1. USER INSTRUCTIONS: SALTYWATR1.0.0

THE LATEST VERSION OF THE SALTYWATR CODE AND MANUAL CAN BE DOWNLOADED AS FREEWARE FROM WATERSCIENCE RESEARCH COMMUNITY (HTTP://WWW.WATERSCI.ORG).

The program Saltywatr1.0.0 is collection of Matlab routines that implement a numerical solution of the shallow-water flow equations for rivers, estuaries, and the coastal sea. Imagine a two-dimensional planar flow, as in a shallow estuary, with spatial variations in velocity, water height, and salt. The model calculates the above flow variables using the conservation equations for water, salt, and momentum. To simulate flow driven by differences in salt concentration, the momentum equation is coupled to the conservation of salt through the hydrostatic pressure term. Saltywatr is capable of modeling domains of irregular geometry, such as estuaries and rivers. The domain is discretized into quadrilaterals (rectangular shapes with sides that are not perpendicular to each other). Saltywatr does not discretize the domain. Topological data is fed to Saltywatr through input files. At the time of that this manual was written the only non-proprietary mesh generating software that generated quadrilaterals was CUBIT; which is available under research license from Sandia National Laboratory in the US (http://endo.sandia.gov/cubit). For more information on discretizing a domain, please see the Waterscience Research Community website, (http://www.watersci.org) .

These routines are work in progress. It is our short-term goal to port the code to C, so that it may be run natively. It might also be easy to port to code that would run in Scilab. Further, there are many capabilities which are yet to be added. We plan to actively develop this software, so if you are interested in participating or have suggestions

for capabilities that you would like implemented, please join Waterscience (membership implies no obligation of any kind) and send us your communications.

This manual is a brief introduction to the Saltywatr routines; its primary purpose is to describe the functions and variables that are used. For more theoretical information and background on what is being done by the programs, please refer to Brice Loose's Masters' thesis, which is also available from (http://www.watersci.org)

1.1 Examples

This distribution comes with one example; we plan to add more. The example is a calculation of the two-dimensional partial dam break simulation which is presented in many papers and in Chaudry (1993). Enter the directory where Saltywatr is stored and enter the directory /example1. Execute the script example1.m and the function will be calculated and graphed. Open the file and modify it to see other possibilities and to familiarize yourself with the program.

1.2 Input Data

Saltywatr requires that you pass it a .mat file with certain domain geometry. This information must be created outside of Saltywatr; as of yet there are no functions that discretize the domain contained within salty water.

First imagine a quadrilateral with the following conventions. The 4 vertices are denoted [A,B,C,D], moving in a counterclockwise order from the lower right hand corner (Fig. 1.1). Each quad. has four neighbors (adjacent quadrilaterals); these are referred to in relation to the directions of the compass. As can be seen in Figure 1.1, the vertices A and B are shared with the adjacent quad. to the EAST.

NOTE: A note about units. Program should work with any set of consistent units, but we recommend the use of SI units (M, L, T)=(kg, m, s).

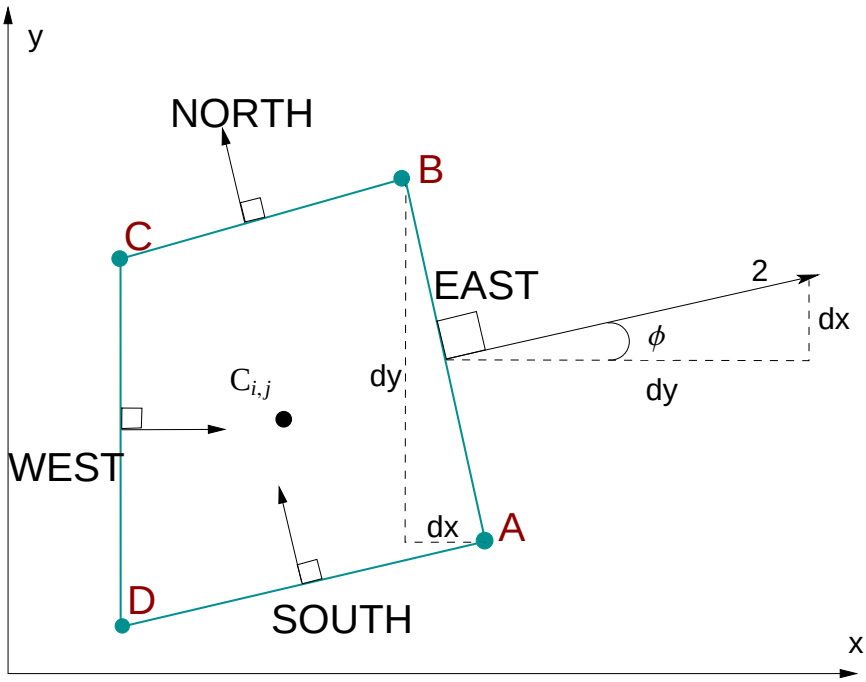

Figure 1.1: Schematic for the naming of sides, vertices, and neighbors in Saltywatr.

The input file, called 'filein', where the actual filename with extension is 'filein.mat' contains the following variables. We define the number of finite volume cells in the domain N. Matlab uses the indexing convention of [rows, columns] or [filas, columnas] or rows × columns to refer to matrices. That is, a [3,2] matrix of ones would be,

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{1.1}$$

We use the same convention

1. *Qx* - A N x 4 matrix containing the x-coordinates for each vertex [A,B,C,D] of the quadrilateral. Listing of quadrilaterals is in no particular order.

2. *Qy* - The same as Qx, but for the y-coordinates of each vertex.

3. *Xc* - A vector of N elements of the x-coordinate for the centroid $C_{i,j}$of each quadrilateral.

4. *Yc* - The same as Xc, but for the y-coordinate.

5. *Zx* - A vector of N elements containing the height (cota) of the bed, over the datum. That is, if the domain has a bedslope in either the x or y direction, this information is contained here.

6. *AreaQ* - A vector of N elements whose values are the area in m^2

7. *ID* - An N x 4 matrix whose row refers to the finite-volume under consideration (i,j) and whose 4 elements are indices to the 4 neighboring finite-volume cells, [E,N,W,S]. This matrix is heavily used; its important to understand its nomenclature. An example of ID is,

$$\begin{array}{cc} & \begin{bmatrix} E & N & W & S \end{bmatrix} \\ \begin{matrix} 2 \\ \vdots \\ N \end{matrix} & \begin{matrix} \rightarrow \\ \\ \rightarrow \end{matrix} \begin{bmatrix} 3 & 25 & 0 & 0 \\ \vdots & \ddots & & \vdots \\ 50 & 42 & 0 & 52 \end{bmatrix} \end{array} \tag{1.2}$$

In this case, we see that row 2, or cell 2 has cell 3 to the East of it, cell 25 to the North of it, and is adjacent to the boundary on the West and South sides. Cells which are along the boundary have a "0" to indicate that there is no neighboring cell on that side. Boundary conditions are implemented using fictitious or *ghost* cells. Later, we will see how the 0's are replaced with information about the ghost cells.

1.3 Description of Functions

1.3.1 Satlywatr.m

This is the primary function that unites all other functions and performs the time-stepping. The first section calculates relevant domain geometry.

The command line call looks like » Saltywatr(flow,fric,filein,fileout); 'filein' is the name of the .mat file containing the domain geometry data described above. 'fileout' is the name of the .mat file that you want to use to save the simulation results. The term 'fric' tells the routine whether the simulation is with/without friction, and 'flow' tell the routine if there are 'inflow' and outflow boundaries and to look for initical condition (ic.m) and boundary condition (bc.m) files. The valid values for fric and flow are 1, or 0. e.g.

» fric = 1 (YES, simulation is with friction). An example would be, if you have an input file name 'resalto.mat' and you want to save the simulation to 'resalto_out.mat', without friction and without inflow or outflow, the call would be:

» Saltywatr(0,0,'resalto','resalto_out'); NOTE: the quotes are necessary.

Within the time-loop, the steps are;

call "predictor.m" to advance solution to time, n+1/2 (half-time step).

extrapolate the variables to cell faces.

implement solid and flow boundary conditions, if any

assemble vectors $\mathbb{F}^{adv}$ and U from primitive variables.

calculate eigenvalues for each cell face

solve Riemann Problem

calculate diffusive flux

calculate friction and slope source terms (if any)

corrector

pass to new time step

1.3.2 predictor.m

Calculates the slope limited gradients and advances the solution to the n+1/2 time step. The operator should not have to call this function directly.

1.3.3 minmod.m

Calculates the minimum modulus of the forward and backward gradients between cells. This is part of slope limiting function.

1.3.4 maxmod.m

Calculates the maximum modulus of the forward and backward gradients between cells. This is part of slope limiting function.

1.3.5 avg2d.m

This is the averaging function which "elects" the limited cell gradients. Calls are made to minmod.m and maxmod.m

1.3.6 ghoster.m

Uses the ID matrix to fill in the boundary nodes with certain values necessary to create ficititous nodes and impose boundary conditions.

1.3.7 ghost2face.m

Implements the boundary conditions for the primitive variables $W = [h, u, v, p]^T$. Routine implements the non-orthogonal boundary condition for velocities and scalar terms.

1.3.8 eddyvisc.m

Calculates the dispersion coefficients D_x and D_y that are used for the salt diffusion.

1.3.9 difus.m

Calculates the salt diffusion in the predictor step (called by predictor.m) by using an upwind scheme.

1.3.10 noslip.m

Implements the second component of the boundary conditions for quadrilaterals, i.e. that the perpendicular velocity $u_\perp$ is equal to zero at the solid boundary.

1.3.11 manning.m

A routine that calculates the water surface height for uniform flow and a given slope, Manning number, discharge, and area.

Bibliography

Chaudry, M. H. (1993), *Open-Channel Flow*, Prentice Hall.