PRediction Of Geospace Radiation Environment and Solar wind parameterS

Work Package 2
Propagation of the Solar Wind from the Sun to L1

Deliverable 2.3
Full user and developer documentation

T. Arber, K. Bennett

December, 2017

This project has received funding from the *European Unions Horizon 2020 research and innovation programme* under grant agreement No 637302.

Document Change Record

Issue	Date	Author	Details
1.0	9 August, 2017	T. Arber	Setup sections headings
1.1	17 December, 2017	T. Arber	Equations and code structure
1.2	20 December, 2017	K. Bennett	Code setup
1.3	20 December, 2017	T. Arber	Final version
1.4	10 September, 2018	T. Arber	Corrected final version

Contents

Summary

Space weather forecasts require reliable knowledge of the MHD variable at L1 and this is required before they are measured there *in situ*. The AWSoM code developed at the University of Michigan will take magnetogram observations and use these to drive MHD simulations, time-accurate, out to around 30 solar radii. These coronal simulations can then be used to drive a fast, spherical geometry inner-heliospheric MHD code (SWIFT) to give predictions at L1. This report is the documentation needed to download, install, compile and run the coupled AWSoM-SWIFT toolchain and then visualise the results at L1.

1 Downloading and setup of the coupled codes

The download, build and setup of the codes used for generating predictions at L1 based on GONG magnetogram data is outlined in the next few sections. The minimum system requirements are listed in Section 5.

The first part of this simulation is carried out using the AWSoM model which is part of the SWMF suite of codes distributed by CSEM at the University of Michigan. The code is freely available after registering for access at the following URL: `http://csem.engin.umich.edu/tools/swmf/downloads.php`

Once registered, you will be able to download a tarball of the code which will be named according to the date of release, e.g. `SWMF_20170626.tgz`. This can be unpacked into a directory named `SWMF_20170626` using the command

```
> tar xzf SWMF_20170626.tgz
```

The second part of the simulation is performed using SWIFT, a 3D spherical-geometry Lagrangian remap code developed and maintained at Warwick University. This code is hosted at Warwick using a GitLab server (an open-source version of GitHub). To obtain the code you must first register for an account at the following URL: `https://cfsa-pmw.warwick.ac.uk/`.

After logging in for the first time, you must next request access to the project by navigating to the URL: `https://cfsa-pmw.warwick.ac.uk/SWIFT` and clicking on the "Request Access" button.

Once registered on the project, the code can be obtained from the git repository using the following command:

```
> git clone --recursive https://cfsa-pmw.warwick.ac.uk/SWIFT/SWIFT
```

Finally, you will need a small tool for generating the satellite trajectory files. This is also hosted on the gitlab repository and can be obtained using the following command:

```
> git clone https://cfsa-pmw.warwick.ac.uk/SWIFT/SPICE
```

For the sake of simplicity, the scripts and examples presented in the remaining sections assume that the two git repositories and the unpacked SWMF tarball all reside in the same directory. We will assume that you start in the top level of this directory heirarchy, so typing the command "ls" will return the entries "SPICE", "SWIFT" and "SWMIF_20170626".

1.1 Building the Code

There are several configuration files and data files that are needed for running the coupled AWSoM/SWIFT simulations. For convenience, a set of example files are supplied with the SWIFT git repository. To access these, switch to the SWIFT directory (`cd SWIFT`) and type

```
> git checkout -b swmf_run origin/swmf_run
```

This will switch to a branch of the repository which contains a directory named `SWMF_FILES` containing the sample files. This directory also contains a few shell scripts which will carry out the job of configuring the codes, building and setting up the data files required for running a simulation on a typical Linux machine. In order to build the codes, just type

```
> ./SWMF_FILES/build.sh
```

The full contents of the build script is supplied in Section 8.1 but we will now outline the basic steps involved. To build SWMF you must first specify the compiler to use, followed by configuring the components and finally use the resulting Makefile to compile the code. There are many possible options available and more details can be found in the accompanying documentation. Begin by switching to the directory that was created when you unpacked the SWMF tarball (eg. `cd SWMF_20170626`). After that, the following set of commands will build the code for the `gfortran` compiler.

```
> ./Config.pl -install -compiler=gfortran
> ./Config.pl -v=Empty,SC/BATSRUS,IH/BATSRUS
> ./Config.pl -o=SC:u=ScChromo,e=MhdWavesPe
> ./Config.pl -o=IH:u=ScChromo,e=MhdWavesPe
> ./Config.pl -g=SC:4,8,4,2000,1,IH:4,4,4,4500,1
> make SWMF
```

In order to convert GONG magnetograms into the harmonics data required for driving AWSoM, an additional "HARMONICS.exe" utility is required which is built as follows:

```
> cd util/DATAREAD/srcMagnetogram
> make HARMONICS
> cd -
```

The remaining programs to be built are SWIFT and SPICE. These are both fairly simple and just use the standard UNIX "make" command. The commands to build SWIFT, along with the Python module required for reading its output are:

```
> cd ../SWIFT
> make
> make sdfutils
```

Finally, to build the SPICE utility the commands are:

```
> cd ../SPICE
> make
> cd ..
```

2 Steady state solution based on Carrington rotation GONG data

To run "SWMF" and generate a steady-state result based on a single full Carrington rotation magnetogram, you can use a script provided for setting up all the required data files:

```
> ./SWMF_FILES/setup.sh
```

By default, this sets up everything required for simulating Carrington rotation number 2098. This number can be changed by using the "-c" flag and specifying the desired Carrington rotation number. For example, to run the steady-state simulation for Carrington rotation 2190 you would issue the following command:

`> ./SWMF_FILES/setup.sh -c 2190` The full contents of the setup script is supplied in Section 8.2

After this you will have a directory named `run_CR2098` in the "SWMF_20170626" directory in which to carry out the simulation (the number following "CR" will change according to the Carrington rotation requested). The directory contains a file named `PARAM.in` which sets the parameters for the run. Note that it also relies on the data files "harmonics.dat" (generated using a GONG magnetogram) and "earth_traj.dat" containing the Earth's trajectory over the given time period. Both of these will need changing if the date of the simulation period is changed.

To run the simulation, change to the "SWMF/run" directory and either run the "SWMF.exe" executable using "mpirun" or submitting the job to the scheduling system if running on a cluster. For example, if running on a 36 core local machine, you would type the following command:

```
> mpirun -np 36 ./SWMF.exe
```

A SLURM job submission script is provided for the University of Warwick cluster. It can be scheduled using the following command:

```
> sbatch job.slurm
```

Once the simulation is complete it should produce output in the "IH/IO2" and "SC/IO2" directories. It also produces a couple of buffer files that are required for driving the "SWIFT" simulation. For the example script, these are the files "R=18Rs_2017_04_29_15_08_00.out" and "R=22Rs_2017_04_29_15_08_00.out". For a different Carrington rotation these files will have different date strings.

To run the "SWIFT" simulation, you must copy these files and the "earth_traj.dat" file into the "SWIFT/Data" directory. You also need to generate a simple text file containing a list of these files. There is a script for accomplishing this task. The commands to run are:

```
> cp earth_traj.dat R*Rs*out ../../SWIFT/Data/

> cd ../../SWIFT

> ./scripts/gen_buffer_list.sh
```

The code can then be run using the command

```
> mpirun -np 36 ./bin/swift
```

3 Daily updated AWSoM steady state plus time-accurate SWIFT

The model coupling AWSoM with SWIFT to generate a prediction at the L1 point using GONG magnetogram data is run on a daily basis at Warwick University. The full simulation is fully automated using the script "autosubmit_predictive.sh" which is provided for reference in the Appendix, Section 8.3. The procedure followed is outlined here.

A new GONG magnetogram is downloaded from the URL `https://gong.nso.edu/data/magmap/QR/bqs`. Currently, the data file is chosen to be the one closest to noon on the current day although the exact time varies. The file is usually available between 14:00 and 16:00 of the same day and it is polled for periodically to ensure that the processing is performed as quickly as possible. Once the magnetogram has been retrieved, it is used to generate a harmonics file and the resulting data used to drive an AWSoM steady-state simulation. The AWSoM run typically requires roughly 14 hours to complete. At the end of this simulation, a buffer file is produced which can be used as a boundary condition for driving the SWIFT simulation.

The next stage is to run the SWIFT code as a time-accurate simulation using the buffers generated by AWSoM. SWIFT is first restarted using a restart dump from the previous day's run. This restart corresponds to the simulation time for the previous day's AWSoM buffer. It reads that buffer in addition to the newly generated one from today's magnetogram and drives the SWIFT simulation forwards a day by interpolation between the two sets of buffer values to generate intermediate times. Once the simulation reaches the time corresponding to the current day's AWSoM buffer, it generates a restart dump to use for starting the next day's simulation and then continues running for a further 4 days keeping the boundary values fixed. Throughout this simulation, various physical quantities are sampled at the L1 Lagrange point to be used for predictions. These are written into a self-describing binary file format.

At the end of the simulation, a script extracts the time-history information from the binary output and writes it to a JSON file which is uploaded to a web page. This process is carried out automatically. An archive of all predictive data can be found at the URL `https://warwick.ac.uk/fac/sci/physics/research/cfsa/people/bennett/swift-data`. Both the AWSoM code and SWIFT run in the Heliographic coordinate system (HGC) which rotates with the Sun. Since it is most convenient for other members of the team to work in Geocentric Solar Magnetic coordinates (GSM), the data is automatically converted to this form before being uploaded.

All data generated by the coupled codes and uploaded to the previously mentioned web archive have been carried out using fixed versions of the code and parameter files. For SWMF, the version used is the 2017-06-26 tarball which corresponds with version 9.20 of the BATS-R-US code and version 2.40 of the SWMF CON library. All SWIFT runs are carried out using version 1.0 of the code. The parameter files used are contained in the "SWMF_FILES" subdirectory of the SWIFT git repository. These are SWMF_FILES/PARAM.in.awsomr for the SWMF code and SWMF_FILES/input.txt for the SWIFT code. The code versions are written as part of the metadata in the JSON files. If either model is updated in future work then these will be archived to a separate web page and the version change clearly documented.

4 Visualisation tools

The SWIFT git repository comes with several tools for visualising the output generated by the AWSoM and SWIFT codes. These are python utilities that make use of the matplotlib library for generating figures and can be found in the "scripts" subdirectory of the SWIFT repository. The most commonly used utility is that for producing time-history plots for the data sampled at the L1 point. Typical usage is as follows:

```
> ./scripts/time_history.py -C \
        SWMF_FILES/sat_earth_traj_n000000.sat Data/timehistory.sdf
```

This plots a time-history of SWIFT data at L1 and compares it with an SWMF simulation and the OMNI data over the same time period. Supplying the "-C" flag will restrict the date range of the plot to just those dates for nearest Carrington rotation number. The results of this plotting command is shown in Figure 1 for Carrington rotation 2098. It is possible to pass any number of data files as input to this script. For example, passing only the SWIFT data file will produce the same plot with the SWMF comparison omitted. Full help on all the available options can be found by passing the "-h" flag.

Since SWIFT is a 3D code, the time-histories from a single point are often insufficient for understanding how the system is behaving as a whole. To help visualise the full 3D model, a script is provided that can produce movies of a 2D slice through the simulation in either the ecliptic plane or the plane perpendicular to the ecliptic and Sun-Earth line. An example use of this script is as follows:

```
> ./scripts/sdf_movie.py -f Data/0000.sdf Velocity_Vr_xy
```

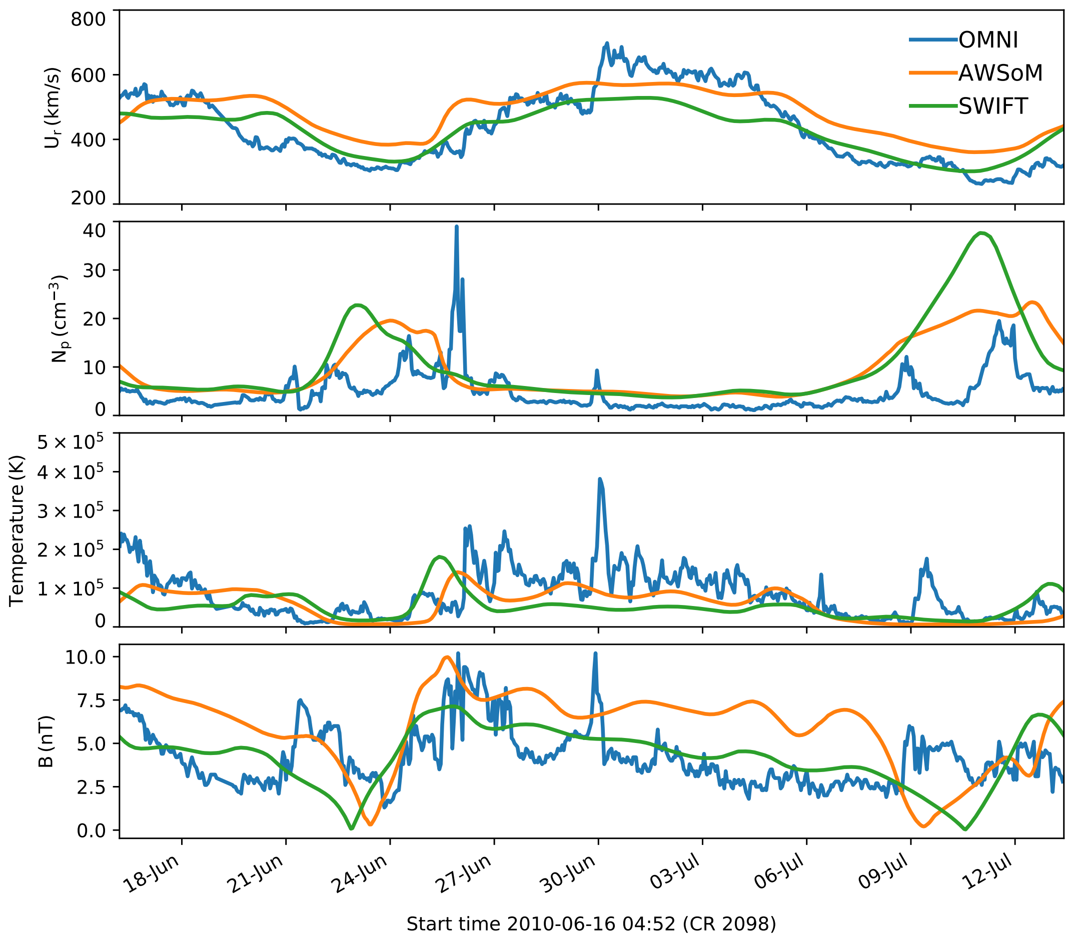

Figure 1: Time history

This will generate a movie of the radial velocity component in the ecliptic plane plotted over the runtime of the simulation. An example frame from this movie is shown in Figure 2. A complete list of variable available for plotting is generated by passing the "-l" flag to the script. Full help on the available options can be found by passing the "-h" flag.

Since the results of the SWIFT simulation are entirely determined by the driver values used from the AWSoM model, it is often useful to be able to plot the contents of the buffer files. SWIFT comes supplied with a script for reading and plotting the contents of these files. It is invoked as follows:

```
> ./scripts/read_buffer.py -pc \
```

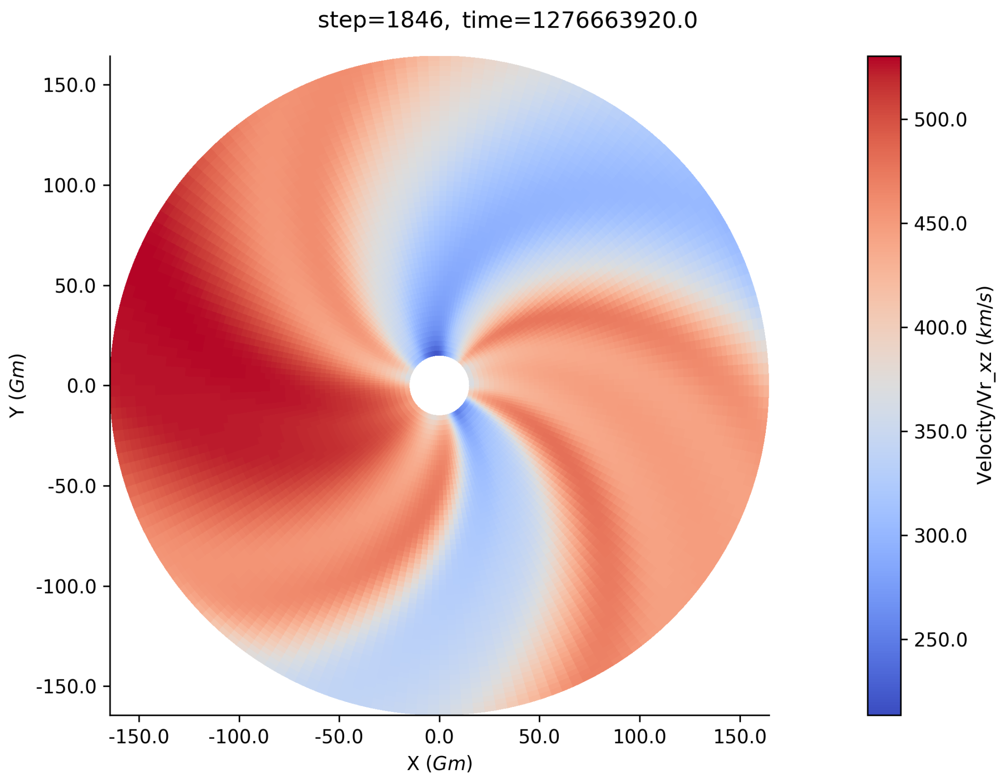

Figure 2: An example movie frame for radial velocity

```
./SWMF_FILES/R=22Rs_2010_06_16_04_52_00.out
```

Here, the "-p" flag specifies that a plot should be generated and the "-c" flag is used to only plot the Cartesian components of the velocity and magnetic field vectors. The plot generated by this command is presented in Figure 3.

In addition to the python utilites described above, SWIFT also comes with several libraries for reading the data into general purpose visualisation tools. All of the scripts supplied make use of the Python matplotlib library and an SDF python module for loading the data files. The SDF module is built and installed as part of the SWIFT build procedure described above. Loading data from within a Python interpreter can be carried out as follows:

```
>>> import sdf_helper as sh
>>> vars = sh.getdata('Data/0000.sdf')
```

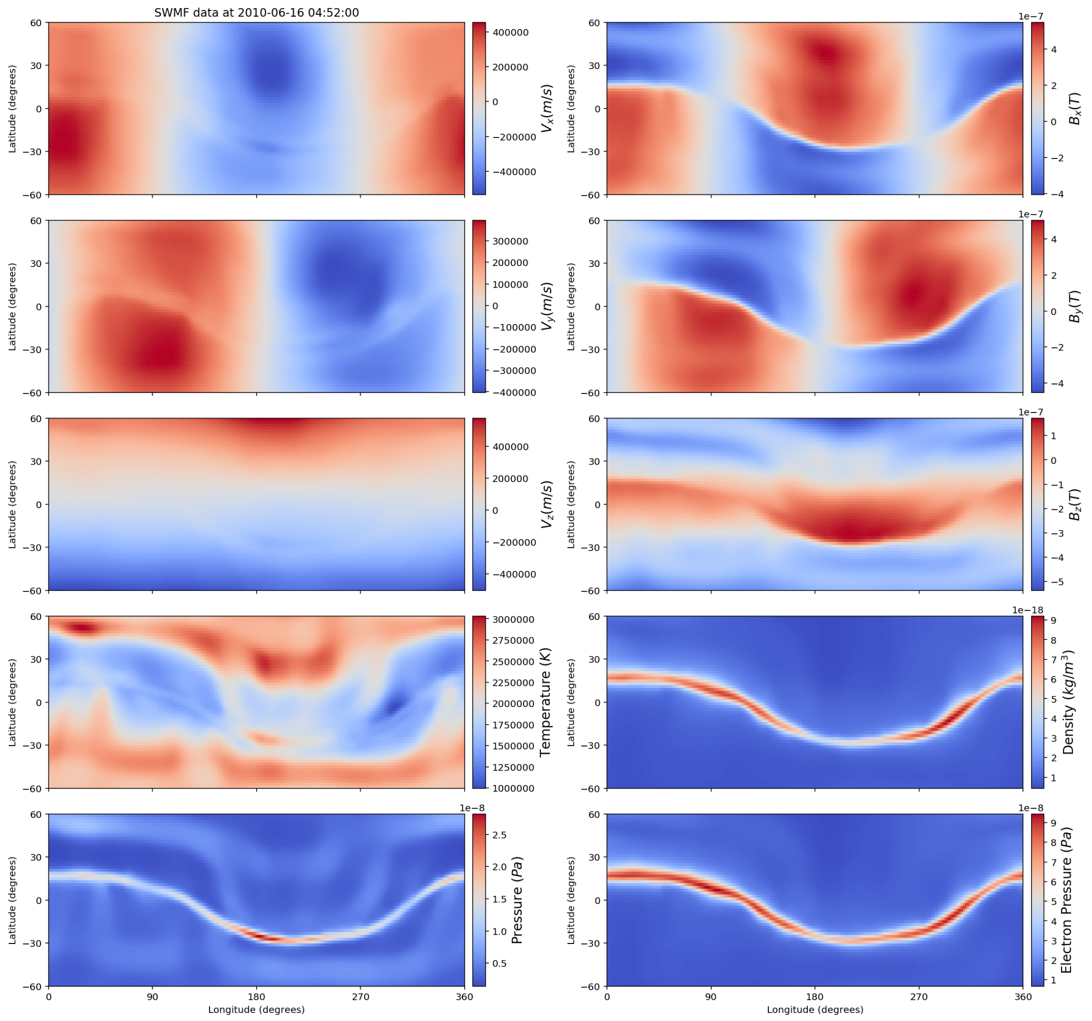

Figure 3: Buffer values at 22.5 $R_\odot$ produced by AWSoM

```
>>> sh.list_variables(vars)
```

The "vars" variable will then contain a data structure containing metadata for all of the variables contained in the SDF file. The "list_variables" function call will list all of the variables along with their types and dimensions. Each variable has a "data" member which will contain a NumPy array holding that variable's data values. Note that this is implemented using a callback mechanism so that the data is only read from the file when it is first used. Each variable also has a "grid" member which references the grid on which that variable is defined.

The SDF subdirectory of the SWIFT repository also comes with reader plugins for

MatLab, IDL and VisIt.

5 System requirements

All of the codes involved in the coupled simulation assume the use of a UNIX based operating system. Most things should work on MacOS although not everything has been fully tested on such a setup. Since most large parallel machines are based on Linux, we will restrict ourselves to listing the packages required for running on a standard Ubuntu installation.

Downloading the Warwick git repositories which contain the "SWIFT" and "SPICE" codes requires the "git" command-line utility to be installed on your system. For Ubuntu, this just requires the "git" package to be installed.

There are several dependencies that are required in order for the build process to be carried out successfully. In particular, it is necessary to have a working Fortran compiler and MPI library. It is also necessary to have Python and Perl installed. For a basic Ubuntu installation, the following packages must be installed: curl, make, g++, gfortran, libopenmpi-dev, perl, python, libpython-dev, python-pip.

For running the code and analysing the output, it is also necessary to have the following packages: python-astropy, python-matplotlib. Finally, there are a couple of python modules that do not come as a standard package in Ubuntu. The preferred method of obtaining these is via the "pip" python package manager. The first of these modules is "ai.cdas" which is necessary for downloading the OMNI data used in generating comparison plots. This can easily be added by the user with the command

```
> pip install --user ai.cdas
```

The second library that is needed is the "spiceypy" module which provides a python interface to the SPICE coordinate system library. This is used when transforming the data from HGC coordinates to GSM coordinates. The library can be installed using the following command:

```
> pip install --user --no-binary spiceypy spiceypy
```

6 Equations solved and numerical implementation

6.1 Core Equations

The core method in SWIFT is based around that for the *Lare2d* documented in Reference [1]. This reference describes the core algorithm for a Cartesian grid. In this report we summarise the basic equations solved and how the SWIFT implementation differs from that of *Lare2d*.

The equations solved in SWIFT, in S.I. units, are the standard ideal-MHD equations modified for two-temperature, shock viscosity and thermal conduction.

$$\begin{aligned}
\frac{\partial\rho}{\partial t} &= -\nabla\cdot(\rho\mathbf{v}) && (1)\\
\frac{\mathrm{D}\mathbf{v}}{\mathrm{D}t} &= \frac{1}{\rho}\mathbf{j}\times\mathbf{B} - \frac{1}{\rho}\nabla(P_e+P_i) + \mathbf{F}_{shock} - \rho\mathbf{g} && (2)\\
\frac{\partial\mathbf{B}}{\partial t} &= -\nabla\times\mathbf{E} && (3)\\
\frac{\mathrm{D}\epsilon_i}{\mathrm{D}t} &= -\frac{P_i}{\rho}\nabla\cdot\mathbf{v} + Q_{shock} && (4)\\
\frac{\mathrm{D}\epsilon_e}{\mathrm{D}t} &= -\frac{P_e}{\rho}\nabla\cdot\mathbf{v} - \nabla.\mathbf{q} && (5)\\
\mathbf{E}+\mathbf{v}\times\mathbf{B} &= 0.0 && (6)\\
\nabla\times\mathbf{B} &= \mu_0\mathbf{j} && (7)
\end{aligned}$$

Where D/Dt is the advective derivative, ρ is the total mass density, $\mathbf{v}$ is the centre-of-mass fluid velocity, $\mathbf{B}, \mathbf{E}$ are the magnetic and electric field in MHD, $\mathbf{j}$ is the current density $\mathbf{g}$ is the acceleration due to gravity and ϵ_e and ϵ_i are the electron and ion specific internal energy desities. The non-ideal MHD terms $\mathbf{F}_{shock}$ and Q_{shock} are the shock viscosity and its associated ion heating, described below, and thermal conduction for the electrons $\nabla.\mathbf{q}$, where $\mathbf{q}$ is the electron heat flux.

Definitions useful for converting between ϵ and the more familiar pressure and tem-

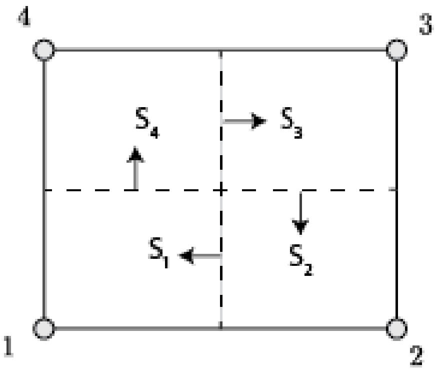

Figure 4: Labels used for edge viscosity.

perature are

$$\begin{aligned} P &= \frac{\rho k_B T}{\mu_m} \\ \epsilon &= \frac{P}{\rho(\gamma-1)} = \frac{k_B T}{\mu_m(\gamma-1)} \end{aligned}$$

where μ_m is the reduced mass, i.e. the average mass of all particles in the plasma. Hence $\mu_m = m_p$ for neutral hydrogen atoms (m_p is the proton mass) and $\mu_m = 0.5m_p$ for fully ionised hydrogen.

6.2 Edge viscosity

The shock viscosity in SWIFT is based on the edge viscosity formulation in Reference [3]. This is presented first for Euler's equation only and MHD added later. First find the area weighted nodal density and sound speed ρ_v and cs_v. Then for each edge of the cell find $cs_i = \min(cs_v^i, cv_v^{i+1})$ where i labels the nodes of the cell as in Figure 4 and is cyclic so for $i = 1$, $i - 1 = 4$ etc.

Next define the edge density through $\rho_i = 2\rho_v^i \rho_v^{i+1}/(\rho_v^i + \rho_v^{i+1})$. Then on edge i the associated viscous force is

$$\mathbf{f}_i = \rho_i\{\bar{c}_2 \Delta v_i + (\bar{c}_2^2 \Delta v_i^2 + c_1^2 cs_i^2)^{1/2}\}(1-\psi_i)(\Delta\hat{\mathbf{v}}_i.\mathbf{S}_i)\Delta\mathbf{v}_i \tag{8}$$

with $\bar{c}_2 = c_2(\gamma+1)/4$ and usually $c_1 = c_2 = 1$. The velocity difference is $\Delta\mathbf{v}_i = \mathbf{v}_i - \mathbf{v}_{i+1}$ and the edge force is only applied if $\Delta\hat{\mathbf{v}}_i.\mathbf{S}_i < 0$, i.e. cell edge compression. In these expressions the median mesh vector $\mathbf{S}_i$ is normal to the median mesh, points anti-clockwise and has magnitude of the distance from the centre of the cell to the edge i. This viscous force is applied to node i and $-\mathbf{f}_i$ is applied to cell $i+1$. Thus the viscous force applied to node 1, due to the edge viscosity, is $\mathbf{f}_1^{visc} = \mathbf{f}_1 - \mathbf{f}_4$ from this cell. There will be 4 similar contributions to the total viscous force on this node from srounding cells. The compatible heating in the cell due to shock viscosity on its edges is

$$\rho \frac{D\epsilon_i}{Dt} = -\frac{1}{C_v} \sum_{i=1}^{4} \mathbf{f}_i . \Delta \mathbf{v}_i = \rho Q_{shock} \tag{9}$$

The function ψ_i acts as a limiter to turn off viscosity in smooth regions of flow. It is defined through

$$\psi_i = \max \left\{ 0, \min \left(\frac{r_{Li} + r_{ri}}{2}, 2r_{Li}, 2r_{ri} \right) \right\} \tag{10}$$

with

$$r_{Li} = \frac{\Delta \mathbf{v}_a . \Delta \hat{\mathbf{v}}_i}{\Delta \mathbf{x}_a . \Delta \hat{\mathbf{x}}_i} \frac{|\Delta \mathbf{x}_i|}{|\Delta \mathbf{v}_i|} \quad ; \quad r_{ri} = \frac{\Delta \mathbf{v}_b . \Delta \hat{\mathbf{v}}_i}{\Delta \mathbf{x}_b . \Delta \hat{\mathbf{x}}_i} \frac{|\Delta \mathbf{x}_i|}{|\Delta \mathbf{v}_i|} \tag{11}$$

Here the indexing $\Delta \mathbf{x}_a$ differs from that in Reference [3] as these differences are taken along an index line, not around cell edges. For example for edge 1 $\Delta \mathbf{v}_1 = \mathbf{v}_1 - \mathbf{v}_2$ where the subscripts match the node labelling scheme in Figure 4. If this is cell (ix, iy) then $\Delta \mathbf{v}_a$ is the same difference but for cell $(ix+1, iy)$ and $\Delta \mathbf{v}_a$ that for cell $(ix-1, iy)$.

When this viscosity is applied to MHD problems we simply replace the sound speed by an effective fast speed c_f defined through $c_f^2 = c_s^2 + v_A^2$ where v_A is the local Alfvén speed.

6.3 Thermal conduction

The thermal conduction is user configurable to be either a flux limited Braginskii (Spitzer-Harm) thermal flux or the simplified model from Hollweg used in AWSoM [2]. The Braginskii thermal conduction model, in the presence of a magnetic field, in SWIFT is of the form

$$\rho \frac{\partial \epsilon}{\partial t} = \nabla . \left(\vec{k} . \nabla T \vec{n} \right) + \nabla . \left(\frac{b_{min}^2}{B^2 + b_{min}^2} \kappa \nabla T \right)$$

where $\vec{k} = \kappa\vec{n}$, $\vec{n} = \vec{B}/(B^2 + b_{min}^2)^{1/2}$ and $\kappa = \kappa_0 T^{\frac{5}{2}}$. In the limit $b_{min} \to 0$ this recovers the Braginskii parallel thermal conductivity. Firstly the conduction is written in terms of the heat flux vector $\vec{q}$ so that

$$\vec{q} = \left(\vec{k}.\nabla T\right)\vec{n} + \frac{b_{min}^2}{B^2 + b_{min}^2}\kappa\nabla T$$

Then the specific energy density in each cell is updated using super-stepping [4]

It is possible that sufficient heating occurs so that qx exceeds the free streaming heat flux $q_f = v_{the}k_BT$. In this case the Braginskii flux needs to be limited. This is done by calculating the components of the Braginskii (Spitzer-Harn) heat flux, e.g. qsh_x and the free streaming limit q_f and then finding the non-linear limited flux through

$$q_{nl} = \frac{1}{(1/q_sh + 1/q_f)}$$

The heat flux must be limited not to $q_f = v_{the}k_BT$ but to $q_f = F_Lv_{the}k_BT$ where F_L is a flux limiter which is user configurable but is typically of order one.

As a simpler, and much faster, approximation Hollweg [5] suggested that for the solar wind a quick-fix electron heat flux could be through taking $\mathbf{q} = \alpha_e P_e \mathbf{v}$ where α_e is a tunable parameter of order one. SWIFT can use either the Hollweg model or a flux limited Braginkii conduction.

6.4 Centrifugal and geometry terms

The coupled AWSoM-SWIFT simulations are run in HGR coordinates so the centrfugal and coriolis forces $-\rho(\mathbf{\Omega} \times (\mathbf{\Omega} \times \mathbf{r} + 2\mathbf{\Omega} \times \mathbf{r})$ are added to the moment equation. In addition to this spherical geometry terms are added. These arise due to the Lagrangian step updating the scalar components of the vector $\mathbf{v}$, i.e. v_r, v_θ, v_ϕ in

$$\mathbf{v} = v_r\hat{\mathbf{r}} + v_\theta\hat{\theta} + v_\phi\hat{\phi}$$

So The Lagrangian equations require the expanion

$$\frac{\mathrm{D}\mathbf{v}}{\mathrm{D}t} = \hat{\mathbf{r}}\frac{\mathrm{D}v_r}{\mathrm{D}t} + v_r\frac{\mathrm{D}\hat{\mathbf{r}}}{\mathrm{D}t} + \hat{\theta}\frac{\mathrm{D}v_\theta}{\mathrm{D}t} + v_\theta\frac{\mathrm{D}\hat{\theta}}{\mathrm{D}t} + \hat{\phi}\frac{\mathrm{D}v_r}{\mathrm{D}t} + v_\phi\frac{\mathrm{D}\hat{\phi}}{\mathrm{D}t}$$

with the derivatives of the unit vectors specified in spherical coordiates in the usual way [6].

7 Grid definitions

The mathematical and algorithm description in the previous section, along wth the main *LareXd* reference [1] are sufficient to follow the implementation as in SWIFT except for the grid definitions. This is best explained by stepping up through 1D then 2D and finally 3D computational cells in Cartesian geometry and then the scale factors needed to implement this on a spherical grid.. In 1D there are nx computational cells. These are labelled from $ix = 1$ up to $ix = nx$. The variables used in the code are not defined at the same point in a cell. We therefore need to be able to access the location of different parts of the grid to set up the initial conditions. For this xb_i is the position of the right hand boundary of a cell and xc_i the position of the cell centre. This means that the left hand boundary of the computational domain is at xb_0. The width of each cell is dxb_i and the distance between cell centres is dxc_i. In this coordinate system the velocities are all defined at the cell boundaries and all scalars (density, pressure, specific internal energy density) are defined at cell centres. This is shown in Figure 7. The magnetic field components are defined at different locations for each component. B_x is defined at the cell boundary (xb_i) while the B_y and B_z components are cell centred. Note that this staggering is essential for the accuracy and conservation properties of SWIFT.

In 2D the velocities are defined at cell corners, the scalars and out of plane magnetic field component at cell centres. The remaining magnetic field components are defined at cell edges. This is shown in Figure 6.

The 3D Cartesian is a natural extension of this staggering and is shown in Figure 7. Here scalars are volume centred, velocities are defined at cell vertices and magnetic field components are defined as face centred. Gravity is a function only of the z coordinate and is defined at the cell vertices, i.e. same location as the velocity.

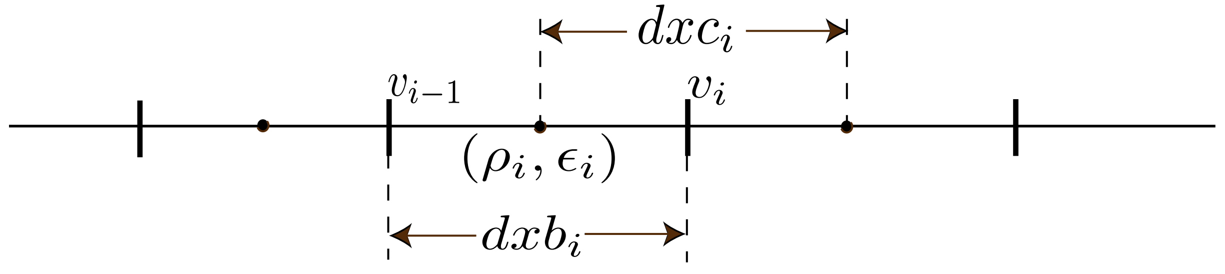

Figure 5: 1D Staggered Grid

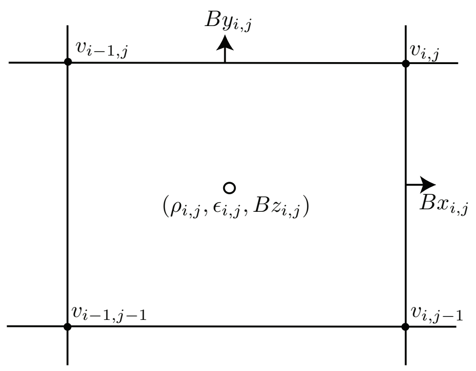

Figure 6: 2D Staggered Cartesian Grid

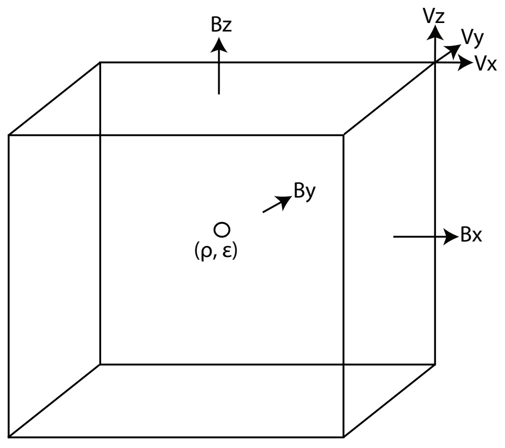

Figure 7: 3D Staggered Cartesian Grid

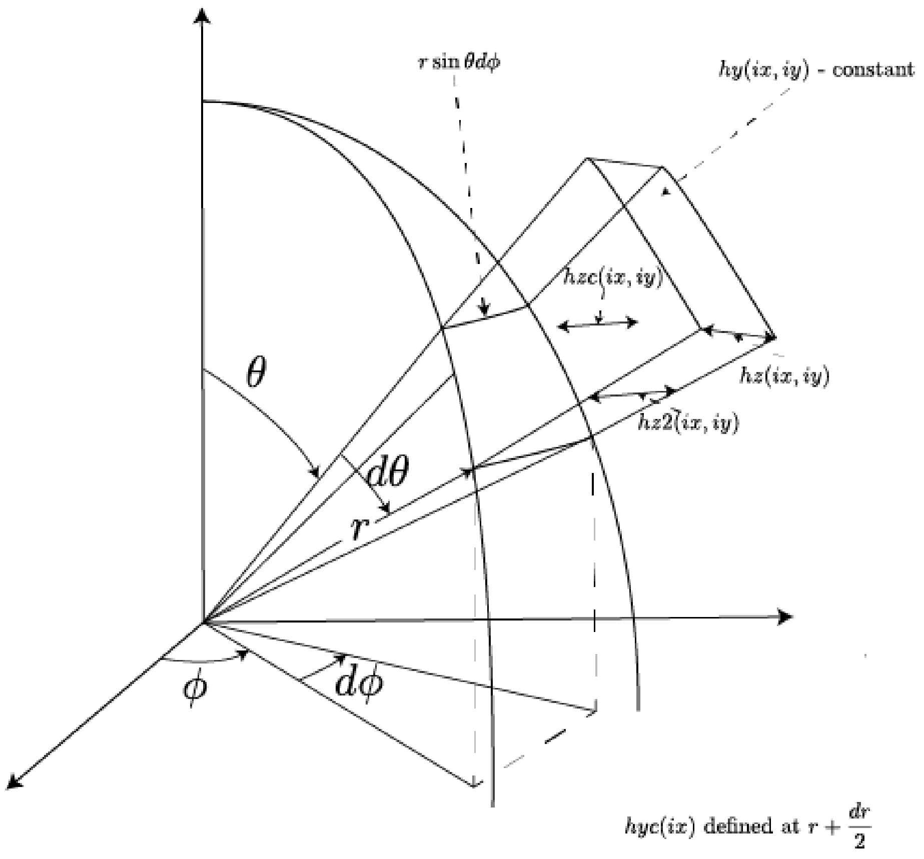

Figure 8: Scale factors used in SWIFT for spherical coordinates

To implement the numerical derivatives, volume averages and fluxes needed for a Lagrangian-remap code [1] additional scale factors are required. These are shown in Figure 8 using the variable names and notion of the SWIFT code.

8 Appendix

8.1 The build.sh script

```
#! /bin/bash

cd $(dirname ${BASH_SOURCE[0]})
dir=$(pwd)
cd ..

# Build SWMF
########################################################################

cd ../SWMF*

```

```
cluster=0
hostname -f | grep tinis > /dev/null && cluster=1

./Config.pl -uninstall

# Always compile the HARMONICS tool in serial
./Config.pl -install -nompi -compiler=gfortran
compiler=gfortran

(cd share/Library/src
make LIB)
(cd util/DATAREAD/srcMagnetogram
make HARMONICS)

mv bin/HARMONICS.exe .

./Config.pl -uninstall

mkdir bin
mv HARMONICS.exe bin/

# Now switch to parallel flags

if [ $cluster -eq 0 ]; then
  ./Config.pl -install -compiler=gfortran
  compiler=gfortran
else
  module purge 2> /dev/null
  module load intel impi HDF5 2> /dev/null
  export I_MPI_F90=ifort
  #./Config.pl -install -nompi -compiler=ifort
  ./Config.pl -install -compiler=mpiifort -hdf5
  compiler=intel
fi

./Config.pl -v=Empty,SC/BATSRUS,IH/BATSRUS
./Config.pl -o=SC:u=ScChromo,e=MhdWavesPe
./Config.pl -o=IH:u=ScChromo,e=MhdWavesPe
./Config.pl -g=SC:4,8,4,2000,1,IH:4,4,4,4500,1

make SWMF

cd -


# Build SWIFT
##########################################################################

make cleanall
make -j COMPILER=$compiler
make sdfutils


# Build SPICE
######################################################################

cd ../SPICE
make
cd -

```

8.2 The setup.sh script

```
#! /bin/bash

# Carrington rotation number to use
CR=2098

# steady-state magnetogram file prefix
pre=mqs

# Parse command-line arguments
usage() {
  echo "Usage: $0 [-c <Carrington rotation>] [-r <rundir>]" 1>&2; exit 1;
}

while getopts ":c:r:" o; do
  case "${o}" in
    c) CR=${OPTARG} ;;
    r) rundir=${OPTARG} ;;
    *) usage ;;
  esac
done

# Setup SWMF
######################################################################

cd $(dirname ${BASH_SOURCE[0]})
dir=$(pwd)
topdir=$(pwd)/../..
cd -

# Get magnetogram data
######################################################################

# Download the next GONG file if not already done
cd $topdir/SWMF*
swmfdir=$(pwd)
[ -d GONG_DATA ] || mkdir GONG_DATA
```

```
cd GONG_DATA

cd $dir
crdates=$(grep "^$CR" gongmonths.txt)
IFS=', ' read -r -a cra <<< "$crdates"
cd -

# Mid date for Carrington rotation
tt=${cra[3]:0:6}${cra[4]:0:4}
if [ ${tt:0:2} -gt 50 ]; then
  tt=19$tt
else
  tt=20$tt
fi
dd=${tt:0:8}

ftp=https://gong.nso.edu/data/magmap
fitspath=$ftp/QR/$pre/${dd:0:6}/mr$pre${dd:2:6}/
fitsfile=mr$pre${dd:2:6}t${tt:8:4}c${CR}_000.fits.gz
fullfitsfile=$fitspath/$fitsfile

if [ ! -e $fitsfile ]; then
  # Download magnetogram file
  curl -O $fullfitsfile
fi

# Start date for Carrington rotation
t0=${cra[1]:0:6}${cra[2]:0:4}
if [ ${t0:0:2} -gt 50 ]; then
  t0=19$t0
else
  t0=20$t0
fi
d0=${t0:0:8}

Y=${t0:0:4}
M=${t0:4:2}
D=${t0:6:2}
h=${t0:8:2}
m=${t0:10:2}
date=${Y}_${M}_${D}_${h}_${m}

echo $date

# Create run directory
########################################################################

cd $swmfdir
if [ "$rundir"x = x ]; then
  rundir=$swmfdir/run_CR$CR
```

```
else
  if [ ! -d "$(dirname $rundir)" ]; then
    rundir=$swmfdir/$rundir
  fi
fi

# Make rundir, moving old one out of the way if necessary
cd $swmfdir
[ -e $rundir ] && mv $rundir $(date +"${rundir}_%F_%R")
make rundir RUNDIR=$rundir

# Copy parameter files into the run directory
cd $dir
cp HARMONICS.in LAYOUT.in PARAM.in.awsomr run.sh $rundir/
cp -r ../scripts/spice_files $rundir/data

# Adjust values in PARAM.in to match magnetogram
cd $rundir
sed "s/[0-9]*\([^0-9]*iYear\)/$Y\1/;   s/[0-9]*\([^0-9]*iMonth\)/$M\1/; \
     s/[0-9]*\([^0-9]*iDay\)/$D\1/;    s/[0-9]*\([^0-9]*iHour\)/$h\1/; \
     s/[0-9]*\([^0-9]*iMinute\)/$m\1/; s/[0-9]*\([^0-9]*iSecond\)/0\1/" \
    PARAM.in.awsomr > PARAM.in

# Make satellite trajectory files

# earth
sed "s/.*\( #START\)/$Y-$M-$D $h:$m  \1/; \
     s/.*\( #FILENAME\)/earth_traj.dat  \1/; \
     s/.*\( #SATELLITE\)/EARTH  \1/" data/input.txt > __tmp__

$topdir/SPICE/bin/cstates < __tmp__

# stereo-a
sed "s/.*\( #START\)/$Y-$M-$D $h:$m  \1/; \
     s/.*\( #FILENAME\)/sta_traj.dat  \1/; \
     s/.*\( #SATELLITE\)/STEREO AHEAD  \1/" data/input.txt > __tmp__

$topdir/SPICE/bin/cstates < __tmp__

# stereo-b
sed "s/.*\( #START\)/$Y-$M-$D $h:$m  \1/; \
     s/.*\( #FILENAME\)/stb_traj.dat  \1/; \
     s/.*\( #SATELLITE\)/STEREO BEHIND  \1/" data/input.txt > __tmp__

$topdir/SPICE/bin/cstates < __tmp__

rm __tmp__

# Convert magnetogram to harmonics file
```

```
module list > /dev/null 2>&1
if [ $? -eq 0 ]; then
  source /etc/profile.d/00-modulepath.sh 2> /dev/null
  source /etc/profile.d/z00_lmod.sh 2> /dev/null
  source /etc/profile.d/impi.sh 2> /dev/null

  module purge 2> /dev/null
  module load intel impi Python/2.7.12 2> /dev/null
fi
source $topdir/py/bin/activate > /dev/null 2>&1

$swmfdir/util/DATAREAD/srcMagnetogram/read_fits.py \
    -Out new $swmfdir/GONG_DATA/$fitsfile
$swmfdir/bin/HARMONICS.exe

echo "Start date $Y$M${D}t$h$m" at $(date)
echo `pwd`
echo CR$CR > start.txt
```

8.3 The autosubmit_predictive.sh script

```
#! /bin/bash

# Carrington rotation number or date to start at ("now" for current time)
# Date is used if both specified
CR=2190
start_date="2017-10-01 12:00:00"
start_date="now"
# Date format (only needed for BSD systems)
date_format="%Y-%m-%d %H:%M:%S"

# Time interval between updates (in seconds)
interval=$((60*60*24))

# time-accurate magnetogram file prefix
pre=bqs

# Setup SWMF
#######################################################################

cd $(dirname ${BASH_SOURCE[0]})
dir=$(pwd)
topdir=$dir/../..
cd -

# Get magnetogram data
#######################################################################

# Download the next GONG file if not already done
```

```
cd $topdir/SWMF*
swmfdir=$(pwd)
[ -d GONG_DATA ] || mkdir GONG_DATA
cd GONG_DATA

# Check for GNU or BSD version of date command
date -j > /dev/null 2>&1
gnu=$?
if [ $gnu -eq 1 ]; then
  arg="-d @"
else
  arg="-jf %s "
fi

prev=previous_gong_file.txt
#ftp=ftp://gong2.nso.edu
ftp=https://gong.nso.edu/data/magmap

if [ -e $prev ]; then
  prev_date=$(cat $prev)
  need_date=$((prev_date+interval))
  start_date=
else
  if [ "$start_date"x != x ]; then
    # Use specified start date
    if [ "$start_date" = "now" ]; then
      need_date=$(date +%s)
    else
      if [ $gnu -eq 1 ]; then
        need_date=$(date -d "$start_date" +%s)
      else
        need_date=$(date -jf "%Y-%m-%d %H:%M:%S" "$start_date" +%s)
      fi
    fi
  else
    # Use Carrington rotation instead

    cd $dir
    crdates=$(grep "^$CR" gongmonths.txt)
    IFS=', ' read -r -a cra <<< "$crdates"
    cd -

    # Start date for Carrington rotation
    t0=${cra[1]:0:6}${cra[2]:0:4}
    if [ ${t0:0:2} -gt 50 ]; then
      t0=19$t0
    else
      t0=20$t0
    fi
    d0=${t0:0:8}
    dt="$d0 ${t0:8:2}:${t0:10:2}:00"
    if [ $gnu -eq 1 ]; then
      need_date=$(date -d "$dt" +%s)
    else
      need_date=$(date -jf "%Y%m%d %H:%M:%S" "$dt" +%s)
    fi
  fi

  prev_date=$((need_date - interval))
fi

got_date=-1
if [ $need_date -gt 0 ]; then
  # Check for files within half-interval of the start time
  sec=$((need_date - interval/2))
  d1=$(date $arg$sec +"%Y%m%d")
  sec=$((need_date + interval/2))
  d2=$(date $arg$sec +"%Y%m%d")
  got_date=-1
  mindt=$((2*interval))
  fitsfile=""
  for f in $(ls mr$pre${d1:2}t* mr$pre${d0:2}t* \
                mr$pre${d2:2}t* 2>/dev/null | sort -u); do
    if [ ${f:5:2} -gt 50 ]; then
      ml=19
    else
      ml=20
    fi

    dt="$ml${f:5:6} ${f:12:2}:${f:14:2}:00"
    if [ $gnu -eq 1 ]; then
      sec=$(date -d "$dt" +%s)
    else
      sec=$(date -jf "%Y%m%d %H:%M:%S" "$dt" +%s)
    fi

    ds=$((sec-need_date))
    if [ $ds -lt 0 ]; then
      ds=$((-ds))
    fi

    # Find file with minimum time difference from starting time
    if [ $ds -lt $mindt -a $sec -gt $prev_date ]; then
      got_date=$sec
      mindt=$ds
      fitsfile=$f
    fi
  done
  # If minimum time difference is greater than 1 hour, need to
  # download file
  if [ $mindt -gt $((55*60)) ]; then
    got_date=-1
  else
    need_date=-1
  fi
fi

if [ $need_date -gt 0 ]; then
  sec1=$((need_date - interval))
  sec2=$((need_date + interval))
  d0=$(date $arg$need_date  +"%Y%m%d")
  d1=$(date $arg$sec1 +"%Y%m%d")
  d2=$(date $arg$sec2 +"%Y%m%d")
  got_date=-1
  mindt=$((2*interval))
  fitsfile=""
  fullfitsfile=""
  echo future
  for dd in $(echo -e "$d0\n$d1\n$d2" | sort -u); do
    echo $dd
    fitspath=$ftp/QR/$pre/${dd:0:6}/mr$pre${dd:2:6}/
    echo $fitspath
    if [ "${ftp:0:1}" = "f" ]; then
      flist=$(curl --list-only -s $fitspath)
    else
      flist=$(curl --list-only -s $fitspath | grep href | grep fits \
              | sed 's/.*href=//' | cut -f2 -d\")
    fi
    for f in $flist; do
      if [ ${f:5:2} -gt 50 ]; then
        ml=19
      else
        ml=20
      fi

      dt="$ml${f:5:6} ${f:12:2}:${f:14:2}:00"
      if [ $gnu -eq 1 ]; then
        sec=$(date -d "$dt" +%s)
      else
        sec=$(date -jf "%Y%m%d %H:%M:%S" "$dt" +%s)
      fi

      ds=$((sec-need_date))
      if [ $ds -lt 0 ]; then
        ds=$((-ds))
      fi

      echo "found $f (ds $ds)"
      # Find file with minimum time difference from starting time
      if [ $ds -lt $mindt -a $sec -gt $prev_date ]; then
        got_date=$sec
        mindt=$ds
        fitsfile=$f
        fullfitsfile=$fitspath/$fitsfile
      fi
    done
  done

  if [ $mindt -gt $((60*30)) -a "$start_date" != "now" ]; then
    got_date=-1
  else
    # Download magnetogram file
    curl -O $fullfitsfile
    echo curl -O $fullfitsfile
  fi
fi

if [ $got_date -lt 0 ]; then
  echo exiting
  exit
fi


# Get date information from the magnetogram file

gzip -dc $fitsfile | head -1 | strings | tr '=' '\n' > head.txt
date=$(grep -A1 DATE-OBS head.txt | tail -1 | cut -f2 -d\')
time=$(grep -A1 TIME-OBS head.txt | tail -1 | cut -f2 -d\')
phishift=$(grep -A1 MAPEDGE head.txt | tail -1 | awk '{print$1}')
rm head.txt

Y=${date:0:4}
M=${date:5:2}
D=${date:8:2}
h=${time:0:2}
m=${time:3:2}
date=${Y}_${M}_${D}_${h}_${m}

echo $got_date > $prev

# Create run directory
######################################################################

rundir=$swmfdir/$(date $arg$got_date  +"run_%Y%m%dt%H%M")

# Make rundir, moving old one out of the way if necessary
cd $swmfdir
```

```
[ -e $rundir ] && mv $rundir $(date +"${rundir}_%F_%R")
make rundir RUNDIR=$rundir

# Copy parameter files into the run directory
cd $dir
cp HARMONICS.in LAYOUT.in PARAM.in.awsomr $rundir/
cp -r ../scripts/spice_files $rundir/data
cp run_pred.sh $rundir/run.sh

# Adjust values in PARAM.in to match magnetogram
cd $rundir
sed "s/[0-9]*\([^0-9]*iYear\)/$Y\1/;   s/[0-9]*\([^0-9]*iMonth\)/$M\1/; \
     s/[0-9]*\([^0-9]*iDay\)/$D\1/;    s/[0-9]*\([^0-9]*iHour\)/$h\1/; \
     s/[0-9]*\([^0-9]*iMinute\)/$m\1/; s/[0-9]*\([^0-9]*iSecond\)/0\1/; \
     s/^[0-9\.]*\(.*PhiShift\)/$phishift\1/"  PARAM.in.awsomr > PARAM.in

sed "s/^[0-9\.]*\(.*tSimulationMax\)/0.\1/" PARAM.in.awsomr > __tmp__
mv __tmp__ PARAM.in.awsomr

# Make satellite trajectory files

# earth
sed "s/.*\( #START\)/$Y-$M-$D $h:$m  \1/; \
     s/.*\( #FILENAME\)/earth_traj.dat  \1/; \
     s/.*\( #SATELLITE\)/EARTH  \1/" data/input.txt > __tmp__

$topdir/SPICE/bin/cstates < __tmp__

# stereo-a
sed "s/.*\( #START\)/$Y-$M-$D $h:$m  \1/; \
     s/.*\( #FILENAME\)/sta_traj.dat  \1/; \
     s/.*\( #SATELLITE\)/STEREO AHEAD  \1/" data/input.txt > __tmp__

$topdir/SPICE/bin/cstates < __tmp__

# stereo-b
sed "s/.*\( #START\)/$Y-$M-$D $h:$m  \1/; \
     s/.*\( #FILENAME\)/stb_traj.dat  \1/; \
     s/.*\( #SATELLITE\)/STEREO BEHIND  \1/" data/input.txt > __tmp__

$topdir/SPICE/bin/cstates < __tmp__

rm __tmp__

# Convert magnetogram to harmonics file

module list > /dev/null 2>&1
if [ $? -eq 0 ]; then
  source /etc/profile.d/00-modulepath.sh 2> /dev/null
  source /etc/profile.d/z00_lmod.sh 2> /dev/null
  source /etc/profile.d/impi.sh 2> /dev/null

  module purge 2> /dev/null
  module load intel impi Python/2.7.12 2> /dev/null
fi
source $topdir/py/bin/activate > /dev/null 2>&1

$swmfdir/util/DATAREAD/srcMagnetogram/read_fits.py \
    -Out new $swmfdir/GONG_DATA/$fitsfile
$swmfdir/bin/HARMONICS.exe

echo "Start date $Y$M${D}t$h$m" at $(date)
echo ‘pwd‘
echo $Y$M${D}t$h$m > start.txt

job=$(sbatch --parsable run.sh)
echo $job > ../prev_job.txt
```

References

[1] T D Arber, A W Longbottom, C L Gerrard, and A M Milne. A Staggered Grid, Lagrangian–Eulerian Remap Code for 3-D MHD Simulations. *Journal of Computational Physics*, 171(1):151–181, 2001.

[2] B van der Holst, Igor V Sokolov, X Meng, Meng Jin, Iv W B Manchester, G Tóth, and Tamas I Gombosi. ALFVÉN WAVE SOLAR MODEL (AWSoM): CORONAL HEATING. *Astrophysical Journal*, 782(2):81, 2014.

[3] E. J. Caramana et al., *J. Comp. Phys.* **144** 70 (1998)

[4] C. Meyer, D. Balsara and T. Aslam, *MNRAS* **422** 2101 (2012)

[5] J. V. Hollweg, *Reviews of Geophysics and Space Physics* **16**, No. 4, 689 (1978)

[6] J. D. Huba, *NRL Plasma Formulary. Washington DC: Naval Research Laboratory* (1998)