# SPHERLS User Guide 1.0

Chris Geroux

February 3, 2016

# Chapter 1

# Introduction

This manual is primarily designed to be used to get a user up and running quickly. It also includes some information on how to go about modifying and developing SPHERLS. The reference manuals are useful for determining specific information about variables, functions and classes.

## 1.1 Overview

SPHERLS stands for Stellar Pulsation with a Horizontal Eulerian Radial Lagrangian Scheme. There are three components to SPHERLS: SPHERLS itself which does the hydrodynamics calculations, SPHERLSgen which creates starting models, and SPHERLSanal which is able to manipulate the output files. All three components have their own reference manuals where details of the various classes, functions, and variables are described, however the details on how to use these three components is described here. For the most part SPHERLS and SPHERLSgen are used directly, while SPHERLSanal is primarly used indirectly through the use of python scripts.

SPHERLS calculates the radial pulsation motions together with the horizontal convective flow of stars. The radial pulsation can be described by a radial grid velocity, moving the grid inward and outward with the pulsation. The movement of the grid is defined by the motion required to maintaining the mass in a spherical shell through out the calculation. This motion is determined so that it will change the volume of the shell so the newly calculated density when multiplied with the new volume will produce the same shell mass for all times. The total motion of the stellar material is simply the combination of the three velocity components and the grid velocity. The convective motion is the radial velocity minus the grid velocity, combined with the theta and phi velocities. This is because the grid velocity describes the

bulk motion of the pulsation so subtracting it out leaves only the convective motions.

SPHERLS solves the normal hydrodynamic equations of, mass, momentum, and energy conservation. The form of the mass equation, momentum conservation, and energy conservation are:

$$\frac{dM}{dt} + \oint_{\mathbb{S}} (\rho\vec{v}) \cdot \hat{n} d\sigma = 0 \tag{1.1}$$

$$\frac{\partial\vec{v}}{\partial t} + (\vec{v}\cdot\nabla)\vec{v} = -\frac{1}{\rho}\nabla P + \nabla\cdot\tau - \nabla\phi \tag{1.2}$$

$$\frac{\partial E}{\partial t} + (\vec{v}\cdot\nabla)E + P\frac{d\mathbb{V}}{dt} = \epsilon - \frac{1}{\rho}\nabla\cdot\ F + \frac{D_t\mathcal{K}^{3/2}}{L} + \frac{1}{\rho}\nabla\cdot\left(\frac{\mu_t}{\rho P_{rt}}\nabla E\right) \tag{1.3}$$

where $\tau$ is the stress tensor for zero bulk viscosity, $E$ is the specific internal energy, $\mathbb{V}$ is the specific volume, and $F$ is the radiative flux. In addition to these conservation equations an equation of state is needed, in this case the OPAL equation of state and opacities, and the Alaxander opacities at low temperatures are used. The equation of state tables are functions of density and temperature, and produce the energy, pressure, opacity, and adiabatic index of the gas for a given temperature and density. In adiabatic calculations, it is also possible to use a $\gamma$-law gas equation of state and in that case an initial energy profile must be specified.

The simulation grid is broken up into two main sections, the 1D region towards the center of the star, the multi-dimensional region towards the surface. The inner part of the multi-dimensional region solves all the conservation equations explicitly, in that the new values for the conserved quantities are directly calculated from the information in the previous time step. In the outer parts of the multi-dimensional region the energy conservation equation is calculated semi-implicitly, which means that the new values are dependent on the new values averaged with the old values to correctly time center the equation. This semi-implicit energy conservation equation can be perturbed and linearised producing a set of linear equations the size of the region being solved implicitly. The solution of these linear equations provide corrections for the temperature. The corrected temperature can then be used to solve for a new set of corrections this processes is repeated until the value of the new temperature converges (the size of the corrections is smaller than some specified amount). The equation of state is a function of temperature and not energy which is why the temperature is perturbed and not the energy. This set of equations for the temperature corrections are solved using the PETSC library.

More info to be included here:

- mention that SPHERLS is parallelized
- Different ways in which SPHERLS can be used, 1D,2D,3D, Adiabatic,Non-adiabatic, implicit, debugging options/test
- mention very quickly how SPHERLS can be used (e.g. give very quick example of running SPHERLS)

## 1.2 Program Flow

This should give a rough idea of how SPHERLS works, but should be kept quite high level

- Describe the grids
- The order of calculation
- When parts of the grid are updated
- when models are dumped

# Chapter 2

# Installing SPHERLS

There are a number of optional and required third party libraries used by SPHERLS. This section will describe how to install these third party libraries as well as SPHERLS. However, as the third party libraries often contain their own installation documentation the following section should be thought of as the minimal set of options and steps required to install those third party libraries for use with SPHERLS and not a complete documentation of how those third party libraries should be installed in general. One should consult the third party library's documentation for the full details of their installation.

There are three flavours of the SPHERLS distribution package available: one which includes all third party libraries, one which includes only required third party libraries, and one which does not include any third party libraries. It is recommended to use the package including all third party libraries. This version helps to ensure the greatest functionality of SPHERLS and also that these libraries are compatible both with SPHERLS and the below installation instructions. Other version of the distribution package can be desirable to reduce download size, if the third party libraries are already installed, or if you wish to try out newer versions of the third party libraries and do not want to download the additional data. The included third party libraries can be found under the `lib/` directory under the root package directory.

A few words on installation before we get into the details of the specific third party package installations. In order for the SPHERLS configuration script to find the required libraries and include files automatically they have to be installed in at least one of the directories that it looks for them in. The configuration script looks for the libraries and include files it requires in the following locations: `/lib`, `/include`, `/usr/lib` `/usr/-include`, `/usr/local/lib`, `/usr/local/include`, `/home/$USER/lib`, and `/home/$USER/include`. If you install the required libraries in places other

than these locations you will have to manually tell the SPHERLS configuration script where to find them. Running `configure -h` will list the available options to tell the script where to find these include files and libraries.

I am going to assume that the user is installing on a Linux like system in a bash shell. Other shells will likely work, but you will need to look up some shell specific things such as how you set environment variables in your specific shell. The install instructions below assume you do not have root access and must do an install to your home directory (a per-user install). The install location for per-user level binaries, libraries, include files and documentation that SPHERLS automatically checks is in `~/bin`, `~/lib`, `~/include`, and `~/share` directories respectively (`~` is just the Linux short hand for `/home/-$USER`). Be aware that if you have these directories in your home directory already and are not using them as a standard place to install per-user packages you will likely want to rename your pre-existing directories or a bunch of additional files will be added to them from the installations of various packages below. Alternatively, you can install the libraries and binaries to any directory on your machine just by changing `--prefix=/home/$USER/`, mentioned below, to point to where you want to install it. But as mentioned if you install to a non-standard location you will have to tell the SPHERLS configure script where you put things. Also `--prefix` should always be specified as some of these libraries don't install to standard locations on their own, preferring to install into the directory where they are built.

In this section I will often mention setting environment variables, or tell you to add a line starting with `export` to your `.bashrc` file. These just set the necessary environment variables for SPHERLS to build, install, and run. These settings can also be easily adapted to form the basis of a module file if one wishes to use GNU environment modules to manage loading and unloading environment settings.

## 2.1 Requirements

SPHERLS requires gcc/g++, OpenMPI, and PETSc to compile and run at all. It also makes heavy use of Python scripts some of which depend on Cython. So while you technically can run the basic binaries without Python and Cython in practise you really won't want to use SPHERLS without these.

The below set of instructions do not explain how to install the GCC (GNU Compiler Collection) but assume that your system already has this installed. If you are on a large HPC cluster you likely also have OpenMPI already available to you. Feel free to use the site OpenMPI installation, but if you find later tests produce unexpected results or you have difficulty building

SPHERLS you may wish to try using a per-user install of an OpenMPI version known to work with SPHERLS and building with that.

The below libraries can be optionally installed but will greatly enhance the usefulness of SPHERLS and are greatly recommended. The full list of dependencies of these third party libraries is not included, but some library dependencies have been noted when I have specifically had to install these dependencies on various machines.

**python** python scripts are supplied for various analysis and convenience operations. Having python is highly recommended and will make using SPHERLS far more enjoyable. Many SPHERLS python scripts use the following python libraries:

- **numpy** numerical python, fast array operations
- **matplotlib** for creating plots
- **scipy** for interpolating in equation of state and opacity tables when making new files
- **evtk** a python library to make writing vtk files easier, the SPHERLS build process actually makes this for you if Cython support is enabled.
- **Cython** needed to build evtk, and also to allow Python scripts access to the equation of state and opacity C++ classes used in SPHERLS.

**fftw3** used for frequency analysis by SPHERLSanal.

**hdf4** for converting model dumps to hdf4 file format for visualizing, used by SPHERLSanal and some Python scripts.

- **jpeg** needed by hdf4 library
- **zlib** needed by hdf4 library

**pdflatex** used to create documentation from src/doc/ latex files. Use `make docs` to build documentation. Note that the `make docs` command requires an additional option to be set during the configuration process `--enable-make-docs`.

## 2.2 Installing OpenMPI

- Download OpenMPI from the open mpi site or use the version in `libs/` directory. Versions 1.6.3 and 1.6.5 have been tested and known to work

with SPHERLS. That is not to say that other versions of OpenMPI will not work with SPHERLS, only that those versions are the only one which have been explicitly tested.

- add library path to `LD_LIBRARY_PATH`
- `./configure --prefix= <path-to-final-location-of-install>`
- `make`
- `install`
- finally the bin, man, and lib paths will need environment variables set:

```
export OPENMPI=<path-to-final-location-of-install>
export PATH=<path-to-final-location-of-install>/bin:${PATH}
export MANPATH=<path-to-final-location-of-install>/man:${MANPATH}
export LD_LIBRARY_PATH=<path-to-final-location-of-install>/lib-
:${LD_LIBRARY_PATH}
```

## 2.3 Installing PETSC Library

Version `petsc-lite-3.1-p8`, has been tested to work with SPHERLS. `petsc--lite-3.2-p7` is known to be incompatible, which as of this writing is the current version of the PETSc library. At some point in the future support for the newer version of the library maybe added. The below commands will install PETSc into your home directory.

- Download PETSc library, from the PETSc website or get it from the `libs/` directory in the spherls package
- Then untar and unzip it with `tar -xzf petsc-lite-3.1-p8.tar`
- To install the library change into the directory made when you extracted the archive and type the following commands:

  1. `./configure PETSC_DIR=$PWD --prefix=/home/$USER/ --with-c++-support --with-c-support --with-shared --download-f-blas-lapack=1 --with-x11=no`

where `$USER` is the environment variable corresponding to your username. Note that the `PETSC_DIR=$PWD` needs to come first. Also `--download-f-blas-lapack=1` only works if there is a fortran compiler present, which isn't strictly needed otherwise. Using `--download-c-blas-lapack=1` seems to work when a Fortran compiler isn't available.

2. `make all` Often at the end of the configuration stage the configuration script will give the command to make the library. One should use this over the above, if given.
3. `make install` as with the `make all` the makeFile will also likely tell you the command needed for the installation, which should be used over the one provided here.
4. `make PETSC_DIR=/home/$USER/lib test` will test the code. Again the install process will give you a more specific command that you should use over the one given here.

## 2.4 Installing Cython

Cython allows you to build some extra python modules to use in creating visualization output such as vtk files for use in VisIt, and for making the equation of state routines available to the python analysis scripts. Finally there are dependencies on cython in scripts which run some simple tests such as checking that results are reproduced and that restarting a calculation works correctly. Therefore it is highly recommended that cython is installed. There is a version tested to work under the `libs/` directory.

- `tar -xzf Cython-0.19.1.tar.gz`
- `cd Cython-0.19.1`
- `python setup.py install --prefix=/home/$USER`
- This will require you adding the line `export PYTHONPATH=/home/$USER/-lib/python2.7/site-packages/:$PYTHONPATH` to `.bashrc` so that python can find your installation of Cython, the exact path may depend on your version of python and of course where you install Cython.

## 2.5 Installing Python Modules

In this section I will just briefly give some suggests about how to install some of the additional python modules I mention. One can often install python

modules into home directory locations with the `pip` tool. For example

```
pip install --install-option="--prefix
=<path-to-install-directory>" numpy
```

If the specified module, in this case `numpy` is already present on the current machine, an upgraded version can be installed in the users home directory if desired with the `--upgrade` option. Another helpful option is the `--user` option which was required when installing the `matplotlib` module. In order to have python pick up these home directory installs one must add to the `PYTHONPATH` environment variable with something like

```
export PYTHONPATH=<path-to-install-directory>
/lib/python2.7/site-packages:${PYTHONPATH}.
```

The exact details of this path to add will depend on the version of python you have available to you on your system.

## 2.6 Installing FFTW Library

- Download the FFTW Library from the FFTW website. Or there is a version available in the `lib/` directory. Version fftw-3.2.2, fftw-3.3.1, and fftw-3.3.3 have been tested to work with SPHERLS.
- Then untar and unzip the downloaded with something like `tar -xzf fftw-3.2.2.tar.gz`
- To install the library change into the directory made when you extracted the archive and type the following commands:
  1. `./configure --prefix=<path-to-final-location-of-library>`
  2. `make`
  3. `make install`

## 2.7 Installing HDF4 Library

Building these libraries requires gfortran and the jpeg library. If using the SPHERLS distribution package including third party optional libraries, version 4.2.8 of the HDF4 library is available in the `libs` subdirectory. Otherwise this library can be downloaded from the hdfgroup website. The most

recent version tested to work with SPHERLS is version 4.2.8, version 4.2.7 has also been tested to work. To build the version included with the SPHERLS distribution package use the following set of commands:

- `cd <SPHERLS package directory>/libs`
- `tar -xzf hdf-4.2.8.tar.gz`
- `cd hdf-4.2.8`
- Run `./configure --prefix=<path-to-final-location-of-library> CFLAGS="-fPIC" CXXFLAGS="-fPIC"`.

  Note that for the hdf library the `<path-to-final-location-of-library>` should be set if doing a global install as well as doing a per user install as the default install directory is inside the build directory, which is not what is usually wanted. So if doing a global install it will probably want to be set to `/usr/local` or for a per-user install `/home/$USER`.

- `make`
- `make install`

The above commands have only been tested with the version including with the SPHERLS distribution package other versions may require slight modifications to the above commands but should be reasonably similar.

## 2.8 Installing evtk

The SPHERLS build process actually builds evtk for you, so you probably do not need to do these steps unless for some reason you wish to install evtk yourself. As mentioned previously, evtk is a python module which allows one to create vtk files.

- `tar -xzf evtk.tar.gz`
- `cd evtk`
- `python setup.py install --prefix=/home/$USER`
- This will require you adding the line `export PYTHONPATH=/home/-$USER/lib/python2.7/site-packages/:$PYTHONPATH` to `.bashrc` so that python can find your installation of evtk, the exact path may depend on your version of python also you only need to have this line in your `.bashrc` file once

## 2.9 Installing SPHERLS

After all the required and optional third party libraries are available and installed into places that the SPHERLS configuration scripts looks for them building and installing SPHERLS should be as easy as:

- `./configure --prefix=/home/$USER`
- `make`
- `make install`

If you have the raw git repository do the following

- `autoreconf --install`
  provided you have the gnu build tools installed, this makes the configuration script
- `./configure --prefix=/home/$USER`
- `make`
- `make install`

Common problems which result during the execution of the above command are: not finding the installed third party libraries as they were installed in different locations. There are numerous options available in the configuration script for manually specifying these installation locations of various library and include files. Run `./configure -h` to get a list of the options and specify the location of the missing libraries and or include files. In addition to not finding various libraries, the various optional components may also be turned off during the configuration step. For example if you don't have hdf4 libraries and don't want to use that functionality including the option –disable-hdf will build without hdf4 capabilities.

After you have built SPHERLS and installed it to a location of your choosing, you will need to setup a number of environment variables. First you will need to add the path to the petsc library location to the `LD_LIBRARY_PATH` variable. This can be done with adding the line

```
export LD_LIBRARY_PATH=<path_to_petsc_lib_dir>:${LD_LIBRARY_PATH}
```

to your `.bashrc` file or some other file which you source settings from. You should also add the path to the SPHERLS bin directory where SPHERLS was installed to your path

`export PATH=<path_to_spherls_bin_dir>:${PATH}`.

Finally python will need to know where to load the SPHERLS python modules from, this can be done by

`export PYTHONPYTH=<path_to_spherls_bin_dir>:${PYTHONPATH}`.

# Chapter 3

# Using SPHERLS

Once you have built SPHERLS and set up the necessary environment variables the first thing to do is run some simple tests to make sure it will run as expected. There are python scripts which you can use to run these tests and are found in the bin directory where SPHERLS was installed. Note that you will need to have built with Cython support enabled to do this step, so it is highly recommended that you have installed Cython and built SPHERLS with it enabled. There are two test scripts which can be run `test_restart.py` and `test_calculation.py`. These will be found under your SPHERLS install directory in the `bin` directory. They should also be findable from your path as they are executable and we have added the `bin` directory to your `PATH` environment variable. Python scripts are distinguished from compiled executables with their `.py` extension. Lets examine what the `test_restart.py` script does in detail first.

This script and many other scripts have many optional and required arguments which can be specified. Nearly all scripts can be executed with the `-h` option to display a brief description of the script and how to use it, as well as listing available options. I would suggest running the `test_restart.py` script with the `-h` option to see these available options. The script help says that it tests restarts of the application to ensure that it produces the same output as if there had not been a restart. Also the `-k` option allows us to keep all the temporary output this script creates while running the tests. This script actually runs SPHERLS several times and compares output of the different runs. We will examine the steps this script takes in order to illustrate some aspects of working with SPHERLS. The output that is generated when you run `test_restart.py -k` should look something like that below, but with different paths reflecting where SPHERLS was installed on your system.

```
1:making a temporary directory, "./test3DNARestarts" to store test data ... SUCCESS
```

```
 2:changing into directory "./test3DNARestarts" ...
 3:making "SPHERLS.xml" ...
 4:running "/home/cgeroux/apps/spherls/bin/SPHERLS" for 2 time steps ... SUCCESS
 5:remaking "SPHERLS.xml" ...
 6:combining binary dump "./RestartTest2_t[0-*]" for restart ... SUCCESS
 7:restarting "/home/cgeroux/apps/spherls/bin/SPHERLS" for 1 time step from dump "RestartTest1_t00172487" ... SUCCESS
 8:combining binary files for dump "./RestartTest2_t[0-*]" ... SUCCESS
 9:diffing last dumps ... SUCCESS
10:making a temporary directory, "./test2DNARestarts" to store test data ... SUCCESS
11:changing into directory "./test2DNARestarts" ...
12:making "SPHERLS.xml" ...
13:running "/home/cgeroux/apps/spherls/bin/SPHERLS" for 2 time steps ... SUCCESS
14:remaking "SPHERLS.xml" ...
15:combining binary dump "./RestartTest2_t[0-*]" for restart ... SUCCESS
16:restarting "/home/cgeroux/apps/spherls/bin/SPHERLS" for 1 time step from dump "RestartTest1_t00137887" ... SUCCESS
17:combining binary files for dump "./RestartTest2_t[0-*]" ... SUCCESS
18:diffing last dumps ... SUCCESS
19:making a temporary directory, "./test1DNARestarts" to store test data ... SUCCESS
20:changing into directory "./test1DNARestarts" ...
21:making "SPHERLS.xml" ...
22:running "/home/cgeroux/apps/spherls/bin/SPHERLS" for 2 time steps ... SUCCESS
23:remaking "SPHERLS.xml" ...
24:combining binary dump "./RestartTest2_t[0-*]" for restart ... SUCCESS
25:restarting "/home/cgeroux/apps/spherls/bin/SPHERLS" for 1 time step from dump "RestartTest1_t00000001" ... SUCCESS
26:combining binary files for dump "./RestartTest2_t[0-*]" ... SUCCESS
27:diffing last dumps ... SUCCESS
```

Lets go through the steps the script executed. After the script creates a new directory to run a test in and changing into it (lines 1-2), it creates a `SPHERLS.xml` file (line 3). This file contains all the settings to perform a calculation with SPHERLS. This file has settings such as where the input file is located, where the output should go, and how many time-steps SPHERLS should take. There are also many more settings contained in this file and it will be discussed in more detail in section 3.2. The script then runs SPHERLS for 2 time-steps, and then remakes the configuration file (line 5) so that the calculation will resume at the second time step and run for one time step. This then allows the output from these two runs to be compared, as they should be at the same time, the output should be identical.

Before SPHERLS is run the second time, the output from the first run is combined into one file (line 6). SPHERLS uses domain decomposition to split up the computational work across multiple separate cpus, this means the physical grid representing the star is split into local grids on each cpu. Each cpu is responsible for computations only on that local grid. As a result the simplest and most efficient way to write out the grid state is to have each cpu write out its local portion of the grid to a file. To have a single file containing the whole grid then requires these separate files to be combined. This combing step is required before a restart occurs and SPHERLS only reads these combined files as input and only generates the distributed files as output. If you look at the directory `test3DNARestarts` created by the script you will see the files `SPHERLS.xml` and `SPHERLS_first_run.xml` configuration files, which correspond the first and second run of SPHERLS. You will also see a `log.txt` file which contains output from the `test_restart.py` script as well as output actually generated from the SPHERLS executable. Blocks (or even single lines) of output from SPHERLS are preceded by the

file and line number in the source code where that output was generated (e.g. `src/SPHERLS/main.cpp:main:144:0:`) this helps to distinguish it form the output generated by the test script. It is also handy to locate the code which generated a given output. You can also see output from python scripts which combine the output files, which have lines starting with `__main__:combine_bin_files:`. Note that this output is actually generated from the function `combine_bin_files` in Python script `combine_bins.py`. The `test_restart.py` script executes the `combine_bins.py` script through the command line to combine the files.

Once the output files from the first run are combined a second run of one time step is initiated (line 7). Finally the output from the second run is combined (line 8) and compared to the output from the first run (line 9). This whole process is repeated to test restarts for runs of 1D, 2D, and 3D models.

For the most part, the `test_restart.py` script is simply issuing commands on the command line to perform these tests. These commands are combination of usual linux commands such as `mkdir` and `cd`, and SPHERLS specific commands. The command it uses to actually run the SPHERLS executable is `mpirun -np <number-of-cores> SPHERLS`. This is just the usual `mpirun` command with the `SPHERLS` executable as the argument. The SPHERLS executable expects to read settings from the `SPHERLS.xml` configuration file, and will fail with an error message such as

```
XML Parsing error inside file 'SPHERLS.xml'.
Error: File not found
At line 0, column 0.
src/SPHERLS/main.cpp:main:275:0:src/xmlFunctions.cpp:openXMLFile:268: error parsing the file "SPHERLS.xml" possibly no global "data" node
```

if it doesn't find that file in the current working directory.

Bits to include in this section are (some of these might be better in a different section, perhaps Developing SPHERLS):

- Generating a starting model
- The XML configuration file
- Starting a calculation
- getting data
- watchzones
- model dumps
- debug information

- post calculation analysis
- python scripts and plotting
- Adiabatic Calculations
- 1D, 2D, and 3D
- $\gamma$-law gas
- Sedov Blast wave test
- Non-Adiabatic Calculations
- 1D, 2D, and 3D
- Tabulate EOS
- Different versions of the energy equation
- LES models
- creating a new EOS file using `eos_interp.py`

## 3.1 Generating a Starting Model

## 3.2 The Configuration file

describe the xml configuration file

## 3.3 Running SPHERLS

describe the command, and the different ways to run SPHERLS, using mpi- the different environements and what might need to be done if you have a different sort of environment

## 3.4 Processing Output

### 3.4.1 Profile Files

### 3.4.2 2D Slices

### 3.4.3 HDF Files

### 3.4.4 VTK Files

## 3.5 Viewing Output

### 3.5.1 Plotting Profiles

### 3.5.2 Plotting 2D Slices

# Chapter 4

# Modifing & Developing SPHERLS

This section should include

- Basic layout/design of the code
- model output
- data monitoring
- watch zones
- peak KE tracking
- internal/versus external variables
- message passing
- grid layout
- ranges of grids
- boundary regions
- grid updating
- How to document SPHERLS
- Premade test for SPHERLS after modification
- reference calculations
- restart test

- calculation test (if not modifying calcluation part of SPHERLS)
- How to modify SPHERLS
- Common changes
- How to add a new internal variable

  1. **Add to the internal variable count:** Decide in what cases the variable will be needed, 1D calculations, 2D calculations, when there is a gamma law gas or a tabulated equation of state, adiabatic or non-adiabatic etc. Then once decided it can be added to the total number of internal variables Grid::nNumIntVars by increasing the value by one in the function modelRead in the section below the comment "set number of internal variables ..." under the appropriate if block. If the specific if block for the situation you need isn't there, you can create your own, and add it there.
  2. **Create a new variable ID:** In the grid.h file under the Grid class are variable ID's. These ID's simply indicate the location of the variable in the array. One must add a new ID for the new variable as an integer. The value of the ID is set in the function modelRead in the same section as the number of internal variables. The value used should be the last integer after the last pre-existing variable ID. This should also be Grid::nNumVars + Grid::nNumIntVars -1. The ID should also be initialized to -1, so that the code knows when it isn't being used. This is done in the grid class constructor, Grid::Grid. Simply add a line in the constructor setting your new ID = -1.
  3. **Set variable infos:** Decide what the dimensions of the new variable will be. It can be cell centered or interface centered. It can also be only 1D, 2D, or 3D. Of course it will be only 1D if the entire calculation is 1D, or 2D if the calculation is 2D, but if the calculation is 3D it could also only be 2D, or 1D, and if 2D it could be only 1D. Also decide if the variable will change with time, dependent variables are only initialized and not updated during the calculations. This information is given to SPHERLS in the setInternalVarInf function in the physEquations.cpp file. The variable that is set is Grid::nVariables. It is a 2D array, the first index corresponds to the particular variable in question, the ID you made in the previous step can be used as the first index of this array. The second index refers to one of the three directions (0-2) or the time dimension (3). If the variable is centered in

the grid in direction 0 (r-direction) then this array element should have a value of 0. If the variable is interface centered in the grid in direction 0, then this array element should have a value of 1. If it isn't defined in direction 0 (for example the theta independent variable isn't defined in the 0 direction) then it should be -1. This is the same for the other 2 directions. The last element (3) should be either 0 not updated every time step, or 1 if updated every timestep. There are various sections here which allows one to set variable information based on which conditions are the variable is defined in. Put these variable infos into the most general case in which the variable is defined. At the end of this function variables are automatically adjusted depending on what the number of dimensions the model uses, so this does not need to be considered unless the variable is not used at all for a specific case of dimensions. For example a variable which is defined at cell center for all three cases for the number of dimensions (1D, 2D, 3D) will be automatically adjusted to be not defined in the 3rd direction when only doing 2D calculations, and similarly for 1D only defined in 1st direction and not defined in the 2nd or 3rd directions.

4. **Add functions:** Finally to do anything usefull with your new internal variable functions must be added to initialize the values of the variables, and to update them with time if needed. Initialization functions are called within the initInternalVars function in the physEquations.cpp file. The details of these functions will depend on what the individual variables are intended for. Functions to be called every timestep must be called from the main program loop in the file main.cpp in the appropriate order.

- How to add a new external variable
- How to add a new physics functions
- Function naming conventions
- Grid variables
- indecies and their ranges
- SPHERLS debugging tips

## 4.1 The Equations

I will want to give a detailed description of the equations used (probably copied from my notes wiki) so that the reader can easily see a 1-1 correspondence between the equation and the terms in SPHERLS.

## 4.2 Message Passing

Explain message passing in SPHERLS