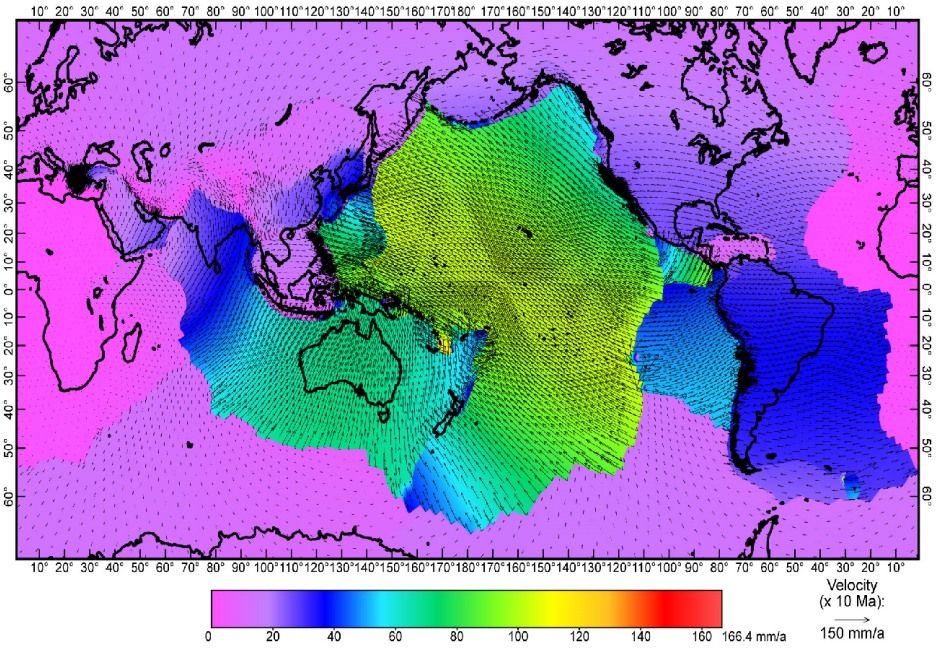
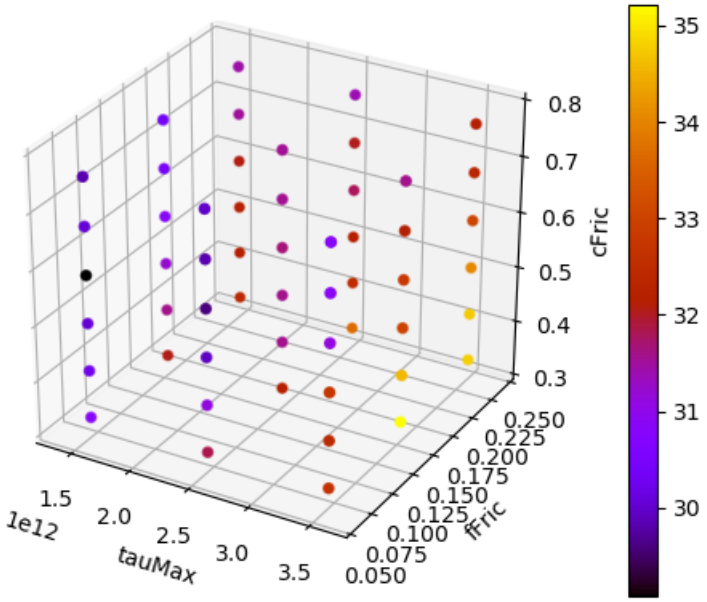
ShellSet v1.1.0 - Parallel Dynamic Neotectonic Modelling A User Guide

Jon B. May[1], Peter Bird[2,1], and Michele M. C. Carafa[1]

[1]Istituto Nazionale di Geofisica e Vulcanologia (INGV)
[2]Department of Earth, Planetary, and Space Sciences, UCLA





Contents

Please contact jonbryan.may@ingv.it with any queries, issues or suggestions.

1 Introduction

ShellSet is an MPI (Message Passing Interface) parallel combination of three existing programs: OrbData5, Shells and OrbScore2. These programs were originally written, and are currently maintained, by Professor Peter Bird and may be found on his website here[1]. This combination into a single MPI framework allowed parallel test options to be added, section 3.1.1 and 3.1.2; the simplification of user input for the three program units, section 3.1.3; additional command line arguments (CLAs) depending on the parallel test option chosen, section 4.2; and the use of a single output file for all models run within a single test, section 5.

Within this guide each individual simulation is referred to as a "model", a combination of models as a "test". Linked websites within the text are repeated inside footnotes for printed versions.

1.1 Repository content

The following table shows an outline of the files and directories found within the ShellSet repository:

Name	Outline
DOCS	Directory containing user guide & other documentation
EXAMPLES	Directory containing two real world examples & a list input example
INPUT	Directory containing all input files
src	Directory containing all Fortran & Python program files
LICENSE	File containing copy of GNU software licence under which ShellSet is released
Makefile	Makefile with 3 versions of ShellSet
README	README file for ShellSet program

Table 1: ShellSet repository information

1.2 Prerequisites & environment setup

ShellSet requires an Intel Fortran compiler, MPI and Intel MKL. Each of these is available for free from Intel through the Intel oneAPI Base & HPC toolkits.

Users with Windows machines that do not currently have access to a Linux environment should read the following installation and setup instructions. Users with an existing Linux environment or Linux OS machine can skip ahead to the paragraph related to the installation of the oneAPI toolkits. ShellSet will function in any Linux environment however, for brevity, only the installation instructions for Windows Subsystem for Linux version 2 (WSL2; free from Microsoft) are given. WSL2 has some advantages over a virtual machine including allowing user access to GPU or other accelerators, although these are not required for ShellSet.

1.2.1 Install WSL2

The first step is to install WSL2 onto a Windows host, the installation depends on the Windows OS version and build number. For complete instructions follow one of the following two links:

1. for newer versions of Windows follow this link[2],
2. users with older Windows versions should use the following link[3].

The shared directory between the WSL2 environment and Windows host is located here: \\wsl$\Ubuntu-20.04\home, adjusting for the Linux flavour and version.

1.2.2 Install Intel oneAPI

The next step is the installation of the Intel oneAPI toolkits, instructions for which can be found here[4], a minimum for ShellSet is the installation of the Base & HPC toolkits. For a complete list of all Intel oneAPI toolkits

[1] http://peterbird.name
[2] https://docs.microsoft.com/en-us/windows/wsl/install
[3] https://docs.microsoft.com/en-us/windows/wsl/install-manual
[4] https://www.intel.com/content/www/us/en/develop/documentation/installation-guide-for-intel-oneapi-toolkits-linux/top/installation/install-using-package-managers/apt.html

available and their contents see here[5]. The installation instructions will depend on the flavour of Linux and each step must be followed from within the Linux environment. In some cases a final setup step for the ifort compiler is required, this can be done (in Ubuntu) by adding the following line to the ".bashrc" file before restarting the terminal:
source /opt/intel/oneapi/setvars.sh > /dev/null 2>&1

1.3 Download and unzip

Once a suitable Linux environment is setup it is possible to start using ShellSet. This user guide should have been included with the complete ShellSet package, if not then the most recent version is available here[6] and the linked Zenodo page, all users should cite the program using its DOI (given on both the GitHub and Zenodo pages). When the download is complete, move the zipped package into a suitable location, for WSL users this should be accessible to the WSL terminal, and unzip the package.

Once unzipped (**unzip ShellSet.zip**), the user should navigate into the newly created directory (**cd ShellSet**) from where they can compile the program ready for use.

2 Compiling

ShellSet comes complete with a Makefile to simplify its compilation. The standard version of ShellSet may be compiled by navigating into the ShellSet directory and entering the following (case sensitive) command:
make ShellSet

Including the basic ShellSet version, the Makefile contains three build targets:

1. **ShellSet** - build the basic version of ShellSet
2. **debug** - build using extra debug flags (initially "-check all -traceback -debug")
3. **optimal** - build using extra optimising flags (initially "-xHost -ipo -unroll-aggressive")

and two clean options:

1. **clean** - clean directory and some sub-directories
2. **cleanall** - clean directory, remove some sub-directories and all .exe files

Each target creates its own private directory where it stores its dependency files: "lib" for **make ShellSet**; "lib_DB" for **make debug**; and "lib_OPT" for **make optimal**. The Makefile allows for all three targets to be built from the same source files without sharing any dependency on compiled files, this means all three targets can be built without any problems from compile flags (such as -ipo), and may be built at the same time using the command **make ShellSet debug optimal**.

To alter the current debug or optimisation flags users should update the Makefile macro **DBFLAGS** or **OPTFLAGS** respectively. For a list of compiler options see the Intel alphabetical option list here[7]. Older versions of the ifort compiler may require a change to the "MKLFLAGS" macro in the Makefile, from "-qmkl=parallel" to "-mkl=parallel".

The compile line flag "-diag-disable 10145" stops the warning "ifort: warning #10145: no action performed for file [*filename*]". The linker line flag "-diag-disable 10182" stops the warning "ifort: warning #10182: disabling optimization; runtime debug checks enabled". These may be removed at the user's discretion.

ShellSet v1.1.0 was initially released with a Makefile containing a direct path to installed Intel MKL libraries of a specific version which could be problematic for users with different versions. The Makefile has been updated in the GitHub repository to use *${MKLROOT}* to point to a default installation location which is defined at installation time by the installer. User's with non-default installation locations should update their Makefile accordingly by altering the **INCL** macro paths to point to their installation location.

3 Basic Program Setup

This section explains the input files required for ShellSet. Files required for ShellSet's constituent parts (OrbData, Shells, OrbScore) are well explained in the relevant sections found here[8]. The three files outlined within this section can all be edited using the GUI provided with the package, see section 6.

[5] https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html#gs.8a3psb
[6] https://github.com/JonBMay/ShellSet
[7] https://www.intel.com/content/www/us/en/develop/documentation/fortran-compiler-oneapi-dev-guide-and-reference/top/compiler-reference/compiler-options/alphabetical-option-list.html
[8] http://peterbird.name/guide/home.htm

3.1 ShellSet input

The following subsections explain the input files needed for each model selection option of ShellSet. Although Fortran is not case sensitive in its internal variable names, these input files are. Care must be taken in insert the file name using the correct case, for variable name case see column 1 of table 3. All input files must be saved in the directory **INPUT**.

3.1.1 List input option

The simplest of the two model selection options is the direct list input choice. This allows the user to input direct values for defined parameters yielding a specific number of simulations. The format for the file "ListInput.in" is shown in listing 1, with an example in listing 2.

```
Number Variables, Number Models
Var 1 Name
Var 2 Name
...
Var N Name
---------- Var1:
Var 1 value 1
Var 1 value 2
...
Var 1 value N
---------- Var2:
Var 2 value 1
Var 2 value 2
...
Var 2 value N
---------- Var3:
...
```

Listing 1: Format for the input file "ListInput.in"

```
2,3
fFric
tauMax
---------- Var1:
0.9
0.86
---------- Var2:
1.000E+12
2.000E+12
```

Listing 2: Example "ListInput.in"

This example generates two models using the two variables, model 1 using values 0.9, & 1.000E+12 and model 2 using 0.86 & 2.000E+12.

3.1.2 Grid search option

If using the grid search option the user should update the file "GridInput.in" with the required information. Listing 3 shows the layout of the file, listing 4 shows an example file.

```
Number Variables, Number Cells kept, Number Search Levels (incl. initial)
---------- Var 1:
Var 1 Name
Var 1 Min Value
Var 1 Max Value
Var 1 Number of models
---------- Var 2:
Var 2 Name
Var 2 Min Value
Var 2 Max Value
Var 2 Number of models
---------- Var 3:
...
```

Listing 3: Format for the input file "GridInput.in"

```
2,2,2
---------- Var 1:
fFric
0.02
0.82
2
---------- Var 2:
tauMax
1.0E12
4.0E12
2
```

Listing 4: Example "GridInput.in"

This example generates 4 models for 2 variables within the defined parameter search area. For the second level the program keeps the best 2 cells from the first, in each of the kept cells another 4 models are generated, for a total of 8 models in level 2 and 12 in total. Further levels would continue in the same fashion. The cells to keep are decided by ranking each model by a misfit score, each model being assumed to represent the entire cell.

ShellSet treats the minimum and maximum values for each variable as "strict" limits that will never be included in the search. Because of this, and so as not to focus the search away from the boundaries, ShellSet shifts the variable values selected towards these set limits. The following is an example using the fFric values given in listing 4. First, ShellSet calculates the distance between any neighbouring models:

$$StepSize = \frac{MaxVal - MinVal}{NumMods} = \frac{0.82 - 0.02}{2} = 0.4$$

then finds the first model using a half step size:

$$\text{model1} = 0.02 + \frac{0.4}{2} = 0.22$$

and any remaining models using the step size:

$$\text{model}\mathbf{N} = \text{model}\mathbf{N\text{-}1} + StepSize$$

In the example shown in listing 4 there are only two models, with fFric values of 0.22 and 0.62.

The method employed by ShellSet to create a grid of models would result in repeated models in certain circumstances. ShellSet automatically checks for, and prevents, repeated models from being run. Each repeat of a model is still reported in all output files, however with reduced information. The supplied grid search examples have skipped repeated models.

3.1.3 Input file for OrbData, Shells, OrbScore

The input files required for the original program units are not altered, see here[9] for information about them. ShellSet reads the names of these files from the input file "InputFiles.in". An example of this file is shown in listing

[9] http://peterbird.name/guide/home.htm

5. The user should take care to enter the names correctly as missing necessary files will halt the program but missing optional files may not.

One important note is that the first line of the parameter input file will be repeated inside each of the Shells output files (named beginning with fEarth, vEarth & qEarth) on the third line, limited to 100 characters in length. It is advisable to put a small description of the test run as you can see in the parameter input file provided within the ShellSet package (iEarth5-049.in). The first and second lines of the output files are again copied from the first line of the finite element grid and boundary conditions files respectively.

The first entries into the file define the parameter input file, finite element grid and (optional) laterally varying lithospheric rheology information (see section 3.2.4). These files are used by all of the program units, with the parameter input file & finite element grid being necessary. In all sections an **X** denotes an optional file which has not been used.

```
All:
iEarth5-049.in             Parameter input file
Earth5R.feg                Grid input file
X                          non-default Lithospheric Rheologies
-------------------------------------------------------------------------------------------

OrbData:
ETOPO20.grd      elevation/topography/bathymetry
age_1p5.grd      age of seafloor
CRUST2.grd       thickness of crust
delta_ts.grd     S-wave travel-time anomaly in upper mantle
-------------------------------------------------------------------------------------------

Shells (Main work loop):
Earth5R-type4AplusA.bcs  Boundary condition file
PB2002_plates.dig        Outlines of Plates
PB2002_boundaries.dig    Plate-Pair boundaries
------------------------------
X  Approx Vel solution
X  Mantle Flow -> HOC79ii.dig (iConve=1), Baum887.dig (iConve=2), PB2002_plates.dig (iConve=3 or 4)
X  Torque & Force balance -> Only if iConve=6
-------------------------------------------------------------------------------------------

Shells (Final OR Non-Iterating run):
Earth5R-type4A.bcs       Boundary condition file
PB2002_plates.dig        Outlines of Plates
PB2002_boundaries.dig    Plate-Pair boundaries
------------------------------
X  Approx Vel solution
X  Mantle Flow -> HOC79ii.dig (iConve=1), Baum887.dig (iConve=2), PB2002_plates.dig (iConve=3 or 4)
X  Torque & Force balance -> Only if iConve=6
-------------------------------------------------------------------------------------------

OrbScore:
GPS2006_selected_subset.gps                  (.gps | Geodetic Velocity)
robust_interpolated_stress_for_OrbScore2.dat  (.dat | Stress Direction)
aggregated_offset_rates.dig                  (.dig | Fault Slip Rate)
magnetic_PB2002.dat                          (.dat | Seafloor Spreading Rates)
GCMT_shallow_m5p7_1977-2017.eqc              (.eqc | Smoothed Seismic Correlation)
Fouch_2004_SKS_splitting-selected.dat        (.dat | Seismic Anisotropy | Only if SC)
-------------------------------------------------------------------------------------------
```

Listing 5: Simplified example of InputFiles.in

OrbData this section requires 4 additional input files containing "general" information about the model domain. If OrbData is required by ShellSet it uses information within these files to update nodal data within the finite element grid input file. OrbData does not alter the topology or node-locations in the model grid, or alter the model domain, in any way; it only makes alterations to the nodal values of lithospheric-structure parameters such as elevation, heat-flow, layer-thicknesses, and density and geotherm parameters.

Since OrbData is only run if certain parameters are altered these files are considered optional by ShellSet, that is: if OrbData is run the files are necessary, if not then this section of "InputFiles.in" is skipped. See table 3 for the list of which variables would cause OrbData to be run.

Shells (both parts) the first of the 2 sections is read when Shells is run within the "main loop" (figure 1) while the Shells Final section (second of the 2 sections) is read if ShellSet is performing tests which do not iterate Shells or when the "main loop" has successfully finished. The Shells & Shells Final sections of the file have the same layout and therefore the following description fits both.

The short, dashed line denotes the boundary between required files and optional files. The first 3 files (above the dashed line) are required by Shells and contain important information about the internals of the domain.

Below the short, dashed lines are the optional files, these are only required for certain simulation types, mostly depending on the value of the parameter **iConve**, which is defined inside the parameter input file. If an optional filename is given then the program will check for its existence, if the file does not exist then the program continues as if a name were not given while noting that an error has occurred. A value of **X** for the optional file will prevent the program from searching for or attempting to use that file.

When ShellSet is iterating Shells, it is possible to change the files between the two sections. Listing 5 shows an example where the boundary conditions are altered between the iterated Shells ("main loop") and the final Shells run.

The value of **iConve**, set within the parameter input file, alters some basic behaviour of the Shells part of ShellSet. **iConve** should take a value from 0-6 inclusive, table 2 shows the link between the value and expected input file. **iConve** values 0 & 6 will iterate Shells (see figure 1): 0 does not expect an input file and assumes no lower mantle flow at the first Shells run; 6 expects an existing torque file (generated by Shells) for the first Shells run entered in the "Shells" section of 5. Both 0 & 6 will then iterate Shells using the generated torque output files within the main loop of figure 1. All other **iConve** values do not iterate and skip directly to the "Shells Final" run (figure 1), each expects an input file entered into the "Shells Final" section of 5.

Value	Example expected file	Iterating?
0	None	✓
1	HOC79ii.dig	
2	Baum887.dig	
3	PB2002_plates.dig	
4	PB2002_plates.dig	
5	[legacy code; no published example]	
6	Existing torque file (existing Shells qEarth output)	✓

Table 2: iConve value options with example input files and behaviour

OrbScore files listed for OrbScore are all optional. Each of the files is used to generate one of the 5 misfits, or 1 score, generated for each model by OrbScore (section 3.3). ShellSet will automatically generate results for every misfit and/or score type where a file is entered (and exists), which will be stored inside the output file. In listing 5 each of the 5 misfits and the 1 score are generated due to the presence of every file. The Seismic Anisotropy (SA) also uses the "Outlines of plates" file from the "Shells Final" section. More information on the misfit options can be found in section 3.3.

3.2 Program personalisation & Variable mapping

3.2.1 Variable mapping

Table 3 shows a list of user input variables in ShellSet format and their Fortran-source-code equivalents in Shells. The first column lists the variable names that the user will write into the input files (ListInput.in & GridInput.in) to tell the program which variables are being updated, the second column shows how the same variable appears in the Fortran source code for Shells. A mixture of these variable names must be used when creating a new variable check subroutine, as can be seen in section 3.2.2, depending on whether the subroutine is checking for altered input variables or fixed input variables, care must be taken to reference the correct version of the variable name.

3.2.2 Variable Check routines

ShellSet is designed in such that a working thread will call a subroutine to check variable values against specified rules. Currently the file "MOD_VarCheck.f90" contains a single globally valid example subroutine "EarthChk",

User input text	Fortran source code	OrbData?
fFric	fFric	
cFric	cFric	
Biot	Biot	
Byerly	Byerly	
aCreep_C	aCreep(1)	
aCreep_M	aCreep(2)	
bCreep_C	bCreep(1)	
bCreep_M	bCreep(2)	
cCreep_C	cCreep(1)	
cCreep_M	cCreep(2)	
dCreep_C	dCreep(1)	
dCreep_M	dCreep(2)	
eCreep	eCreep	
tAdiab	tAdiab	✓
gradie	gradie	✓
zBAsth	zBAsth	✓
trHMax	trHMax	✓
tauMax	tauMax(1) & tauMax(2)	
tauMax_S	tauMax(1)	
tauMax_L	tauMax(2)	
rhoH2O	rhoH2O	✓
rhoBar_C	rhoBar(1)	✓
rhoBar_M	rhoBar(2)	✓
rhoAst	rhoAst	✓
gMean	gMean	
oneKm	oneKm	
radius	radius	
alphaT_C	alphaT(1)	✓
alphaT_M	alphaT(2)	✓
conduc_C	conduc(1)	✓
conduc_M	conduc(2)	✓
radio_C	radio(1)	✓
radio_M	radio(2)	✓
tSurf	tSurf	✓
temLim_C	temLim(1)	✓
temLim_M	temLim(2)	✓

Table 3: This table shows the mapping of variable names between how the user should type the names in input files for ShellSet (column 1) and how the variables appear within Fortran source code for Shells (column 2). This is to assist in any addition to the variable check subroutine (section 3.2.2) as well as any further updates. The third column indicates whether an alteration to this parameter's value (for example in grid search) would require OrbData to update the finite element grid input file.

which checks the values for two pairs of variables. To avoid altering main code regions users should add any localised variable check subroutines to the same module, then add a call from "EarthChk" to any new subroutine(s). Users less proficient in Fortran can use "EarthChk" as a template for new subroutines.

An example section of "EarthChk" is shown in listing 6. This is the first half of a check for a linked pair of variables, namely rhoBar_C and rhoBar_M. The global rule is: **rhoBar_C < rhoBar_M**.

```
if(trim(VarNames(i)) == 'rhoBar_C') then
  tested = .False.
  do j = i+1,size(VarNames)
   if(trim(VarNames(j)) == 'rhoBar_M') then
     tested = .True.
     if(VarValues(j) <= VarValues(i)) then
       Run = .False.
     end if
   end if
  end do

  if(.not. tested) then
    if(rhoBar(2) <= VarValues(i)) then
     Run = .False.
   end if
  end if
...
```

Listing 6: Simplified example of "EarthChk" subroutine, found in MOD_VarChk.f90

The subroutine first checks the user input variable names for rhoBar_C, if it finds this then it will check for rhoBar_M. If it finds both then it marks them as *tested* and checks their values against the rule. If it finds only rhoBar_C then *tested* will be *False* and the subroutine will use the global memory value (which is read from the parameter input file) for rhoBar_M, which from table 3 is rhoBar(2). The second half (not shown) of this section of the check performs the reverse checks, starting from first finding rhoBar_M. Omitted from this simplified section is the error message that is printed to an error file. This error is always non-fatal to the whole model set since different models are unique but models which fail this check are skipped and given an unrealistically large misfit score.

3.2.3 Updating variables

ShellSet offers the possibility of updating variables between the first and second invocations of Shells. The variables which can be updated in this way are those listed in table 3. As an example, when using option iConve=6, it is common to start a model with a trHMax=0.0 for the initial Shells run before updating it. Once updated, the value is fixed for each further Shells call within that model. Starting a new model resets all variables to the values given within the parameter input file or the updated values which define the model.

ShellSet is programmed to make this change if and only if a suitable "UpVar.in" file exists within **INPUT**. The name for this file is hard-coded into the program so this option is switched on (or off) by the existence (or not) of this file. Listing 7 shows the layout of the file, it is simply a list of variable names (from column 1 of table 3) and the new values to be used from the second Shells invocation.

```
Var 1  Val 1
Var 2  Val 2
...
Var N  Val N
```

Listing 7: Example of UpVar.in

3.2.4 Laterally Varying Lithospheric Rheology (Requires OrbWin)

Since version v5.0 of Shells (which is used in ShellSet) it has been possible to design a lithosphere with lateral variations in its rheologic (strength) parameters (in addition to lateral variations in elevation, heat-flow, and layer-thicknesses that have always been supported). This is achieved by labelling the elements within the finite element grid file and creating an additional file which contains the local values to be used for the strength parameters. The term used for a set of strength parameters is "Lithospheric Rheology" and its abbreviation is "LR" in some contexts.

The parameters which can be altered are the 4 which describe frictional rheology at low temperature, including an option for lower effective friction in fault elements (fFric, cFric, Biot, and Byerly) and the 9 describing the dislocation-creep flow-laws of the crust and mantle-lithosphere layers of the lithosphere (aCreep_C, aCreep_M, bCreep_C, bCreep_M, cCreep_C, cCreep_M, dCreep_C, dCreep_M, and eCreep).

A Lithospheric Rheology index number (LR#) is a non-negative integer that is a key to a set of different rheologic constants, identifying a different "zone" with different geologic-column of rock-types in that part of the lithosphere. For example, all seafloor could be described by LR0, and all continents by LR1. LR# labels may be applied (and usually should be applied) to both continuum elements and fault elements. For an element to have the default rheology (=LR0) just omit any LR# specification.

It is important to note that only the parameters making up the default rheology (LR0) are varied by the ShellSet search processes; all rheologies described by LR1, LR2, ... , LRn are held fixed.

Any number of these zones may be added to the model, and multiple disconnected zones (for example distant fault zones) may use the same LR# and therefore parameter values.

To use this option in ShellSet the user must create a finite element grid file with suitable labels linking the necessary elements to a particular LR# label. This can be done manually however is more simply achieved using the interactive graphical interface of OrbWin, described later in this section.

An input file is also required which defines the parameter values for each of the LR# labels present in the finite element grid. An example input file containing a single zone definition is shown in listing 8. This file is then fed to ShellSet on the third input line of "InputFiles.in" (non-default Lithospheric Rheologies), see listing 5.

LR#	fFric	cFric	Biot	Byerly	aCreep_C	aCreep_M	bCreep_C	bCreep_M	cCreep_C	cCreep_M	dCreep_C	dCreep_M	eCreep
1	0.1025	0.85	1.00	0.00	1.85E+06	5.30E+04	10524.	16494.	0.00793	0.01037	5.00E+08	5.00E+08	0.266667

Listing 8: Example input file defining new "zone" parameters to be used with an appropriate finite element grid file. The file contains 2 lines, the first with variable names while the second defines the values for each variable in that numbered zone. ShellSet ignores the first header line of variable names; however, the parameter order is strict and should be followed exactly.

OrbWin Although not included in ShellSet it is necessary to note the importance of OrbWin. The creation of a finite element grid file for ShellSet (and OrbData, Shells & OrbScore) is done using OrbWin. A Windows executable of OrbWin can be found, along with a user manual and some example files, here[10].

Note that all new finite element grids generated by OrbWin will need to use OrbData as some initial nodal information required in simulations are initially undefined. For this purpose OrbData is available as a standalone download here[11].

3.3 Selecting misfits and/or score

OrbScore allows the calculation of up to 5 misfits (in which a high value is bad) and 1 score (where high is good): Geodetic Velocity (GV); Seafloor Spreading Rates (SSR); Stress Direction (SD); Fault Slip Rate (FSR); Smoothed Seismicity Correlation (SC); and Seismic Anisotropy (SA). ShellSet is setup to calculate and report all misfits and score for which the necessary files exist and are known to ShellSet. The following table details the names of files for each misfit or score type which are provided with ShellSet. A parameter input file and a Finite Element Grid file (produced by OrbWin) are always required, examples of both are also supplied.

These files are known to ShellSet through the "InputFiles.in" input file, an example section of which is shown in listing 5. The user should complete the final section of this file, beneath the line "OrbScore" for the majority of the files and the "Outlines of Plates" line within the "Shells (Final OR Non-Iterating run)" portion of the file. (For logical consistency this filename cannot be changed between the Shells and OrbScore routines, so there is no input option within the OrbScore section.)

The computation of the seismic anisotropy misfit overlaps with the computation of the smoothed seismic correlation score. Therefore, it is not possible to use different files as input for this pair of quality measures. Smoothed seismic correlation is the only one of these 6 measures which is a score (in which a greater value is better); all 5 other measures are misfits (in which a lower value is preferred).

[10] http://peterbird.name/oldFTP/OrbWin/
[11] http://peterbird.name/oldFTP/neotec/SHELLS/OrbData/OrbData5/
[12] http://peterbird.name/guide/Step_22.htm

Misfit/Score option	Code	Name in INPUT directory
Geodetic Velocity (misfit)	GV	GPS2006_selected_subset.gps
Stress Direction (misfit)	SD	robust_interpolated_stress_for_OrbScore2.dat
Seafloor Spreading Rate (misfit)	SSR	magnetic_PB2002.dat
Fault Slip Rate (misfit)	FSR	slip_rate_format.txt
Smoothed Seismic Correlation (score)	SC	GCMT_shallow_m5p7_1977-2017.eqc
Seismic Anisotropy (misfit)	SA	GCMT_shallow_m5p7_1977-2017.eqc Fouch_2004_SKS_splitting-selected.dat PB2002_plates.dig (also used by Shells)

Table 4: Files required for each OrbScore2 misfit/scoring options. For a complete list of all IO file options and descriptions see the scoring section of the guide to neotectonic dynamic modelling here[12].

3.3.1 Geometric mean

ShellSet allows the option of calculating a geometric mean misfit for a combination of the existing misfit measures; see table 4. The selection of the smoothed seismic correlation (SC) is not allowed within the geometric mean as this is a score and not a misfit. The geometric mean works by multiplying the values for each selected misfit and then taking the n^{th} root, see equation 1.

$$\sqrt[n]{x_1 \times x_2 \times x_3 \times ... \times x_n} \tag{1}$$

The geometric mean may be used to select best cells inside the grid search model selection or may be calculated for informational purposes only and will be output in the standard output files. By default, ShellSet calculates the geometric mean using all run misfits for every model and reports the value in the final position of the output file.

The reason that use of the geometric mean measure (for ranking trial models within a test) is suggested is that the ordering/ranking of models by the geometric-mean measure is independent of the physical units used within each misfit measure (e.g., degrees or radians; m/s or mm/a). A simple arithmetic average of misfits would not have this desirable property.

4 Launch & Run

4.1 Terminal command

ShellSet must be launched using two or more MPI threads. If more MPI threads are requested than ever needed the program will automatically shut down with a warning message so as not to waste compute resources. The maximum number of threads needed is simply calculated: for the list model option it is the number of models+1; for the grid model selection it is the number of models in the initial level+1.

An example invocation line for the list option is shown here:

mpirun -np 5 ./ShellSet.exe -Iter 3 -InOpt List

This begins a test using five mpi threads, therefore a minimum of four models must be run, and four models will run in parallel. The models are taken from the list model selection option and each will perform three iterations of the "main loop" (see figure 1).

This is an example invocation line for a grid search option:

mpirun -np 5 ./ShellSet.exe -Iter 3 -InOpt Grid -MR GV

The grid search also needs a misfit score to use in selecting the best cells at each level, in this example the geodetic velocity is selected (**-MR GV**), for a complete explanation of these and other command line argument options see section 4.2 and table 5.

4.2 Command Line Arguments

Each CLA is both case and type-sensitive. For example -MC expects its first input to be upper case, for the second and third entering a real or integer in place of the other will cause an error.

Many of the options are quite trivial and explained well in table 5, however some require extra explanation:

-AEF tells ShellSet to stop if a model error is detected. This is not recommended but can aid in debugging of a test. Defined non-fatal errors, such as those generated inside the "EarthChk" subroutine, are never fatal.

-MC requests that ShellSet run misfit convergence. To perform misfit convergence ShellSet generates a misfit score for each iteration of the "main loop" from the iteration previous to the defined number of iterations at which the check is performed. The next misfit score is compared to the proceeding misfit score by way of comparison using the supplied multiplied range. If the next model misfit score is within an acceptable range of the previous model

CLA	Type	Role	Options	Grid	List
-Iter	Integer	Maximum iterations of "Main Loop" (see figure 1), only required for specific values of iConve (see table 2)	User choice	**C**	**C**
-InOpt	String	Choose between list and grid options	"List" or "Grid"	**R**	**R**
-Dir	String	Working directory (created), defaults to "test" if not entered, will overwrite existing directory	User choice	**O**	**O**
-AEF		Make all errors fatal to the program	Present means True	**O**	**O**
-V		Produce extra information in a new output file (see section 5.3)	Present means True	**O**	**O**
-MC	String Integer Real	Perform misfit convergence: 1) See -MR 2) Iterations of "Main Loop" before check performed (see figure 1) 3) Multiplied range that defines a "converged" pair of models	1) GM or any of 6 options in -MR 2) User choice 3) Range [0,1] e.g: GV,3,0.1	**O**	**O**
-MR	String	Misfit type used to select best models (Only used by grid search)	User choice: GV, SD, SSR, FSR, SC, SA See table 4	**R**	**X**
-ML	Real	Create special file to store models with misfit scores better than user defined value (see section 5.2)	User choice	**O**	**X**
-KL	Real	Misfit score at which Grid Search program will stop	User choice	**O**	**X**
-GM	Integer & String(s)	Combine multiple misfit score options into a geometric mean to use in selecting best models (replaces -MR, required if GM used in -MC)	User choice: e.g: 3,GV,SD,FSR See table 4 (excl. SC)	**O**	**X**

Table 5: Command Line Arguments (CLAs), their role and user options. All CLAs are case sensitive. All real valued inputs are formatted and read to 4 decimal places (xxx.yyyy). **R** - required, **O** - optional, **C** - conditional, **X** - not used.

misfit score, that is if $M_N \in [M_{N-1} * (1 - mr), M_{N-1} * (1 + mr)]$, where mr is the multiplied range and M_N is the misfit for model N, then the two models are considered converged and ShellSet will exit the "main loop" at this step.

-KL supplies a misfit score to ShellSet that, when reached, will stop the program in a controlled way. Should this option be selected, and a misfit score reach the value, ShellSet will print information inside the "Models.txt" and, if run, the "Models_Lim.txt" files.

-GM tells ShellSet to use the geometric mean to select the best cells in the grid search and informs ShellSet of the number and selected misfits to use in its calculation. It also serves to define the geometric mean if the **-MC** option is selected.

There are also several CLAs which do not begin the main program of ShellSet but instead print some information, these are referred to as non-running CLAs. Table 6 shows the different options available.

CLA	Role
-help	Print ShellSet help
-info	Print general ShellSet information

Table 6: Non-running Command Line Arguments (CLAs) and their role. All CLAs are case sensitive.

4.3 ShellSet error files

There are three different types of error that ShellSet can produce: (1) Fatal, always fatal to the entire test set as by definition it is an error that would cause a failure for every model - for example a missing required file; (2) Model-specific, errors that would cause only that model to fail - for example failing a variable check; and (3) non-fatal errors, errors that are not fatal to a model but require reporting - for example not finding an optional file.

Each error encountered by a thread produces a personal file containing a message related to the cause of the error. The files are named using the thread ID number of the thread creating the file to aid locating a specific error. The files are located inside the "Error" directory which is automatically created upon detection of any type of error. The detection of a Fatal error will always halt the program, Model-specific errors will not unless the CLA **-AEF** (see section 4.2) is used, and non-fatal errors will never halt ShellSet. The presence of the **-AEF** CLA will cause the deletion of the empty "ModelError.txt" file.

4.4 Worker Thread

Figure 1 shows a flowchart which outlines the process that a working MPI thread goes through. Diamonds represent decisions, rectangles results and parallelograms IO with the boss thread. Arrows represent the program flow, black for general flow, green and red for true or false responses respectively. As an example, the first few steps will be outlined.

1. **Receive Input** - receive several inputs, including the variable values and a logical parameter stating whether to run a model or not. *False* moves the thread to **Exit**, *True* moves the thread to the next step

2. **Input Check** Depending on the variables being altered the thread will run a check of the values using supplied subroutines in MOD_VarCheck.f90 (see section 3.2.2). Failing this moves the thread to **Large Misfit**, passing this moves to the next step.

3. **Run OrbData?** Here the thread checks whether or not to run the program unit OrbData. This is a simple check on which variables are being updated, certain variables require an update to the Finite Element Grid (FEG) file while others do not require any update. If OrbData is not needed then the thread moves to **SHELLS**, otherwise it moves to **OrbData**, after which it moves to **SHELLS** and into the "Main Loop"

The main loop is iterated up to the number of iterations supplied as a CLA (using the option **-Iter**, see section 4.2), after which it will run Shells again with optionally altered input files (see section 3.1.3). There are two other ways for the thread to leave the loop: one of the Shells runs fails to converge during its solution step, which would return a large misfit to the boss thread; the misfit convergence option is run which could allow exiting the loop early. The program continues normally even if the convergence criteria are not met within the iterations defined by the user. The final result stored in the output files is always the final run of OrbScore performed on the results from the final run of Shells (outside of the main loop), regardless of any misfit convergence results.

For more information on the individual program units (OrbData, Shells & OrbScore) please see the information here[13].

5 Outputs

For every test ShellSet will produce one output file, "Models.txt", and can optionally produce two other output files "Models_Lim.txt" and "Verbose**X**.txt" where **X** represents the MPI thread number.

5.1 Models.txt

This output file is created automatically in the working directory and contains all of the models run within the test, in the order in which they finish. The layout of the file alters slightly depending on the method for model selection. In both cases the first line is used to display the invocation line while the second line shows the order in which the output information is written.

If the list model selector is chosen then the second line will contain the following information: global model number; MPI thread which ran the model; VarValues(**X**), where **X** denotes the number of variables tested; a list of all the misfit scores calculated in the order in which they are printed in the rest of the file. For example, a test of two variables running all misfit options:
global model, ThID, VarValues(2), GV, SSR, SD, FSR, SC, SA

If the grid search model selector is chosen then the second line will contain the following information: global model number; MPI thread which ran the model; grid search level of the model; the cell from the previous level that this model is contained within; VarValues(**X**), where **X** denotes the number of variables tested; a list of all the misfits scores calculated in the order in which they are printed in the rest of the file. For example, a test of two variables running all misfit options using geodetic velocities to pick the best models:
global model, ThID, Level, Cell, VarValues(2), GV, SSR, SD, FSR, SC, SA
It is important to note that the misfit score chosen to "drive" the grid search (GV here) is always displayed in the first position.

[13] http://peterbird.name

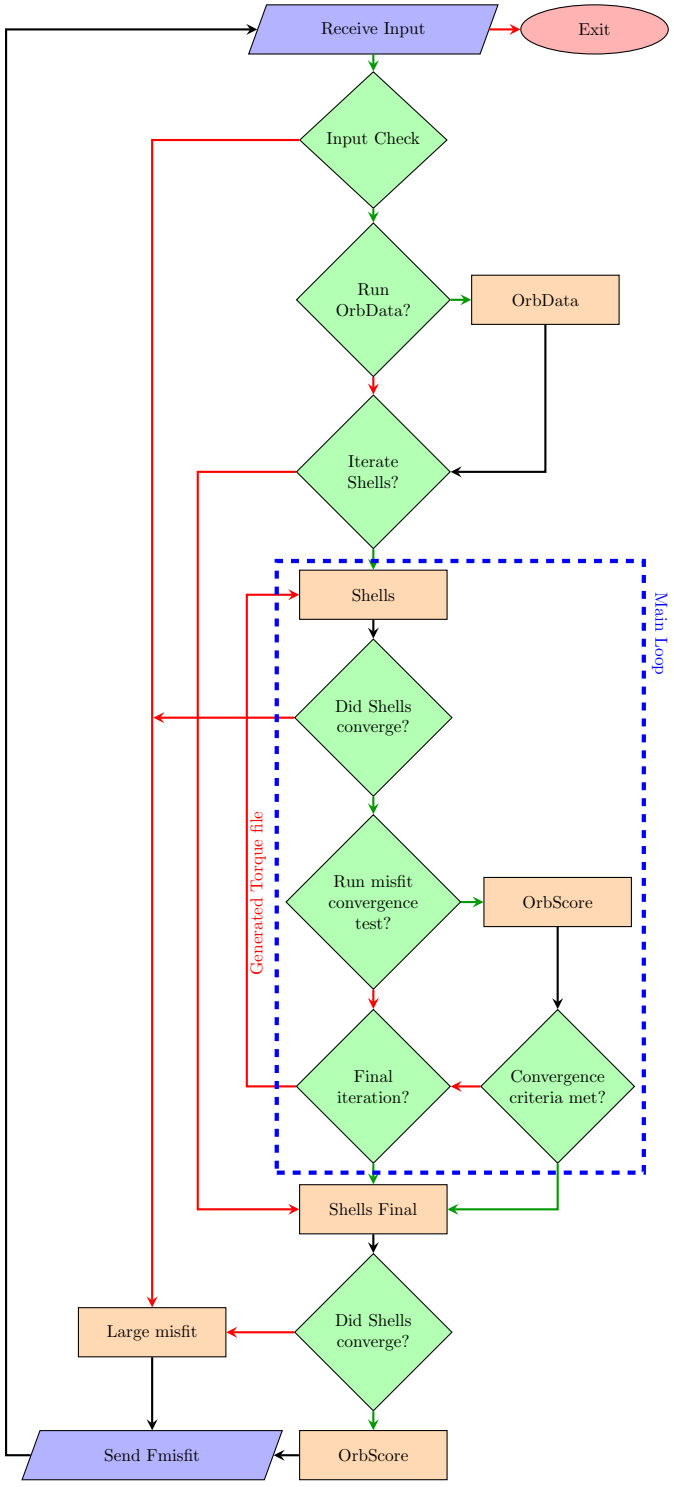


Figure 1: Worker Flow Chart. Yes or True responses in green, No or False responses in red, general flow in black, main work loop in blue dashed box

5.1.1 Scatter Plot

ShellSetScatter.py is a Python program that can create scatter plots in 1D, 2D & 3D from both the grid search and list input test results. The user must control the entries at the head of the file in the **Modify** section. The entries **TestDir**, **ModInFile** and **ResFile** must correspond to an existing directory and files.

Once these entries are correct the program will create labelled scatter plots using the information within the files. When plotting a grid search test, the program will create a new plot at each level and a final plot of all levels while the list input tests generate only a single plot. An example of the 3D scatter plot generated for a grid search test can be seen on the cover page.

All generated plots are automatically saved in their initial view into a new directory within the working directory supplied by the user. The user can alter the file type of the generated images by modifying the **OutType** variable. The supported formats are: eps, pdf, pgf, png, ps, raw, rgba, svg, and svgz.

The variable **ErrLvl** controls the matplotlib error level, the options are: "notset", "debug", "info", "warning", "error", and "critical". The initial setting at "error" prevents warning messages caused by file types which do not support transparency in their output.

Lines exist within each **Plot** code portion to control the range for each plotted dimension, these lines can be uncommented and used as needed.

5.2 Models_Lim.txt

The Models_Lim.txt file is an optional output for the grid search model selector. It's creation is assumed *False* unless switched on at run-time using the **-ML** CLA (see table 5 and section 4.2). If created, the file will contain a list of all models that obtained a misfit score for the driving misfit that is strictly better than its defined value. This file does not impact the creation of Models.txt.

5.3 Verbose.txt

The Verbose**X**.txt, where **X** represents the MPI thread number of the creating thread, is an optional output for either model selector. If switched on, using the **-V** CLA (see table 5 and section 4.2), it will contain all of the information that was previously printed to the terminal by the original three program units but which is now suppressed. This file is very useful in debugging or learning the specifics of the program units; however, seasoned users may prefer to leave this option off.

5.4 OrbData, Shells, OrbScore

The output files for OrbData, Shells and OrbScore have been renamed compared to the standalone versions. The filenames for Shells and OrbScore are generated by adding the global model number and main loop iteration number to the original file name and unit number, for example the Shells output torque file for the 3rd model number on its 2nd iteration would be saved as: qEarth_3_2.24. OrbData output files are never updated and so they are renamed using only the global model number, for example the same model would produce the OrbData output file: FEG_3.14.

ShellSet organises these files into program unit and thread specific directories within the working directory. For example, Shells output files generated by MPI thread 3 will be stored in the directory: ThID_3_Shells_output.

6 GUI

ShellSet comes with a simple GUI to aid the user in the selection and setup of the model input (grid search or list); creation or editing of input files; and program initialisation. The GUI is written in Python and can be launched within any Linux terminal. It is designed to simplify the setup of tests and experienced users may still update any files manually with any text editor if they prefer before launching the program using the command line.

Specifically, the GUI simplifies the information set out in section 3.1 and therefore the user should understand this section, including the "InputFiles.in" and the parameter input file. Use of the GUI also requires knowledge of the variable names, table 3, and the scoring options described in table 4 and section 3.3.1.

The main GUI window is shown in figure 2. At the top of the window users are asked to select the ShellSet version that they wish to run, the selected option should match the version created using the Makefile. The default is set to standard.

The second section of the main window is under the heading "Update/Create input files", within which there are four options. Each option is outlined in a section below.

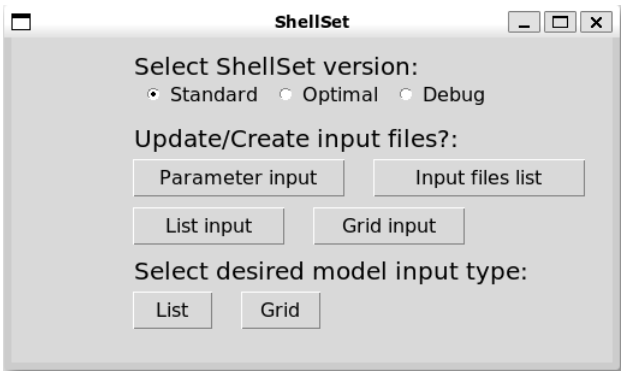


Figure 2: Initial window of ShellSet GUI

6.1 Update or Create input files

Each of the four buttons "Parameter input", "Input files list", "List input" and "Grid input", open respective setup windows. The top two, "Parameter input" & "Input files list", allow the creation or new, or adaption of existing, input files. "List input" and "Grid input" always create new input files.

6.1.1 Parameter input

This will allow the input of all standard input parameters, such as in the example file iEarth5-049.in. The window will be filled with example values or with values from an existing parameter input file which the user can define by updating the variable **ParamFile** in ShellSetGUI.py. The reason that 2 columns appear is that some parameters have distinct values for Crust and Mantle layers of the model; the left column is for Crust and the right column is for Mantle (also, some integer choices for parameter iConve require an additional real-number in the right column). Another important note is that the first 9 rows collectively describe the basic or default Lithospheric Rheology (LR0), in the case that lateral-variation of Lithospheric Rheology was specified in the finite-element grid (FEG) input file. It is this basic or default LR0 that can be automatically varied with the ShellSet test process. At the creation of a new parameter input file (clicking save) the old file is copied to a file with the prefix "OLD_", only one previous version is kept in this fashion.

6.1.2 Input files list

This will allow the specification of all necessary input files, such as in the example file InputFiles.in. The first window will ask the user to specify which (if any) of the files related to the six OrbScore scoring options are to be altered, this is to prevent an incorrect file being altered or an update forgotten. The next window will be filled with example filenames (those in the example InputFiles.in file) or with filenames from an existing input file which the user should define by updating the variable **InFile** in ShellSetGUI.py. The files are expected to be found in **INPUT**, a check exists within the GUI so non-existent (e.g., mis-typed, or mis-located) filenames should be caught. As with the parameter input file, when a new input files file is created the previous version is copied to a file with the prefix "OLD_".

6.1.3 List input & Grid input

The first window opened by these options asks for basic information related to the setup of the test in terms of models. After entering this the next window will ask for information related to each variable, one by one. Variables are case sensitive so entries should follow the naming rules given in table 3

6.2 Model input type

The third and final section is "Select desired model input type", here the user must decide whether they wish to use the grid search or a list of models as input. It is within one of these options that the user may start the program.

Clicking on either option will load a window asking which, if any, misfit criteria should be used in the misfit convergence test. Since misfit convergence is optional this selection is also optional. It is also possible to use the geometric mean and define the criteria to combine to calculate that, an explanation can be found by clicking the "?" box.

Clicking Enter moves to the final window (and pre-fills the misfit convergence inputs if selected) where the user is expected to enter information related to how they wish the program to run. Each option has been explained in section 4.2. clicking Start begins the program with the input setup.

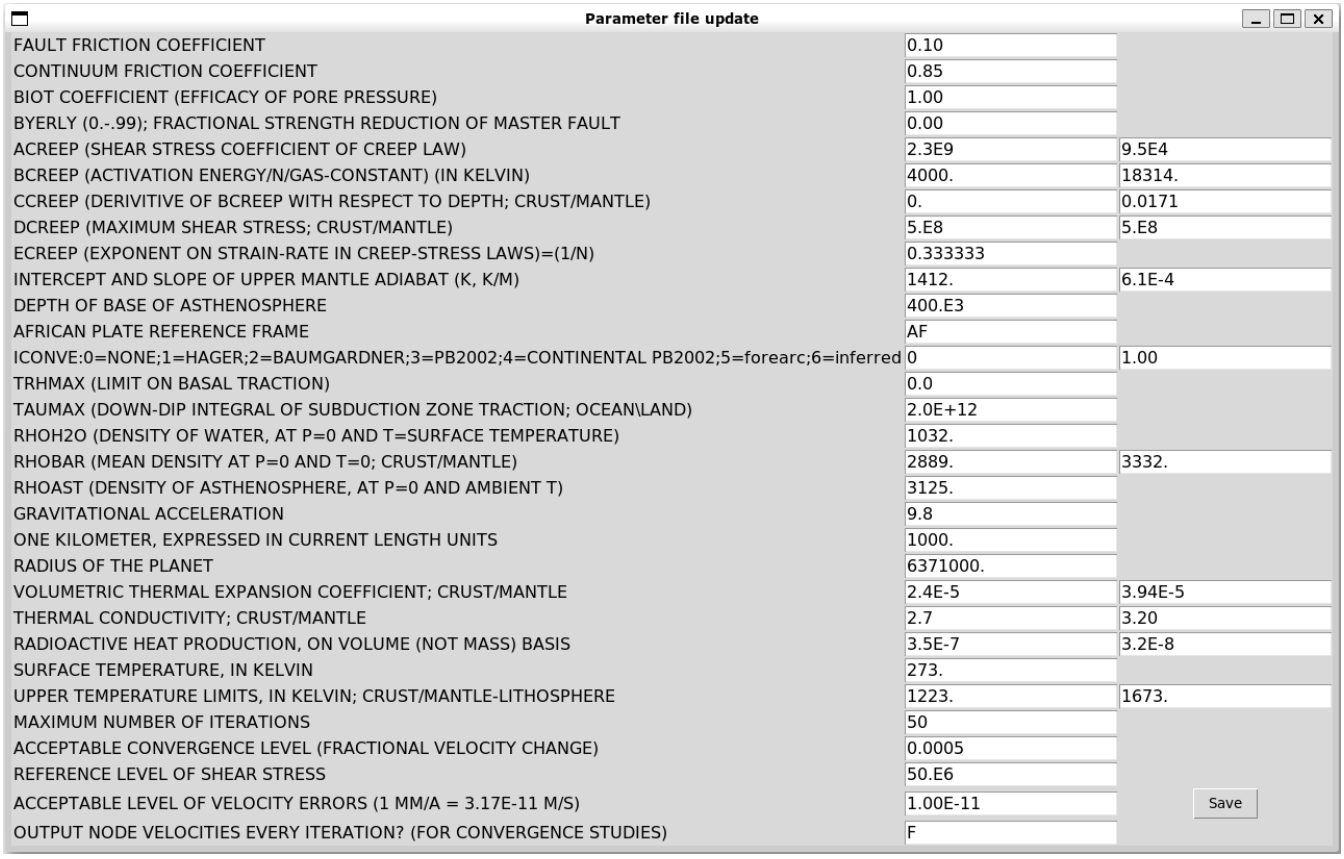


Figure 3: Parameter input file window of ShellSet GUI. This window is filled either with set default values or values from an existing parameter input file if one is provided.

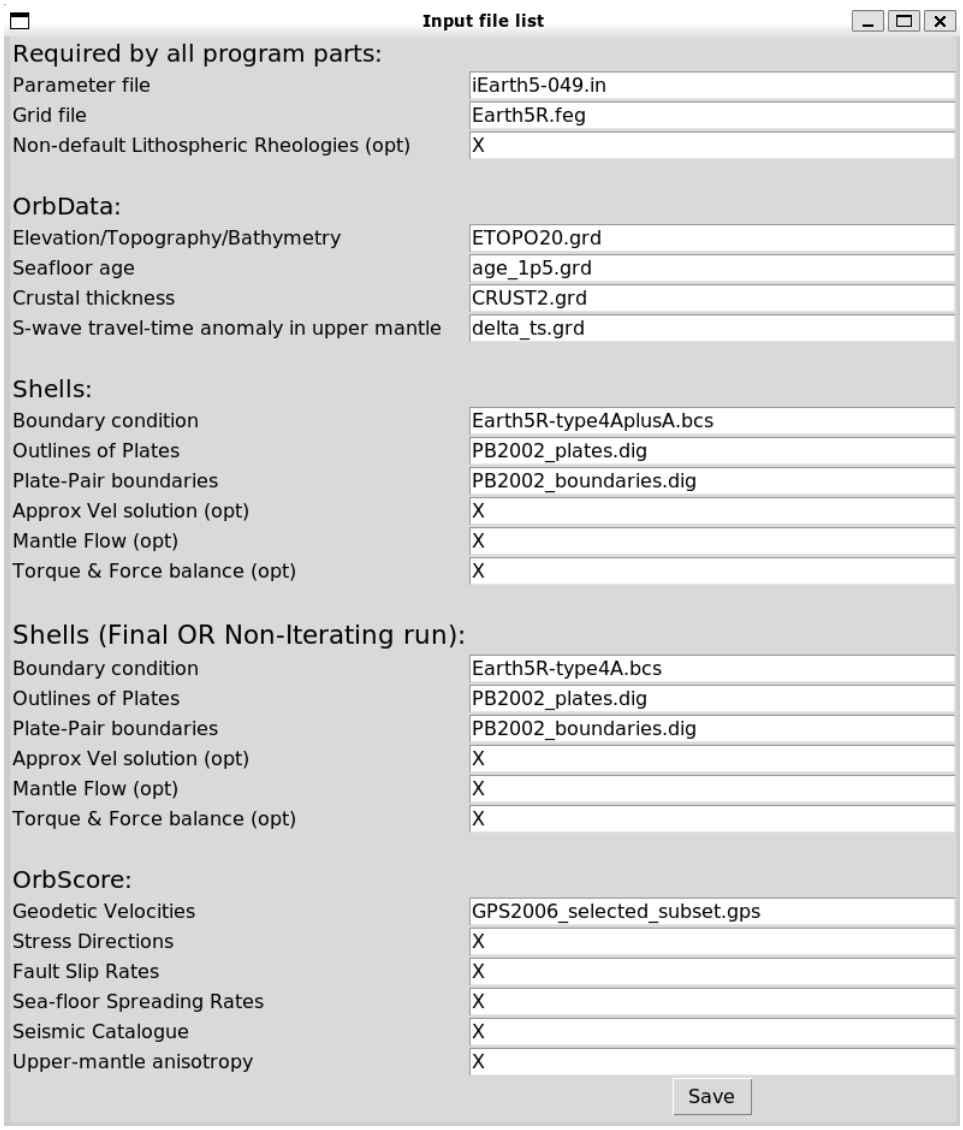
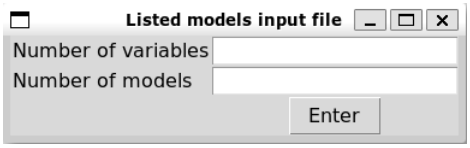
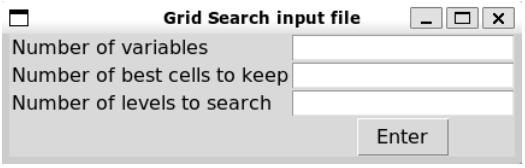


Figure 4: Input file list file creation window. The OrbScore section is completely visible due to the selected options at a previous window. **X** can be input for all ignored optional files to prevent minor error messages during ShellSet execution.

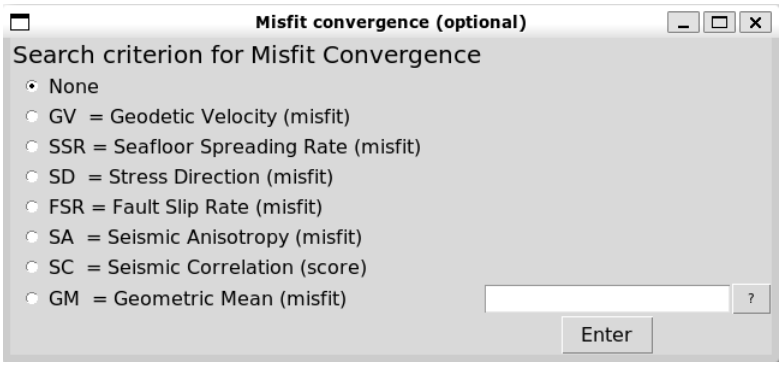


(a) First window for list input file creation.

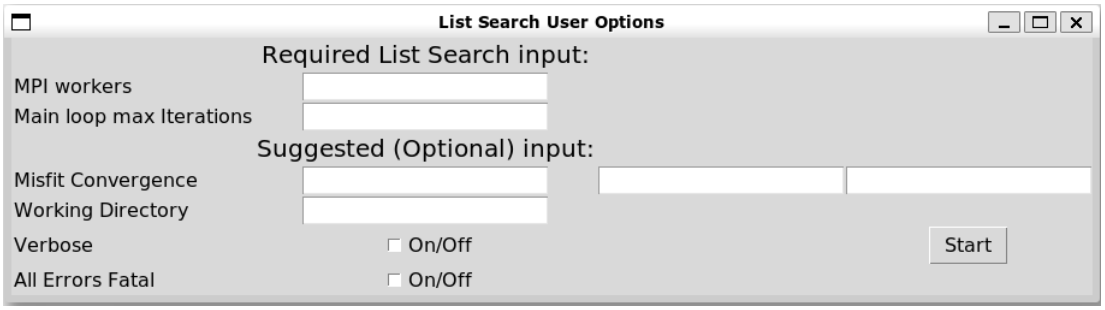


(b) First window for grid search file creation

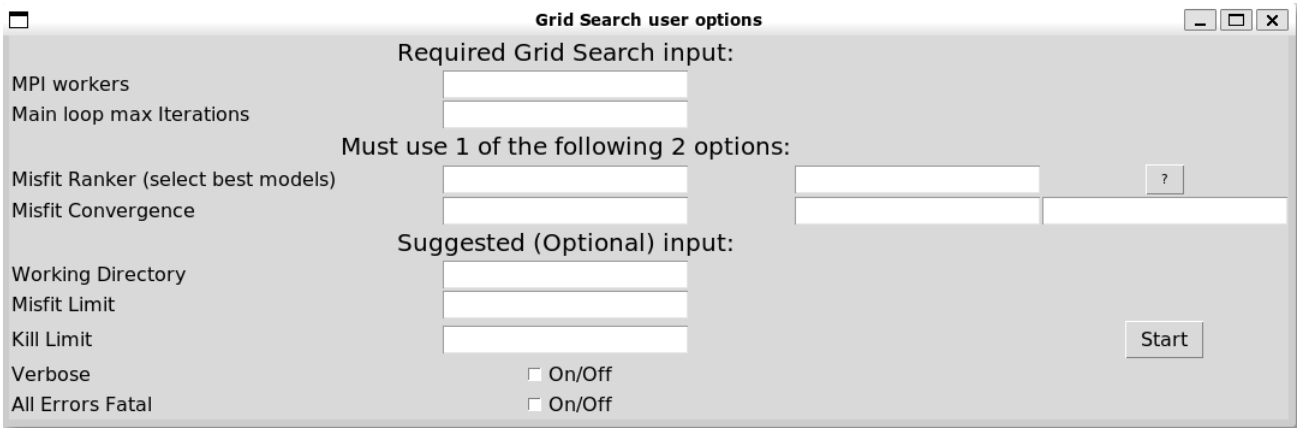
Figure 5: First windows for input file creation. After pressing enter a further window will open requiring data for each variable, this window will loop for the number of required variables.



(a) Misfit convergence window, this is an optional feature for tests which iterate Shells, non-iterating tests do not use this option.



(b) Setup window for list input option, this window finalises the ShellSet test setup and initialises ShellSet.



(c) Setup window for grid search option, this window finalises the ShellSet test setup and initialises ShellSet.

Figure 6: Selecting either the "List" or "Grid" option as the model type will load the window shown in figure 6a, then one of figures figure 6b or figure 6c. Through these windows the user will finalise the setup and initialise the ShellSet program.

7 Examples

ShellSet includes 3 examples - 2 grid search and 1 list input. To reduce the size of the package the examples do not include all output files from ShellSet, only the main results file (Models.txt), generated plots, and the input files required to reproduce the results are included.

To perform any example the user should simply copy the files: "InputFiles.in" & "GridInput.in" or "ListInput.in" from the example directory into **INPUT** of the main working directory. At initial download all other required input files should already exist inside **INPUT** however the user should control this, all input files are also given within the example directory. Once copied the test is started with the command:
mpirun -np X ./ShellSet.exe -Iter 4 -InOpt Grid -Dir GridEx1 -GM 4,SSR,GV,SD,SA for grid example 1,
mpirun -np X ./ShellSet.exe -Iter 4 -InOpt Grid -Dir GridEx2 -GM 5,SSR,GV,SD,SA,FSR for grid example 2, or
mpirun -np X ./ShellSet.exe -Iter 4 -InOpt List -Dir ListEx1 -GM 5,SSR,GV,SD,SA,FSR for the list example. In each case the letter **X** should be replaced with a suitable value depending on the computing system, noting that ShellSet needs at least 2 and the tests would allow at most 10.

Exact results will depend on the system and software (compiler version, etc), however general results should be similar to those found in the output files and plots included with each example.

7.1 List input

The list example contains the results of 9 models taken from the first level of the grid search examples. The results differ slightly compared to the grid search due to slight differences between the values for fFric, these differences result from the fact that ShellSet reports only the first 5 decimal places for this parameter but the models within the grid search are run with greater accuracy. The geometric mean values should be compared to the 2^{nd} grid search example as this test uses the same definition for the geometric mean.

7.2 Grid search

Each of the 2 examples demonstrates a grid search over 3 levels in a 2D parameter space. Through the 3 levels of the search ShellSet performed 5 3*3 grids to test 41 distinct models after accounting for 4 repeated models.

The first example is a direct copy of the work outlined in "Stresses that drive the plates from below: Definitions, computational path, model optimization, and error analysis" (Bird et al. (2008))[14] and is shown to demonstrate the speed and simplicity with which ShellSet can perform parameter testing. The second example includes an additional data set, which was not available to the previous authors, which has been refined from published data. Both examples found models which improve on the existing global model.

A Copyright

ShellSet is released with the GNU GPL v3.0 licence. You should find a copy of the licence file within the downloaded ShellSet package, and an outline at the head of each program file.

[14] https://doi.org/10.1029/2007JB005460