arrive at the overall governing equation simply expand the terms in equation 2 in terms of Darcy's law, resulting in equation 3 below:

$$K_x \frac{\partial^2 h}{\partial x^2} + K_y \frac{\partial^2 h}{\partial y^2} + W = S_s \frac{\partial h}{\partial t} \qquad \textbf{(3)}$$

where

K_x and K_y [L/T] represent the hydraulic conductivity of the soil in the X-axis and Y-axis, respectively

Ss [unitless] represents the specific storage, a parameter which describes the volume of water released (or added) to aquifer storage based on a unit decline (or increase) in hydraulic head

$\frac{\partial h}{\partial t}$ **[L/T]** represents a change in hydraulic head over time

W [L^3/T] represents any additional volumetric fluxes (sinks/sources) within the control volume

Simply put, the first and second terms in equation 3 represent the sum of flow in the X-axis and Y-axis, respectively, the W represents any additional fluxes of water which are not attributed to base flow (e.g. if a groundwater supply well is pumping water out of the aquifer, or inflow from precipitation) and the terms on the right represent the change in water stored in the aquifer over some time. Please note that the cross-sectional area term which was included in Darcy's Law

(equation 1) is not explicitly included in equation 3. However, it will reappear when the problem is expressed in terms of a finite difference solution (see section 4.0)

Equation 3 is therefore the principle equation which describes groundwater flow in 2-dimensions and is the equation that must be solved in groundwater modeling problems (Anderson & Woessner, 1991). As a partial differential equation, the equations for groundwater flow in 1-, 2- or 3-dimensions lend themselves well to a solution using finite difference methods and is the solution method used in the MATLAB code presented in this report. The following section describes the finite difference solution methodology for equation 3.

4.0 Solution Using Finite Difference Method

Frind (2003) provides a detailed description of how to solve equation 3 using the finite difference methodology and was an essential source of guidance during this project. The user manual to MODFLOW-2005 (a widely used groundwater modeling code) also provided valuable insight as to how groundwater modeling applications typically work (Harbaugh, 2005).

To apply the finite difference solution method, the partial differential equation from the previous section (i.e. equation 3) must be re-written in an algebraic or discretized format. If an elemental control volume is considered, the flow across each interface of the control volume can be rewritten in terms of Darcy's law, as shown in Figure 2 below:

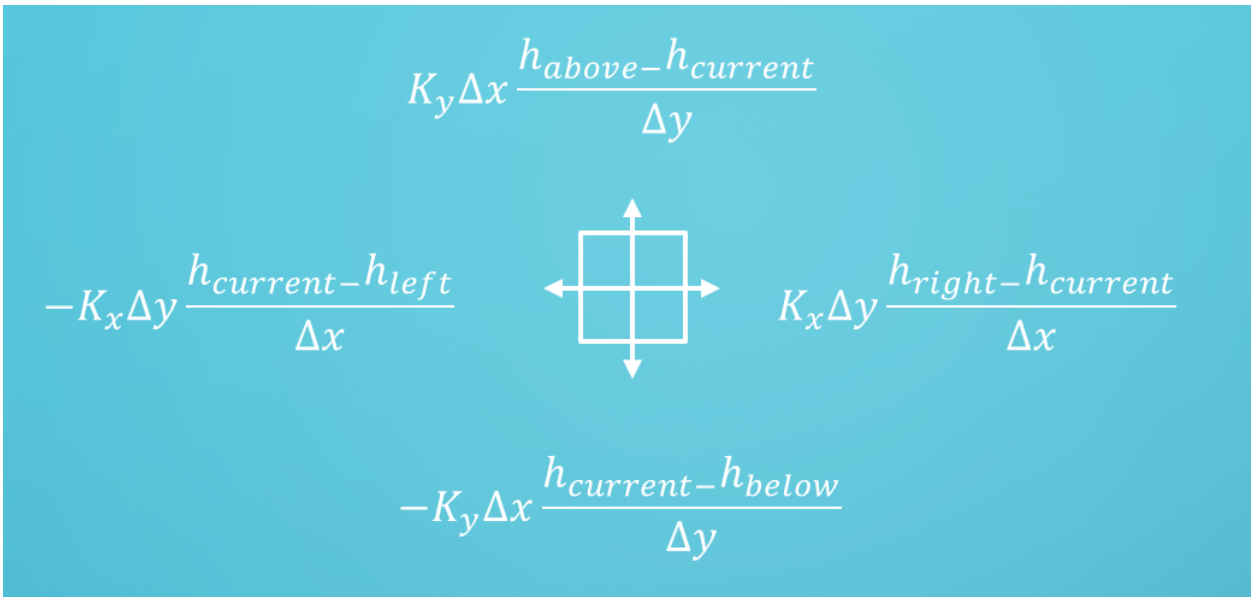

Figure 2: Control Volume and Expressions for Groundwater Flow in Each Direction

In Figure 2 the flow across each cell interface is expressed in terms of the hydraulic conductivity in the direction of flow (i.e. Kx and Ky; note that some soils are anisotropic and will favour flow in one axis), the cross-sectional area of flow (i.e. Δx and Δy for flow along the X-axis and Y-axis, respectively) and the change in hydraulic gradient across the cell interface (i.e. the change in hydraulic head over some distance Δx or Δy).

If flow through the cell is generally to the left and down, we can say that water is leaving the cell on the left-interface and bottom-interface, which is why a negative symbol precedes those terms. These would represent the 'Mass Out' term in equation 2. Similarly, flow across the right-interface and top-interface would represent the 'Mass In' term in equation 2. Therefore, combining these terms together and rewriting the mass conservation equation gives equation 4 below, which is the overall governing equation for groundwater flow in 2-dimensions:

$$-K_x \Delta y \frac{h_{i,j}-h_{i-1,j}}{\Delta x} - K_y \Delta x \frac{h_{i,j}-h_{i,j-1}}{\Delta y} + K_y \Delta x \frac{h_{i,j+1}-h_{i,j}}{\Delta y} + K_x \Delta y \frac{h_{i+1,j}-h_{i,j}}{\Delta x} + W = S_s \frac{\partial h}{\partial t} \qquad \textbf{(4)}$$

where **K_x and K_y [L/T]** represent the hydraulic conductivity of the soil in the X-axis and Y-axis, respectively

Δx and Δy [L] represent the cross-sectional area of flow in the X-axis and Y-axis, respectively

$h_{i,j}$ [L] represents the hydraulic head of the current cell

i and j [-] are cell-indexing values in the X-axis and Y-axis respectively, used to generalize the equation instead of writing 'left', 'right', 'up', 'down'

Ss [unitless] represents the specific storage, a parameter which describes the volume of water released (or added) to aquifer storage based on a unit decline (or increase) in hydraulic head

$\frac{\partial h}{\partial t}$ [L/T] represents a change in hydraulic head over time

W [L^3/T] represents any additional volumetric fluxes (sinks/sources) within the control volume

Equation 4 can be rearranged to give equation 5 below:

$$K_x \Delta y \left(\frac{h_{i-1,j}-2h_{i,j}+h_{i+1,j}}{\Delta x^2}\right) + K_y \Delta x \left(\frac{h_{i,j-1}-2h_{i,j}+h_{i,j+1}}{\Delta y^2}\right) + W = S_s \frac{\partial h}{\partial t} \qquad \textbf{(5)}$$

We can see that equation 5 is simply equation 3 rewritten with an approximation for the 2nd differential terms in the X- and Y-direction, and with cross-sectional area terms included.

In the MATLAB code presented here, equation 4 is further simplified by assuming that the solution will be steady state. As such, the terms on the right-hand side of the equation are equal to zero, resulting in equation 6 (note: the sources/sink term W has been moved to the right-hand side):

$$-K_x \Delta y \frac{h_{i,j}-h_{i-1,j}}{\Delta x} - K_y \Delta x \frac{h_{i,j}-h_{i,j-1}}{\Delta y} + K_y \Delta x \frac{h_{i,j+1}-h_{i,j}}{\Delta y} + K_x \Delta y \frac{h_{i+1,j}-h_{i,j}}{\Delta x} = W \qquad \textbf{(6)}$$

The next step in the finite difference solution is to discretize the model domain, which is easily accomplished by specifying the model size in the X- and Y-directions, and the total number of rows and columns. In other words, consider the model domain as a series of 'cells' which are treated in the same way as the elemental control volume mentioned at the beginning of the section. This allows the discrete terms Δx and Δy to be calculated using equations 7 and 8 below:

$$\Delta x = \frac{X_{max}-X_{min}}{\#\ of\ columns} \qquad \textbf{(7)}$$

$$\Delta y = \frac{Y_{max} - Y_{min}}{\#\ of\ rows} \qquad \textbf{(8)}$$

Please note that equation 6 includes 5 unknown terms: the heads in the current cell and all neighboring cells. Of course, a single equation with 5 unknowns is not solvable. However, equation 6 is applied to all cells within the model domain. Furthermore, boundary conditions are applied to all cells on the exterior of the model domain, either as known heads (type 1 boundary) or known fluxes (type 2 boundary). Therefore, as the system of equations is expanded across the model domain the result is a system of n-equations with n-unknowns, for a system with n-cells. Boundary conditions will be revisited later in this section.

Once the model domain is discretized the system of equations must be re-written in matrix format [A]*[x]=[b], where [x] is a matrix which represents all unknown values (or heads in each cell), [A] represents the coefficient matrix, and [b] represents all known values. In the case of the groundwater modeling problem it may be easier to consider the system of equations as follows:

$$[Conductance\ Matrix] * [Heads\ Matrix] = [Known\ Values] \qquad \textbf{(9)}$$

In equation 9, the [*Conductance Matrix*] contains all the coefficients which are multiplied with the unknown head terms. Referring to equation 6 we can see that all terms which describe flow to neighboring cells in the X-direction (left and right, or i-1 and i+1 respectively) have the following coefficient term *a*, as shown in equation 10:

$$a = \frac{K_x * \Delta y}{\Delta x} \qquad \textbf{(10)}$$

Similarly, in equation 6 all terms which describe flow to neighboring cells in the Y-direction (above and below, or j+1 and j-1 respectively) have the following coefficient term *b*, as shown in equation 11:

$$b = \frac{K_y * \Delta x}{\Delta y} \qquad \textbf{(11)}$$

And finally, all terms on the left-hand side of equation 6 have a term which describes flow to/from the current cell, or in other words all terms contain a value for head in the current cell. As such, we can define a coefficient *d* for head in the current cell as shown in equation 12:

$$d = 2(a + b) \qquad \textbf{(12)}$$

Figure 3 illustrates the coefficients applicable to the current and neighboring cells for an *interior* cell.

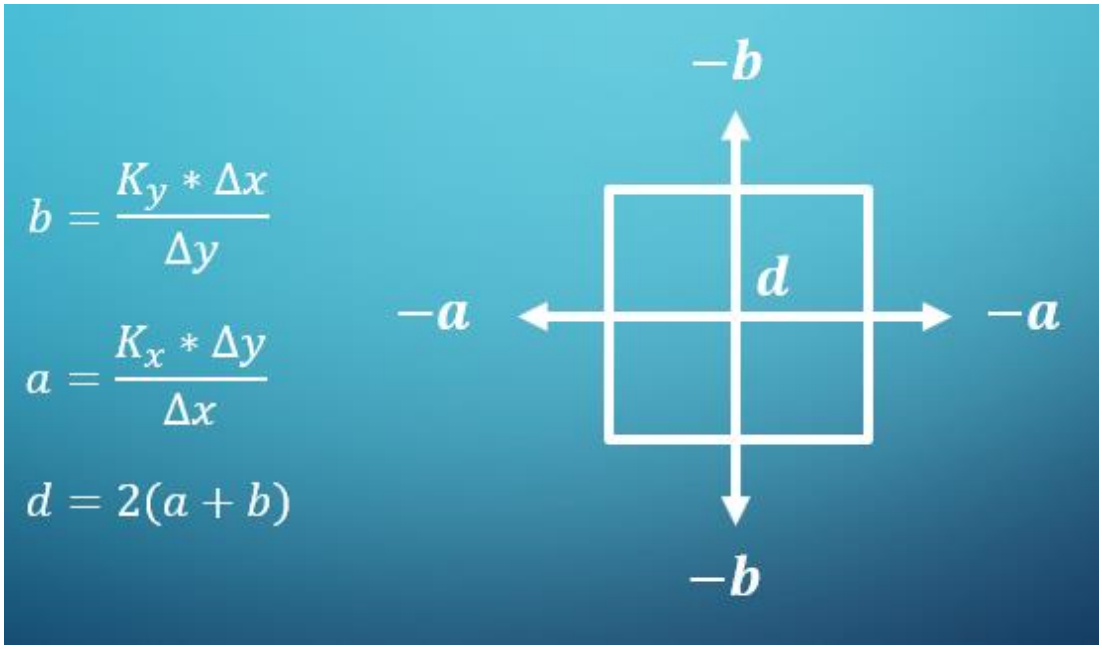

Figure 3: Conductance Coefficient Template for Interior Cell (adapted from Frind, 2003)

Please note that the conductance template shown in Figure 3 does not apply to cells along the top and bottom row of the model domain. Referring to Figure 1 we can see that a type 2 boundary condition ('no-flow') is applied to the upper and lower edge of the model domain. As such, the conductance template for cells along the top and bottom row must be altered. Figure 4 illustrates the conductance template for a cell along the bottom row of the model domain, where q_b represents the magnitude of the applied flux (note: the magnitude q_b for the upper and lower boundary of the model domain is 0, and therefore nothing is added to the right hand side (RHS) of the matrix equation).

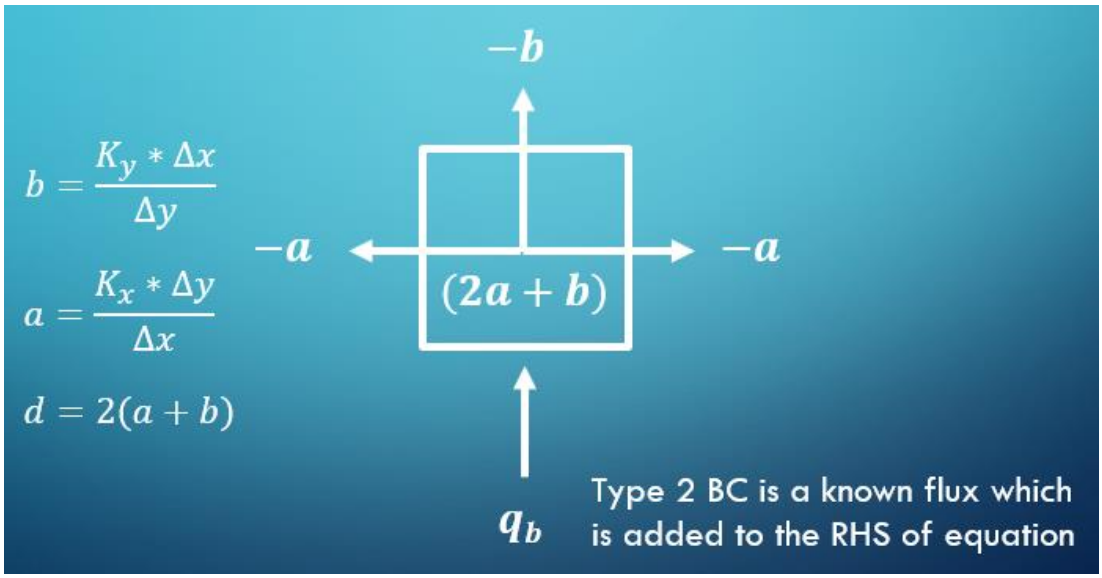

Figure 4: Conductance Coefficient Template for Interior Cell (adapted from Frind, 2003)

In equation 9, the [Heads Matrix] would contain all unknown head values throughout the model domain. Since the heads in the first and last column of the model domain are defined in this MATLAB code these would not be included in the [Heads Matrix]. Therefore, for a model domain with **n** rows and m columns there would be **n*(m-2)** unknown head values.

Finally, the right-hand side of equation 9 contains all known values. In the conceptual model for this MATLAB code (Figure 1) there are two sources of know values: the defined head values in the first and last column of the model domain (i.e. the type 1 boundary conditions), and the defined values for the 'recharge' boundary condition (i.e. the type 2 boundary conditions for interior cells).

Each model cell containing a defined head (type 1) boundary condition will have exactly 1 neighboring cell with an unknown head (i.e. the interior cell to the right or left of the type 1 boundary). As such, the defined head value is multiplied by the **a**-coefficient (equation 10) and is moved to the right-hand side of equation 9, with the index within the [Known Values] matrix corresponding to the index of the neighboring cell within the [Heads Matrix].

With respect to the type 2/recharge boundary condition, the defined recharge rate ([L/T]) is multiplied by the interfacial area of flow (i.e. the area of the cell, $\Delta x * \Delta y$) to achieve a flux term with units [L3/T], and is moved to the right-hand side of equation 9, with the index within the [Known Values] matrix corresponding to the index of the neighboring cell within the [Unknown Heads] matrix.

Please note that the details regarding the construction of the [Conductance Matrix], [Heads Matrix] and [Known Values] matrices will be revisited in greater detail in section 6.0.

Once the system of equations has been successfully constructed it can be solved using any method typically used to solve systems of linear equations. The MATLAB code presented in this report is based on the Gauss-Seidel method, and further details are provided in section 5.0.

5.0 Reviewing the MATLAB Code

With an understanding of the governing equation for groundwater flow and the general methodology for the finite difference solution it is possible to apply these concepts in MATLAB. This section of the report introduces the 2-dimensional groundwater modeling MATLAB code and discusses how it functions. Section 5.1 introduces the user interface and how the application can be used, whereas section 5.2 focuses on the structure of the code and the callback function when the 'Run' button is clicked.

5.1 User Interface

The user interface for the groundwater modeling application is shown in Figure 5. The user interface includes three different panels which require user input before the code can be run, and two panels which display results. A list of instructions are also included at the top of the

interface to help guide a new user. Also please note that tooltips are included for most fields, so new users can simply scroll the computer mouse over each field to learn more.

The first user-input fields appear under the **'Define Model Domain'** panel. The fields within this panel allow the user to specify the overall size of the model domain in the X-axis using the '**Xmin'** and '**Xmax'** fields, and the overall size of the model domain in the Y-axis using the '**Ymin'** and '**Ymax'** fields. The user must also specify the total number of rows and columns using the **'# of Columns'** and **'# of Rows'** fields. Please note that the minimum value for the # rows/columns fields is set to 5. This minimum value is required in order for the code to function properly, and also has the added benefit of ensuring a certain level of refinement in the model domain. The user can click the **'Calculate dX and dY'** button to populate the **'dX'** and **'dY'** fields, which indicate the width/length of the resulting cells using equations 13 and 14, respectively.

$$dX = \Delta x = \frac{X_{max} - X_{min}}{\#\ of\ Columns} \qquad \textbf{(13)}$$

$$dY = \Delta y = \frac{Y_{max} - Y_{min}}{\#\ of\ Rows} \qquad \textbf{(14)}$$

Please note that it isn't necessary for the user to click the **'Calculate dX and dY'** button for the code to function properly. However, to numerical error is minimized when dX=dY and therefore a button to check the resulting dX and dY is provided to help the user with the grid design.

Finally, a **'Model Depth'** field is included as well. Please note that this field is not actually used to solve the 2-dimensional groundwater modeling problem. This field is used primarily to calculate the volumetric flux results which appear in the **'Cell-Specific Results'** section. These results will be discussed later in this section. It is important to highlight the presence of this user input field because without prior knowledge it does give the impression that this is a 3-dimensional model domain.

The second batch of user input fields appear in the **'Define Model Parameters/Boundary Conditions'** panel. Under the section for **'Parameters'** are fields for Kx and Ky, which are the hydraulic conductivities of the modeled soil material in the X- and Y-directions respectively. Kx and Ky must both be positive numbers. A third user input field for **'Neff'** is also included. 'Neff' stands for effective porosity, which is a property of the soil material with a value between 0 and 1. The effective porosity is essentially a measure of the open space within a unit of soil, and represents the amount of soil volume which is available for water to flow through (water flows through pores in the soil, not through the soil particles themselves). As with the 'Model Depth' parameter, 'Neff' is not a required parameter for the solution of the groundwater model itself, but is rather used to calculate the groundwater velocities which appear in the **'Cell Specific Results'** panel, and will be discussed later in this section.

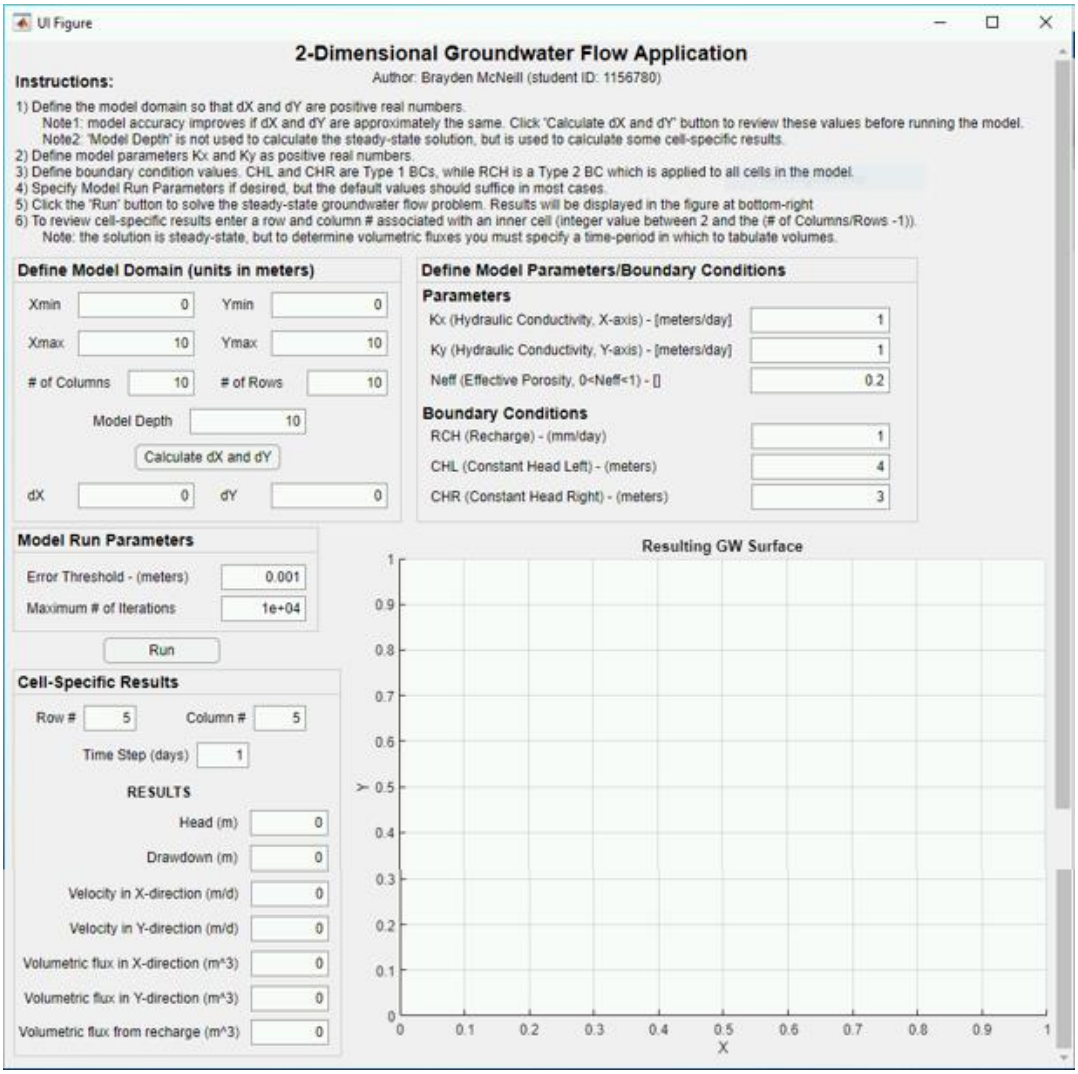

Figure 5: User Interface for the 2-Dimensional Groundwater Flow Application

The **'Define Model Parameters/Boundary Conditions'** panel also includes input fields for the boundary conditions discussed in previous sections (note: the type 2/no-flow boundary conditions at the upper and lower model boundary are fixed and have a magnitude of 0 by definition, therefore these boundaries are not editable). Three input fields are available, including a volumetric flux for recharge/precipitation (i.e. the **'RCH'** field which represents the type 2 boundaries illustrated with green dots in Figure 1), and two input fields for the type 1/constant head boundary conditions in row 1 (i.e. the **'CHL'** or 'Constant Head Left') and in the last row (i.e. the **'CHR'** or 'Constant Head Right'). A positive **'RCH'** value is typical and would indicate an influx of water from precipitation. However, it is possible to enter a negative value which could be conceptualized as evapotranspiration losses to the atmosphere. Any values can be entered for **'CHR'** and **'CHL'**, and it is the difference in magnitude between these two boundaries which will be the primary driver of groundwater flow in the model.

Finally, two user-input fields appear under the **'Model Run Parameters'** panel, **'Error Threshold'** and **'Maximum # of Iterations'**. Of course the **'Error Threshold'** field indicates the level of accuracy required from the solution before it is considered 'solved', while the **'Maximum # of Iterations'** field allows the code to terminate in cases where the solution is not converging quickly. A warning message will appear if the maximum number of iterations is reached, and the results for the current iteration will be displayed anyway. An example of this warning is shown in Figure 6.

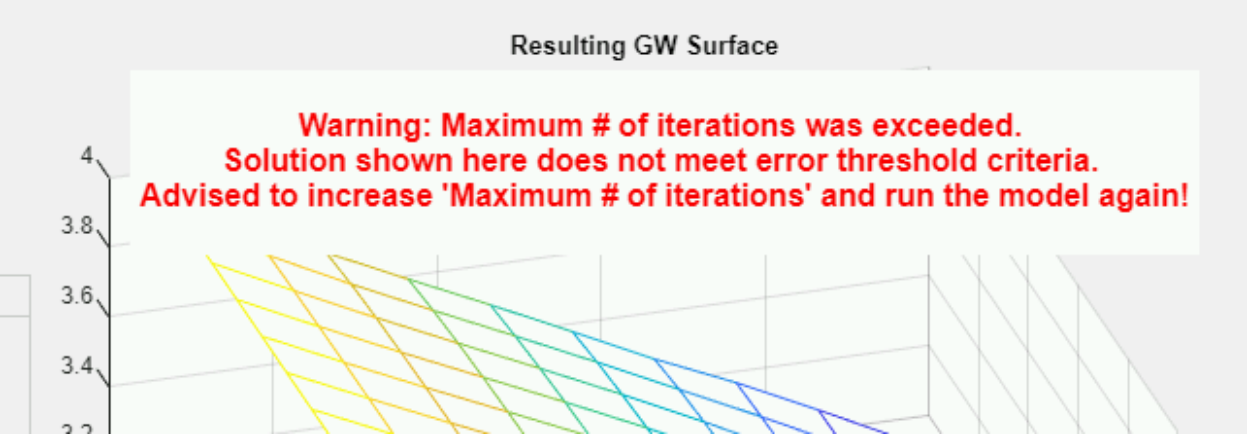

Figure 6: Warning Message Displayed if Maximum # of Iterations is Reached

Once all user input fields have been specified as desired by the user, click the **'Run'** button under the **'Model Run Parameters'** panel to run the code and solve the groundwater flow problem. The code should typically execute successfully, and the results will then be displayed in the **'Resulting GW Surface'** graph and the **'Cell-Specific Results'** panel.

The primary results of the groundwater modeling code (i.e. a single groundwater elevation value for each model cell) is displayed as a 3-dimensional mesh under the **'Resulting GW Surface'** graph at the bottom right of the user interface.

Additional cell-specific results are displayed on the bottom left. The user must specify which specific cell to display the results for using the **'Row #'** and **'Column #'** fields. If the user specifies a row or column which is outside the range of the model a warning message is displayed indicating the error and no results will be displayed (Figure 7). The user must also specify a period of time over which to calculate these results using the **'Time Step'** field. While the solution is steady-state and therefore time-independent, a particular time period is required in order to calculate volumetric fluxes. The default time step is set to 1 day.

Cell-specific information includes the following:

- **Head** – the calculated groundwater elevation at the desired cell
- **Drawdown** – the difference between the initial head and calculated head at the desired cell
    - Note: initial head is the average of CHL and CHR

- **Groundwater Velocity in X- and Y-direction** – indicates the velocity of groundwater flow in each direction at the desired cell
- **Volumetric Flux in X- and Y-direction** – indicates the volume of water passing through the cell in each direction over a specified time period (i.e. 'time step')
- **Volumetrix flux from recharge** – indicates the volume of water which entered the cell from the recharge boundary condition over a specified time period (i.e. 'time step')

The equations used to calculate groundwater velocities (equation 15), baseflow fluxes (equation 16) and fluxes from recharge (equation 17) are included below. Please note that equation 15 and 16 are shown for the X-direction, equations for the Y-direction are very similar:

$$Velocity_X = \frac{1}{2} * \left(\frac{\frac{K_X(Head\ _{Current} - Head\ _{Downstream})}{\Delta x}}{Neff} + \frac{\frac{K_X(Head\ _{Upstream} - Head\ _{Current})}{\Delta x}}{Neff} \right) \quad \textbf{(15)}$$

$$Volumetric\ Flux_X = Velocity_X * \Delta y * Model\ Depth * TimeStep \quad \textbf{(16)}$$

$$Recharge\ Flux = RCH * 0.001 * \Delta x * \Delta y * TimeStep \quad \textbf{(17)}$$

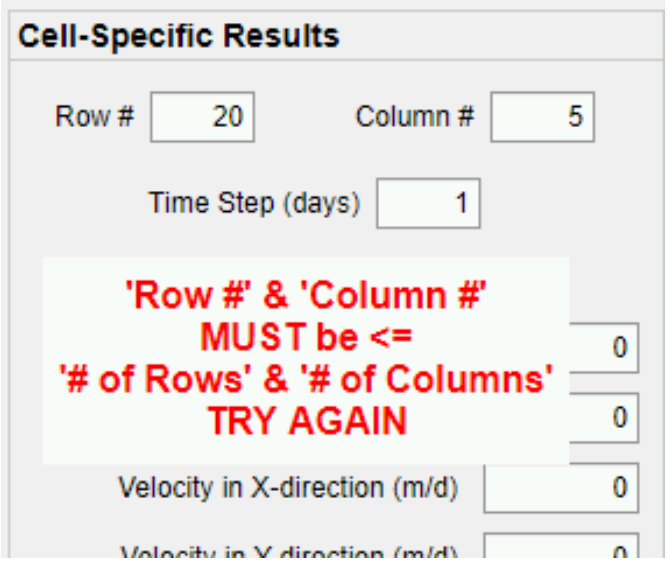

Figure 7: Warning Message Displayed when Specified Cell is Outside Range of Model

This concludes the review of the user interface for the 2-dimensional groundwater modeling code. In the next section the 'Run' button callback function is discussed in detail.

5.2 Run Button Callback Function

The most important code with respect to the 2-dimensional groundwater modeling application is contained within the callback function when the 'Run' button is pressed by the user. This section of the code is approximately 300 lines long. In this section of the report the 'Run' button functionality will be discussed in detail, but please note that this section of the code does include considerable annotation which should make it possible for any user to review and understand the solution procedure.

The procedure has five general steps, each of which is discussed in further detail in the subsections to follow. The six steps are:

1. Initialize Variables
2. Assemble the [Conductance Matrix]
3. Assemble the [Known Values Matrix]
4. Solve using Gauss-Seidel Method
5. Organize and Display the Results

5.2.1 Initializing Variables (Lines 108-166)

The first section of the code is quite simply to initialize all necessary variables and to retrieve or calculate the variables based on the user-input fields. Most variables are simply based on user-input fields, but the dX and dY variables (which correspond to Δx and Δy, respectively) are calculated using equations 13 and 14 from section 5.1. The values for dX and dY are also displayed in the related user-input fields, in case the user hasn't already used the **'Calculate dX and dY'** button.

5.2.2 Assemble the Conductance Matrix (Lines 168-292)

`As discussed in section 4.0, the finite difference solution requires the system of equations to be in the form [A]*[x]=[b], where matrix [A] contains the coefficients for unknown variables [x]. In the context of the groundwater modeling problem, matrix [A] is called the [Conductance Matrix] and it contains a coefficients **a**, **b** and **d** which were introduced in section 4.0. Please note that in the MATLAB code this matrix is called **'GlobalM'**.

As such, the first task in this section of the code is to calculate the values for coefficients a, b and d using equations 10, 11 and 12 above (Lines 168-173).

The second task is to create an empty matrix of the correct size (Lines 186-187). The [Conductance Matrix] must be a square two-dimensional matrix, with the total number of rows/columns being the same as the total number of unknown head values within the model domain. Recall that heads are defined for the first and last column, and as such the [Conductance Matrix] the number of rows and columns in the matrix can be calculated using equation 18:

$$\#\ of\ Unknowns = (\#Rows) * (\#Columns - 2) \qquad \textbf{(18)}$$

Once the empty [Conductance Matrix] is initialized the next task is to populate it with the coefficients previously calculated. This task is somewhat complicated since there are multiple cells throughout the model domain which require special considerations. The simplest cells are those interior cells which are at least 1 row/column removed from the edges of the model domain, since these have four neighboring cells (see Figure 3). However, all four corner cells are unique within the scheme of the model domain, since they each have different neighbors (for example, the cell at the top-left of the model domain has a neighbor to the right and below, whereas the cell at the bottom-right corner has a neighbor to the left and above). Cells in the second and second last columns (not counting corner cells) are also unique, since they border on

a type 1 boundary condition. Furthermore, cells in the first and last rows (not counting corner cells) are also unique since they border on a type 2 boundary condition.

With these complications in mind, lines 189-292 of the code populate the [Conductance Matrix] with the desired coefficients. The code begins by entering the required coefficients for the corner cells (lines 200-230), followed by a series of five 'for loops' which enter the remaining coefficients for the second column (lines 232-242), then the second last column (lines 244-254), then the first/bottom row (line 256-266), then the last/top row (line 268-278) and finally all remaining interior cells (line 280-292).

The result is a diagonally dominant, symmetric, pentadiagonal matrix. An example [Conductance Matrix] for a model with five rows and columns is shown in Figure 8. Note the banded nature of the matrix, with all d-coefficients appearing on the diagonal, offset by bands of b- and a-coefficients for corresponding neighbor cells. Entries on the diagonal which are highlighted in yellow indicate cells bordering a type 2 boundary (i.e. corner cells, or cells in the top/bottom row). In these cases, a different template must be used for the conductance terms (see Figures 3 and 4 from section 4.0).

Figure 8: Example [Conductance Matrix] for a 5x5 Model Domain

5.2.3 Assemble the [Known Values] Matrix (Lines 294-313)

Once the [Conductance Matrix] is assembled the next step is to assemble all known values to the right-hand side of the typical matrix equation [A]*[x]=[b]. All known values (i.e. related to type 1 and 2 boundary conditions) must be moved to the right-side of the equation. Please note that in the MATLAB code this [Known Value Matrix] is called **'RHS'**.

There are two tasks to populate the [Known Value Matrix]. The first task is to address the flow terms which are related to the constant head boundary condition on the left side (i.e. CHL) and right side (i.e. CHR) of the model domain. Since the cells connected to CHL and CHR are all directly

to the right or left of CHL/CHR the a-coefficient is used to calculate the fluxes between these cells. Therefore, we apply the following equations to calculate the flux terms which appear in the [Known Value Matrix] for CHL and CHR, respectively (note: these equations are repeated and applied to the 'RHS' matrix once for each row):

$$Known\ Flux\ from\ CHL = a * CHL \quad \textbf{(19)}$$

$$Known\ Flux\ from\ CHR = a * CHR \quad \textbf{(20)}$$

The next task to populate the [Known Value Matrix] is to calculate the flux due to the recharge (i.e. RCH) boundary condition. This flux is calculated using equation 21 below, and is applied to ALL entries in the [Known Value Matrix] (since this flux applies to ALL interior cells):

$$Recharge\ Flux = RCH * 0.001 * \Delta x * \Delta y \quad \textbf{(19)}$$

The final resulting [Known Value Matrix] would look similar to Figure 9, an example based on a 5x5 model domain. The first five values in the matrix correspond to the CHL boundary, the final five entries would correspond to the CHR boundary, and a small flux (0.004 m^3 in this case) is applied to all entries:

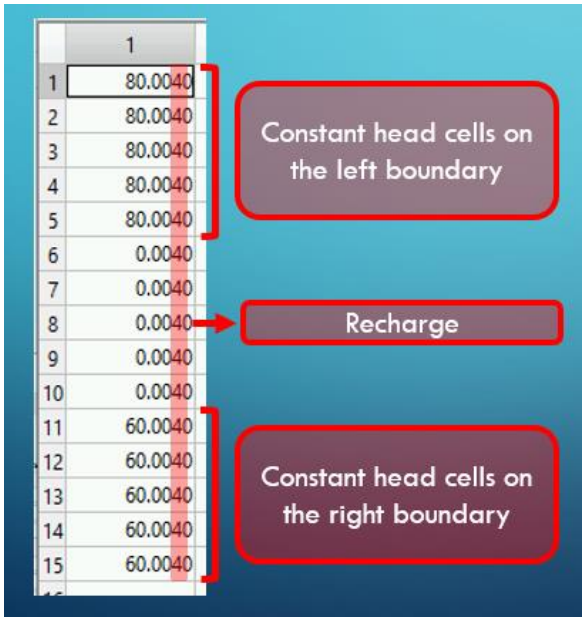

Figure 9: Example [Conductance Matrix] for a 5x5 Model Domain

5.2.4 Solve Using the Gauss-Seidel Method (Lines 315-352)

Once the [Conductance Matrix] and [Known Value Matrix] are assembled it is a simple matter to apply the Gauss-Seidel solution method to determine the final solution. Fortunately, the conductance matrix is *always* diagonally dominant (based on the fast that the d-coefficient is always greater than a- or b-coefficients), which means that the Gauss-Seidel solution method will always converge. Before running the Gauss-Seidel method a few small tasks are performed to prepare. First a matrix of initial values equal to (CHR+CHL)/2 is generated ('initialHead'). Another matrix ('heads') is also created to store the estimated head values for the current iteration. Yet

another matrix ('Error') is generated to store the relative approximate error for each cell in the model domain. Finally, a conditional for loop is executed to run the Gauss-Seidel method. The for loop is run continuously until the maximum error value throughout the entire model domain is less than the error threshold applied by the user OR the maximum number of iterations is met.

5.2.5 Organize and Display the Results (Lines 354-413)

Finally, once the Gauss-Seidel solution method has run its course, the final step in the code is simply to organize the results and to display them in an easily interpreted format. At this stage the results are contained in a matrix which contains a single row, and the head values for the cells containing CHL and CHR are not included, even though these should be included in the final plot.

Therefore, the first few lines in this section of the code simply create a new matrix ('finalHeads') with the correct dimensionality to contain the final results. The values from CHL, CHR and the 'heads' matrix are then transcribed into the 'finalHeads' matrix, which is then reshaped into a matrix with a number of rows and columns equivalent to the number of rows and columns within the model domain.

In order to display the results in a mesh diagram, additional variables for the X and Y coordinate are also required. Therefore, a matrix of X and Y coordinates are created. All these results are used to display the final groundwater table/elevations using the mesh command.

The last lines in the code (lines 397-410) calculate the cell-specific results as described in section 5.1.

6.0 Assumptions, Limitations and Sources of Error

Please note that several assumptions and limitations are implicit in the functioning of this 2-dimensional groundwater modeling application. This section reviews some of these assumptions and limitations and provides general guidelines to ensure that the model is used appropriately. This section also discusses potential sources of error in the code and methods which may be used to reduce these errors.

6.1 Assumptions and Limitations of the MATLAB Code

Many of the limitations of the MATLAB code are implicitly tied to the assumptions which were made as the code was developed, and these can be easily uncovered by comparing the functionality of the MATLAB code to any of the readily available modeling codes which are available throughout the world (i.e. open-source codes and/or commercial codes). For example, the MATLAB code assumes that the solution is steady state, but no groundwater system is truly at equilibrium. Commercial groundwater modeling applications overcome this hurdle by supporting transient solutions based on time-varying inputs. This, unfortunately, is beyond the capabilities of this simple MATLAB application.

The MATLAB code also assumes that a real-world aquifer can be properly modeled based on the conceptual model which is implicit to the code (see Figure 1). In most cases the conceptual model would not be sufficiently like the real-world aquifer, which of course imposes a source of error on the eventual solution compared to real-world observations. With commercial applications this source of error is overcome by providing the user with enough flexibility in recreating the real-world environment, for example by allowing the user to define an irregularly shaped model domain and providing the user with control over the types and locations of boundary conditions.

However, even in a real-world setting which is very similar to the conceptual model there are assumptions in the solution which introduce errors. For example, the current code assumes that the aquifer is homogeneous, although in the real-world there are always variations in hydraulic conductivities at different scales. Even if the MATLAB code did provide the ability to assign different hydraulic conductivity values to different cells, the assumption is that the parameters are at least continuous throughout the volume of a given cell – an assumption which is not strictly true. The MATLAB code also assumes that flow only occurs in the directions of the principle axes, although this is also false (there is no 'grid' in the real world). The MATLAB code also assumes that gradients in hydraulic potential in one direction only drive groundwater flow in that direction. However, in the real world it is possible (and common) for hydraulic gradients in the X-direction to drive 'cross-flow' in the Y-direction. These and many other assumptions make the code less reliable at modeling real-world groundwater flow.

Fortunately, with respect to testing the code for accuracy (see section 7.2) the MATLAB code and application used for testing (Visual MODFLOW Flex) are subject to the same assumptions and limitations, and therefore should present similar solutions. The assumptions and limitations inherent in the MATLAB code are more important when comparing the solution from MATLAB against real-world observational data.

6.2 Sources of Error in the Numerical Solution

There is only one major source of error related to the Gauss-Seidel solution method, specifically round-off errors. Another concern is that solution method is stepwise in the sense that a new estimate is calculated one node at a time, and the new estimate is then used to calculate estimates at subsequent nodes. The result is that errors are cumulative within each solution iteration, resulting in higher relative error in the final nodes calculates by the Gauss-Seidel method. This results in some directional bias in the relative approximate error of the solution. See section 7.2 for more details on this directional bias.

7.0 Benchmark Testing

The MATLAB code was tested against two criteria: the 'run' time (i.e. the elapsed time between clicking the 'Run' button and the solution being displayed) and the relative approximate accuracy compared to a commercial groundwater modeling code called Visual MODFLOW Flex. Both types of test were run using the same model composition (i.e. size, boundary condition values, etc.). The composition/parameters of the testing model are shown in Table 1:

Xmin	0	**Kx**	1
Xmax	10	**Ky**	1
Ymin	0	**Neff**	0.2
Ymax	10	**RCH**	10
# Rows/dY	Variable	**CHL**	10
# Columns/dX	Variable	**CHR**	5

Table 1: Composition of Model Used for Benchmark Testing

7.1 Testing Run Times

To test the overall efficiency of the MATLAB code the model described in Table 1 was run several times with different levels of grid refinement and with different levels of accuracy (using the Error Threshold variable). The run time was tabulated using the stopwatch functionality in MATLAB, and the total number of iterations required for the Gauss-Seidel method to reach the solution were also recorded. The results of the run time tests are displayed in Table 2.

	Grid Refinement			
Error Threshold	**5x5 grid (25 nodes)**	**10x10 grid (100 nodes)**	**20x20 grid (400 nodes)**	**40x40 grid (1600 nodes)**
0.001	0.031 (10)	0.078 (21)	0.654 (49)	16.662 (90)
0.0001	0.042 (17)	0.090 (51)	0.948 (97)	34.532 (265)
0.00001	0.033 (24)	0.094 (88)	1.456 (195)	53.835 (444)
0.000001	0.086 (30)	0.112 (125)	2.346 (362)	84.298 (735)
0.0000001	0.043 (37)	0.122 (163)	3.201 (529)	157.556 (1439)

Table 2: Results of Run Time Testing – Run Time in Seconds (# of Iterations Required)

The results of run time testing indicate that for smaller grids with fewer nodes the model run times are quite fast (<1 second). However, as the size of the model increases, especially above 400-500 nodes, there is a rapid increase in model run times at all accuracy levels. Of course, there is also a general trend of increasing run times as the error threshold is decreased, but overall it would seem that the number of nodes is the primary cause of excessive model run times.

While the model run times are quite short with smaller models, users familiar with commercial groundwater modeling software may be discouraged by the longer model run times with larger models. In fact, the 'large' model tested here is quite small by the standards of commercial groundwater models which may have hundreds of thousands or even millions of nodes. These models may take hours to run, but that would be expected for exceptionally large models. But a model run time of approximately 2.5 minutes for a simple model with only 1600 nodes does not compare favorably to commercial groundwater modeling codes.

7.2 Accuracy Testing with Visual MODFLOW Flex

Unfortunately, there is no analytical solution available for a 2-dimensional groundwater model with the desired boundary conditions. However, the accuracy of the solution provided by the MATLAB code can be easily compared to commercial applications with the same functionality. A series of accuracy tests were performed by comparing the results of a model built in the MATLAB code against an identical model built in Visual MODFLOW Flex (VMOD).

To compare the results from the two applications an identical model (described in Table 1) was built in both applications, and the relative approximate error of each model cell was calculated using equation 20. Finally, the average of the relative approximate errors across all cells was calculated, and the results are presented in Table 3. Please note that the error threshold for these tests were set to 0.001 for all model runs.

$$\in_a = \frac{Result\ from\ MATLAB - Result\ from\ VMOD}{Result\ from\ VMOD} x100\% \quad \textbf{(20)}$$

Grid Refinement (RowsxColumns)	Relative Approximate Error (ε_a)
5x5	**3.67%**
10x10	**2.51%**
20x20	**2.64%**
40x40	**2.60%**

Table 3: Relative Approximate Error of MATLAB Code Compared to Visual MODFLOW at Various Levels of Refinement

From the results in Table 3 it is clear that the MATLAB code does result in relatively accurate solutions, at least compared to one popular commercial application with the same functionality. It's somewhat unusual that the relative approximate error is in the same range at various levels of grid refinement. However, this may be expected since both the MATLAB code and Visual MODFLOW Flex are numerical solutions to the same governing equations, and as such they are subject to the same sources of numerical errors.

Since the solution method is stepwise in fashion there is a directional bias inherent in the relative approximate error of the MATLAB solution. This directional bias is an artefact of the solution methodology, which begins by calculating a new estimated groundwater level at node 1, and then moving to subsequent nodes. At each subsequent node the estimate from earlier cells are used to estimate a new value in the current cell, which means that errors from previous nodes tend to accumulate in subsequent nodes as the solution progresses. This can be resolved by simply lowering the 'Error Threshold', which ensures that all nodes/cells in the model domain will achieve a certain level of accuracy. But overall the directional bias will still be evident. This can be illustrated by reviewing the relative approximate error on a node-by-node basis, as shown in Figure 10 below. Figure 10 illustrates the relative approximate error for all nodes within a 10x10 model domain (with 'Error Threshold' of 0.001).

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10
Row 10	0	0.0157	0.0345	0.0463	0.0558	0.0638	0.0678	0.0554	0.0372	0
Row 9	0	0.0134	0.0269	0.0378	0.0466	0.0532	0.0558	0.05	0.035	0
Row 8	0	0.0113	0.0221	0.0315	0.0393	0.0453	0.048	0.0449	0.0322	0
Row 7	0	0.0096	0.0187	0.0267	0.0338	0.0392	0.0421	0.0404	0.0298	0
Row 6	0	0.0083	0.016	0.023	0.0292	0.0343	0.0375	0.0366	0.0276	0
Row 5	0	0.0072	0.0138	0.0196	0.0251	0.0297	0.033	0.0329	0.0254	0
Row 4	0	0.0061	0.0115	0.0162	0.0208	0.0249	0.0283	0.0291	0.023	0
Row 3	0	0.005	0.009	0.0122	0.0156	0.0192	0.0224	0.024	0.02	0
Row 2	0	0.0037	0.006	0.0069	0.009	0.0117	0.0145	0.0164	0.0161	0
Row 1	0	0.0027	0.003	-0.0012	0	0.0021	0.0035	0.0042	0.0117	0

Figure 10: Relative Approximate Error for all Nodes in a 10x10 Grid

The Gauss-Seidel solution methodology for the MATLAB code dictates that a new estimate will be calculated at the node in Row1/Column2 first, followed by the remaining nodes in Column2, before moving to Row3, then Row4, etc. As a result, errors tend accumulate as the row increases within each column (i.e. $\varepsilon_{Row10} > \varepsilon_{Row1}$), and that errors tend to also accumulate with each successive column (i.e. $\varepsilon_{Column9} > \varepsilon_{Column2}$). It is also notable that the relative error in Row1 for each successive column is smaller than the error in Row10 for the previous column. That's because the estimate for any node in Row1 depends on the estimate for the neighboring cell in Row1. In effect as the solution transitions from one column to the next it truncates the accumulated error. Nevertheless, each successive node in Row1 continues to accumulate the error from its previous neighbor, resulting in an overall increase in error with each successive column.

Overall it can be said that the accuracy of the MATLAB code is reasonably low, and the application can be used with confidence in situations where the conceptual model is valid, and the assumptions/limitations of the MATLAB code are not violated.

8.0 Conclusion

In conclusion, the 2-dimensional groundwater modeling application designed using the MATLAB application editor successfully estimates groundwater elevations based on the fixed conceptual model presented in Figure 1. The solution method is based on the finite difference analysis, using the Gauss-Seidel method to iteratively converge on a solution. There are many limitations implicit to the solution, and there are also many assumptions which may further invalidate the results when compared to real-world observational data. However, when compared to a commercially available groundwater modeling code with the same implicit assumptions (i.e. same conceptual model) and sources of error, the MATLAB code does provide a relatively accurate solution with relative approximate error less than 5% at various levels of grid refinement.

One interesting result of the code is that it accumulates numerical errors (round off error) as the solution progresses through the model domain from one node to the next, resulting in a directional bias in the final estimated groundwater elevation. While this directional bias can not be eliminated within the code as it currently exists, the overall error can be easily reduced by specifying a lower error threshold.

References

Anderson, M. & Woessner, W. (1991). Applied groundwater modeling (pp. 381–381). http://search.proquest.com/docview/17408058/

Frind, E.O. (2003). Fundamentals of Groundwater Modelling (Course Text). Waterloo, ON: University of Waterloo, Department of Earth Sciences.

Harbaugh, A.W. (2005). MODFLOW-2005, the U.S. Geological Survey modular ground-water model—The ground-water flow process: U.S. Geological Survey Techniques and Methods 6 – A16. Retrieved from: http://pubs.er.usgs.gov/publication/tm6A16