ParFlow User's Manual

Software Revision: 3.0
2001
UCRL-MA-123204

Steven Ashby, Chuck Baldwin, Bill Bosl, Robert Falgout, Richard Hornung, Steven Smith, Carol Woodward

Contents

Chapter 1

Introduction

ParFlow is a groundwater flow and contaminant transport simulation code designed to run efficiently in a multi-processor computing environment on a variety of available machines. This simulator is being built by scientists from the Center for Applied Scientific Computing (CASC), Environmental Programs, and the Environmental Protection Department. ParFlow is able to utilize the computing power that is available on today's supercomputers, enabling it to more accurately model the effects of subsurface heterogeneity on fluid transport processes. Heterogeneity is present on many scales in the subsurface, ranging from the microscopic capillary processes to macroscopic variations in structure, such as fractures and faults. ParFlow attempts to account for heterogeneity and model its effects on contaminant flow.

This manual describes how to use ParFlow, and is intended for geoscientists and environmental scientists and engineers. In Chapter 2, we describe how to install ParFlow. Then, we lead the user through a simple ParFlow run. In Chapter 3, we describe the ParFlow system in more detail. Chapter 5 describes the formats of the various files used by ParFlow.

Chapter 2

Getting Started

This chapter is an introduction to setting up and running ParFlow. In § 2.1, we describe how to install ParFlow. In § 2.2, we lead the user through a simple groundwater problem, supplied with the ParFlow distribution.

2.1 Installing ParFlow

ParFlow requires a Standard ANSI C and Fortran 77 compiler (or f2c) to build code. ParFlow also requires `TCL/TK` version 8.0 (or higher). `TCL/TK` can be obtained from:
`http://www.sunlabs.com/research/tcl/`.

The following steps are designed to take you through the steps of installing ParFlow from a source distribution. This assumes that a configuration file exists for your system. If one does not exist you will need to create a configuration file for your system in the `config` directory of the ParFlow source tree.

1. The following environment variables should be set up in your `.profile` file (if you are using `bash` or a bourne syntax shell):

```
export PARFLOW_SRC=~/parflow/src
export PARFLOW_DIR=~/parflow/exe
export PARFLOW_HELP=~/parflow/docs
export PARFLOW_HTML_VIEWER=/usr/local/bin/netscape
```

   If you are using a `csh` like shell you will need the following in your `.cshrc` file:

```
setenv PARFLOW_SRC ~/parflow/src
setenv PARFLOW_DIR ~/parflow/exe
setenv PARFLOW_HELP ~/parflow/docs
setenv PARFLOW_HTML_VIEWER /usr/local/bin/netscape
```

The PARFLOW_HTML_VIEWER variable should be set to a HTML browser of your choice. This is used to choose the browser that will be used to view the online documentation. The other variables point to locations of directory structures used by ParFlow. The variable PARFLOW_DIR specifies the location of the installed version of ParFlow This is where executables and support files will be placed. If you have a directory which is shared on multiple architectures you can set different PARFLOW_DIRs on the different machines (for example ~/parflow/exe.t3d and ~/parflow/exe.irix). PARFLOW_SRC is the location of the source code for ParFlow and affiliated tools. PARFLOW_HELP is the location of the HTML help files. We will use the ~/parflow directory as the root directory for building ParFlow in this user manual; you can use a different directory if you wish.

2. You should also add $PARFLOW_DIR/bin to your PATH environment variable. If you are using bash or a bourne syntax shell):

   ```
   PATH=$PATH:$PARFLOW_DIR/bin
   ```

   If you are using a csh like shell:

   ```
   set path=($path $PARFLOW_DIR/bin)
   ```

3. Extract the source files from the distribution compressed tar file. This example assumes the parflow.tar.Z file is in your home directory and you are building it in a directory /parflow.

   ```
   mkdir ~/parflow
   cd ~/parflow
   zcat ../parflow.tar.Z | tar xf -
   ```

4. Next, we need to build the parflow and other tools which make up the ParFlow suite.

   ```
   cd $PARFLOW_SRC
   ./build install
   ./build install docs
   ```

   The first command will build ParFlow and the bundled tools and install them in the $PARFLOW_DIR directory. The second command will build and install the documentation.

   After the tools are built, be sure to execute the Unix rehash function if you are using a csh like shell.

   ```
   rehash
   ```

2.2 Running the Sample Problem

Here, we assume that ParFlow is already built. The following steps will allow you to run a simple test problem supplied with the distribution.

1. We first create a directory in which to run the problem, then copy into it some supplied default input files. So, do the following anywhere in your `$HOME` directory:

   ```
   mkdir foo
   cd foo
   cp $PARFLOW_DIR/test/default_single.pftcl .
   chmod 640 *
   ```

   We used the directory name `foo` above; you may use any name you wish. The last line changes the permissions of the files so that you may write to them.

2. Run ParFlow using the pftcl file as a TCL script

   ```
   tclsh default_single.pftcl
   ```

You have now successfully run a simple ParFlow problem. For more information on running ParFlow, see § 3.2.

Visualizing the Permeability

Now suppose we want to visualize some of the output. In particular, let's look at the permeability field generated by ParFlow in this run. Follow these steps:

1. Start avs using the `pfavs` command. This will start AVS (the visualization system we are currently using), load the ParFlow module library, and set the default network directory to your installed ParFlow network directory.

2. Click on `Network Editor`. Then from the `AVS Network Editor`, click on `Read Network`, and select file `slice3.net`. This will read in the `slice3.net` network which we will use to look at 3 slice planes of the permeability data.

3. The panel buttons `read parflow macro` and `read parflow` should already be selected (highlighted). From the `File Browser`, select `default_single.out.perm.pfb`. A picture of the permeability field used in the problem should appear in the `Geometry Viewer` window.

For more information on visualizing ParFlow output, see § 3.4.

Adding a Pumping Well

Let us change the input problem by adding a pumping well:

1. Edit the file `default_single.pftcl` using your favorite text editor.

2. Add the following lines to the input file near where the existing well information is in the input file. You need to replace the "Wells.Names" line with the one included here to get both wells activated (this value lists the names of the wells):

```
pfset Wells.Names {snoopy new_well}

pfset Wells.new_well.InputType               Recirc

pfset Wells.new_well.Cycle     constant

pfset Wells.new_well.ExtractionType     Flux
pfset Wells.new_well.InjectionType            Flux

pfset Wells.new_well.X     10.0
pfset Wells.new_well.Y     10.0
pfset Wells.new_well.ExtractionZLower      5.0
pfset Wells.new_well.ExtractionZUpper      5.0
pfset Wells.new_well.InjectionZLower      2.0
pfset Wells.new_well.InjectionZUpper      2.0

pfset Wells.new_well.ExtractionMethod     Standard
pfset Wells.new_well.InjectionMethod          Standard

pfset Wells.new_well.alltime.Extraction.Flux.water.Value              5.0
pfset Wells.new_well.alltime.Injection.Flux.water.Value      7.5
pfset Wells.new_well.alltime.Injection.Concentration.water.tce.Fraction 0.1
```

For more information on defining the problem, see § 3.1.

Converting the Pressure Output

Next, let us convert the pressure output computed by ParFlow to hydraulic head. Do the following:

1. Start the `xpftools` utility. This is a utility which is used to do file conversions and other simple operations on ParFlow output files.

2. To read in the data, use the `Data` pulldown menu and select the `Load ParFlow Binary` option.

3. In the file browser select the `default_single.out.press.00000.pfb` file.

4. Select the `Compute Hydraulic Head` option from the `Functions` pulldown menu.

5. In the `Compute Hydraulic Head` popup, us the selector error next to the `Pressure Head` field to select `dataset0` (the `default_single.out.press.00000.pfb` file).

6. Press the `Compute Function` button at the bottom of the panel. This creates a new dataset labeled `dataset1`.

7. To save the hydraulic head data in a ParFlow file, `Close` the `Hydraulic Head` panel and select `Save ParFlow Data` from the `Dataset` pulldown menu.

8. In the file selector box type the filename `default_single.out.hhead.00000.pfb` and save the dataset.

   You have now computed and created a new ParFlow binary file with the hydraulic head.

For more information on manipulating and analyzing ParFlow data, see § 3.6.

Chapter 3

The ParFlow System

The ParFlow system is still evolving, but at present, it has three basic components. GMS is used to create geometry input, ParFlow is the simulator itself, and AVS (or AVS/Express) are used to visualize the results. We discuss how to define the problem in § 3.1, how to run ParFlow in § 3.2, and how to visualize the results in § 3.4. There is also a utility providing a set of functions for manipulating ParFlow data. This utility is discussed in § 3.6. Lastly, § 3.7 discusses the contents of a directory of test problems provided with ParFlow.

3.1 Defining the Problem

Defining the problem may involve several steps. One of these steps requires the use of GMS (Groundwater Modeling System) [9, 10] to define complicated geometries such as hydrostratigraphic layers. These geometries are then converted to the `.pfsol` file format (§ 5.4) using the appropriate PFTools conversion utility (§ 3.6).

The "main" ParFlow input file is the `.pftcl` file. This input file is a TCL script with some special routines to create a database which is used as the input for ParFlow. See § 5.1 for details on the format of this file. The input values into ParFlow are defined by a key/value pair. For each key you provide the associated value using the `pfset` command inside the input script. To set the computational grid for the problem you would enter:

```
#-----------------------------------------------------------------------------
# Computational Grid
#-----------------------------------------------------------------------------
pfset ComputationalGrid.Lower.X              -10.0
pfset ComputationalGrid.Lower.Y               10.0
pfset ComputationalGrid.Lower.Z                1.0

pfset ComputationalGrid.DX                     8.8888888888888893
pfset ComputationalGrid.DY                    10.666666666666666
pfset ComputationalGrid.DZ                     1.0
pfset ComputationalGrid.NX                      18
pfset ComputationalGrid.NY                      15
pfset ComputationalGrid.NZ                       8
```

The value is normally a single string, double, or integer. In some cases, in particular for a list of names, you need to supply a space seperated sequence. This can be done using either a double quote or bracies.

```
pfset Geom.domain.Patches "left right front back bottom top"

pfset Geom.domain.Patches {left right front back bottom top}
```

For commands longer than a single line, the TCL continuation character can be used,

```
pfset Geom.domain.Patches "very_long_name_1 very_long_name_2 very_long_name_3 \
                           very_long_name_4 very_long_name_5 very_long_name_6"
```

Since the input file is a TCL script you can use any feature of TCL to define the problem. This manual will make no effort to teach TCL so refer to one of the available TCL manuals for more information ("Practical Programming in TCL and TK" by Brent Welch [11] is a good starting point). This is NOT required, you can get along fine without understanding TCL/TK.

Looking at the example programs in the `test` directory is one of the best ways to understand what a ParFlow input file looks like. See § 3.7.

3.2 Running ParFlow

Once the problem input is defined, you need to add a few things to the script to make it execute ParFlow. First you need to add the TCL commands to load the ParFlow command package.

```
#
# Import the ParFlow TCL package
#
lappend auto_path $env(PARFLOW_DIR)/bin
package require parflow
namespace import Parflow::*
```

This loads the `pfset` and other ParFlow commands into the TCL shell.

Since this is a script you need to actually run ParFlow. These are normally the last lines of the input script.

```
#-----------------------------------------------------------------------------
# Run and Unload the ParFlow output files
```

```
#-----------------------------------------------------------------------------
pfrun default_single
pfundist default_single
```

The `pfrun` command runs ParFlow with the database as it exists at that point in the file. The argument is the name to give to the output files (which will normally be the same as the name of the script). Advanced users can set up multiple problems within the input script by using different output names.

The `pfundist` command takes the output files from the ParFlow run and undistributes them. ParFlow uses a virtual file system which allows files to be distributed across the processors. The `pfundist` takes these files and collapses them into a single file. On some machines if you don't do the `pfundist` you will see many files after the run. Each of these contains the output from a single node; before attempting using them you should undistribute them.

Since the input file is a TCL script run it using TCL:

```
tclsh runname.pftcl
```

NOTE: Make sure you are using TCL 8.0 or later. The script will not work with earlier releases.

For a network of workstations, you need to specify the computer systems that are going to be used in the virtual machine (VM). To add a machine to the VM you should set the `Process.HostNames` variable to be a space separated list of the machine names.

```
pfset Process.Hostnames ``bert.llnl.gov ernie.llnl.gov bigbird.llnl.gov''
```

One output file of particular interest is the `<run name>.out.log` file. This file contains information about the run such as number of processes used, convergence history of algorithms, timings and MFLOP rates.

3.3 Restarting a Run

Occasionally a ParFlow run may need to be restarted because either a system time limit has been reached and ParFlow has been prematurely terminated or the user specifically sets up a problem to run in segments. In order to restart a run the user needs to know the conditions under which ParFlow stopped. If ParFlow was prematurely terminated then the user must examine the output files from the last "timed dump" to see if they are complete. If not then those data files should be discarded and the output files from the next to last "timed dump" will be used in the restarting procedure. As an important note, if any set of "timed dump" files are deleted remember to also delete corresponding lines in the well output file or recombining the well output files from the individual segments afterwards will be difficult. It is not necessary to delete lines from the log file as you will only be noting information from it. To summarize, make sure all the important output data files are complete, accurate and consistent with each other.

Given a set of complete, consistent output files - to restart a run follow this procedure :

1. Note the important information for restarting :

- Write down the dump sequence number for the last collection of "timed dump" data.
- Examine the log file to find out what real time that "timed dump" data was written out at and write it down.

2. Prepare input data files from output data files :
   - Take all the concentration output files with the sequence number from above and convert them from a ParFlow Scattered Binary File § 5.3 to a ParFlow Binary File § 5.2 (including a ".dist" file) using PFTools § 3.6 and the *pfdist* utility in the input script.
3. Change the Main Input File § 5.1 :
   - Edit the .pftcl file (you may want to save the old one) and utilize the read concentration input file option to specify all of the input files you created above as initial conditions for concentrations.
4. Restart the run :
   - Utilizing an editor recreate all the input parameters used in the run except for the following two items :
     - Use the dump sequence number from step 1 as the start_count.
     - Use the real time that the dump occured at from step 1 as the start_time.

3.4 Visualizing Output with AVS

We currently use AVS (Application Visualization System) [2, 3, 4] to visualize ParFlow output. AVS modules and networks are provided in directory `$PARFLOW_DIR/avs` for this purpose. To access these modules and networks within AVS, type

```
pfavs &
```

This will start AVS, load the ParFlow module library, and set the default network directory to your `$PARFLOW_DIR/avs/networks` directory. Click on `Network Editor` to open the network editor. Here, you will be able to load supplied ParFlow networks (click on `Read Network`) and use ParFlow modules to either construct new networks or modify old networks These provided networks may be used to visualize ParFlow output. Documentation for ParFlow modules is found online using the help capability provided with AVS. See the AVS documentation for details on how to use AVS.

3.5 Visualizing Output with AVS/Express

3.5.1 Running and Interacting With AVS/Express

Before running AVS/Express you may need to set some environment variables. See the AVS/Express documentation for more information.

A script is included for starting AVS/Express for the local Express installation. This can be used as a template for other systems. To start Express with the ParFlow modules loaded:

```
pfexpress
```

3.5.2 Using the AVS/Express Applications

First, some notes that apply to all of the applications. A quick way to reload the file currently being viewed is to press return in the text area that contains the current file name. To print the contents of a viewer window (or save it to a `.ps` or `.eps` file), use the `Print` editor in the `Editors` menu of the viewer. To get to other user interfaces, such as those for the grid scaling and downsizing, use the module stack pulldown menu on the left side of the screen.

Animate In the file dialog brought up by the "Select file in sequence..." button, the user should select one of the `.pfsb` files from the sequence that will be animated. The application will then load the first file in the sequence and do the isosurface on it. The desired isosurface level should be selected at this point also. Start, end, and stride are used to select the files to include in the animation. The indices in these fields are from 0 to the number of `.pfsb` files in the series minus one. If cycle is checked, the animation will continuously cycle while run is checked. Checking reset will cause the display to be reset to the start file. Animate only works with `.pfsb` files.

Brick Everything here should be self-explanatory. The dials or sliders control the upper bounding planes of the box. The dials have the immediate toggle set, causing them to update the display as they are being modified. The sliders do not have this toggle set.

ExcavateBrick The "opposite" of Brick, but with the same user interface.

Isosurface Like Animate, but without the animation stuff and only allows you to view one file at a time. Also allows you to view `.pfb` files.

Orthograph Axis labels and names are now working, although they tend to get jumbled together at the origin. The Probe Value field in the user interface gives the value of the current probe location in the plot. You can probe the value of an isoline by control-left mouse button clicking anywhere on it. You can also choose up to two isolines at specific locations by modifying the sliders in the isoline user interface in the module stack. If more precision is necessary for specifying these isolines values, a new user interface can easily be added for it.

One note on a semi-bug: If the plot shrinks down to become very small, try hitting the Normalize button. If that doesn't get it back to normal, flip between different axes a time or two, and hopefully that should work. As a last-ditch effort, delete and reload the application. This problem seems to be rare, but I have always been able to solve it without having to reload the application.

Slice3 The same user interface as Brick.

Of these applications, there is currently one known bug. In Isosurface, loading a `.pfb` file after a `.pfsb` file causes minimum values in some places to become messed up, but the application is still usable. The is due to an unfortunate "feature" in the AVS/Express UIdial module. To avoid this, delete and reload the application when you want to visualize a `.pfb` file after a `.pfsb` file.

3.6 Manipulating Data

3.6.1 Introduction to the ParFlow TCL commands (PFTCL)

Several tools for manipulating data are provided in PFTCL command set. In order to use them you need to load the ParFlow package into the TCL shell. If you are doing simple data manipulation the `xpftools` provides GUI access to most of these features. All of these tools are accessible inside of a ParFlow input script. You can use them to do post and pre processing of datafiles each time you execute a run.

```
#
# To Import the ParFlow TCL package
#
lappend auto_path $env(PARFLOW_DIR)/bin
package require parflow
namespace import Parflow::*
```

Use `pfhelp` to get a list of commands.

PFTCL assigns identifiers to each data set it stores. For example, if you read in a file called `foo.pfb`, you get the following:

```
parflow> pfload foo.pfb
dataset0
```

The first line is typed in by the user and the second line is printed out by PFTCL. It indicates that the data read from file `foo.pfb` is associated with the identifier `dataset0`.

To exit use the standard TCL command `exit`.

3.6.2 PFTCL Commands

The following gives a list of ParFlow commands and instructions for their use: Note that commands that perform operations on data sets will require an identifier for each data set it takes as input.

`pfaxpy alpha x y`

This command computes y = alpha*x+y where alpha is a scalar and x and y are identifiers representing data sets. No data set identifier is returned upon successful completion since data set y is overwritten.

`pfcvel conductivity phead`

This command computes the Darcy velocity in cells for the conductivity data set represented by the identifier ‘conductivity’ and the pressure head data set represented by the identifier ‘phead’. (note: This ”cell” is not the same as the grid cells; its corners are defined by the grid vertices.) The identifier of the data set created by this operation is returned upon successful completion.

`pfdelete dataset`

This command deletes the data set represented by the identifier ‘dataset’.

`pfdiffelt datasetp datasetq i j k digits [zero]`

This command returns the difference of two corresponding coordinates from ‘datasetp’ and ‘datasetq’ if the number of digits in agreement (significant digits) differs by more than ‘digits’ significant digits and the difference is greater than the absolute zero given by ‘zero’.

`pfdist filename`

Distribute the file onto the virtual file system. This utility must be used to create files which ParFlow can use as input. ParFlow uses a virtual file system which allows each node of the parallel machine to read from the input file independentaly. The utility does the inverse of the pfundist command. If you are using a ParFlow binary file for input you should do a pfdist just before you do the pfrun. This command requires that the processor topology and computational grid be set in the input file so that it knows how to distribute the data.

`pfflux conductivity hhead`

This command computes the net Darcy flux at vertices for the conductivity data set ‘conductivity’ and the hydraulic head data set given by ‘hhead’. An identifier representing the flux computed will be returned upon successful completion.

`pfgetelt dataset i j k`

This command returns the value at element (i,j,k) in data set ‘dataset’. The i, j, and k above must range from 0 to (nx - 1), 0 to (ny - 1), and 0 to (nz - 1) respectively. The values nx, ny, and nz are the number of grid points along the x, y, and z axes respectively. The string ‘dataset’ is an identifier representing the data set whose element is to be retrieved.

`pfgetgrid dataset`

This command returns a description of the grid which serves as the domain of data set ‘dataset’. The format of the description is given below.

- `(nx, ny, nz)`
  The number of coordinates in each direction.
- `(x, y, z)`
  The origin of the grid.

- `(dx, dy, dz)`
  The distance between each coordinate in each direction.

The above information is returned in the following Tcl list format: nx ny nz x y z dx dy dz

`pfgridtype gridtype`

This command sets the grid type to either cell centered if 'gridtype' is set to 'cell' or vetex centered if 'gridtype' is set to 'vertex'. If no new value for 'gridtype' is given, then the current value of 'gridtype' is returned. The value of 'gridtype' will be returned upon successful completion of this command.

`pfhhead phead`

This command computes the hydraulic head from the pressure head represented by the identifier 'phead'. An identifier for the hydraulic head computed is returned upon successful completion.

`pflistdata dataset`

This command returns a list of pairs if no argument is given. The first item in each pair will be an identifier representing the data set and the second item will be that data set's label. If a data set's identifier is given as an argument, then just that data set's name and label will be returned.

`pfload [file format] filename`

Loads a dataset into memory so it can be manipulated using the other utilities. A file format may preceed the filename in order to indicate the file's format. If no file type option is given, then the extension of the filename is used to determine the default file type. An identifier used to represent the data set will be returned upon successful completion.

File type options include:

- `pfb`
  ParFlow binary format. Default file type for files with a '.pfb' extension.
- `pfsb`
  ParFlow scattered binary format. Default file type for files with a '.pfsb' extension.
- `sa`
  ParFlow simple ASCII format. Default file type for files with a '.sa' extension.
- `sb`
  ParFlow simple binary format. Default file type for files with a '.sb' extension.
- `rsa`
  ParFlow real scattered ASCII format. Default file type for files with a '.rsa' extension

`pfloadsds filename dsnum`

This command is used to load Scientific Data Sets from HDF files. The SDS number 'dsnum' will be used to find the SDS you wish to load from the HDF file 'filename'. The data set loaded into memory will be assigned an identifier which will be used to refer to the data set until it is deleted. This identifier will be returned upon successful completion of the command.

`pfmdiff datasetp datasetq digits [zero]`

If 'digits' is greater than or equal to zero, then this command computes the grid point at which the number of digits in agreement (significant digits) is fewest and differs by more than 'digits' significant digits. If 'digits' is less than zero, then the point at which the number of digits in agreement (significant digits) is minimum is computed. Finally, the maximum absolute difference is computed. The above information is returned in a Tcl list of the following form: mi mj mk sd adiff

Given the search criteria, (mi, mj, mk) is the coordinate where the minimum number of significant digits 'sd' was found and 'adiff' is the maximum absolute difference.

`pfnewdata {nx ny nz} {x y z} {dx dy dz} label`

This command creates a new data set whose dimension is described by the lists nx ny nz, x y z, and dx dy dz. The first list, describes the dimensions, the second indicates the origin, and the third gives the length intervals between each coordinate along each axis. The 'label' argument will be the label of the data set that gets created. This new data set that is created will have all of its data points set to zero automatically. An identifier for the new data set will be returned upon successful completion.

`pfnewlabel dataset newlabel`

This command changes the label of the data set 'dataset' to 'newlabel'.

`pfphead hhead`

This command computes the pressure head from the hydraulic head represented by the identifier 'hhead'. An identifier for the pressure head is returned upon successful completion.

`pfsavediff datasetp datasetq digits [zero] -file filename`

This command saves to a file the differences between the values of the data sets represented by 'datasetp' and 'datasetq' to file 'filename'. The data points whose values differ in more than 'digits' significant digits and whose differences are greater than 'zero' will be saved. Also, given the above criteria, the minimum number of digits in agreement (significant digits) will be saved.

If 'digits' is less than zero, then only the minimum number of significant digits and the coordinate where the minimum was computed will be saved.

In each of the above cases, the maximum absolute difference given the criteria will also be saved.

`pfsave dataset -filetype filename`

This command is used to save the data set given by the identifier 'dataset' to a file 'filename' of type 'filetype' in one of the ParFlow formats below.

File type options include:

- pfb ParFlow binary format.
- sa ParFlow simple ASCII format.
- sb ParFlow simple binary format.
- vis Vizamrai binary format.

`pfsavesds dataset -filetype filename`

This command is used to save the data set represented by the identifier 'dataset' to the file 'filename' in the format given by 'filetype'.

The possible HDF formats are:

- -float32
- -float64
- -int8
- -uint8
- -int16
- -uint16
- -int32
- -uint32

`pfstats dataset`

This command prints various statistics for the data set represented by the identifier 'dataset'. The minimum, maximum, mean, sum, variance, and standard deviation are all computed. The above values are returned in a list of the following form: min max mean sum variance (standard deviation)

`pfvmag datasetx datasety datasetz`

This command computes the velocity magnitude when given three velocity components. The three parameters are identifiers which represent the x, y, and z components respectively. The identifier of the data set created by this operation is returned upon successful completion.

`pfvvel conductivity phead`

This command computes the Darcy velocity in cells for the conductivity data set represented by the identifier 'conductivity' and the pressure head data set represented by the identifier 'phead'. The identifier of the data set created by this operation is returned upon successful completion.

`pfprintdata dataset`

This command executes ‘pfgetgrid’ and ‘pfgetelt’ in order to display all the elements in the data set represented by the identifier ‘dataset’.

`pfprintdiff datasetp datasetq digits [zero]`

This command executes ‘pfdiffelt’ and ‘pfmdiff’ to print differences to standard output. The differences are printed one per line along with the coordinates where they occur. The last two lines displayed will show the point at which there is a minimum number of significant digits in the difference as well as the maximum absolute difference.

`pfprintgrid dataset`

This command executes pfgetgrid and formats its output before printing it on the screen. The triples (nx, ny, nz), (x, y, z), and (dx, dy, dz) are all printed on seperate lines along with labels describing each.

`pfprintlist [dataset]`

This command executes pflistdata and formats the output of that command. The formatted output is then printed on the screen. The output consists of a list of data sets and their labels one per line if no argument was given or just one data set if an identifier was given.

`pfprintmdiff datasetp datasetq digits [zero]`

This command executes ‘pfmdiff’ and formats that command’s output before displaying it on the screen. Given the search criteria, a line displaying the point at which the difference has the least number of significant digits will be displayed. Another line displaying the maximum absolute difference will also be displayed.

`printstats dataset`

This command executes ‘pfstats’ and formats that command’s output before printing it on the screen. Each of the values mentioned in the description of ‘pfstats’ will be displayed along with a label.

`pfundist filename, pfundist runname`

The command undistributes a ParFlow output file. ParFlow uses a distributed file system where each node can write to its own file. The pfundist command takes all of these individual files and collapses them into a single file.

The arguments can be a runname or a filename. If a runname is given then all of the output files associated with that run are undistributed.

Normally this is done after every pfrun command.

3.7 Directory of Test Cases

ParFlow comes with a directory containing a few simple input files for use as templates in making new files and for use in testing the code. This section gives a brief descriptionn of the problems in this driectory.

Chapter 4

Model Equations

In this chapter, we discuss the model equations used by ParFlow for both its multiphase flow and transport model and the variably saturated flow model. In section 4.1 we describe the multi-phase flow equations, and in section 4.2 we describe the transport equations. Lastly, section 4.4 describes the Richards' equation model for variably saturated flow as implemented in ParFlow.

4.1 Multi-Phase Flow Equations

The flow equations are a set of *mass balance* and *momentum balance* (Darcy's Law) equations, given respectively by,

$$\frac{\partial}{\partial t}(\phi S_i) \; + \; \nabla \cdot \vec{V}_i \; - \; Q_i \; = \; 0, \tag{4.1}$$

$$\vec{V}_i \; + \; \lambda_i \cdot (\nabla p_i \; - \; \rho_i \vec{g}) \; = \; 0, \tag{4.2}$$

for $i = 0, \ldots, n_p - 1$ ($n_p \in \{1, 2, 3\}$), where

$$\lambda_i \; = \; \frac{\bar{k} k_{ri}}{\mu_i}, \tag{4.3}$$

$$\vec{g} \; = \; [0, 0, -g]^T, \tag{4.4}$$

Table 4.1 defines the symbols in the above equations, and outlines the symbol dependencies and units. Here, ϕ describes the fluid capacity of the porous medium, and S_i describes the content of phase i in the porous medium, where we have that $0 \leq \phi \leq 1$ and $0 \leq S_i \leq 1$. The coefficient $\bar{k}$ is considered a scalar here. We also assume that ρ_i and μ_i are constant. Also note that in ParFlow, we assume that the relative permeability is given as $k_{ri}(S_i)$. The Darcy velocity vector is related to the *velocity vector*, $\vec{v}_i$, by the following:

$$\vec{V}_i = \phi S_i \vec{v}_i. \tag{4.5}$$

To complete the formulation, we have the following n_p *consititutive relations*

$$\sum_i S_i = 1, \tag{4.6}$$

Table 4.1: Notation and units for flow equations.

symbol	quantity	units
$\phi(\vec{x},t)$	porosity	[]
$S_i(\vec{x},t)$	saturation	[]
$\vec{V}_i(\vec{x},t)$	Darcy velocity vector	$[LT^{-1}]$
$Q_i(\vec{x},t)$	source/sink	$[T^{-1}]$
λ_i	mobility	$[L^3TM^{-1}]$
$p_i(\vec{x},t)$	pressure	$[ML^{-1}T^{-2}]$
ρ_i	mass density	$[ML^{-3}]$
$\vec{g}$	gravity vector	$[LT^{-2}]$
$\bar{k}(\vec{x},t)$	intrinsic permeability tensor	$[L^2]$
$k_{ri}(\vec{x},t)$	relative permeability	[]
μ_i	viscosity	$[ML^{-1}T^{-1}]$
g	gravitational acceleration	$[LT^{-2}]$

$$p_{i0} \;=\; p_{i0}(S_0), \qquad i = 1, \ldots, n_p - 1. \tag{4.7}$$

where, $p_{ij} = p_i - p_j$ is the *capillary pressure* between phase i and phase j. We now have the $3n_p$ equations, (4.1), (4.2), (4.6), and (4.7), in the $3n_p$ unknowns, $S_i, \vec{V}_i$, and p_i.

For technical reasons, we want to rewrite the above equations. First, we define the *total mobility*, λ_T, and the *total velocity*, $\vec{V}_T$, by the relations

$$\lambda_T \;=\; \sum_i \lambda_i, \tag{4.8}$$

$$\vec{V}_T \;=\; \sum_i \vec{V}_i. \tag{4.9}$$

After doing a bunch of algebra, we get the following equation for p_0:

$$-\sum_i \{\nabla \cdot \lambda_i \left(\nabla(p_0 \;+\; p_{i0}) \;-\; \rho_i \vec{g}\right) \;+\; Q_i\} \;=\; 0. \tag{4.10}$$

After doing some more algebra, we get the following $n_p - 1$ equations for S_i:

$$\frac{\partial}{\partial t}(\phi S_i) \;+\; \nabla \cdot \left(\frac{\lambda_i}{\lambda_T}\vec{V}_T \;+\; \sum_{j \neq i} \frac{\lambda_i \lambda_j}{\lambda_T}(\rho_i - \rho_j)\vec{g} \right) \;+\; \sum_{j \neq i} \nabla \cdot \frac{\lambda_i \lambda_j}{\lambda_T} \nabla p_{ji} \;-\; Q_i \;=\; 0. \tag{4.11}$$

The capillary pressures p_{ji} in (4.11) are rewritten in terms of the constitutive relations in (4.7) so that we have

$$p_{ji} \;=\; p_{j0} \;-\; p_{i0}, \tag{4.12}$$

where by definition, $p_{ii} = 0$. Note that equations (4.11) are analytically the same equations as in (4.1). The reason we rewrite them in this latter form is because of the numerical scheme we are using. We now have the $3n_p$ equations, (4.10), (4.11), (4.9), (4.2), and (4.7), in the $3n_p$ unknowns, $S_i, \vec{V}_i$, and p_i.

Table 4.2: Notation and units for transport equation.

symbol	quantity	units
$\phi(\vec{x})$	porosity	[]
$c_{i,j}(\vec{x},t)$	concentration fraction	[]
$\vec{V}_i(\vec{x},t)$	Darcy velocity vector	$[LT^{-1}]$
λ_j	degradation rate	$[T^{-1}]$
$\rho_s(\vec{x})$	density of the solid mass	$[ML^{-3}]]$
$F_{i,j}(\vec{x},t)$	mass concentration	$[L^3M^{-1}]$
n_I	number of injection wells	[]
$\gamma_k^{I;i}(t)$	injection rate	$[T^{-1}]$
$\Omega_k^I(\vec{x})$	injection well region	[]
$\bar{c}_{ij}^k()$	injected concentration fraction	[]
n_E	number of extraction wells	[]
$\gamma_k^{E;i}(t)$	extraction rate	$[T^{-1}]$
$\Omega_k^E(\vec{x})$	extraction well region	[]

4.2 Transport Equations

The transport equations in ParFlow are currently defined as follows:

$$\begin{aligned}\left(\frac{\partial}{\partial t}(\phi c_{i,j}) \;+\; \lambda_j \;\phi c_{i,j}\right) \quad &+ \quad \nabla\cdot\left(c_{i,j}\vec{V}_i\right)\\ &= \\ -\left(\frac{\partial}{\partial t}((1-\phi)\rho_s F_{i,j}) \;+\; \lambda_j\;(1-\phi)\rho_s F_{i,j}\right) \quad &+ \quad \sum_k^{n_I}\gamma_k^{I;i}\chi_{\Omega_k^I}\left(c_{i,j}-\bar{c}_{ij}^k\right) \quad - \quad \sum_k^{n_E}\gamma_k^{E;i}\chi_{\Omega_k^E}c_{i,j}\end{aligned} \tag{4.13}$$

where $i = 0,\ldots,n_p-1$ ($n_p \in \{1,2,3\}$) is the number of phases, $j = 0,\ldots,n_c-1$ is the number of contaminants, and where $c_{i,j}$ is the concentration of contaminant j in phase i. Recall also, that χ_A is the characteristic function of set A, i.e. $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ if $x \notin A$. Table 4.2 defines the symbols in the above equation, and outlines the symbol dependencies and units. The equation is basically a statement of mass conservation in a convective flow (no diffusion) with adsorption and degradation effects incorporated along with the addition of injection and extraction wells. These equations will soon have to be generalized to include a diffusion term. At the present time, as an adsorption model, we take the mass concentration term ($F_{i,j}$) to be instantaneous in time and a linear function of contaminant concentration :

$$F_{i,j} = K_{d;j}c_{i,j}, \tag{4.14}$$

where $K_{d;j}$ is the distribution coefficient of the component ($[L^3M^{-1}]$). If 4.14 is substituted into 4.13 the following equation results (which is the current model used in ParFlow) :

$$(\phi + (1-\phi)\rho_s K_{d;j})\frac{\partial}{\partial t}c_{i,j} \quad + \quad \nabla\cdot\left(c_{i,j}\vec{V}_i\right)$$

Table 4.3: Notation and units for reformulated flow equations.

symbol	quantity	units
$\vec{V}_i$	Darcy velocity vector	$[LT^{-1}]$
$\bar{K}_i$	hydraulic conductivity tensor	$[LT^{-1}]$
h_i	pressure head	$[L]$
γ	constant scale factor	$[ML^{-2}T^{-2}]$
$\vec{g}$	gravity vector	$[LT^{-2}]$

$$=$$

$$-\ (\phi + (1-\phi)\rho_s K_{d;j})\lambda_j c_{i,j} \ + \ \sum_k^{n_I} \gamma_k^{I;i} \chi_{\Omega_k^I} \left(c_{i,j} - \bar{c}_{ij}^k \right) \ - \ \sum_k^{n_E} \gamma_k^{E;i} \chi_{\Omega_k^E} c_{i,j} \tag{4.15}$$

4.3 Notation and Units

In this section, we discuss other common formulations of the flow and transport equations, and how they relate to the equations solved by ParFlow.

We can rewrite equation (4.2) as

$$\vec{V}_i \ + \ \bar{K}_i \cdot (\nabla h_i \ - \ \frac{\rho_i}{\gamma} \vec{g}) \ = \ 0, \tag{4.16}$$

where

$$\bar{K}_i \ = \ \gamma \lambda_i, \tag{4.17}$$

$$h_i \ = \ (p_i \ - \ \bar{p})/\gamma. \tag{4.18}$$

Table 4.3 defines the symbols and their units. We can then rewrite equations (4.10) and (4.11) as

$$-\ \sum_i \left\{ \nabla \cdot \bar{K}_i \left(\nabla (h_0 \ + \ h_{i0}) \ - \ \frac{\rho_i}{\gamma} \vec{g} \right) \ + \ Q_i \right\} \ = \ 0, \tag{4.19}$$

$$\frac{\partial}{\partial t}(\phi S_i) \ + \ \nabla \cdot \left(\frac{\bar{K}_i}{\bar{K}_T} \vec{V}_T \ + \ \sum_{j \neq i} \frac{\bar{K}_i \bar{K}_j}{\bar{K}_T} \left(\frac{\rho_i}{\gamma} - \frac{\rho_j}{\gamma} \right) \vec{g} \right) \ + \ \sum_{j \neq i} \nabla \cdot \frac{\bar{K}_i \bar{K}_j}{\bar{K}_T} \nabla h_{ji} \ - \ Q_i \ = \ 0. \tag{4.20}$$

Note that $\bar{K}_i$ is supposed to be a tensor, but we treat it as a scalar here. Also, note that by carefully defining the input to ParFlow, we can use the units of equations (4.19) and (4.20). To be more precise, let us denote ParFlow input symbols by appending the symbols in table 4.1 with (I), and let $\gamma = \rho_0 g$ (this is a typical definition). Then, we want:

$$\bar{k}(I) \ = \ \gamma \bar{k}/\mu_0; \tag{4.21}$$

$$\mu_i(I) \ = \ \mu_i/\mu_0; \tag{4.22}$$

$$\begin{aligned} p_i(I) &= h_i; && (4.23)\\ \rho_i(I) &= \rho_i/\rho_0; && (4.24)\\ g(I) &= 1. && (4.25)\end{aligned}$$

By doing this, $\bar{k}(I)$ represents hydraulic conductivity of the base phase $\bar{K}_0$ (e.g. water) under saturated conditions (i.e. $k_{r0} = 1$). Though ParFlow input has been defined this way in the past, this is not the recommended procedure since it may lead to confusion and mistakes.

4.4 Richards' Equation

The form of Richards' equation implemented in ParFlow is given as,

$$\frac{\partial(S(p)\rho(p)\phi)}{\partial t} - \nabla\cdot(K(p)\rho(p)(\nabla p - \rho(p)\vec{g})) = Q, \text{ in } \Omega, \tag{4.26}$$

where Ω is the flow domain, p is the water pressure, S is the water saturation, ϕ is the porosity of the medium, $K(p)$ is the hydraulic conductivity and Q is the water source/sink term (includes wells). The hydraulic conductivity can be written as,

$$K(p) = \frac{\bar{k}k_r(p)}{\mu} \tag{4.27}$$

Boundary conditions can be stated as,

$$\begin{aligned} p &= p_D, \text{ on } \Gamma^D, && (4.28)\\ -K(p)\nabla p\cdot\mathbf{n} &= g_N, \text{ on } \Gamma^N, && (4.29)\end{aligned}$$

where $\Gamma^D \cup \Gamma^N = \partial\Omega$, $\Gamma^D \neq \emptyset$, and $\mathbf{n}$ is an outward pointing, unit, normal vector to Ω. This is the mixed form of Richards' equation. Note here that due to the constant (or passive) air phase pressure assumption, Richards' equation ignores the air phase except through its effects on the hydraulic conductivity, K. An initial condition,

$$p = p^0(x), \; t = 0, \tag{4.30}$$

completes the specification of the problem.

Chapter 5

ParFlow Files

In this chapter, we discuss the various file formats used in ParFlow. To help simplify the description of these formats, we use a pseudocode notation composed of *fields* and *control constructs*.

A field is a piece of data having one of the *field types* listed in Table 5.1 (note that field types may have one meaning in ASCII files and another meaning in binary files). Fields are denoted by enclosing the field name with a `<` on the left and a `>` on the right. The field name is composed of alphanumeric characters and underscores (`_`). In the defining entry of a field, the field name is also prepended by its field type and a `:`. The control constructs used in our pseudocode have the keyword names `FOR`, `IF`, and `LINE`, and the beginning and end of each of these constructs is delimited by the keywords `BEGIN` and `END`.

The `FOR` construct is used to describe repeated input format patterns. For example, consider the following file format:

```
<integer : num_coordinates>
FOR coordinate = 0 TO <num_coordinates> - 1
BEGIN
   <real : x>  <real : y>  <real : z>
END
```

The field `<num_coordinates>` is an integer specifying the number of coordinates to follow. The `FOR` construct indicates that `<num_coordinates>` entries follow, and each entry is composed of the

Table 5.1: Field types.

field type	ASCII	binary
`integer`	integer	XDR integer
`real`	real	-
`string`	string	-
`double`	-	IEEE 8 byte double
`float`	-	IEEE 4 byte float

three real fields, `<x>`, `<y>`, and `<z>`. Here is an example of a file with this format:

```
3
2.0 1.0 -3.5
1.0 1.1 -3.1
2.5 3.0 -3.7
```

The `IF` construct is actually an `IF/ELSE` construct, and is used to describe input format patterns that appear only under certain circumstances. For example, consider the following file format:

```
<integer : type>
IF (<type> = 0)
BEGIN
   <real : x>  <real : y>  <real : z>
END
ELSE IF (<type> = 1)
BEGIN
   <integer : i>  <integer : j>  <integer : k>
END
```

The field `<type>` is an integer specifying the "type" of input to follow. The `IF` construct indicates that if `<type>` has value 0, then the three real fields, `<x>`, `<y>`, and `<z>`, follow. If `<type>` has value 1, then the three integer fields, `<i>`, `<j>`, and `<k>`, follow. Here is an example of a file with this format:

```
0
2.0 1.0 -3.5
```

The `LINE` construct indicates fields that are on the same line of a file. Since input files in ParFlow are all in "free format", it is used only to describe some output file formats. For example, consider the following file format:

```
LINE
BEGIN
   <real : x>
   <real : y>
   <real : z>
END
```

The `LINE` construct indicates that the three real fields, `<x>`, `<y>`, and `<z>`, are all on the same line. Here is an example of a file with this format:

```
2.0 1.0 -3.5
```

Comment lines may also appear in our file format pseudocode. All text following a `#` character is a comment, and is not part of the file format.

5.1 Main Input File (.pftcl)

The main ParFlow input file is a TCL script. This might seem overly combersome at first but the basic input file structure is not very complicated (although it is somewhat verbose). For more advanced users, the TCL scripting means you can very easily create programs to run ParFlow. A simple example is creating a loop to run several hundred different simulations using different seeds to the random field generators. This can be automated from within the ParFlow input file.

The basic idea behind ParFlow input is a simple database. The database contains entries which have a key and a value associated with that key. This is very similiar in nature to the Windows 95/NT registry and several other systems. When ParFlow runs, it queries the database you have created by key names to get the values you have specified.

The command `pfset` is used to create the database entries. A simple ParFlow input script contains a long list of `pfset` commands.

It should be noted that the keys are "dynamic" in that many are built up from values of other keys. For example if you have two wells named *northwell* and *southwell* then you will have to set some keys which specify the parameters for each well. The keys are built up in a simple sort of heirarchy.

The following sections contain a description of all of the keys used by ParFlow. For an example of input files you can look at the `test` subdirectory of the ParFlow distribution. Looking over some examples should give you a good feel for how the file scripts are put together.

Each key's entry has the form:

type **KeyName** [default value]
Description
Example Useage:

The "type" is one of integer, double, string, list. Integer and double are IEEE numbers. String is a text string (for example, a filename). Strings can contain spaces if you use the proper TCL syntax (i.e. using double quotes). These types are standard TCL types. Lists are strings but they indicate the names of a series of items. For example you might need to specify the names of the geometries. You would do this using space seperated names (what we are calling a list) "layer1 layer2 layer3".

The descriptions that follow are organized into functional areas. An example for each database entry is given.

Note that units used for each physical quantity specified in the input file must be consistent with units used for all other quantities. The exact units used can be any consistent set as ParFlow does not assume any specific set of units. However, it is up to the user to make sure all specifications are indeed consistent.

5.1.1 Input File Format Number

integer **FileVersion** [no default]

This gives the value of the input file version number that this file fits.
Example Useage:

```
pfset FileVersion 4
```

As development of the ParFlow code continues, the input file format will vary. We have thus included an input file format number as a way of verifying that the correct format type is being used. The user can check in the `parflow/config/file_versions.h` file to verify that the format number specified in the input file matches the defined value of `PFIN_VERSION`.

5.1.2 Geometries

Here we define all "geometrical" information needed by ParFlow. For example, the domain (and patches on the domain where boundary conditions are to be imposed), lithology or hydrostratigraphic units, faults, initial plume shapes, and so on, are considered geometries.

This input section is a little confusing. Two items are being specified, geometry inputs and geometries. A geometry input is a type of geometry input (for example a box or an input file). A geometry input can contain more than one geometry. A geometry input of type Box has a single geometry (the square box defined by the extants of the two points). A SolidFile input type can contain several geometries.

list **GeomInput.Names** [no default]
This is a list of the geometry input names which define the containers for all of the geometries defined for this problem.
Example Useage:

```
pfset GeomInput.Names    "solidinput indinput boxinput"
```

string **GeomInput.***geom_input_name***.InputType** [no default]
This defines the input type for the geometry input with *geom_input_name*. This key must be one of: **SolidFile, IndicatorField**, **Box**.
Example Useage:

```
pfset GeomInput.solidinput.InputType  SolidFile
```

list **GeomInput.***geom_input_name***.GeomNames** [no default]
This is a list of the names of the geometries defined by the geometry input. For a geometry input type of Box, the list should contain a single geometry name. For the SolidFile geometry type this should contain a list with the same number of gemetries as were defined using GMS. The order of geometries in the SolidFile should match the names. For IndicatorField types you need to specify the value in the input field which matches the name using GeomInput.*geom_input_name*.Value.
Example Useage:

```
pfset GeomInput.solidinput.GeomNames "domain bottomlayer \
                                      middlelayer toplayer"
```

string **GeomInput.***geom_input_name***.Filename** [no default]
For IndicatorField and SolidFile geometry inputs this key specifies the input filename which contains the field or solid information.
Example Useage:

```
pfset GeomInput.solidinput.FileName   ocwd.pfsol
```

integer **GeomInput.***geometry_input_name***.Value** [no default]
For IndicatorField geometry inputs you need to specify the mapping between values in the input file and the geometry names. The named geometry will be defined whereever the input file is equal to the specifed value.
Example Useage:

```
pfset GeomInput.sourceregion.Value   11
```

For box geometries you need to specify the location of the box. This is done by defining two corners of the the box.

double **Geom.***box_geom_name***.Lower.X** [no default]
This gives the lower X real space coordinate value of the previously specified box geometry of name *box_geom_name*.
Example Useage:

```
pfset Geom.background.Lower.X   -1.0
```

double **Geom.***box_geom_name***.Lower.Y** [no default]
This gives the lower Y real space coordinate value of the previously specified box geometry of name *box_geom_name*.
Example Useage:

```
pfset Geom.background.Lower.Y   -1.0
```

double **Geom.***box_geom_name***.Lower.Z** [no default]
This gives the lower Z real space coordinate value of the previously specified box geometry of name *box_geom_name*.
Example Useage:

```
pfset Geom.background.Lower.Z   -1.0
```

double **Geom.***box_geom_name***.Upper.X** [no default]
This gives the upper X real space coordinate value of the previously specified box geometry of name *box_geom_name*.
Example Useage:

```
pfset Geom.background.Upper.X   151.0
```

double **Geom.***box_geom_name***.Upper.Y** [no default]
This gives the upper Y real space coordinate value of the previously specified box geometry of name *box_geom_name*.
Example Useage:

```
pfset Geom.background.Upper.Y   171.0
```

double **Geom.***box_geom_name***.Upper.Z** [no default]
This gives the upper Z real space coordinate value of the previously specified box geometry of name *box_geom_name*.
Example Useage:

```
pfset Geom.background.Upper.Z   11.0
```

list **Geom.***geom_name***.Patches** [no default]
Patches are defined on the surfaces of geometries. Currently you can only define patches on Box geometries and on the the first geometry in a SolidFile. For a Box the order is fixed (left right front back bottom top) but you can name the sides anything you want.
For SolidFiles the order is printed by the conversion routine that converts GMS to SolidFile format.
Example Useage:

```
pfset Geom.background.Patches   "left right front back bottom top"
```

Here is an example geometry input section which has three geometry inputs.

```
#---------------------------------------------------------
# Geometries:
#---------------------------------------------------------

#
# This defines the three geometry input names
#
pfset GeomInput.Names   "solidinput indinput boxinput"

#
# For a solid file geometry input type you need to specify the names
# of the gemetries and the filename
#
pfset GeomInput.solidinput.InputType  SolidFile

#
```

```
# The names of the geometries contained in the solid file.  Order is
# important and defines the mapping.  First geometry gets the first name.
#
pfset GeomInput.solidinput.GeomNames
                   "domain bottomlayer middlelayer toplayer"

#
# Filename that contains the geometry
#
pfset GeomInput.solidinput.FileName   ocwd.pfsol

#
# Order is important here, must match what is solid file, what is
# printed by the conversion routine.
#
pfset Geom.domain.Patches  "henry frank jane betsy al nellie"

#
# An indicator field is a 3D field of values.  The values within the
# field can be mapped to ParFlow geometries
# Indicator fields must match the computation grid exactly!
#
pfset GeomInput.indinput.InputType    IndicatorField
pfset GeomInput.indinput.GeomNames    "sourceregion concenregion"
pfset GeomInput.indinput.FileName     ocwd.pfb

#
# Here we set up the mapping between values in the field and
# ParFlow geometries
#
pfset GeomInput.sourceregion.Value    11
pfset GeomInput.concenregion.Value    21

#
# A box is just a box defined by two points.
#
pfset GeomInput.boxinput.InputType    Box
pfset GeomInput.boxinput.GeomName     background

pfset Geom.background.Lower.X         -1.0
pfset Geom.background.Lower.Y         -1.0
pfset Geom.background.Lower.Z         -1.0
pfset Geom.background.Upper.X         151.0
pfset Geom.background.Upper.Y         171.0
pfset Geom.background.Upper.Z         11.0

#
# Order is fixed, but you can change the name
#
pfset Geom.background.Patches         "left right front back bottom top"
```

5.1.3 Timing Information

The data given in the timing section describe all the "temporal" information needed by ParFlow. The data items are used to describe time units for later sections, sequence iterations in time, indicate actual starting and stopping values and give instructions on when data is printed out.

double **TimingInfo.BaseUnit** [no default]
This key is used to indicate the smallest common unit of time for describing "time cycling data" in, currently, the well and boundary condition sections. The lengths of the cycles in those sections will be integer multiples of this value, therefore it needs to be the smallest divisor which produces an integral result for every "real time" cycle interval length needed.
Example Useage:

```
pfset TimingInfo.BaseUnit      1.0
```

integer **TimingInfo.StartCount** [no default]
This key is used to indicate the time step number that will be associated with the first advection cycle in a transient problem. The value **-1** indicates that advection is not to be done. The value **0** indicates that advection should begin with the given initial conditions. Values greater than **0** are intended to mean "restart" from some previous "checkpoint" time-step, but this has not yet been implemented.
Example Useage:

```
pfset TimingInfo.StartCount    0
```

double **TimingInfo.StartTime** [no default]
This key is used to indicate the starting time for the simulation.
Example Useage:

```
pfset TimingInfo.StartTime     0.0
```

double **TimingInfo.StopTime** [no default]
This key is used to indicate the stopping time for the simulation.
Example Useage:

```
pfset TimingInfo.StopTime      100.0
```

double **TimingInfo.DumpInterval** [no default]
This key is the real time interval at which time-dependent output should be written. A value of **0** will produce undefined behavior. If the value is negative, output will be dumped out every n time steps, where n is the absolute value of the integer part of the value.
Example Useage:

```
pfset TimingInfo.DumpInterval  10.0
```

For *Richards' equation cases only* input is collected for time step selection. Input for this section is given as follows:

list **TimeStep.Type** [no default]
This key must be one of: **Constant** or **Growth**. The value **Constant** defines a constant time step. The value **Growth** defines a time step that starts as dt_0 and is defined for other steps as $dt^{new} = \gamma dt^{old}$ such that $dt^{new} \leq dt_{max}$ and $dt^{new} \geq dt_{min}$.
Example Useage:

```
pfset TimeStep.Type      Constant
```

double **TimeStep.Value** [no default]
This key is used only if a constant time step is selected and indicates the value of the time step for all steps taken.
Example Useage:

```
pfset TimeStep.Value      0.001
```

double **TimeStep.InitialStep** [no default]
This key specifies the initial time step dt_0 if the **Growth** type time step is selected.
Example Useage:

```
pfset TimeStep.InitialStep    0.001
```

double **TimeStep.GrowthFactor** [no default]
This key specifies the growth factor γ by which a time step will be multiplied to get the new time step when the **Growth** type time step is selected.
Example Useage:

```
pfset TimeStep.GrowthFactor      1.5
```

double **TimeStep.MaxStep** [no default]
This key specifies the maximum time step allowed, dt_{max}, when the **Growth** type time step is selected.
Example Useage:

```
pfset TimeStep.MaxStep      86400
```

double **TimeStep.MinStep** [no default]
This key specifies the minimum time step allowed, dt_{min}, when the **Growth** type time step is selected.
Example Useage:

```
pfset TimeStep.MinStep      1.0e-3
```

5.1.4 Domain

The domain may be represented by any of the solid types in § 5.1.2 above that allow the definition of surface patches. These surface patches are used to define boundary conditions in § 5.1.17 and § 5.1.18 below. Subsequently, it is required that the union of the defined surface patches equal the entire domain surface. NOTE: This requirement is NOT checked in the code.

string **Domain.GeomName** [no default]
This key specifies which of the named geometries is the problem domain.
Example Useage:

```
pfset Domain.GeomName    domain
```

5.1.5 Phases and Contaminants

list **Phase.Names** [no default]
This specifies the names of phases to be modeled. Currently only 1 or 2 phases may be modeled.
Example Useage:

```
pfset Phase.Names    "water"
```

list **Contaminant.Names** [no default]
This specifies the names of contaminants to be advected.
Example Useage:

```
pfset Contaminants.Names   "tce"
```

5.1.6 Gravity, Phase Density and Phase Viscosity

double **Gravity** [no default]
Specifies the gravity constant to be used.
Example Useage:

```
pfset Gravity 1.0
```

string **Phase.*phase_name*.Density.Type** [no default]
This key specifies whether density will be a constant value or if it will be given by an equation of state of the form $(rd)exp(cP)$, where P is pressure, rd is the density at atmospheric pressure, and c is the phase compressibility constant. This key must be either **Constant** or **EquationOfState**.
Example Useage:

```
pfset Phase.water.Density.Type  Constant
```

double **Phase.*phase_name*.Density.Value** [no default]
This specifies the value of density if this phase was specified to have a constant density value for the phase *phase_name*.
Example Useage:

```
pfset Phase.water.Density.Value   1.0
```

double **Phase.*phase_name*.Density.ReferenceDensity** [no default]
This key specifies the reference density if an equation of state density function is specified for the phase *phase_name*.
Example Useage:

```
pfset Phase.water.Density.ReferenceDensity   1.0
```

double **Phase.*phase_name*.Density.CompressibilityConstant** [no default]
This key specifies the phase compressibility constant if an equation of state density function is specified for the phase *phase—-name*.
Example Useage:

```
pfset Phase.water.Density.CompressibilityConstant   1.0
```

string **Phase.*phase_name*.Viscosity.Type** [Constant]
This key specifies whether viscosity will be a constant value. Currently, the only choice for this key is **Constant**.
Example Useage:

```
pfset Phase.water.Viscosity.Type   Constant
```

double **Phase.*phase_name*.Viscosity.Value** [no default]
This specifies the value of viscosity if this phase was specified to have a constant viscosity value.
Example Useage:

```
pfset Phase.water.Viscosity.Value   1.0
```

5.1.7 Chemical Reactions

double **Contaminants.***contaminant_name***.Degradation.Value** [no default]

This key specifies the half-life decay rate of the named contaminant, *contaminant_name.* At present only first order decay reactions are implemented and it is assumed that one contaminant cannot decay into another.
Example Useage:

```
pfset Contaminants.tce.Degradation.Value         0.0
```

5.1.8 Permeability

In this section, permeability property values are assigned to grid points within geometries (specified in § 5.1.2 above) using one of the methods described below. Permeabilities are assumed to be a diagonal tensor with entries given as,

$$\begin{pmatrix} k_x(\mathbf{x}) & 0 & 0 \\ 0 & k_y(\mathbf{x}) & 0 \\ 0 & 0 & k_z(\mathbf{x}) \end{pmatrix} K(\mathbf{x}),$$

where $K(\mathbf{x})$ is the permeability field given below. Specification of the tensor entries (k_x, k_y and k_z) will be given at the end of this section.

The random field routines (*turning bands* and *pgs*) can use conditioning data if the user so desires. It is not necessary to use conditioning as ParFlow automatically defaults to not use conditioning data, but if conditioning is desired, the following key should be set:

string **Perm.Conditioning.FileName** ["NA"]

This key specifies the name of the file that contains the conditioning data. The default string **NA** indicates that conditioning data is not applicable.
Example Useage:

```
pfset Perm.Conditioning.FileName   "well_cond.txt"
```

The file that contains the conditioning data is a simple ascii file containing points and values. The format is:

```
nlines
x1 y1 z1 value1
x2 y2 z2 value2
.  .  .    .
.  .  .    .
.  .  .    .
xn yn zn valuen
```

The value of *nlines* is just the number of lines to follow in the file, which is equal to the number of data points.

The variables *xi,yi,zi* are the real space coordinates (in the units used for the given parflow run) of a point at which a fixed permeability value is to be assigned. The variable *valuei* is the actual permeability value that is known.

Note that the coordinates are not related to the grid in any way. Conditioning does not require that fixed values be on a grid. The PGS algorithm will map the given value to the closest grid point and that will be fixed. This is done for speed reasons. The conditioned turning bands algorithm does not do this; conditioning is done for every grid point using the given conditioning data at the location given. Mapping to grid points for that algorithm does not give any speedup, so there is no need to do it.

NOTE: The given values should be the actual measured values - adjustment in the conditioning for the lognormal distribution that is assumed is taken care of in the algorithms.

The general format for the permeability input is as follows:

list **Geom.Perm.Names** [no default]
This key specifies all of the geometries to which a permeability field will be assigned. These geometries must cover the entire computational domain.
Example Useage:

```
pfset GeomInput.Names   "background domain concen_region"
```

string **Geom.**geometry_name**.Perm.Type** [no default]
This key specifies which method is to be used to assign permeability data to the named geometry, *geometry_name*. It must be either **Constant**, **TurnBands**, **ParGuass**, or **PFBFile**. The **Constant** value indicates that a constant is to be assigned to all grid cells within a geometry. The **TurnBand** value indicates that Tompson's Turning Bands method is to be used to assign permeability data to all grid cells within a geometry [8]. The **ParGauss** value indicates that a Parallel Gaussian Simulator method is to be used to assign permeability data to all grid cells within a geometry. The **PFBFile** value indicates that premeabilities are to be read from the "ParFlow Binary" file. Both the Turning Bands and Parallel Gaussian Simulators generate a random field with correlation lengths in the 3 spatial directions given by λ_x, λ_y, and λ_z with the geometric mean of the log normal field given by μ and the standard deviation of the normal field given by σ. In generating the field both of these methods can be made to stratify the data, that is follow the top or bottom surface. The generated field can also be made so that the data is normal or log normal, with or without bounds truncation. Turning Bands uses a line process, the number of lines used and the resolution of the process can be changed as well as the maximum normalized frequency $K_{\max}$ and the normalized frequency increment δK. The Parallel Gaussian Simulator uses a search neighborhood, the number of simulated points and the number of conditioning points can be changed.
Example Useage:

```
pfset Geom.background.Perm.Type   Constant
```

double **Geom.***geometry_name***.Perm.Value** [no default]
This key specifies the value assigned to all points in the named geometry, *geometry_name*, if

the type was set to constant.
Example Useage:

```
pfset Geom.domain.Perm.Value   1.0
```

double **Geom.***geometry_name***.Perm.LambdaX** [no default]
This key specifies the x correlation length, λ_x, of the field generated for the named geometry, *geometry_name*, if either the Turning Bands or Parallel Gaussian Simulator are chosen.
Example Useage:

```
pfset Geom.domain.Perm.LambdaX   200.0
```

double **Geom.***geometry_name***.Perm.LambdaY** [no default]
This key specifies the y correlation length, λ_y, of the field generated for the named geometry, *geometry_name*, if either the Turning Bands or Parallel Gaussian Simulator are chosen.
Example Useage:

```
pfset Geom.domain.Perm.LambdaY   200.0
```

double **Geom.***geometry_name***.Perm.LambdaZ** [no default]
This key specifies the z correlation length, λ_z, of the field generated for the named geometry, *geometry_name*, if either the Turning Bands or Parallel Gaussian Simulator are chosen.
Example Useage:

```
pfset Geom.domain.Perm.LambdaZ   10.0
```

double **Geom.***geometry_name***.Perm.GeomMean** [no default]
This key specifies the geometric mean, μ, of the log normal field generated for the named geometry, *geometry_name*, if either the Turning Bands or Parallel Gaussian Simulator are chosen.
Example Useage:

```
pfset Geom.domain.Perm.GeomMean   4.56
```

double **Geom.***geometry_name***.Perm.Sigma** [no default]
This key specifies the standard deviation, σ, of the normal field generated for the named geometry, *geometry_name*, if either the Turning Bands or Parallel Gaussian Simulator are chosen.
Example Useage:

```
pfset Geom.domain.Perm.Sigma   2.08
```

integer **Geom.***geometry_name***.Perm.Seed** [1]
This key specifies the initial seed for the random number generator used to generate the field for the named geometry, *geometry_name*, if either the Turning Bands or Parallel Gaussian Simulator are chosen. This number must be positive.
Example Useage:

```
pfset Geom.domain.Perm.Seed   1
```

integer **Geom.***geometry_name***.Perm.NumLines** [100]
This key specifies the number of lines to be used in the Turning Bands algorithm for the named geometry, *geometry_name*.
Example Useage:

```
pfset Geom.domain.Perm.NumLines   100
```

double **Geom.***geometry_name***.Perm.RZeta** [5.0]
This key specifies the resolution of the line processes, in terms of the minimum grid spacing, to be used in the Turning Bands algorithm for the named geometry, *geometry_name*. Large values imply high resolution.
Example Useage:

```
pfset Geom.domain.Perm.RZeta   5.0
```

double **Geom.***geometry_name***.Perm.KMax** [100.0]
This key specifies the the maximum normalized frequency, $K_{\max}$, to be used in the Turning Bands algorithm for the named geometry, *geometry_name*.
Example Useage:

```
pfset Geom.domain.Perm.KMax   100.0
```

double **Geom.***geometry_name***.Perm.DelK** [0.2]
This key specifies the normalized frequency increment, δK, to be used in the Turning Bands algorithm for the named geometry, *geometry_name*.
Example Useage:

```
pfset Geom.domain.Perm.DelK   0.2
```

integer **Geom.***geometry_name***.Perm.MaxNPts** [no default]
This key sets limits on the number of simulated points in the search neighborhood to be used in the Parallel Gaussian Simulator for the named geometry, *geometry_name*.
Example Useage:

```
pfset Geom.domain.Perm.MaxNPts   5
```

integer **Geom.***geometry_name***.Perm.MaxCpts** [no default]
This key sets limits on the number of external conditioning points in the search neighborhood to be used in the Parallel Gaussian Simulator for the named geometry, *geometry_name*.
Example Useage:

```
pfset Geom.domain.Perm.MaxCpts   200
```

string **Geom.***geometry_name***.Perm.LogNormal** [”LogTruncated”]
The key specifies when a normal, log normal, truncated normal or truncated log normal field is to be generated by the method for the named geometry, *geometry_name*. This value must be one of **Normal**, **Log**, **NormalTruncated** or **LogTruncate** and can be used with either Turning Bands or the Parallel Gaussian Simulator.
Example Useage:

```
pfset Geom.domain.Perm.LogNormal   "LogTruncated"
```

string **Geom.***geometry_name***.Perm.StratType** [”Bottom”]
This key specifies the stratification of the permeability field generated by the method for the named geometry, *geometry_name*. The value must be one of **Horizontal**, **Bottom** or **Top** and can be used with either the Turning Bands or the Parallel Gaussian Simulator.
Example Useage:

```
pfset Geom.domain.Perm.StratType  "Bottom"
```

double **Geom.***geometry_name***.Perm.LowCutoff** [no default]
This key specifies the low cutoff value for truncating the generated field for the named geometry, *geometry_name*, when either the NormalTruncated or LogTruncated values are chosen.
Example Useage:

```
pfset Geom.domain.Perm.LowCutoff   0.0
```

double **Geom.***geometry_name***.Perm.HighCutoff** [no default]
This key specifies the high cutoff value for truncating the generated field for the named geometry, *geometry_name*, when either the NormalTruncated or LogTruncated values are chosen.
Example Useage:

```
pfset Geom.domain.Perm.HighCutoff   100.0
```

string **Geom.***geometry_name***.Perm.FileName** [no default]
This key specifies that permeability values for the specified geometry, *geometry_name*, are given according to a user-supplied description in the “ParFlow Binary” file whose filename is given as the value. For a description of the ParFlow Binary file format, see § 5.2. The ParFlow Binary file associated with the named geometry must contain a collection of permeability values corresponding in a one-to-one manner to the entire computational grid. That is to say, when the contents of the file are read into the simulator, a complete permeability description for the entire domain is supplied. Only those values associated with computational cells residing within the geometry (as it is represented on the computational grid) will be copied into data structures used during the course of a simulation. Thus, the values associated with cells outside of the geounit are irrelevant. For clarity, consider a couple of different scenarios. For example, the user may create a file for each geometry such that appropriate permeability values are given for the geometry and “garbage” values (e.g., some flag value) are given for the rest of the computational domain. In this case, a separate

binary file is specified for each geometry. Alternatively, one may place all values representing the permeability field on the union of the geometries into a single binary file. Note that the permeability values must be represented in precisely the same configuration as the computational grid. Then, the same file could be specified for each geounit in the input file. Or, the computational domain could be described as a single geouint (in the ParFlow input file) in which case the permeability values would be read in only once.
Example Useage:

```
pfset Geom.domain.Perm.FileName "domain_perm.pfb"
```

string **Perm.TensorType** [no default]
This key specifies whether the permeability tensor entries k_x, k_y and k_z will be specified as three constants within a set of regions covering the domain or whether the entries will be specified cell-wise by files. The choices for this key are **TensorByGeom** and **TensorByFile**.
Example Useage:

```
pfset Perm.TensorType     TensorByGeom
```

string **Geom.Perm.TensorByGeom.Names** [no default]
This key specifies all of the geometries to which permeability tensor entries will be assigned. These geometries must cover the entire computational domain.
Example Useage:

```
pfset Geom.Perm.TensorByGeom.Names   "background domain"
```

double **Geom.***geometry_name***.Perm.TensorValX** [no default]
This key specifies the value of k_x for the geometry given by *geometry_name.*
Example Useage:

```
pfset Geom.domain.Perm.TensorValX   1.0
```

double **Geom.***geometry_name***.Perm.TensorValY** [no default]
This key specifies the value of k_y for the geometry given by *geom_name.*
Example Useage:

```
pfset Geom.domain.Perm.TensorValY   1.0
```

double **Geom.***geometry_name***.Perm.TensorValZ** [no default]
This key specifies the value of k_z for the geometry given by *geom_name.*
Example Useage:

```
pfset Geom.domain.Perm.TensorValZ   1.0
```

string **Geom.***geometry_name***.Perm.TensorFileX** [no default]

This key specifies that k_x values for the specified geometry, *geometry_name*, are given according to a user-supplied description in the "ParFlow Binary" file whose filename is given as the value. The only choice for the value of *geometry_name* is "domain".
Example Useage:

```
pfset Geom.domain.Perm.TensorByFileX   "perm_x.pfb"
```

string **Geom.***geometry_name***.Perm.TensorFileY** [no default]

This key specifies that k_y values for the specified geometry, *geometry_name*, are given according to a user-supplied description in the "ParFlow Binary" file whose filename is given as the value. The only choice for the value of *geometry_name* is "domain".
Example Useage:

```
pfset Geom.domain.Perm.TensorByFileY   "perm_y.pfb"
```

string **Geom.***geometry_name***.Perm.TensorFileZ** [no default]

This key specifies that k_z values for the specified geometry, *geometry_name*, are given according to a user-supplied description in the "ParFlow Binary" file whose filename is given as the value. The only choice for the value of *geometry_name* is "domain".
Example Useage:

```
pfset Geom.domain.Perm.TensorByFileZ   "perm_z.pfb"
```

5.1.9 Porosity

Here, porosity values are assigned within geounits (specified in § 5.1.2 above) using one of the methods described below.

The format for this section of input is:

list **Geom.Porosity.GeomNames** [no default]

This key specifies all of the geometries on which a porosity will be assigned. These geometries must cover the entire computational domain.
Example Useage:

```
pfset GeomInput.Names   "background"
```

string **Geom.***geometry_name***.Porosity.Type** [no default]

This key specifies which method is to be used to assign porosity data to the named geometry, *geometry_name*. The only choice currently available is **Constant** which indicates that a constant is to be assigned to all grid cells within a geometry.
Example Useage:

```
pfset Geom.background.Porosity.Type   Constant
```

double **Geom.***geometry_name***.Porosity.Value** [no default]
This key specifies the value assigned to all points in the named geometry, *geometry_name*, if the type was set to constant.
Example Useage:

```
pfset Geom.domain.Porosity.Value   1.0
```

5.1.10 Retardation

Here, retardation values are assigned for contaminants within geounits (specified in § 5.1.2 above) using one of the functions described below. The format for this section of input is:

list **Geom.Retardation.GeomNames** [no default]
This key specifies all of the geometries to which the contaminants will have a retardation function applied.
Example Useage:

```
pfset GeomInput.Names   "background"
```

string **Geom.***geometry_name.contaminant_name***.Retardation.Type** [no default]
This key specifies which function is to be used to compute the retardation for the named contaminant, *contaminant_name*, in the named geometry, *geometry_name.* The only choice currently available is **Linear** which indicates that a simple linear retardation function is to be used to compute the retardation.
Example Useage:

```
pfset Geom.background.tce.Retardation.Type   Linear
```

double **Geom.***geometry_name.contaminant_name***.Retardation.Value** [no default]
This key specifies the distribution coefficient for the linear function used to compute the retardation of the named contaminant, *contaminant_name*, in the named geometry, *geometry_name.* The value should be scaled by the density of the material in the geometry.
Example Useage:

```
pfset Geom.domain.Retardation.Value   0.2
```

5.1.11 Full Multiphase Mobilities

Here we define phase mobilities by specifying the relative permeability function. Input is specified differently depending on what problem is being specified. For full multi-phase problems, the following input keys are used. See the next section for the correct Richards' equation input format.

string **Phase.***phase_name***.Mobility.Type** [no default]
This key specifies whether the mobility for *phase_name* will be a given constant or a polynomial of the form, $(S - S_0)^a$, where S is saturation, S_0 is irreducible saturation, and a is some exponent.

The possibilities for this key are **Constant** and **Polynomial**.
Example Useage:

```
pfset Phase.water.Mobility.Type   Constant
```

double **Phase.***phase_name***.Mobility.Value** [no default]
This key specifies the constant mobility value for phase *phase_name*.
Example Useage:

```
pfset Phase.water.Mobility.Value   1.0
```

double **Phase.***phase_name***.Mobility.Exponent** [2.0]
This key specifies the exponent used in a polynomial representation of the relative permeability. Currently, only a value of 2.0 is allowed for this key.
Example Useage:

```
pfset Phase.water.Mobility.Exponent   2.0
```

double **Phase.***phase_name***.Mobility.IrreducibleSaturation** [0.0]
This key specifies the irreducible saturation used in a polynomial representation of the relative permeability. Currently, only a value of 0.0 is allowed for this key.
Example Useage:

```
pfset Phase.water.Mobility.IrreducibleSaturation   0.0
```

5.1.12 Richards' Equation Relative Permeabilities

The following keys are used to describe relative permeability input for the Richards' equation implementation. They will be ignored if a full two-phase formulation is used.

string **Phase.RelPerm.Type** [no default]
This key specifies the type of relative permeability function that will be used on all specified geometries. Note that only one type of relative permeability may be used for the entire problem. However, parameters may be different for that type in different geometries. For instance, if the problem consists of three geometries, then **VanGenuchten** may be specified with three different sets of parameters for the three different goemetries. However, once **VanGenuchten** is specified, one geometry cannot later be specified to have **Data** as its relative permeability. The possible values for this key are **Constant, VanGenuchten, Haverkamp, Data,** and **Polynomial**.
Example Useage:

```
pfset Phase.RelPerm.Type   Constant
```

The various possible functions are defined as follows. The **Constant** specification means that the relative permeability will be constant on the specified geounit. The **VanGenuchten** specification means that the relative permeability will be given as a Van Genuchten function [7] with the form,

$$k_r(p) = \frac{(1 - \frac{(\alpha p)^{n-1}}{(1+(\alpha p)^n)^m})^2}{(1 + (\alpha p)^n)^{m/2}}, \tag{5.1}$$

where α and n are soil parameters and $m = 1 - 1/n$, on each region. The **Haverkamp** specification means that the relative permeability will be given in the following form [6],

$$k_r(p) = \frac{A}{A + p^\gamma}, \tag{5.2}$$

where A and γ are soil parameters, on each region. The **Data** specification is currently unsupported but will later mean that data points for the relative permeability curve will be given and ParFlow will set up the proper interpolation coefficients to get values between the given data points. The **Polynomial** specification defines a polynomial relative permeability function for each region of the form,

$$k_r(p) = \sum_{i=0}^{degree} c_i p^i. \tag{5.3}$$

list **Phase.RelPerm.GeomNames** [no default]
This key specifies the geometries on which relative permeability will be given. The union of these geometries must cover the entire computational domain.
Example Useage:

```
pfset Phase.RelPerm.Geonames   domain
```

double **Geom.*geom_name*.RelPerm.Value** [no default]
This key specifies the constant relative permeability value on the specified geometry.
Example Useage:

```
pfset Geom.domain.RelPerm.Value    0.5
```

integer **Phase.RelPerm.VanGenuchten.File** [0]
This key specifies whether soil parameters for the VanGenuchten function are specified in a pfb file or by region. The options are either 0 for specification by region, or 1 for specification in a file. Note that either all parameters are specified in files (each has their own input file) or none are specified by files. Parameters specified by files are: α and N.
Example Useage:

```
pfset Phase.RelPerm.VanGenuchten.File   1
```

string **Geom.*geom_name*.RelPerm.Alpha.Filename** [no default]
This key specifies a pfb filename containing the alpha parameters for the VanGenuchten function cell-by-cell. The ONLY option for *geom_name* is "domain".
Example Useage:

```
pfset Geom.domain.RelPerm.Alpha.Filename   alphas.pfb
```

string **Geom.***geom_name***.RelPerm.N.Filename** [no default]
This key specifies a pfb filename containing the N parameters for the VanGenuchten function cell-by-cell. The ONLY option for *geom_name* is "domain".
Example Useage:

```
pfset Geom.domain.RelPerm.N.Filename   Ns.pfb
```

double **Geom.***geom_name***.RelPerm.Alpha** [no default]
This key specifies the α parameter for the Van Genuchten function specified on *geom_name*.
Example Useage:

```
pfset Geom.domain.RelPerm.Alpha  0.005
```

double **Geom.***geom_name***.RelPerm.N** [no default]
This key specifies the N parameter for the Van Genuchten function specified on *geom_name*.
Example Useage:

```
pfset Geom.domain.RelPerm.N   2.0
```

double **Geom.***geom_name***.RelPerm.A** [no default]
This key specifies the A parameter for the Haverkamp relative permeability on *geom_name*.
Example Useage:

```
pfset Geom.domain.RelPerm.A  1.0
```

double **Geom.***geom_name***.RelPerm.Gamma** [no default]
This key specifies the the γ parameter for the Haverkamp relative permeability on *geom_name*.
Example Useage:

```
pfset Geom.domain.RelPerm.Gamma  1.0
```

integer **Geom.***geom_name***.RelPerm.Degree** [no default]
This key specifies the degree of the polynomial for the Polynomial relative permeability given on *geom_name*.
Example Useage:

```
pfset Geom.domain.RelPerm.Degree  1
```

double **Geom.***geom_name***.RelPerm.Coeff.***coeff_number* [no default]
This key specifies the *coeff_number*th coefficient of the Polynomial relative permeability given on *geom_name*.
Example Useage:

```
pfset Geom.domain.RelPerm.Coeff.0  0.5
pfset Geom.domain.RelPerm.Coeff.1  1.0
```

NOTE: For all these cases, if only one region is to be used (the domain), the background region should NOT be set as that single region. Using the background will prevent the upstream weighting from being correct near Dirichlet boundaries.

5.1.13 Phase Sources

The following keys are used to specify phase source terms. The units of the source term are $1/T$. So, for example, to specify a region with constant flux rate of L^3/T, one must be careful to convert this rate to the proper units by dividing by the volume of the enclosing region. For *Richards' equation* input, the source term must be given as a flux multiplied by density.

string **PhaseSources.*phase_name*.Type** [no default]
This key specifies the type of source to use for phase *phase_name*. Possible values for this key are **Constant** and **PredefinedFunction**. **Constant** type phase sources specify a constant phase source value for a given set of regions. **PredefinedFunction** type phase sources use a preset function (choices are listed below) to specify the source. Note that the **PredefinedFunction** type can only be used to set a single source over the entire domain and not separate sources over different regions.
Example Useage:

```
pfset PhaseSources.water.Type   Constant
```

list **PhaseSources.*phase_name*.GeomNames** [no default]
This key specifies the names of the geometries on which source terms will be specified. This is used only for **Constant** type phase sources. Regions listed later "overlay" regions listed earlier.
Example Useage:

```
pfset PhaseSources.water.GeomNames   "bottomlayer middlelayer toplayer"
```

double **PhaseSources.*phase_name*.Geom.*geom_name*.Value** [no default]
This key specifies the value of a constant source term applied to phase *phase _name* on geometry *geom_name*.
Example Useage:

```
pfset PhaseSources.water.Geom.toplayer.Value   1.0
```

string **PhaseSources.*phase_name*.PredefinedFunction** [no default]
This key specifies which of the predefined functions will be used for the source. Possible values for this key are **X, XPlusYPlusZ, X3Y2PlusSinXYPlus1, X3Y4PlusX2PlusSinXYCosYPlus1, XYZTPlus1** and **XYZTPlus1PermTensor**.
Example Useage:

```
pfset PhaseSources.water.PredefinedFunction   XPlusYPlusZ
```

The choices for this key correspond to sources as follows:

X: source $= 0.0$

XPlusYPlusX: source $= 0.0$

X3Y2PlusSinXYPlus1: source $= -(3x^2y^2 + y\cos(xy))^2 - (2x^3y + x\cos(xy))^2 - (x^3y^2 + \sin(xy) + 1)(6xy^2 + 2x^3 - (x^2 + y^2)\sin(xy))$
This function type specifies that the source applied over the entire domain is as noted above. This corresponds to $p = x^3y^2 + \sin(xy) + 1$ in the problem $-\nabla \cdot (p\nabla p) = f$.

X3Y4PlusX2PlusSinXYCosYPlus1: source $= -(3x^22y^4 + 2x + y\cos(xy)\cos(y))^2 - (4x^3y^3 + x\cos(xy)\cos(y) - \sin(xy)\sin(y))^2 - (x^3y^4 + x^2 + \sin(xy)\cos(y) + 1)(6xy^4 + 2 - (x^2 + y^2 + 1)\sin(xy)\cos(y) + 12x^3y^2 - 2x\cos(xy)\sin(y))$
This function type specifies that the source applied over the entire domain is as noted above. This corresponds to $p = x^3y^4 + x^2 + \sin(xy)\cos(y) + 1$ in the problem $-\nabla \cdot (p\nabla p) = f$.

XYZTPlus1: source $= xyz - t^2(x^2y^2 + x^2z^2 + y^2z^2)$
This function type specifies that the source applied over the entire domain is as noted above. This corresponds to $p = xyzt + 1$ in the problem $\frac{\partial p}{\partial t} - \nabla \cdot (p\nabla p) = f$.

XYZTPlus1PermTensor: source $= xyz - t^2(x^2y^23 + x^2z^22 + y^2z^2)$
This function type specifies that the source applied over the entire domain is as noted above. This corresponds to $p = xyzt + 1$ in the problem $\frac{\partial p}{\partial t} - \nabla \cdot (Kp\nabla p) = f$, where $K = diag(1\ \ 2\ \ 3)$.

5.1.14 Capillary Pressures

Here we define capillary pressure. Note: this section needs to be defined *only* for multi-phase flow and should not be defined for single phase and Richards' equation cases. The format for this section of input is:

string **CapPressure.*phase_name*.Type** ["Constant"]
This key specifies the capillary pressure between phase 0 and the named phase, *phase_name*. The only choice available is **Constant** which indicates that a constant capillary pressure exists between the phases.
Example Useage:

```
pfset CapPressure.water.Type   Constant
```

list **CapPressure.*phase_name*.GeomNames** [no default]
This key specifies the geometries that capillary pressures will be computed for in the named phase, *phase_name*. Regions listed later "overlay" regions listed earlier. Any geometries not listed will be assigned 0.0 capillary pressure by ParFlow.
Example Useage:

```
pfset CapPressure.water.GeomNames   "domain"
```

double **Geom.***geometry_name***.CapPressure.***phase_name***.Value** [0.0]
This key specifies the value of the capillary pressure in the named geometry, *geometry_name*, for the named phase, *phase_name*.
Example Useage:

```
pfset Geom.domain.CapPressure.water.Value   0.0
```

Important note: the code currently works only for capillary pressure equal zero.

5.1.15 Saturation

This section is *only* relevant to the Richards' equation cases. All keys relating to this section will be ignored for other cases. The following keys are used to define the saturation-pressure curve.

string **Phase.Saturation.Type** [no default]
This key specifies the type of saturation function that will be used on all specified geometries. Note that only one type of saturation may be used for the entire problem. However, parameters may be different for that type in different geometries. For instance, if the problem consists of three geometries, then **VanGenuchten** may be specified with three different sets of parameters for the three different goemetries. However, once **VanGenuchten** is specified, one geometry cannot later be specified to have **Data** as its saturation. The possible values for this key are **Constant, VanGenuchten, Haverkamp, Data, Polynomial** and **PFBFile**.
Example Useage:

```
pfset Phase.Saturation.Type   Constant
```

The various possible functions are defined as follows. The **Constant** specification means that the saturation will be constant on the specified geounit. The **VanGenuchten** specification means that the saturation will be given as a Van Genuchten function [7] with the form,

$$s(p) = \frac{s_{sat} - s_{res}}{(1 + (\alpha p)^n)^m} + s_{res}, \tag{5.4}$$

where s_{sat} is the saturation at saturated conditions, s_{res} is the residual saturation, and α and n are soil parameters with $m = 1 - 1/n$, on each region. The **Haverkamp** specification means that the saturation will be given in the following form [6],

$$s(p) = \frac{\alpha(s_{sat} - s_{res})}{A + p^{\gamma}} + s_{res}, \tag{5.5}$$

where A and γ are soil parameters, on each region. The **Data** specification is currently unsupported but will later mean that data points for the saturation curve will be given and ParFlow will set up the proper interpolation coefficients to get values between the given data points. The **Polynomial** specification defines a polynomial saturation function for each region of the form,

$$s(p) = \sum_{i=0}^{degree} c_i p^i. \tag{5.6}$$

The **PFBFile** specification means that the saturation will be taken as a spatially varying but constant in pressure function given by data in a ParFlow binary (.pfb) file.

list **Phase.Saturation.GeomNames** [no default]
This key specifies the geometries on which saturation will be given. The union of these geometries must cover the entire computational domain.
Example Useage:

```
pfset Phase.Saturation.Geonames   domain
```

double **Geom.*geom_name*.Saturation.Value** [no default]
This key specifies the constant saturation value on the *geom_name* region.
Example Useage:

```
pfset Geom.domain.Saturation.Value    0.5
```

integer **Phase.Saturation.VanGenuchten.File** [0]
This key specifies whether soil parameters for the VanGenuchten function are specified in a pfb file or by region. The options are either 0 for specification by region, or 1 for specification in a file. Note that either all parameters are specified in files (each has their own input file) or none are specified by files. Parameters specified by files are α, N, SRes, and SSat.
Example Useage:

```
pfset Phase.Saturation.VanGenuchten.File   1
```

string **Geom.*geom_name*.Saturation.Alpha.Filename** [no default]
This key specifies a pfb filename containing the alpha parameters for the VanGenuchten function cell-by-cell. The ONLY option for *geom_name* is “domain”.
Example Useage:

```
pfset Geom.domain.Saturation.Filename   alphas.pfb
```

string **Geom.*geom_name*.Saturation.N.Filename** [no default]
This key specifies a pfb filename containing the N parameters for the VanGenuchten function cell-by-cell. The ONLY option for *geom_name* is “domain”.
Example Useage:

```
pfset Geom.domain.Saturation.N.Filename   Ns.pfb
```

string **Geom.***geom_name***.Saturation.SRes.Filename** [no default]
This key specifies a pfb filename containing the SRes parameters for the VanGenuchten function cell-by-cell. The ONLY option for *geom_name* is “domain”.
Example Useage:

```
pfset Geom.domain.Saturation.SRes.Filename   SRess.pfb
```

string **Geom.***geom_name***.Saturation.SSat.Filename** [no default]
This key specifies a pfb filename containing the SSat parameters for the VanGenuchten function cell-by-cell. The ONLY option for *geom_name* is “domain”.
Example Useage:

```
pfset Geom.domain.Saturation.SSat.Filename   SSats.pfb
```

double **Geom.***geom_name***.Saturation.Alpha** [no default]
This key specifies the α parameter for the Van Genuchten function specified on *geom_name*.
Example Useage:

```
pfset Geom.domain.Saturation.Alpha  0.005
```

double **Geom.***geom_name***.Saturation.N** [no default]
This key specifies the N parameter for the Van Genuchten function specified on *geom_name*.
Example Useage:

```
pfset Geom.domain.Saturation.N   2.0
```

Note that if both a Van Genuchten saturation and relative permeability are specified, then the soil parameters should be the same for each in order to have a consistent problem.

double **Geom.***geom_name***.Saturation.SRes** [no default]
This key specifies the residual saturation on *geom_name*.
Example Useage:

```
pfset Geom.domain.Saturation.SRes   0.0
```

double **Geom.***geom_name***.Saturation.SSat** [no default]
This key specifies the saturation at saturated conditions on *geom_name*.
Example Useage:

```
pfset Geom.domain.Saturation.SSat   1.0
```

double **Geom.***geom_name***.Saturation.A** [no default]
This key specifies the A parameter for the Haverkamp saturation on *geom_name*.
Example Useage:

```
pfset Geom.domain.Saturation.A   1.0
```

double **Geom.*geom_name*.Saturation.Gamma** [no default]
This key specifies the the γ parameter for the Haverkamp saturation on *geom_name*.
Example Useage:

```
pfset Geom.domain.Saturation.Gamma   1.0
```

integer **Geom.*geom_name*.Saturation.Degree** [no default]
This key specifies the degree of the polynomial for the Polynomial saturation given on *geom_name*.

Example Useage:

```
pfset Geom.domain.Saturation.Degree   1
```

double **Geom.*geom_name*.Saturation.Coeff.*coeff_number*** [no default]
This key specifies the *coeff_number*th coefficient of the Polynomial saturation given on *geom_name*.

Example Useage:

```
pfset Geom.domain.Saturation.Coeff.0   0.5
pfset Geom.domain.Saturation.Coeff.1   1.0
```

string **Geom.*geom_name*.Saturation.FileName** [no default]
This key specifies the name of the file containing saturation values for the domain. It is assumed that *geom_name* is "domain" for this key.
Example Useage:

```
pfset Geom.domain.Saturation.FileName  "domain_sats.pfb"
```

5.1.16 Internal Boundary Conditions

In this section, we define internal Dirichlet boundary conditions by setting the pressure at points in the domain. The format for this section of input is:

string **InternalBC.Names** [no default]
This key specifies the names for the internal boundary conditions. At each named point, x, y and z will specify the coordinate locations and h will specify the hydraulic head value of the condition. This real location is "snapped" to the nearest gridpoint in ParFlow.
NOTE: Currently, ParFlow assumes that internal boundary conditions and pressure wells are separated by at least one cell from any external boundary. The user should be careful of this when defining the input file and grid.
Example Useage:

```
pfset InternalBC.Names   "fixedvalue"
```

double **InternalBC.*internal_bc_name*.X** [no default]
This key specifies the x-coordinate, x, of the named, *internal_bc_name*, condition.
Example Useage:

```
pfset InternalBC.fixedheadvalue.X   40.0
```

double **InternalBC.***internal_bc_name***.Y** [no default]
This key specifies the y-coordinate, y, of the named, *internal_bc_name*, condition.
Example Useage:

```
pfset InternalBC.fixedheadvalue.Y   65.2
```

double **InternalBC.***internal_bc_name***.Z** [no default]
This key specifies the z-coordinate, z, of the named, *internal_bc_name*, condition.
Example Useage:

```
pfset InternalBC.fixedheadvalue.Z   12.1
```

double **InternalBC.***internal_bc_name***.Value** [no default]
This key specifies the value of the named, *internal_bc_name*, condition.
Example Useage:

```
pfset InternalBC.fixedheadvalue.Value   100.0
```

5.1.17 Boundary Conditions: Pressure

Here we define the pressure boundary conditions. The Dirichlet conditions below are hydrostatic conditions, and it is assumed that at each phase interface the pressure is constant. *It is also assumed here that all phases are distributed within the domain at all times such that the lighter phases are vertically higher than the heavier phases.*

Boundary condition input is associated with domain patches (see § 5.1.4). Note that different patches may have different types of boundary conditions on them.

list **BCPressure.PatchNames** [no default]
This key specifies the names of patches on which pressure boundary conditions will be specified. Note that these must all be patches on the external boundary of the domain and these patches must “cover” that external boundary.
Example Useage:

```
pfset BCPressure.PatchNames    "left right front back top bottom"
```

string **Patch.***patch_name***.BCPressure.Type** [no default]
This key specifies the type of boundary condition data given for patch *patch_name*. Possible values for this key are **DirEquilRefPatch, DirEquilPLinear, FluxConst, FluxVolumetric, PressureFile, FluxFile** and **ExactSolution**. The choice **DirEquilRefPatch** specifies that the pressure on the specified patch will be in hydrostatic equilibrium with a constant reference pressure given on a reference patch. The choice **DirEquilPLinear** specifies that the pressure on the specified patch will be in hydrostatic equilibrium with pressure given along a piecewise line at elevation

$z = 0$. The choice **FluxConst** defines a constant normal flux boundary condition through the domain patch. This flux must be specified in units of $[L]/[T]$. For *Richards' equation*, fluxes must be specified as a mass flux and given as the above flux multiplied by the density. Thus, this choice of input type for a Richards' equation problem has units of $([L]/[T])([M]/[L]^3)$. The choice **FluxVolumetric** defines a volumetric flux boundary condition through the domain patch. The units should be consistent with all other user input for the problem. For *Richards' equation* fluxes must be specified as a mass flux and given as the above flux multiplied by the density. The choice **PressureFile** defines a hydraulic head boundary condition that is read from a properly distributed .pfb file. Only the values needed for the patch are used. The choice **FluxFile** defines a flux boundary condition that is read form a properly distributed .pfb file defined on a grid consistent with the pressure field grid. Only the values needed for the patch are used. The choice **ExactSolution** specifies that an exact known solution is to be applied as a Dirichlet boundary condition on the respective patch. Note that this does not change according to any cycle. Instead, time dependence is handled by evaluating at the time the boundary condition value is desired. The solution is specified by using a predefined function (choices are described below). NOTE: This type of boundary condition input is for *Richards' equation cases only!*
Example Useage:

```
pfset Patch.top.BCPressure.Type  DirEquilRefPatch
```

string **Patch.*patch_name*.BCPressure.Cycle** [no default]
This key specifies the time cycle to which boundary condition data for patch *patch_name* corresponds.
Example Useage:

```
pfset Patch.top.BCPressure.Cycle   Constant
```

string **Patch.*patch_name*.BCPressure.RefGeom** [no default]
This key specifies the name of the solid on which the reference patch for the **DirEquilRefPatch** boundary condition data is given. Care should be taken to make sure the correct solid is specified in cases of layered domains.
Example Useage:

```
pfset Patch.top.BCPressure.RefGeom   domain
```

string **Patch.*patch_name*.BCPressure.RefPatch** [no default]
This key specifies the reference patch on which the **DirEquilRefPatch** boundary condition data is given. This patch must be on the reference solid specified by the Patch.*patch_name*.BCPressure.RefGeom key.
Example Useage:

```
pfset Patch.top.BCPressure.RefPatch    bottom
```

double **Patch.*patch_name*.BCPressure.*interval_name*.Value** [no default]
This key specifies the reference pressure value for the **DirEquilRefPatch** boundary condition

or the constant flux value for the **FluxConst** boundary condition, or the constant volumetric flux for the **FluxVolumetric** boundary condition.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.Value  -14.0
```

double **Patch.***patch_name***.BCPressure.***interval_name***.***phase_name***.IntValue** [no default]
Note that the reference conditions for types **DirEquilPLinear** and **DirEquilRefPatch** boundary conditions are for phase 0 *only*. This key specifies the constant pressure value along the interface with phase *phase_name* for cases with two phases present.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.water.IntValue   -13.0
```

double **Patch.***patch_name***.BCPressure.***interval_name***.XLower** [no default]
This key specifies the lower x coordinate of a line in the xy-plane.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.XLower  0.0
```

double **Patch.***patch_name***.BCPressure.***interval_name***.YLower** [no default]
This key specifies the lower y coordinate of a line in the xy-plane.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.YLower  0.0
```

double **Patch.***patch_name***.BCPressure.***interval_name***.XUpper** [no default]
This key specifies the upper x coordinate of a line in the xy-plane.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.XUpper  1.0
```

double **Patch.***patch_name***.BCPressure.***interval_name***.YUpper** [no default]
This key specifies the upper y coordinate of a line in the xy-plane.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.YUpper  1.0
```

integer **Patch.***patch_name***.BCPressure.***interval_name***.NumPoints** [no default]
This key specifies the number of points on which pressure data is given along the line used in the type **DirEquilPLinear** boundary conditions.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.NumPoints   2
```

double **Patch.***patch_name***.BCPressure.***interval_name.point_number***.Location** [no default]
This key specifies a number between 0 and 1 which represents the location of a point on the line on which data is given for type **DirEquilPLinear** boundary conditions. Here 0 corresponds to the lower end of the line, and 1 corresponds to the upper end.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.0.Location   0.0
```

double **Patch.***patch_name***.BCPressure.***interval_name.point_number***.Value** [no default]
This key specifies the pressure value for phase 0 at point number *point_number* and $z = 0$ for type **DirEquilPLinear** boundary conditions. All pressure values on the patch are determined by first projecting the boundary condition coordinate onto the line, then linearly interpolating between the neighboring point pressure values on the line.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.0.Value   14.0
```

string **Patch.***patch_name***.BCPressure.***interval_name***.FileName** [no default]
This key specifies the name of a properly distributed `.pfb` file that contains boundary data to be read for types **PressureFile** and **FluxFile**. For flux data, the data must be defined over a grid consistent with the pressure field. In both cases, only the values needed for the patch will be used. The rest of the data is ignored.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.FileName   ocwd_bc.pfb
```

string **Patch.***patch_name***.BCPressure.***interval_name***.PredefinedFunction** [no default]
This key specifies the predefined function that will be used to specify Dirichlet boundary conditions on patch *patch_name*. Note that this does not change according to any cycle. Instead, time dependence is handled by evaluating at the time the boundary condition value is desired. Choices for this key include **X, XPlusYPlusZ, X3Y2PlusSinXYPlus1, X3Y4PlusX2PlusSinXYCosYPlus1, XYZTPlus1** and **XYZTPlus1PermTensor**.
Example Useage:

```
pfset Patch.top.BCPressure.alltime.PredefinedFunction  XPlusYPlusZ
```

The choices for this key correspond to pressures as follows.

X: $p = x$

XPlusYPlusZ: $p = x + y + z$

X3Y2PlusSinXYPlus1: $p = x^3y^2 + \sin(xy) + 1$

X3Y4PlusX2PlusSinXYCosYPlus1: $p = x^3y^4 + x^2 + \sin(xy)\cos y + 1$

XYZTPlus1: $p = xyzt + 1$

XYZTPlus1PermTensor: $p = xyzt + 1$

5.1.18 Boundary Conditions: Saturation

Note: this section needs to be defined *only* for multi-phase flow and should *not* be defined for the single phase and Richards' equation cases.

Here we define the boundary conditions for the saturations. Boundary condition input is associated with domain patches (see § 5.1.4). Note that different patches may have different types of boundary conditions on them.

list **BCSaturation.PatchNames** [no default]
This key specifies the names of patches on which saturation boundary conditions will be specified. Note that these must all be patches on the external boundary of the domain and these patches must "cover" that external boundary.
Example Useage:

```
pfset BCSaturation.PatchNames    "left right front back top bottom"
```

string **Patch.***patch_name***.BCSaturation.***phase_name***.Type** [no default]
This key specifies the type of boundary condition data given for the given phase, *phase_name*, on the given patch *patch_name*. Possible values for this key are **DirConstant**, **ConstantWTHeight** and **PLinearWTHeight**. The choice **DirConstant** specifies that the saturation is constant on the whole patch. The choice **ConstantWTHeight** specifies a constant height of the water-table on the whole patch. The choice **PLinearWTHeight** specifies that the height of the water-table on the patch will be given by a piecewise linear function.

Note: the types **ConstantWTHeight** and **PLinearWTHeight** assume we are running a 2-phase problem where phase 0 is the water phase.
Example Useage:

```
pfset Patch.left.BCSaturation.water.Type  ConstantWTHeight
```

double **Patch.***patch_name***.BCSaturation.***phase_name***.Value** [no default]
This key specifies either the constant saturation value if **DirConstant** is selected or the constant water-table height if **ConstantWTHeight** is selected.
Example Useage:

```
pfset Patch.top.BCSaturation.air.Value 1.0
```

double **Patch.***patch_name***.BCSaturation.***phase_name***.XLower** [no default]
This key specifies the lower x coordinate of a line in the xy-plane if type **PLinearWTHeight** boundary conditions are specified.
Example Useage:

```
pfset Patch.left.BCSaturation.water.XLower -10.0
```

double **Patch.***patch_name***.BCSaturation.***phase_name***.YLower** [no default]
This key specifies the lower y coordinate of a line in the xy-plane if type **PLinearWTHeight**

boundary conditions are specified.
Example Useage:

```
pfset Patch.left.BCSaturation.water.YLower 5.0
```

double **Patch.***patch_name***.BCSaturation.***phase_name***.XUpper** [no default]
This key specifies the upper x coordinate of a line in the xy-plane if type **PLinearWTHeight** boundary conditions are specified.
Example Useage:

```
pfset Patch.left.BCSaturation.water.XUpper  125.0
```

double **Patch.***patch_name***.BCSaturation.***phase_name***.YUpper** [no default]
This key specifies the upper y coordinate of a line in the xy-plane if type **PLinearWTHeight** boundary conditions are specified.
Example Useage:

```
pfset Patch.left.BCSaturation.water.YUpper  82.0
```

integer **Patch.***patch_name***.BCPressure.***phase_name***.NumPoints** [no default]
This key specifies the number of points on which saturation data is given along the line used for type **DirEquilPLinear** boundary conditions.
Example Useage:

```
pfset Patch.left.BCPressure.water.NumPoints 2
```

double **Patch.***patch_name***.BCPressure.***phase_name***.***point_number***.Location** [no default]
This key specifies a number between 0 and 1 which represents the location of a point on the line for which data is given in type **DirEquilPLinear** boundary conditions. The line is parameterized so that 0 corresponds to the lower end of the line, and 1 corresponds to the upper end.
Example Useage:

```
pfset Patch.left.BCPressure.water.0.Location 0.333
```

double **Patch.***patch_name***.BCPressure.***phase_name***.***point_number***.Value** [no default]
This key specifies the water-table height for the given point if type **DirEquilPLinear** boundary conditions are selected. All saturation values on the patch are determined by first projecting the water-table height value onto the line, then linearly interpolating between the neighboring water-table height values onto the line.
Example Useage:

```
pfset Patch.left.BCPressure.water.0.Value  4.5
```

5.1.19 Initial Conditions: Phase Saturations

Note: this section needs to be defined *only* for multi-phase flow and should *not* be defined for single phase and Richards' equation cases.

Here we define initial phase saturation conditions. The format for this section of input is:

string **ICSaturation.***phase_name***.Type** [no default]
This key specifies the type of initial condition that will be applied to different geometries for given phase, *phase_name*. The only key currently available is **Constant**. The choice **Constant** will apply constants values within geometries for the phase.
Example Useage:

```
ICSaturation.water.Type Constant
```

string **ICSaturation.***phase_name***.GeomNames** [no default]
This key specifies the geometries on which an initial condition will be given if the type is set to **Constant**.
Note that geometries listed later "overlay" geometries listed earlier.
Example Useage:

```
ICSaturation.water.GeomNames "domain"
```

double **Geom.***geom_input_name***.ICSaturation.***phase_name***.Value** [no default]
This key specifies the initial condition value assigned to all points in the named geometry, *geom_input_name*, if the type was set to **Constant**.
Example Useage:

```
Geom.domain.ICSaturation.water.Value 1.0
```

5.1.20 Initial Conditions: Pressure

The keys in this section are used to specify pressure initial conditions for Richards' equation cases *only*. These keys will be ignored if any other case is run.

string **ICPressure.Type** [no default]
This key specifies the type of initial condition given. The choices for this key are **Constant, HydroStaticDepth, HydroStaticPatch** and **PFBFile**. The choice **Constant** specifies that the initial pressure will be constant over the regions given. The choice **HydroStaticDepth** specifies that the initial pressure within a region will be in hydrostatic equilibrium with a given pressure specified at a given depth. The choice **HydroStaticPatch** specifies that the initial pressure within a region will be in hydrostatic equilibrium with a given pressure on a specified patch. Note that all regions must have the same type of initial data - different regions cannot have different types of initial data. However, the parameters for the type may be different. The **PFBFile** specification means that the initial pressure will be taken as a spatially varying function given by data in a ParFlow binary (.pfb) file.

Example Useage:

```
pfset ICPressure.Type   Constant
```

list **ICPressure.GeomNames** [no default]
This key specifies the geometry names on which the initial pressure data will be given. These geometries must comprise the entire domain. Note that conditions for regions that overlap other regions will have unpredictable results. The regions given must be disjoint.
Example Useage:

```
pfset ICPressure.GeomNames   "toplayer middlelayer bottomlayer"
```

double **Geom.*geom_name*.ICPressure.Value** [no default]
This key specifies the initial pressure value for type **Constant** initial pressures and the reference pressure value for types **HydroStaticDepth** and **HydroStaticPatch**.
Example Useage:

```
pfset Geom.toplayer.ICPressure.Value  -734.0
```

double **Geom.*geom_name*.ICPressure.RefElevation** [no default]
This key specifies the reference elevation on which the reference pressure is given for type **HydroStaticDepth** initial pressures.
Example Useage:

```
pfset Geom.toplayer.ICPressure.RefElevation  0.0
```

double **Geom.*geom_name*.ICPressure.RefGeom** [no default]
This key specifies the geometry on which the reference patch resides for type **HydroStatic-Patch** initial pressures.
Example Useage:

```
pfset Geom.toplayer.ICPressure.RefGeom   bottomlayer
```

double **Geom.*geom_name*.ICPressure.RefPatch** [no default]
This key specifies the patch on which the reference pressure is given for type **HydorStatic-Patch** initial pressures.
Example Useage:

```
pfset Geom.toplayer.ICPressure.RefPatch   bottom
```

string **Geom.*geom_name*.ICPressure.FileName** [no default]
This key specifies the name of the file containing pressure values for the domain. It is assumed that *geom_name* is “domain” for this key.
Example Useage:

```
pfset Geom.domain.ICPressure.FileName  "ic_pressure.pfb"
```

5.1.21 Initial Conditions: Phase Concentrations

Here we define initial concentration conditions for contaminants. The format for this section of input is:

string **PhaseConcen.***phase_name***.***contaminant_name***.Type** [no default]
This key specifies the type of initial condition that will be applied to different geometries for given phase, *phase_name*, and the given contaminant, *contaminant_name*. The choices for this key are **Constant** or **PFBFile**. The choice **Constant** will apply constants values to different geometries. The choice **PFBFile** will read values from a "ParFlow Binary" file (see § 5.2).
Example Useage:

```
PhaseConcen.water.tce.Type Constant
```

string **PhaseConcen.***phase_name***.GeomNames** [no default]
This key specifies the geometries on which an initial condition will be given, if the type was set to **Constant**.
Note that geometries listed later "overlay" geometries listed earlier.
Example Useage:

```
PhaseConcen.water.GeomNames "ic_concen_region"
```

double **PhaseConcen.***phase_name***.***contaminant_name***.***geom_input_name***.Value** [no default]
This key specifies the initial condition value assigned to all points in the named geometry, *geom_input_name*, if the type was set to **Constant**.
Example Useage:

```
PhaseConcen.water.tce.ic_concen_region.Value 0.001
```

string **PhaseConcen.***phase_name***.***contaminant_name***.FileName** [no default]
This key specifies the name of the "ParFlow Binary" file which contains the initial condition values if the type was set to **PFBFile**.
Example Useage:

```
PhaseConcen.water.tce.FileName "initial_concen_tce.pfb"
```

5.1.22 Known Exact Solution

For *Richards equation cases only* we allow specification of an exact solution to be used for testing the code. Only types that have been coded and predefined are allowed. Note that if this is speccified as something other than no known solution, corresponding boundary conditions and phase sources should also be specified.

string **KnownSolution** [no default]
This specifies the predefined function that will be used as the known solution. Possible choices

for this key are **NoKnownSolution, Constant, X, XPlusYPlusZ, X3Y2PlusSinXYPlus1, X3Y4PlusX2PlusSinXYCosYPlus1, XYZTPlus1** and **XYZTPlus1PermTensor**.
Example Useage:

```
pfset KnownSolution  XPlusYPlusZ
```

Choices for this key correspond to solutions as follows.

NoKnownSolution: No solution is known for this problem.

Constant: $p = \text{constant}$

X: $p = x$

XPlusYPlusZ: $p = x + y + z$

X3Y2PlusSinXYPlus1: $p = x^3y^2 + sin(xy) + 1$

X3Y4PlusX2PlusSinXYCosYPlus1: $p = x^3y^4 + x^2 + \sin(xy)\cos y + 1$

XYZTPlus1: $p = xyzt + 1$

XYZTPlus1: $p = xyzt + 1$

double **KnownSolution.Value** [no default]
This key specifies the constant value of the known solution for type **Constant** known solutions.
Example Useage:

```
pfset KnownSolution.Value  1.0
```

Only for known solution test cases will information on the L^2-norm of the pressure error be printed.

5.1.23 Wells

Here we define wells for the model. The format for this section of input is:

string **Wells.Names** [no default]
This key specifies the names of the wells for which input data will be given.
Example Useage:

```
Wells.Names "test_well inj_well ext_well"
```

string **Wells.***well_name***.InputType** [no default]
This key specifies the type of well to be defined for the given well, *well_name*. This key can be either **Vertical** or **Recirc**. The value **Vertical** indicates that this is a single segmented well whose action will be specified by the user. The value **Recirc** indicates that this is a dual segmented, recirculating, well with one segment being an extraction well and another being an injection well.

The extraction well filters out a specified fraction of each contaminant and recirculates the remainder to the injection well where the diluted fluid is injected back in. The phase saturations at the extraction well are passed without modification to the injection well.

Note with the recirculating well, several input options are not needed as the extraction well will provide these values to the injection well.
Example Useage:

```
Wells.test_well.InputType Vertical
```

string **Wells.***well_name***.Action** [no default]
This key specifies the pumping action of the well. This key can be either **Injection** or **Extraction**. A value of **Injection** indicates that this is an injection well. A value of **Extraction** indicates that this is an extraction well.
Example Useage:

```
Wells.test_well.Action Injection
```

double **Wells.***well_name***.Type** [no default]
This key specfies the mechanism by which the well works (how ParFlow works with the well data) if the input type key is set to **Vectical**. This key can be either **Pressure** or **Flux**. A value of **Pressure** indicates that the data provided for the well is in terms of hydrostatic pressure and ParFlow will ensure that the computed pressure field satisfies this condition in the computational cells which define the well. A value of **Flux** indicates that the data provided is in terms of volumetric flux rates and ParFlow will ensure that the flux field satisfies this condition in the computational cells which define the well.
Example Useage:

```
Wells.test_well.Type Flux
```

string **Wells.***well_name***.ExtractionType** [no default]
This key specfies the mechanism by which the extraction well works (how ParFlow works with the well data) if the input type key is set to **Recirc**. This key can be either **Pressure** or **Flux**. A value of **Pressure** indicates that the data provided for the well is in terms of hydrostatic pressure and ParFlow will ensure that the computed pressure field satisfies this condition in the computational cells which define the well. A value of **Flux** indicates that the data provided is in terms of volumetric flux rates and ParFlow will ensure that the flux field satisfies this condition in the computational cells which define the well.
Example Useage:

```
Wells.ext_well.ExtractionType Pressure
```

string **Wells.***well_name***.InjectionType** [no default]
This key specfies the mechanism by which the injection well works (how ParFlow works with the well data) if the input type key is set to **Recirc**. This key can be either **Pressure** or

Flux. A value of **Pressure** indicates that the data provided for the well is in terms of hydrostatic pressure and ParFlow will ensure that the computed pressure field satisfies this condition in the computational cells which define the well. A value of **Flux** indicates that the data provided is in terms of volumetric flux rates and ParFlow will ensure that the flux field satisfies this condition in the computational cells which define the well.
Example Useage:

```
Wells.inj_well.InjectionType Flux
```

double **Wells.***well_name***.X** [no default]
This key specifies the x location of the vectical well if the input type is set to **Vectical** or of both the extraction and injection wells if the input type is set to **Recirc**.
Example Useage:

```
Wells.test_well.X 20.0
```

double **Wells.***well_name***.Y** [no default]
This key specifies the y location of the vectical well if the input type is set to **Vectical** or of both the extraction and injection wells if the input type is set to **Recirc**.
Example Useage:

```
Wells.test_well.Y 36.5
```

double **Wells.***well_name***.ZUpper** [no default]
This key specifies the z location of the upper extent of a vectical well if the input type is set to **Vectical**.
Example Useage:

```
Wells.test_well.ZUpper 8.0
```

double **Wells.***well_name***.ExtractionZUpper** [no default]
This key specifies the z location of the upper extent of a extraction well if the input type is set to **Recirc**.
Example Useage:

```
Wells.ext_well.ExtractionZUpper 3.0
```

double **Wells.***well_name***.InjectionZUpper** [no default]
This key specifies the z location of the upper extent of a injection well if the input type is set to **Recirc**.
Example Useage:

```
Wells.inj_well.InjectionZUpper 6.0
```

double **Wells.***well_name***.ZLower** [no default]
This key specifies the z location of the lower extent of a vectical well if the input type is set to **Vectical**.
Example Useage:

```
Wells.test_well.ZLower 2.0
```

double **Wells.***well_name***.ExtractionZLower** [no default]
This key specifies the z location of the lower extent of a extraction well if the input type is set to **Recirc**.
Example Useage:

```
Wells.ext_well.ExtractionZLower 1.0
```

double **Wells.***well_name***.InjectionZLower** [no default]
This key specifies the z location of the lower extent of a injection well if the input type is set to **Recirc**.
Example Useage:

```
Wells.inj_well.InjectionZLower 4.0
```

string **Wells.***well_name***.Method** [no default]
This key specifies a method by which pressure or flux for a vertical well will be weighted before assignment to computational cells. This key can only be **Standard** if the type key is set to **Pressure**; or this key can be either **Standard**, **Weighted** or **Patterned** if the type key is set to **Flux**. A value of **Standard** indicates that the pressure or flux data will be used as is. A value of **Weighted** indicates that the flux data is to be weighted by the cells permeability divided by the sum of all cell permeabilities which define the well. The value of **Patterned** is not implemented.
Example Useage:

```
Wells.test_well.Method Weighted
```

string **Wells.***well_name***.ExtractionMethod** [no default]
This key specifies a method by which pressure or flux for an extraction well will be weighted before assignment to computational cells. This key can only be **Standard** if the type key is set to **Pressure**; or this key can be either **Standard**, **Weighted** or **Patterned** if the type key is set to **Flux**. A value of **Standard** indicates that the pressure or flux data will be used as is. A value of **Weighted** indicates that the flux data is to be weighted by the cells permeability divided by the sum of all cell permeabilities which define the well. The value of **Patterned** is not implemented.
Example Useage:

```
Wells.ext_well.ExtractionMethod Standard
```

string **Wells.***well_name***.InjectionMethod** [no default]
This key specifies a method by which pressure or flux for an injection well will be weighted

before assignment to computational cells. This key can only be **Standard** if the type key is set to **Pressure**; or this key can be either **Standard**, **Weighted** or **Patterned** if the type key is set to **Flux**. A value of **Standard** indicates that the pressure or flux data will be used as is. A value of **Weighted** indicates that the flux data is to be weighted by the cells permeability divided by the sum of all cell permeabilities which define the well. The value of **Patterned** is not implemented.
Example Useage:

```
Wells.inj_well.InjectionMethod Standard
```

string **Wells.***well_name***.Cycle** [no default]
This key specifies the time cycles to which data for the well *well_name* corresponds.
Example Useage:

```
Wells.test_well.Cycle "all_time"
```

double **Wells.***well_name.interval_name***.Pressure.Value** [no default]
This key specifies the hydrostatic pressure value for a vectical well if the type key is set to **Pressure**.
Note This value gives the pressure of the primary phase (water) at $z = 0$. The other phase pressures (if any) are computed from the physical relationships that exist between the phases.
Example Useage:

```
Wells.test_well.all_time.Pressure.Value 6.0
```

double **Wells.***well_name.interval_name***.Extraction.Pressure.Value** [no default]
This key specifies the hydrostatic pressure value for an extraction well if the extraction type key is set to **Pressure**.
Note This value gives the pressure of the primary phase (water) at $z = 0$. The other phase pressures (if any) are computed from the physical relationships that exist between the phases.
Example Useage:

```
Wells.ext_well.all_time.Extraction.Pressure.Value 4.5
```

double **Wells.***well_name.interval_name***.Injection.Pressure.Value** [no default]
This key specifies the hydrostatic pressure value for an injection well if the injection type key is set to **Pressure**.
Note This value gives the pressure of the primary phase (water) at $z = 0$. The other phase pressures (if any) are computed from the physical relationships that exist between the phases.
Example Useage:

```
Wells.inj_well.all_time.Injection.Pressure.Value 10.2
```

double **Wells.***well_name.interval_name***.Flux.***phase_name***.Value** [no default]
This key specifies the volumetric flux for a vectical well if the type key is set to **Flux**.

Note only a positive number should be entered, ParFlow assignes the correct sign based on the chosen action for the well.
Example Useage:

```
Wells.test_well.all_time.Flux.water.Value 250.0
```

double **Wells.***well_name*.*interval_name***.Extraction.Flux.***phase_name***.Value** [no default]
This key specifies the volumetric flux for an extraction well if the extraction type key is set to **Flux**.
Note only a positive number should be entered, ParFlow assignes the correct sign based on the chosen action for the well.
Example Useage:

```
Wells.ext_well.all_time.Extraction.Flux.water.Value 125.0
```

double **Wells.***well_name*.*interval_name***.Injection.Flux.***phase_name***.Value** [no default]
This key specifies the volumetric flux for an injection well if the injection type key is set to **Flux**.
Note only a positive number should be entered, ParFlow assignes the correct sign based on the chosen action for the well.
Example Useage:

```
Wells.inj_well.all_time.Injection.Flux.water.Value 80.0
```

double **Wells.***well_name*.*interval_name***.Saturation.***phase_name***.Value** [no default]
This key specifies the saturation value of a vertical well.
Example Useage:

```
Wells.test_well.all_time.Saturation.water.Value 1.0
```

double **Wells.***well_name*.*interval_name***.Concentration.***phase_name*.*contaminant_name***.Value** [no default]
This key specifies the contaminant value of a vertical well.
Example Useage:

```
Wells.test_well.all_time.Concentration.water.tce.Value 0.0005
```

double **Wells.***well_name*.*interval_name***.Injection.Concentration.***phase_name*.*contaminant_name***.Fractic**
[no default]
This key specifies the fraction of the extracted contaminant which gets resupplied to the injection well.
Example Useage:

```
Wells.inj_well.all_time.Injection.Concentration.water.tce.Fraction 0.01
```

Multiple wells assigned to one grid location can occur in several instances. The current actions taken by the code are as follows:

- If multiple pressure wells are assigned to one grid cell, the code retains only the last set of overlapping well values entered.
- If multiple flux wells are assigned to one grid cell, the code sums the contributions of all overlapping wells to get one effective well flux.
- If multiple pressure and flux wells are assigned to one grid cell, the code retains the last set of overlapping hydrostatic pressure values entered and sums all the overlapping flux well values to get an effective pressure/flux well value.

5.1.24 Code Parameters

In addition to input keys related to the physics capabilities and modeling specifics there are some key values used by various algorithms and general control flags for ParFlow. These are described next :

integer **Solver.SadvectOrder** [2]
This key controls the order of the explicit method used in advancing the saturations. This value can be either 1 for a standard upwind first order or 2 for a second order Godunov method.
Example Useage:

```
Solver.SadvectOrder 1
```

integer **Solver.AdvectOrder** [2]
This key controls the order of the explicit method used in advancing the concentrations. This value can be either 1 for a standard upwind first order or 2 for a second order Godunov method.
Example Useage:

```
Solver.AdvectOrder 2
```

double **Solver.CFL** [0.7]
This key gives the value of the weight put on the computed CFL limit before computing a global timestep value. Values greater than 1 are not suggested and in fact because this is an approximation, values slightly less than 1 can also produce instabilities.
Example Useage:

```
Solver.CFL 0.7
```

integer **Solver.MaxIter** [1000000]
This key gives the maximum number of iterations that will be allowed for time-stepping. This is to prevent a run-away simulation.
Example Useage:

```
Solver.MaxIter 100
```

double **Solver.RelTol** [1.0]
This value gives the relative tolerance for the linear solve algorithm.
Example Useage:

```
Solver.RelTol 1.0
```

double **Solver.AbsTol** [1E-9]
This value gives the absolute tolerance for the linear solve algorithm.
Example Useage:

```
Solver.AbsTol 1E-8
```

double **Solver.Drop** [1E-8]
This key gives a clipping value for data written to PFSB files. Data values greater than the negative of this value and less than the value itself are treated as zero and not written to PFSB files.
Example Useage:

```
Solver.Drop 1E-6
```

string **Solver.PrintSubsurf** [True]
This key is used to turn on printing of the subsurface data, Permeability and Porosity. The data is printed after it is generated and before the main time stepping loop - only once during the run. The data is written as a PFB file.
Example Useage:

```
Solver.PrintSubsurf False
```

string **Solver.PrintPressure** [True]
This key is used to turn on printing of the pressure data. The printing of the data is controlled by values in the timing information section. The data is written as a PFB file.
Example Useage:

```
Solver.PrintPressure False
```

string **Solver.PrintVelocities** [False]
This key is used to turn on printing of the x, y and z velocity data. The printing of the data is controlled by values in the timing information section. The data is written as a PFB file.
Example Useage:

```
Solver.PrintVelocities True
```

string **Solver.PrintSaturation** [True]
This key is used to turn on printing of the saturation data. The printing of the data is controlled by values in the timing information section. The data is written as a PFB file.
Example Useage:

```
Solver.PrintSaturation False
```

string **Solver.PrintConcentration** [True]
This key is used to turn on printing of the concentration data. The printing of the data is controlled by values in the timing information section. The data is written as a PFSB file.
Example Useage:

```
Solver.PrintConcentration False
```

string **Solver.PrintWells** [True]
This key is used to turn on collection and printing of the well data. The data is collected at intervals given by values in the timing information section. Printing occurs at the end of the run when all collected data is written.
Example Useage:

```
Solver.PrintWells False
```

5.1.25 Richards' Equation Solver Parameters

The following keys are used to specify various parameters used by the linear and nonlinear solvers in the Richards' equation implementation. For information about these solvers, see [12] and [1].

double **Solver.Nonlinear.ResidualTol** [1e-7]
This key specifies the tolerance that measures how much the relative reduction in the nonlinear residual should be before nonlinear iterations stop. The magnitude of the residual is measured with the l^1 (max) norm.
Example Useage:

```
pfset Solver.Nonlinear.ResidualTol   1e-4
```

double **Solver.Nonlinear.StepTol** [1e-7]
This key specifies the tolerance that measures how small the difference between two consecutive nonlinear steps can be before nonlinear iterations stop.
Example Useage:

```
pfset Solver.Nonlinear.StepTol   1e-4
```

integer **Solver.Nonlinear.MaxIter** [15]
This key specifies the maximum number of nonlinear iterations allowed before iterations stop with a convergence failure.
Example Useage:

```
pfset Solver.Nonlinear.MaxIter   50
```

integer **Solver.Linear.KrylovDimension** [10]
This key specifies the maximum number of vectors to be used in setting up the Krylov subspace in the GMRES iterative solver. These vectors are of problem size and it should be noted that large increases in this parameter can limit problem sizes. However, increasing this parameter can sometimes help nonlinear solver convergence.
Example Useage:

```
pfset Solver.Linear.KrylovDimension   15
```

integer **Solver.Linear.MaxRestarts** [0]
This key specifies the number of restarts allowed to the GMRES solver. Restarts start the development of the Krylov subspace over using the current iterate as the initial iterate for the next pass.
Example Useage:

```
pfset Solver.Linear.MaxRestarts   2
```

string **Solver.Nonlinear.PrintFlag** [HighVerbosity]
This key specifies the amount of informational data that is printed to the `*.out.kinsol.log` file. Choices for this key are **NoVerbosity, LowVerbosity, NormalVerbosity** and **HighVerbosity**. The choice **NoVerbosity** prints no statistics about the nonlinear convergence process. The choice **LowVerbosity** outputs the nonlinear iteration count, the scaled norm of the nonlinear function, and the number of function calls. The choice **NormalVerbosity** prints the same as for **LowVerbosity** and also the global strategy statistics. The choice **HighVerbosity** prints the same as for **NormalVerbosity** with the addition of further Krylov iteration statistics.
Example Useage:

```
pfset Solver.Nonlinear.PrintFlag   NormalVerbosity
```

string **Solver.Nonlinear.EtaChoice** [Walker2]
This key specifies how the linear system tolerance will be selected. The linear system is solved until a relative residual reduction of η is achieved. Linear residuall norms are measured in the l^2 norm. Choices for this key include **EtaConstant, Walker1** and **Walker2**. If the choice **EtaConstant** is specified, then η will be taken as constant. The choices **Walker1** and **Walker2** specify choices for η developed by Eisenstat and Walker [5]. The choice **Walker1** specifies that η will be given by $|\|F(u^k)\| - \|F(u^{k-1}) + J(u^{k-1}) * p\||/\|F(u^{k-1})\|$. The choice **Walker2** specifies that η will be given by $\gamma\|F(u^k)\|/\|F(u^{k-1})\|^{\alpha}$. For both of the last two choices, η is never allowed to be less than 1e-4.
Example Useage:

```
pfset Solver.Nonlinear.EtaChoice   EtaConstant
```

double **Solver.Nonlinear.EtaValue** [1e-4]
This key specifies the constant value of η for the EtaChoice key **EtaConstant**.
Example Useage:

```
pfset Solver.Nonlinear.EtaValue   1e-7
```

double **Solver.Nonlinear.EtaAlpha** [2.0]
This key specifies the value of α for the case of EtaChoice being **Walker2**.
Example Useage:

```
pfset Solver.Nonlinear.EtaAlpha   1.0
```

double **Solver.Nonlinear.EtaGamma** [0.9]
This key specifies the value of γ for the case of EtaChoice being **Walker2**.
Example Useage:

```
pfset Solver.Nonlinear.EtaGamma   0.7
```

string **Solver.Nonlinear.UseJacobian** [False]
This key specifies whether the Jacobian will be used in matrix-vector products or whether a matrix-free version of the code will run. Choices for this key are **False** and **True**. Using the Jacobian will most likely decrease the number of nonlinear iterations but require more memory to run.
Example Useage:

```
pfset Solver.Nonlinear.UseJacobian   True
```

double **Solver.Nonlinear.DerivativeEpsilon** [1e-7]
This key specifies the value of ϵ used in approximating the action of the Jacobian on a vector with approximate directional derivatives of the nonlinear function. This parameter is only used when the UseJacobian key is **False**.
Example Useage:

```
pfset Solver.Nonlinear.DerivativeEpsilon   1e-8
```

string **Solver.Nonlinear.Globalization** [LineSearch]
This key specifies the type of global strategy to use. Possible choices for this key are **InexactNewton** and **LineSearch**. The choice **InexactNewton** specifies no global strategy, and the choice **LineSearch** specifies that a line search strategy should be used where the nonlinear step can be lengthened or decreased to satisfy certain criteria.
Example Useage:

```
pfset Solver.Nonlinear.Globalization   LineSearch
```

string **Solver.Linear.Preconditioner** [MGSemi]
This key specifies which preconditioner to use. Currently, the three choices are **NoPC, MGSemi** and **SMG**. The choice **NoPC** specifies that no preconditioner should be used. The choice **MGSemi** specifies a semi-coarsening multigrid algorithm which uses a point relaxation method. The choice **SMG** specifies a semi-coarsening multigrid algorithm which uses plane relaxations. This method is more robust than **MGSemi**, but generally requires more memory and compute time.
Example Useage:

```
pfset Solver.Linear.Preconditioner   MGSemi
```

string **Solver.Linear.Preconditioner.SymmetricMat** [Symmetric]
This key specifies whether the preconditioning matrix is symmetric. Choices fo rthis key are **Symmetric** and **Nonsymmetric**. The choice **Symmetric** specifies that the symmetric part of the Jacobian will be used as the preconditioning matrix. The choice **Nonsymmetric** specifies that the full Jacobian will be used as the preconditioning matrix. NOTE: ONLY **Symmetric** CAN BE USED IF MGSemi IS THE SPECIFIED PRECONDITIONER!
Example Useage:

```
pfset Solver.Linear.Preconditioner.SymmetricMat     Symmetric
```

integer **Solver.Linear.Preconditioner.***precond_method***.MaxIter** [1]
This key specifies the maximum number of iterations to take in solving the preconditioner system with *precond_method* solver.
Example Useage:

```
pfset Solver.Linear.Preconditioner.SMG.MaxIter    2
```

integer **Solver.Linear.Preconditioner.SMG.NumPreRelax** [1]
This key specifies the number of relaxations to take before coarsening in the specified preconditioner method. Note that this key is only relevant to the SMG multigrid preconditioner.
Example Useage:

```
pfset Solver.Linear.Preconditioner.SMG.NumPreRelax    2
```

integer **Solver.Linear.Preconditioner.SMG.NumPostRelax** [1]
This key specifies the number of relaxations to take after coarsening in the specified preconditioner method. Note that this key is only relevant to the SMG multigrid preconditioner.
Example Useage:

```
pfset Solver.Linear.Preconditioner.SMG.NumPostRelax    0
```

5.2 ParFlow Binary Files (.pfb)

The `.pfb` file format is a binary file format which is used to store ParFlow grid data. The format for the file is:

```
<double : X>    <double : Y>    <double : Z>
<integer : NX>  <integer : NY>  <integer : NZ>
<double : DX>   <double : DY>   <double : DZ>

<integer : num_subgrids>
FOR subgrid = 0 TO <num_subgrids> - 1
BEGIN
   <integer : ix>  <integer : iy>  <integer : iz>
   <integer : nx>  <integer : ny>  <integer : nz>
   <integer : rx>  <integer : ry>  <integer : rz>
   FOR k = iz TO iz + <nz> - 1
   BEGIN
      FOR j = iy TO iy + <ny> - 1
      BEGIN
         FOR i = ix TO ix + <nx> - 1
         BEGIN
            <double : data_ijk>
         END
      END
   END
END
```

5.3 ParFlow Scattered Binary Files (.pfsb)

The `.pfsb` file format is a binary file format which is used to store ParFlow grid data. This format is used when the grid data is "scattered", that is, when most of the data is 0. For data of this type, the `.pfsb` file format can reduce storage requirements considerably. The format for the file is:

```
<double : X>    <double : Y>    <double : Z>
<integer : NX>  <integer : NY>  <integer : NZ>
<double : DX>   <double : DY>   <double : DZ>

<integer : num_subgrids>
FOR subgrid = 0 TO <num_subgrids> - 1
BEGIN
   <integer : ix>  <integer : iy>  <integer : iz>
   <integer : nx>  <integer : ny>  <integer : nz>
   <integer : rx>  <integer : ry>  <integer : rz>
   <integer : num_nonzero_data>
   FOR k = iz TO iz + <nz> - 1
   BEGIN
```

```
         FOR j = iy TO iy + <ny> - 1
         BEGIN
            FOR i = ix TO ix + <nx> - 1
            BEGIN
               IF (<data_ijk> > tolerance)
               BEGIN
                  <integer : i>  <integer : j>  <integer : k>
                  <double : data_ijk>
               END
            END
         END
      END
   END
```

5.4 ParFlow Solid Files (.pfsol)

The `.pfsol` file format is an ASCII file format which is used to define 3D solids. The solids are represented by closed triangulated surfaces, and surface "patches" may be associated with each solid.

Note that unlike the user input files, the solid file cannot contain comment lines.

The format for the file is:

```
<integer : file_version_number>

<integer : num_vertices>
# Vertices
FOR vertex = 0 TO <num_vertices> - 1
BEGIN
   <real : x>  <real : y>  <real : z>
END

# Solids
<integer : num_solids>
FOR solid = 0 TO <num_solids> - 1
BEGIN
   #Triangles
   <integer : num_triangles>
   FOR triangle = 0 TO <num_triangles> - 1
   BEGIN
      <integer : v0> <integer : v1> <integer : v2>
   END
```

```
      # Patches
      <integer : num_patches>
      FOR patch = 0 TO <num_patches> - 1
      BEGIN
         <integer : num_patch_triangles>
         FOR patch_triangle = 0 TO <num_patch_triangles> - 1
         BEGIN
            <integer : t>
         END
      END
   END
```

The field `<file_version_number>` is used to make file format changes more manageable. The field `<num_vertices>` specifies the number of vertices to follow. The fields `<x>`, `<y>`, and `<z>` define the coordinate of a triangle vertex. The field `<num_solids>` specifies the number of solids to follow. The field `<num_triangles>` specifies the number of triangles to follow. The fields `<v0>`, `<v1>`, and `<v2>` are vertex indexes that specify the 3 vertices of a triangle. Note that the vertices for each triangle MUST be specified in an order that makes the normal vector point outward from the domain. The field `<num_patches>` specifies the number of surface patches to follow. The field `num_patch_triangles` specifies the number of triangles indices to follow (these triangles make up the surface patch). The field `<t>` is an index of a triangle on the solid `solid`.

ParFlow `.pfsol` files can be created from GMS `.sol` files using the utility `gmssol2pfsol` located in the `$PARFLOW_DIR/bin` directory. This conversion routine takes any number of GMS `.sol` files, concatenates the vertices of the solids defined in the files, throws away duplicate vertices, then prints out the `.pfsol` file. Information relating the solid index in the resulting `.pfsol` file with the GMS names and material IDs are printed to stdout.

5.5 ParFlow Well Output File (.wells)

A well output file is produced by ParFlow when wells are defined. The well output file contains information about the well data being used in the internal computations and accumulated statistics about the functioning of the wells.
The header section has the following format:

```
   LINE
   BEGIN
      <real : BackgroundX>
      <real : BackgroundY>
      <real : BackgroundZ>
      <integer : BackgroundNX>
      <integer : BackgroundNY>
      <integer : BackgroundNZ>
      <real : BackgroundDX>
      <real : BackgroundDY>
      <real : BackgroundDZ>
   END

   LINE
   BEGIN
      <integer : number_of_phases>
      <integer : number_of_components>
      <integer : number_of_wells>
   END

   FOR well = 0 TO <number_of_wells> - 1
   BEGIN
      LINE
      BEGIN
         <integer : sequence_number>
      END

      LINE
      BEGIN
         <string : well_name>
      END

      LINE
      BEGIN
         <real : well_x_lower>
         <real : well_y_lower>
         <real : well_z_lower>
         <real : well_x_upper>
         <real : well_y_upper>
         <real : well_z_upper>
         <real : well_diameter>
      END

      LINE
      BEGIN
        <integer : well_type>
        <integer : well_action>
      END
   END
```

The data section has the following format:

```
   FOR time = 1 TO <number_of_time_intervals>
```

```
BEGIN
   LINE
   BEGIN
      <real : time>
   END

   FOR well = 0 TO <number_of_wells> - 1
   BEGIN
      LINE
      BEGIN
         <integer : sequence_number>
      END

      LINE
      BEGIN
         <integer : SubgridIX>
         <integer : SubgridIY>
         <integer : SubgridIZ>
         <integer : SubgridNX>
         <integer : SubgridNY>
         <integer : SubgridNZ>
         <integer : SubgridRX>
         <integer : SubgridRY>
         <integer : SubgridRZ>
      END

      FOR well = 0 TO <number_of_wells> - 1
      BEGIN
         LINE
         BEGIN
            FOR phase = 0 TO <number_of_phases> - 1
            BEGIN
               <real : phase_value>
            END
         END

         IF injection well
         BEGIN
            LINE
            BEGIN
               FOR phase = 0 TO <number_of_phases> - 1
               BEGIN
                  <real : saturation_value>
               END
            END

            LINE
            BEGIN
               FOR phase = 0 TO <number_of_phases> - 1
               BEGIN
                  FOR component = 0 TO <number_of_components> - 1
                  BEGIN
                     <real : component_value>
                  END
               END
            END
         END

         LINE
         BEGIN
            FOR phase = 0 TO <number_of_phases> - 1
            BEGIN
               FOR component = 0 TO <number_of_components> - 1
               BEGIN
                  <real : component_fraction>
               END
            END
         END

         LINE
         BEGIN
            FOR phase = 0 TO <number_of_phases> - 1
            BEGIN
               <real : phase_statistic>
            END
         END

         LINE
         BEGIN
            FOR phase = 0 TO <number_of_phases> - 1
            BEGIN
               <real : saturation_statistic>
            END
         END
```

```
         LINE
         BEGIN
            FOR phase = 0 TO <number_of_phases> - 1
            BEGIN
               FOR component = 0 TO <number_of_components> - 1
               BEGIN
                  <real : component_statistic>
               END
            END
         END

         LINE
         BEGIN
            FOR phase = 0 TO <number_of_phases> - 1
            BEGIN
               FOR component = 0 TO <number_of_components> - 1
               BEGIN
                  <real : concentration_data>
               END
            END
         END
      END
   END
END
```

Bibliography

[1] S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124:145–159, 1996.

[2] *AVS: Developer's Guide.* 300 Fifth Ave., Waltham, MA, May 1992. Release 4.

[3] *AVS: User's Guide.* 300 Fifth Ave., Waltham, MA, May 1992. Release 4.

[4] *AVS: Module Reference.* 300 Fifth Ave., Waltham, MA, February 1993. Release 5.

[5] Stanley C. Eisenstat and Homer F. Walker. Choosing the forcing terms in an inexact newton method. *SIAM J. Sci. Comput.*, 17(1):16–32, 1996.

[6] R. Haverkamp and M. Vauclin. A comparative study of three forms of the Richard equation used for predicting one-dimensional infiltration in unsaturated soil. *Soil Sci. Soc. of Am. J.*, 45:13–20, 1981.

[7] M. Th. van Genuchten. A closed form equation for predicting the hydraulic conductivity of unsaturated soils. *Soil Sci. Soc. Am. J.*, 44:892–898, 1980.

[8] A. F. B. Tompson, R. Ababou, and L. W. Gelhar. Implementation of of the three-dimensional turning bands random field generator. *Water Resources Res.*, 25(10):2227–2243, 1989.

[9] United States Department of Defense. *Groundwater Modeling System: Tutorial.* Brigham Young University, Provo, UT, 1994. Version 1.0.

[10] United States Department of Defense. *Groundwater Modeling System: Reference Manual.* Brigham Young University, Provo, UT, 1995. Version 1.1.

[11] B. Welch. *Practical Programming in TCL and TK.* Prentice Hall, 1995.

[12] Carol S. Woodward. A Newton-Krylov-Multigrid solver for variably saturated flow problems. In *Proceedings of the XIIth International Conference on Computational Methods in Water Resources*, June 1998.