ERDC_Documentation

1 Introduction

This documentation will start by examining some of the reasoning and results behind the scripts: 'bv4987_TopoCell.m' and 'bv4987_GWage1D.m', skimming over the functions called. The remaining documentation will briefly describe each function, its importance, and then go into further detail for specific lines of code. As each function comes with a header, the variable definitions, amongst other details, will be skipped, and only some implementation and underlying motivations will be discussed. Note that these problems have been done with the assumptions that all fluid is found in pores, the pores are saturated, and there is no compaction or reactions within the porous matrix.

2 Walkthrough

2.1 bv4987_TopoCell.m

'bv4987_TopoCell.m' was used to generate and analyze the flownet of a specific topographic flow problem whose parameters are defined in the script.

Line 22:

We parameterize the anonymous function 'hb' by initializing after the physical parameters defined in Lines 7-16. Take note that this analytical equation is only used to define the Dirichlet Boundary Condition on the top boundary.

Lines 25 - 27:

After giving minimal information pertaining to the computational domain, we call 'build_grid.m' to construct the rest of our 2D Grid structure.

Lines 30 - 35:

The top boundary is defined to contain the Dirichlet boundary condition, where the Neumann conditions are left empty. By construction, our G operator assigns no flow conditions to all boundaries by default (see 'build_grid.m'). 'g' contains the values of the Dirichlet condition and can be either homogeneous or heterogeneous.

Lines 41 - 44:

We first construct the matrix containing the hydraulic conductivity at the cell centers in the form of the grid for easier implementation. Calling 'comp_mean.m' appropriates the averages of 'K' and converts it into a discrete form, 'K_d', that is defined on the cell faces. This allows for its incorporation into the system matrix, 'L'. The linear operator arises from the discretization of the combination of Darcy's Law and a simplified fluid mass balance equation after the unsteady term is removed resulting in the Poisson Equation given below:

$$-\nabla \cdot (K\nabla h) = f_s \tag{1}$$

Lines 54 - 55:

The flux is separated into its x and y components and reshaped to better represent the physical problem. See implementation of degrees of freedom in 'build_grid.m' for more information.

Lines 61 - 68:

After interpolating both fluxes to the cell centers, we find the stagnation point by minimizing the sum of their absolute values, taking note to eliminate the SE and SW boundaries.

Lines 71 - 74:

The first figure given below was a plot to visualize the gaussian head distribution on the top boundary.

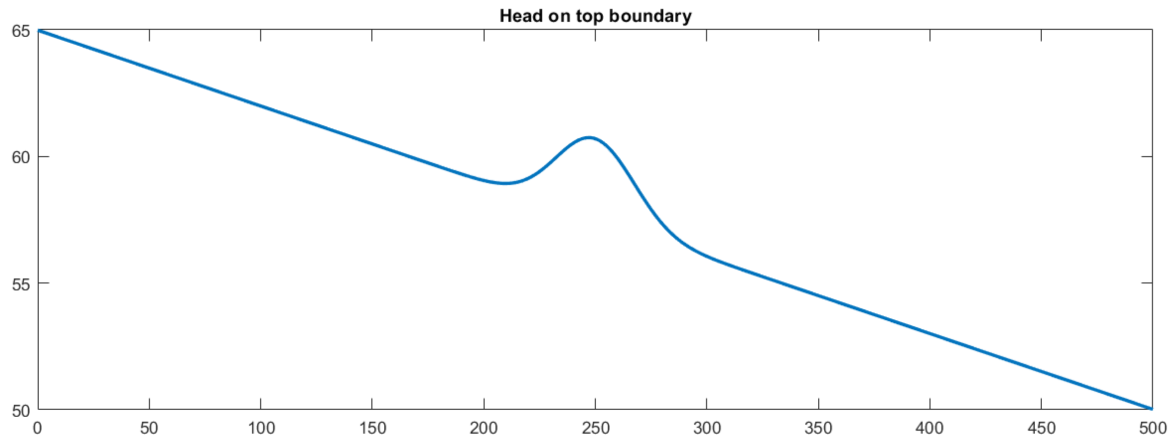

Figure 1: Dirichlet Boundary Condition

Lines 81 - 94:

Here the streamlines where 'q_x' $= 0$ and 'q_y' $= 0$ were plotted along with the flownet. The stagnation point is then represented as a single scatter point. It can be seen that the nullclines agree with the stagnation point from our earlier calculations.

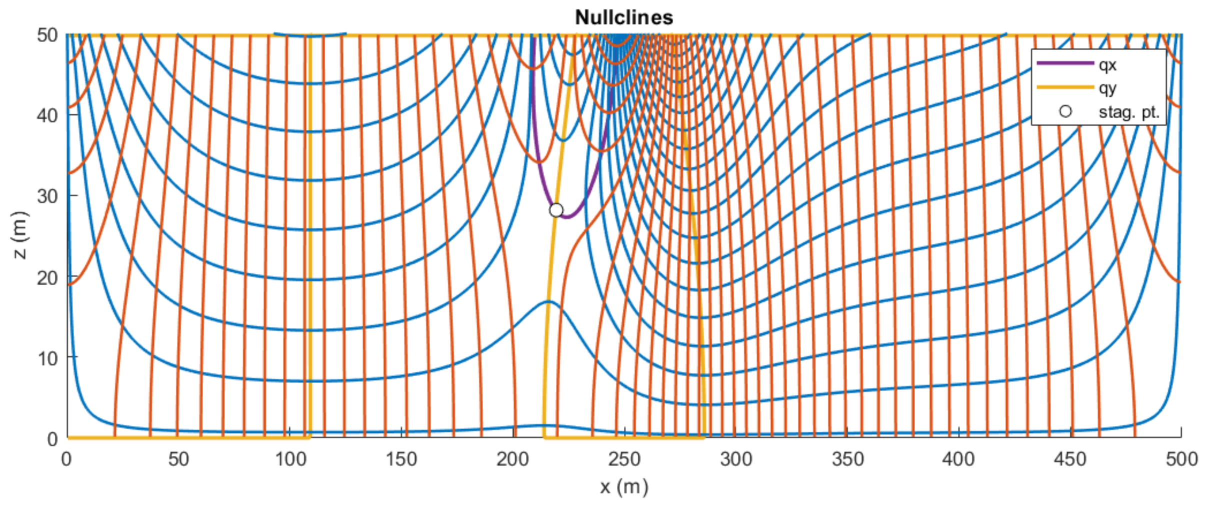

Figure 2: Plot of nullclines

The additional streamlines from ‘q_y‘ can be attributed to some numerical errors.

Lines 97 - 107:

To find the cell boundaries for the flow cells, we take the streamfunction associated with the stagnation point.

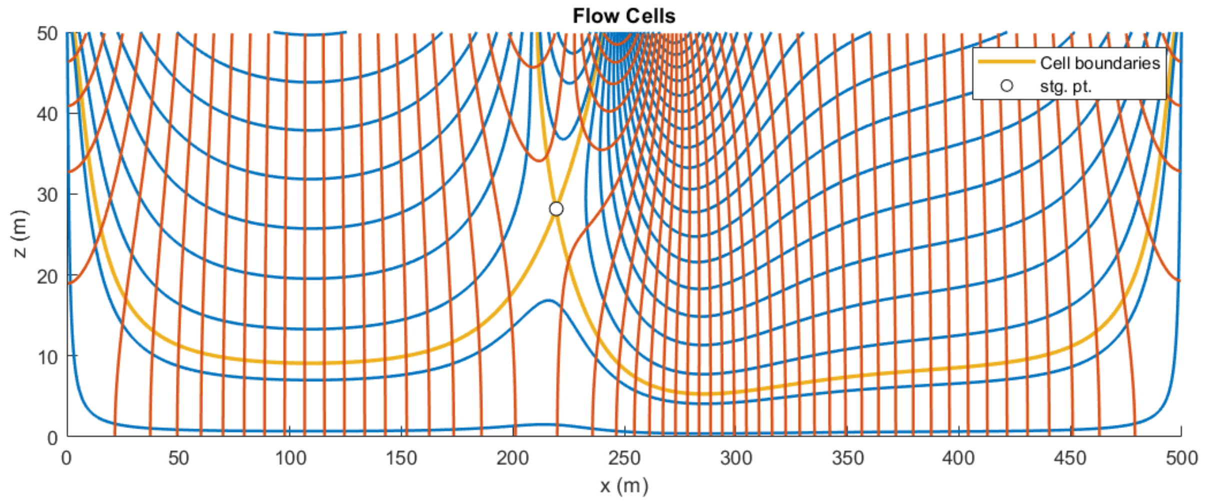

Figure 3: Plot of flow cells

2.2 bv4987_GWage1D.m

This script was written to solve for the advection problem dealing with groundwater age of a one-dimensional aquifer described throughout several sections of the code. Note that sections of this script that are sufficiently similar to ‘bv4987_TopoCell.m‘ will be skipped in this discussion.

Lines 29 - 32:

For this problem, a nonhomogenous Neumann condition was prescribed at the leftmost boundary where the value of the inflow is described by ‘Param.qb‘.

Lines 40 - 41:

Hydraulic conductivity, like porosity in lines 53-54, was defined to be piece-wise from the problem statement.

Lines 56 - 59:

‘ode45.m‘ was utilized to solve the analytical equation modeling groundwater age and implemented in ‘age_ode.m‘. The ode arises from the the original pde:

$$\nabla \cdot (\mathbf{q}a) = \phi \tag{2}$$

For 1D:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x} \cdot (q(x)a(x)) = \phi(x) \\ \frac{\mathrm{d}a}{\mathrm{d}x} q + a\frac{\mathrm{d}q}{\mathrm{d}x} = \phi \end{aligned}$$

Using the simplified fluid mass equation, $\nabla \cdot q = f_s$, we can relate the spatial derivative of our flux to the source term effectively setting it equal to 0.

$$\frac{\mathrm{d}a}{\mathrm{d}x} = \frac{\phi}{q} \tag{3}$$

Line 64:

The linear operator for the advective equation can be drawn from (2).

Line 66 - 67:

To solve for groundwater age, ‘a‘, we need to remove the singularity arising in ‘L2‘ due to the Dirichlet condition acting at the rightmost boundary from the flow problem. ‘solve_lbvp.m‘ can be recycled as it solves the BVP in a reduced space, the boundary matrices, B and N can be reused as well as they already contain information on the problem constraints. As a result, to solve for ‘a‘, only the source term and boundary condition need to be changed. From eqn (2), it can be seen that ‘ϕ‘ is the source. As for the boundary condition, we interpolate the analytical solution to the position on the rightmost computational boundary.

Lines 71- 72:

The numerical and analytical solutions are plotted against each other after some conversions.

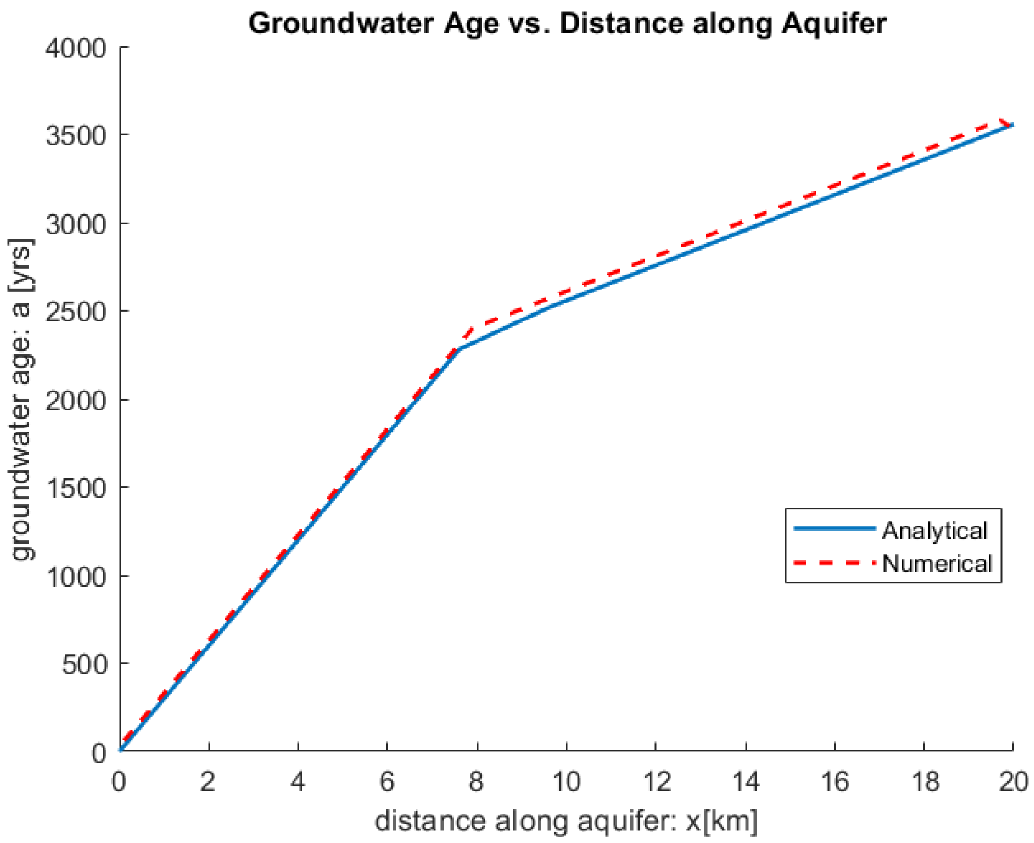

Figure 4: Analytical groundwater age vs. numerical calculation

2.3 build_grid.m

For a majority of this course, we used finite difference methods and implemented several functions to solve specific 1D and 2D flow problems. The first function was 'build_grid.m'. It constructs our computational domain using a staggered grid approach with our hydraulic head specified at the cell centers, and our volumetric flux at the cell faces.

Lines 51 - 85:

These lines are set aside for 1D grids, in our class when dealing with one dimension it was assumed along the x-direction.

Lines 88 - 91:

These fields determines how the solver will compute a given streamfunction. See 'comp_streamfun.m' for more information.

Lines 100 - 106:

Below is an instance of a 2D Grid:

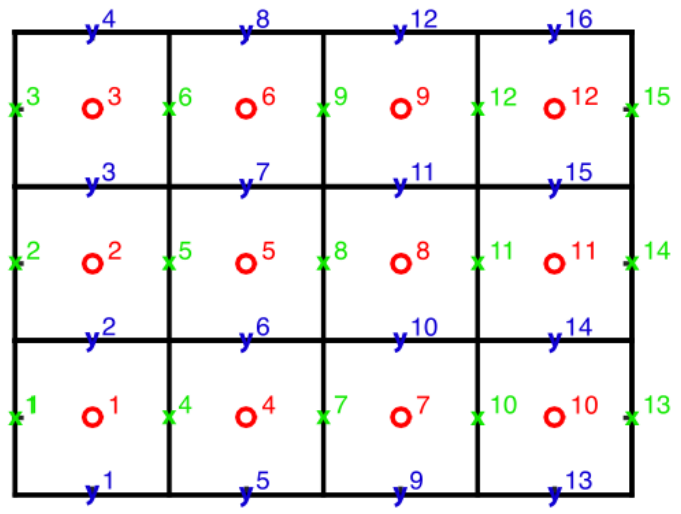

Figure 5: Example 2D Grid Visualization

It can be seen from Figure 5, that the number of x faces is equal to the cell rows times the face columns, and the number of y faces equal the face rows times the cell columns.

Lines 116 - 118:

Each respective cell and face's degrees of freedom were assembled into a grid matching their physical representation, to allow for straightforward implementation of the boundaries.

Lines 129 - 130:

Many of the 2D implementations of the operators and fluxes and heads have the y values "stacked" above the x values, or "x-first". To simulate this the y face degrees of freedom must have the number of x faces added to it to serve as a new baseline.

2.4 build_ops.m

'build_ops.m' discretizes our divergence and gradient operators whose size depends on the grid specifications. Directly discretizing our div-grad operators helps reduce errors by ensuring we conserve energy. The purpose of 'G' is to take us from hydraulic head on the cell centers to fluxes on the cell faces, hence why it has dimensions N by Nf. 'D', on the other hand, takes us from the fluxes on the cell faces to the sources back on the cell centers, and is why it has dimensions Nf by N.

Line 25:

Making use of the adjoint relation between div and grad, we can construct 'G' much quickly and with little trouble.

Line 26:

We implement natural boundary conditions onto 'G' by default. If a nonhomogeneous Neumann condition arises, it is implemented using residuals. See 'comp_flux.m' for more information.

Lines 31 - 40:

From inspection of the divergence matrices for a simple 2D grid, it can be shown that the columns don't "communicate with each other" when dealing with y fluxes. This is due to the linear numbering as seen in Figure 5. The result for 'D_y' is a block diagonal matrix of the 1D divergence matrix case repeated across the main diagonal. For the x fluxes, it is a bit more complicated. From Figure 5 when looking at any row of cells the x faces jump in values equal to the number of rows. Regular grids like Figure 5 with simple and consistent jumps in the dof result in a 'D_x' where the two diagonals in the difference matrix are offset by the number of rows in the grid. This results in a relatively "nice" overall block diagonal matrix. MATLAB's 'kron.m' function allows us to quickly assemble the x and y component from their corresponding 1D 'D' and 'I' matrices.

2.5 build_bnd.m

'build_bnd.m' uses the 'Param' and 'Grid' structures to create a constraint and projection matrix, 'B' and 'N' respectively, using 'I' as the standard basis for the full solution space of the hydraulic head, 'h'. Take note that 'N' isn't a proper projection matrix as it is not square. Additionally, the function makes an effective source term from the Neumann boundary conditions.

Lines 40 - 41:

Here we take advantage of the fact that our computational grid can't tell the difference between a flux and a source term. After multiplying by the surface area of the appropriate cell interface and dividing by the cell volume, we can treat the influx of fluid through any boundary as a source term. Where the prescribed flux is specified by 'Param.qb'.

Lines 44 - 47:

Dirichlet Boundary Conditions are expressed below by:

$$Bh = g \tag{4}$$

Where 'B' is the constraint matrix, 'h' is the hydraulic head, and 'g' is the set of Dirichlet conditions. Since 'I' serves as the standard basis for the full solution space of 'h', we can construct 'B' by pulling the rows of 'I' that correspond to grid boundaries with prescribed heads. To properly construct 'N', (see solve_lbvp), we require it to be an orthonormal nullspace of 'B'. The quickest way to do so is to again pull from 'I', this time removing columns associated with constrained boundaries. This effectively makes 'N' a basis for the reduced solution space of 'h'.

Below is an instance of the three matrices for a given 1D Grid with 5 partitions, and Dirichlet conditions at both ends:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad N = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

2.6 solve_lbvp.m

Despite the output name of ‘h‘, ‘solve_lbvp.m‘ is a somewhat general linear boundary value problem solver. As long as the proper inputs are given, it solves a differential equation in its reduced space removing singularities and improving computational speed.

The governing system of equations that we use to discretize our linear PDE flow problem is given by:

$$Lh = f_s \tag{5}$$

Where ‘L‘ is the system matrix representing any linear partial differential operator.

Since eqn (4) is linear, we can break the unique solution, ‘h‘, into two parts: a particular and homogeneous solution. To solve the BVP, we first convert everything into the reduced system space. By using ‘N‘ we arrive at the relations:

$$h = Nh_r \tag{6}$$

Since ‘N‘ is orthonormal we then get:

$$N^T h = N^T N h_r = I_r h_r = h_r \tag{7}$$

Similarly,

$$f_s = N f_{s,r}, \quad f_{s,r} = N^T f_s \tag{8}$$

Taking these relations and manipulating eqn (5), we can derive a reduced system matrix:

$$\begin{aligned} Lh &= f_s \\ N^T L h &= N^T f_s \\ N^T L N N^T h &= f_{s,r} \\ N^T L N h_r &= f_{s,r} \end{aligned}$$

From inspection, it can be seen that:

$$L_r = N^T L N \tag{9}$$

Where the r subscript denotes an object in the reduced, ”constrained” space.

Line 44:

Note that ‘h_o‘ satisfies eqn (5), while ‘h_p‘ only needs to satisfy eqn (4). To start, we define relations between ‘h_p‘ and its reduced form:

$$h_{p,r} = Bh_p \quad h_p = B^T hp \tag{10}$$

Combining eqns (10) and (4):

$$\begin{aligned} Bh_p &= g \\ B(B^T h_{p,r}) &= g \\ h_{p,r} &= (BB^T)^{-1} g \\ h_p &= B^T (BB^T)^{-1} g \end{aligned}$$

Where ‘BB^T‘ is invertible and allows us to solve a reduced system before matrix multiplying back into the full space.

Line 45:

Since, ‘h_p‘ is known and eqn (5) is linear, we can move it over to the right hand treating it as a source term.

$$\begin{aligned} L(h_o + h_p) &= f_s \\ Lh_o &= f_s - Lh_p \end{aligned}$$

Introducing relations from eqn (9):

$$\begin{aligned} L_r h_{o,r} &= f_r \\ (N^T LN)N^T h_o &= N^T (f_s Lh_p) \\ h_o &= N(N^T LN)^{-1} N^T (f_s - Lh_p) \end{aligned}$$

2.7 comp_flux.m

'comp_flux.m' computes the flux in the interior points through Darcy's Law, reconstructs the boundary fluxes through residuals and determines their direction based on information of the boundaries' degrees of freedom.

Line 37:

From a simplified Darcy's Law given by:

$$q = -K\nabla h \tag{11}$$

From eqn (11), can compute the internal fluxes.

Line 45:

Since our gradient matrix was constructed such that natural Nuemann conditions were the default, we use discrete mass balance on the boundary cells to reconstruct the boundary fluxes. To do so we use the residual.

Lines 48 - 52:

We can reconstruct the flux 'qb' by reversing the reasoning used in build_bnd.m, by noticing that the residual is equal to 'f_n'. Since, a positive 'qb' just indicates an inflow to any specified boundary, we must manipulate the signs to maintain a consistent direction. Note that we the inward normal as the positive sign convention for our fluxes. To implement properly, we check the degrees of freedom of our non-natural Neumann conditions and see if the 'xmin' faces are contained within them. If so they are treated as positive, and the other faces associated to the the 'xmax' faces are treated as negative.

2.8 comp_mean.m

Heterogeneity serves a key role in the behavior of groundwater flow. 'comp_mean.m' takes this into account when dealing with simple layered media. It appropriates means of the hydraulic conductivity from the cell centers to the cell faces allowing us to use them for the fluxes. There are two different mean operations implemented. The arithmetic mean is used for flow along different layers, while harmonic means are called when dealing with flow across layers.

Lines 30 - 32

The means are calculated using the general equation for 2 elements x and y:

$$(\frac{x^p + y^p}{2})^{\frac{1}{p}} \tag{12}$$

Lines 36 - 44:

The means for horizontal and vertical layers are calculated separately then stacked x-first, with anisotropy being encapsulated in the constant, 'kvkh'.

2.9 comp_streamfun.m

'comp_streamfun.m' takes the fluxes specified on cell faces and integrates them along a path to the cell corner of interest computing the streamfunction. In the absence of source/nonzero divergence terms, the value of the integral is path-independent and the starting direction for integration does not matter. In some limiting cases, the errors in a source or sink term can be mitigated by choosing an appropriate integration path. In most cases, a branch cut will arise.

Lines 17 - 27:

The relationship between 'Ψ' and 'q' is given below:

$$\Psi = \int_{\Gamma} q \cdot \hat{n} ds \tag{13}$$

Where Γ is the integration path and s is the arc length variable. Taking advantage of the path independent case, we break the path up into its x and y components:

$$\Psi = \int_{x_A}^{x_B} qydx + \int_{y_A}^{y_B} qxdy \tag{14}$$

The integrals are approximated by multiplying the fluxes at each interface with the face's area and using MATLAB's 'cumsum.m' function to add along each direction's respective path. The streamfunctions are then reshaped to better resemble the actual grid. The location of the origin and path direction is defined in the 'Grid' structure, with the default values being implemented in 'build_grid.m'.

2.10 plot_flownet.m

'plot_flownet.m' allows for a streamlined method to plot flownets.

2.11 flux_upwind.m

'flux_upwind.m' uses a First-Order Upwind scheme to compute an upwind flux matrix, 'A(q)', from the flux vector. In the governing advection-diffusion PDE, 'A(q)' is multiplied with the concentration to get an advective flux vector, 'a'. In this scheme, the advective flux through a given cell face is dependent on the numerical flux through the same cell and the concentration directly upstream. Since concentration, 'c', is defined on the cell centers 'A' has dimensions Nf by N.

Lines 23 - 28:

Since the advective flux is contingent on the upstream concentration, we must check the direction of the flux and construct separate vectors for positive and negative fluxes, 'q_p' and 'q_n', respectively. 'q_p' omits the first flux as there is no concentration defined beyond the left boundary. The advective boundary flux is then handled in the boundary conditions. The same line of reasoning was used for 'q_n' with regards to the rightmost boundary. Lastly, comparing the min and max to 0 allows for the construction of a full vector making it easy to assemble 'A'.

Lines 26 - 28:

A single ‘A‘ matrix is then constructed from the ‘q‘ vectors, and by design, it effectively switches between the two diagonals for the appropriate concentrations when matrix multiplication is done.

2.12 age_ode.m

age_ode.m contains a simple ode with nonconstant coefficients that models the behavior of the 1D aquifer problem handled in ‘bv4987_GWage1D.m‘.