

Guide to Installing and Using FISHES

Kayla C. Lewis

May 22, 2009

1 Purpose

The following sections are intended to constitute a practical guide to installing and using FISHERS. Readers interested in the underlying theory should consult [LL09], [Pat80], and [Lew07].

1.1 Installation

FISHERS is written in FORTRAN 90, and the programs mentioned in this guide for analyzing the output of FISHERS require Matlab[©]¹. The code is compatible with the free FORTRAN 90/95 compilers g95 and gfortran, which can be gotten from <http://www.g95.org> and via the “synaptic” package manager in debian/ubuntu, respectively; it has also been successfully compiled and run using the Intel[©] FORTRAN compiler, ifort. It is recommended that one of these compilers be used, because different FORTRAN compilers have minor differences that can lead to a program compiling on one of them but not another. There are g95 binaries at the above-mentioned website for linux, solaris, windows, etc, and one can even compile the compiler itself from source code if one wishes – see the website for details. In linux, g95 uses the GNU C compiler gcc, and if there are any problems getting g95 to work, it is most likely because one is lacking the correct version of gcc. Regarding Matlab, it is recommended that Matlab 7 or later be used.

Once Matlab and the FORTRAN compiler have been installed, it is time to install FISHERS and the auxiliary programs that come with it. Simply unzip the file fishes.zip in your chosen directory. After you have unzipped the file, it may be necessary to change the user permissions on the resulting files and directories to allow the user to copy, edit, or read files (for unix/linux, use the chmod command).

Before running FISHERS (or any of the auxiliary programs that come with it) for the first time, it is necessary to compile it for the particular machine on which it will run - that is why g95 or ifort is needed. For unix/linux users

¹We plan to switch to python plotting utilities in the near future.

(I will assume that the users are employing linux or unix hereafter), there is a script in the FISHERS directory called “FishComp”, and it will compile FISHERS with the proper parameters, rename the compiled file from “a.out” to “fishes”, and move the result to the run directory, which is where FISHERS must execute, because all the input files and data files for the equations of state are located there. Edit the script and uncomment the lines appropriate for the compiler you plan to use. The script can be executed by typing `./FishComp` from the FISHERS directory. After the file “fishes” has been created in the run directory, one may need to go there and type `chmod 711 fishes`, to apply the right file permissions. Subsequently, to run FISHERS itself, one can simply type `./fishes` from the run directory.

2 User Input Files

When FISHERS runs, it reads the information in the files `InFile0` and `InFile1` to determine the problem geometry, the initial and boundary conditions, and other simulation controlling variables like the time step, iteration tolerances, etc. These user-defined variables will now be described, and they will each be printed in bold to facilitate ease of reference.

2.1 InFile0

A simple example of an `InFile0` is in the `FISHERS/run/` directory (there is also a backup of this file, `InFile0.example`, in case the user would like to review this material later after having replaced the original example files). This file can be read with one’s favorite text editor (the authors choice is vim, with syntax highlighting turned on). Regarding units in these files, SI units are used exclusively, except that salinity is in wt% NaCl and not wt fraction NaCl (although internally FISHERS converts the input salinities from wt% to wt fractions).

The first variables in the file are **NumNodesX**, **NumNodesY**, and **NumNodesZ** - these are the number of nodes in the X (west-east), Y (north-

south), and Z (up-down) directions. FISHES is currently only designed to handle up to two dimensions, so that Y should always be set to 1. In this example, there are 100 nodes total, with 10 nodes from west to east and 10 nodes from top to bottom. The variable **SuppressScreenMessages** should be set to Y if one wishes to run FISHES as a background process and does not want FISHES to send output to the screen while running; otherwise, this variable should be set to N.

The next line starts with the variable **TimeStepIncr**. This variable is for situations such that the user wants the time step to increase with time as the code runs – sometimes many changes happen quickly at once when a simulation starts, and the time step should be set comparatively small during these times, while later it may be possible to obtain meaningful results with a higher time step. For example, setting **TimeStepIncr** to 2 will double the time step size after each time marching step completes. Setting **TimeStepIncr** to 1 keeps the time step the same (i.e., multiplies it by 1 after each time marching step), and this value is the standard one. The code will not allow the time step to increase to more than ten times its initial value² due to application of **TimeStepIncr**. **Dispersivity** and **CorrelationLength** are there for modeling mechanical salt dispersion, but currently the code has numerical dispersion in the salt transport large enough such that physical dispersion should not be added, and the **Dispersivity** should be set to zero³. These variables are there in case the author decides to increase the accuracy of salt transport in the future.

MassFluxGradual should be set to N unless one wants to have the mass flux at the bottom of the system increase gradually with time. If this variable is set to Y, then the next three variables must also be specified, namely, the starting mass flux, the ending mass flux (both in $\text{kg}/\text{m}^2\text{s}$), and the interval of time over which this increase should take place (in seconds)⁴.

²That is, the time step corresponding to the latest user defined interval, see **TotalTime**, **PrintFreq**, and **Delta.t** below.

³When the **Dispersivity** is set to zero, the value of **CorrelationLength** is immaterial.

⁴If **MassFluxGradual** is set to N, the values of the next three variables are immaterial.

TempRelax, **PresRelax**, and **SaliRelax** are for if one wants to use under-relaxation on the temperature, pressure, or salinity during simulations. Under-relaxation slows the rate at which variables evolve with time by requiring the new values to be closer to the old values. Setting **TempRelax** to 0, for instance, would keep the temperature from changing with time at all, while setting it to 1 would cause the temperature to evolve normally. Numbers between 0 and 1 will have intermediate effects. An important thing to keep in mind is that, because under-relaxation causes variables to evolve more slowly in time than they otherwise would, it will be necessary to increase the **TimeCalibration** variable above 1 in there is a significant degree of under-relaxation. **TimeCalibration** should always be set to a whole integer value, and the number to which it is set is the number of extra times FISHES will solve the temperature, pressure, and salinity equations before letting simulation time pass. If one sets **TempRelax**, **PressRelax**, and **SaliRelax** all to 1.D-1, then setting **TimeCalibration** to about 10 will produce results that evolve at roughly the same overall rate that they would without under-relaxation. No under-relaxation is required for most cases.

The variable **TimeStepCheck** determines whether the code automatically checks the appropriateness of the time step size. If the velocity of a fluid phase somewhere in the system is high enough that the fluid traverses more than one control volume during a single time step, the code will both produce a warning and divide the time step by two. The code take similar actions if the line-by-line matrix solver cannot reach a convergent solution for the pressure or the temperature⁵.

RockDens, **CpRock**, **RockThermCond**, and **Grav** are the variables that store the rock density, rock specific heat at constant pressure, rock thermal conductivity, and gravitational acceleration, respectively. **HaliteDens**, **CpHalite**, **FluidThermCond**, and **SaltChemDiff** store the halite (solid salt) density, halite specific heat, fluid thermal conductivity, and salt chemical diffusivity, respectively. **CpHalite** is not currently used for anything in the code it is there for

⁵The system of equations governing salt transport does not need to be solved iteratively (see [LL09]).

if the author decides to include the formation of halite in a more rigorous way in the future. **XNodeWidths**, **YNodeWidths**, and **ZNodeWidths** store the node widths in the X, Y, and Z directions. The C stands for “Constant” and means that the number below it will be the value for every node in the corresponding direction. For instance, in this InFile0, all the nodes in the X direction are set to a width of 10 meter, while all those in the Z direction are set to widths of 10 meters. Alternatively, F stands for Free and when there is an F, all the numbers listed below are read into individual nodes. For example, suppose that the X widths line were as follows: Node widths in the x direction(m)

```
F
0.1
1.0
1.0
```

In this case, the western-most nodes would have west-east widths of 0.1 meters, while next two columns would have west-east widths of 1.0 meter.

Porosity and **Permeability**, **Temperature**, **Pressure**, and **Salinity** all store the initial values of the variables after which they are named (Salinity refers to the bulk salinity). Again, a C means that all the nodes will be set to the constant value of the number immediately below C. For instance, all the nodes have been set to have a porosity of 0.1 (or 10%). The pressures, on the other hand, are set using the F descriptor, and the pressure values are listed afterward corresponding to the nodes in which they will be stored. The last variables in InFile0 allow the user to specify different print frequencies and time step sizes for different intervals of simulation time. The number under “Number of time periods below to execute before stopping” refers to the number of such predefined intervals the user desires to employ. In this example, there is only one such period. The variables **TotalTime**, **PrintFreq**, and **Delta.t** refer to the simulation time interval, print frequency, and time step size, respectively. In this example, the simulation will run for 1.61E7 s (~ 0.5 yrs), printing output every 3.2E6 s (~ 0.1 yr), with a time step size of 1.E5 s.

2.2 InFile1

InFile1 contains information specifying the types of boundary conditions and the boundary values for solving the temperature, pressure, and bulk salinity equations. FISHERS recognizes four types of nodes: constant boundary, flux boundary, upwind boundary, and interior nodes. A constant boundary node is a node that is both on the boundary of the system and that is to be held at a fixed value for the entire simulation, and such node types are designated with a C. A flux boundary node is a node that is both on the boundary and that specifies a constant flux for the entire simulation. It is signified by the letter F. Upwind boundary nodes are nodes on the boundary of the system whose values are determined from their initial values plus the upwind condition. This condition is symbolized with a U, and will be explained more fully below. Finally, an interior node, designated by I, is simply any node that is not on a boundary.

PresNodeTypes is the variable storing the node types with respect to pressure. In this example, the pressures at the top of the system are held constant, while all the other boundary nodes are set at constant mass fluxes (note that the mass fluxes have units of $\text{kg}/\text{m}^2\text{s}$). The interior nodes are marked I. **TempNodeTypes** stores the node types with respect to temperature. In this example, the nodes at the top of the system have all been set to the upwind boundary condition, while the side boundary nodes are set to constant heat fluxes ($\text{J}/\text{m}^2\text{s}$), and the bottom nodes are set to constant temperatures. **SalinNodeTypes** stores the node types with respect to salinity, and these are set similarly to those for the temperature, except that the bottom nodes are set at constant salt mass fluxes instead of constant salinity values. Now that the node types have been set, the specific numerical values for these node types must be recorded. **PresBoundVals** stores the pressure boundary values. Interior nodes are marked 0 simply to reflect again that they are not on the boundary (so these nodes are not set to initial pressures of 0 – the initial pressures for the internal nodes were specified in InFile0 already). The top pressures in this example are held fixed at 260.D5 Pa (260 bars), while the sides and bottom of the system

are set to zero mass flux. **TempBoundVals** stores the temperature boundary values. The temperatures at the top of the system have been set to initial values of 100°C (consistent with the initial values set in InFile0); however, if fluid flows from inside the system out of the top, the temperature at the top node will change to the temperature of the node just below it. For instance, if the node immediately below the upstream condition node were, say, 105°C, then fluid flowing upward out of the system would cause the temperature in the top node to change from 100°C to 105°C. The upstream condition is meant to represent the fact that changes upstream in a moving fluid influence what happens downstream. If fluid flows from the top of the system downward, then the top node will shift back to its initial value of 100°C. The bottom temperatures have been set to a constant 200°C, while the sides of the system are thermally insulated (heat fluxes = 0). **SaliBoundVals** stores the salinity boundary values. The top nodes are set at initial values of 3.2 wt% NaCl, with the upstream condition governing later values. The sides and the bottom of the system are set at zero salt flux.

3 The Output Files

There are five output files to which FISHES writes as it runs, numbered from 0 to 4. In them are stored the initial problem setup information, the bulk fluid properties, the fluid velocities, the individual phase properties, and the surface heat fluxes, respectively. A detailed description of each follows, and the example OutFiles provided in the run directory are those that result from running FISHES with the example InFiles.

3.1 OutFile0

This output file is a summary of all the user-defined variables that have been read into memory for the simulation. If one desires to check that the input files were read in properly with the appropriate values, then this file is the one to

check. Everything in this file is labeled fairly obviously; however, below some variables, like the porosity for instance, the phrase `Slice: 1` will be seen. This phrase is just to indicate that the Y dimension is set to 1 if the code were set up to handle three dimensions, then there would be more than one slice of X-Z values for each variable.

3.2 OutFile1

This file stores the phase indices, liquid volume saturation, bulk densities, pressures, temperatures, bulk salinities, and bulk enthalpies generated during the simulation. Again, one can ignore the `Slice: 1` comments. Note that the phase indices are labeled numerically as

- 2 = liquid + halite
- 1 = pure liquid
- 0 = two-phase (liquid + vapor) mixture
- 1 = pure vapor
- 2 = vapor + halite

The code uses these numbers internally to identify which phase is present at each node.

3.3 OutFile2

In addition to the velocities being vectors, they are stored at the interfaces between nodes, which makes reading the velocity data file slightly more complicated than reading the others. Velocities are listed in `OutFile2` for the liquid and vapor phases separately, and for the X and Z directions separately. In our example setup, there are three nodes in the X direction, and so there are two interfaces in the X direction between nodes. In `OutFile2`, the phrase `X direction interface slice: 1` indicates that what follows is the velocity at the first interface in the X direction. Similarly, `X direction interface slice: 2` means that what follows is the velocity at the second interface in the X direction. Because there are four nodes in the Z direction, there are three interfaces between nodes in

this direction. Hence, there are three slices each for each velocity component in that direction, and for each phase.

3.4 OutFile3

OutFile3 is arranged like OutFile1, except that it stores the individual vapor and liquid salinities, enthalpies, and densities instead of the corresponding bulk properties.

3.5 OutFile4

OutFile4 contains information about the surface heat fluxes – this file is a very recent addition and the module that generates it needs more work. Hence, its contents should only be used if you know what you’re doing, i.e., you understand what the code is doing to get these numbers.

4 Auxiliary Programs

There are five auxiliary programs that will make creating the input files and reading/plotting the results from the output files much easier than otherwise, especially for problem setups that require a large number of nodes. They are described below.

4.1 InHelp

This program is located in the InHelper folder under the run directory. Before one can run InHelp, the first time it must be compiled. To compile it using g95, for instance, type `g95 *.f95` from the InHelper directory, and then type `mv a.out InHelp`. Finally one may need to type `chmod 711 InHelp` to apply the proper permissions to this file. One can then run it in unix/linux by typing `./InHelp` from that directory. Upon running this program, a menu will appear followed by a prompt for the menu selection. The example InFile0 and InFile1 are already in this directory, and these are the files InHelp reads when it is run.

If the user changes, for example, the permeabilities, and then asks InHelp to save, the result will be files just like the example InFile0 and InFile1 except with altered permeability information. When the program writes output, it writes the output to the files InFile0.new and InFile1.new so that if a mistake has been made the original InFile0 and InFile1 will not be affected. By experimenting with the options in the menu and viewing the output, it should become clear what functions most of these options perform. Here is an example. Suppose the user selects option 3, to modify the permeabilities. A new menu will then appear, with the options “1 select starting and ending rows/columns” and “2 return to main menu”. Selecting 1 leads to the prompt “starting row:”, and after specifying the starting row, one is asked to specify the ending row, the starting column, and the ending column. Essentially, a square of nodes is being specified, which will then be filled in with values determined by the user. Suppose for starting row, ending row, starting column, and ending column one has entered 1, 3, 2, 3, respectively. Then the region of space selected will look like the following (where o stands for a node, and x stands for a selected node):

```

o x x
o x x
o x x
o o o

```

After selecting the nodes that will have altered permeabilities, the user is then given the choices constant (C), linear left to right (L), or linear top to bottom (T). Choosing C will set all the selected node permeabilities to one constant value. Choosing L will cause the permeabilities to vary linearly with distance from west to east, and choosing T will cause them to vary linearly with distance from the top toward the bottom of the system.

4.2 analyze

After FISHES has been run, or even while it is still running (as long as there has been some output to the output files), one can run a script in the run directory

called analyze by typing `./analyze`. This script will copy all the input and output files into the Analysis directory, and run a FORTRAN program called MatGen there, which will convert the FISHES output into matrices that can be read by Matlab (make sure to compile MatGen and rename the a.out file to MatGen before running analyze the first time). The resulting files will have the extension `.dat` and be named according to which values are stored in them. For example, the temperatures from OutFile1 will be stored in a data file called `Tempr.dat`.

4.3 Plot0

Once the FISHES output files have been converted to files readable by Matlab, there is a plot program in the Analysis directory called `Plot0.m`. This program is a Matlab program and hence must be run from Matlab; it will plot all the output from FISHES so far, determining automatically whether the output is for 1D or 2D results, and plotting the results in the appropriate format. It can be executed by typing `Plot0(0)` from the Matlab prompt. For one dimensional plots, one can overlay output from several different times by using `Plot(1)` for the first plot, executing analyze again, and using `Plot(0)` thereafter.

4.4 surfsamp

To use `surfsamp`, first copy the input and output files into the `surfsamp` directory. Then run the FORTRAN program `surfsamp`, which will create files that are readable by Matlab, with the surface salinities and temperatures above a given column of nodes together with their output times. After running this program, typing `SurfSample` from Matlab will produce plots of the surface vent salinities and temperatures vs. output times. A horizontal line on the salinity plot refers to equivalent normal seawater salinity (3.2 wt% NaCl).

4.5 pickup

Sometimes it is necessary to take the last output from one simulation and use it as the starting input for the next simulation. The FORTRAN program `pickup` in

the run directory will do this task automatically. It simply reads the last output from the output files and re-writes InFile0 with these values as the initial values. If the user has any questions about how to use this program or any of the others mentioned in this user file, he/she is encouraged to contact the author via email.

References

- [Lew07] K.C. Lewis. *Numerical modeling of two-phase flow in the sodium chloride-water system with applications to seafloor hydrothermal systems*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, 2007.
- [LL09] K.C. Lewis and R.P. Lowell. Numerical modeling of two-phase flow in the NaCl-H₂O system: Introduction of a numerical method and benchmarking. *Journal of Geophysical Research*, 114:B05202, 2009.
- [Pat80] S.V. Patankar. *Numerical Heat Transfer and Fluid Flow (series in computational methods in mechanics and thermal sciences)*. Taylor & Francis Publishers, 1980.