**Installing AxiSEM3D**

Some Notes

Ben Fernando

1. **Aim**

This document is intended to document my installation of AxiSEM3D (specifically on the Leeds ARC), and its use in undertaking synthetic studies of the earthquakes occurring at subduction zones.

This guide is not intended to be a repetition of either the AxiSEM3D papers or the GitHub manual, but provides extra detail specifically relevant to this project.

This guide is distributed with python scripts that you may wish to use to create and modify 3D models, and to analyse synthetic data generated using AxiSEM3D.

*Contributions to this manual were received from Will Eaton and Tim Craig.*

1. **Glossary**

Some terms that are used in this document, and relevant papers, might not be commonplace or otherwise clearly defined. Here they are, if useful:

2.5D: see ‘axisymmetric’.

Axisymmetric: here, a model or method which does not have any longitudinal variation; i.e. where any variation in parameters or physics is confined to within a single plane (‘2D slice’) of the planet. What this means is that any 2D slice through the Earth is identical to all the others. Note that in a 2.5D method, any structure must wrap all the way around the axis: e.g. a high-velocity circular province in the 2D plane will become a torus in the 2.5D version.

AxiSEM: The AxiSymmetric Spectral Element Method: the 2.5D version of the code we’re using here

AxiSEM3D: The AxiSymmetric Spectral Element Method 3D: the 3D version of the above

Geometric model: A 3D model which involves the radial deformation of some interface to change its depth/radius at different latitudes and longitudes. An example is the Moho in Crust 1.0.

Green’s Function: In this case, the response of the simulation domain to a near-impulsive (quasi-delta-function) source. Note that I say near-impulsive because in a computational method of this type the source must have some width (here, a few timesteps). In general this is so much less than the seismic period that it is close enough to a delta function for most purposes.

Jacobian: Computations on individual elements in AxiSEM3D are mapped back to a reference square element, which increases computational speed. As most elements are not square to begin with, the Jacobian (and its inverse) associated with each coordinate transform must be calculated.

Pseudospectral Method: A method of solving the wave equation which uses basis functions to represent key quantities in the equations of motion. See Igel’s Computational Seismology (2016) for more detail.

Spectral Element Method: a fast and efficient way of solving the wave equation, see Igel’s Computational Seismology (2016) for more detail.

Volumetric model: A 3D model which represents a change (with respect to the 1D case) in seismic properties (e.g. vp or vs), such that they vary with latitude and/or longitude and/or depth/radius.

1. **A brief introduction to AxiSEM3D**

AxiSEM3D is a seismological modelling code which allows users to undertake simulations of wave propagation in complex, 3D environments.

3.1 Literature

The following papers provide a reasonably comprehensive background on the method (at least the parts relevant here), though not the practicalities of running it:

* Nissen-Meyer et al, 2014: [AxiSEM: broadband 3-D seismic wavefields in axisymmetric media](https://doi.org/10.5194/se-5-425-2014) (an introduction to the ‘2.5D’ version of AxiSEM3D, known as AxiSEM)
* Leng et al, 2016: [Efficient global wave propagation adapted to 3-D structural complexity: a pseudospectral/spectral-element approach](https://doi.org/10.1093/gji/ggw363) (an introduction to the 3D version)
* Leng et al, 2019: [AxiSEM3D: broad-band seismic wavefields in 3-D global earth models with undulating discontinuities](https://doi.org/10.1093/gji/ggz092) (an extension to the 3D version from the 2016 paper including details of boundary undulations, etc)
* Fernando et al, 2020: [Oceanic high-frequency global seismic wave propagation with realistic bathymetry](https://doi.org/10.1093/gji/ggaa248) (the 3D version extended to include a treatment of oceans - similar computational constraints apply as to subduction zones)
* Haindl et al, 2021: [A 3D complexity-adaptive approach to explore sparsity in elastic wave propagation](https://doi.org/10.1190/geo2020-0490.1) (more on the local/regional scale methods which will probably be of use here)

3.2 Physical and mathematical background

The mathematical background to the AxiSEM3D method is rather complex, but a brief summary might prove useful.

In short, AxiSEM3D solves the 3D equations of motion (which describe the propagation of seismic waves) using a spectral element-pseudospectral method. What this means is that the in-plane parts of the solution are found through one method (spectral methods), whilst the azimuthal (i.e. longitudinal) solutions are found through another (pseudospectral methods).

This may seem like an unusual way to do things, given that the azimuthal solution follows exactly the same physics as the radial and meridional parts. However, what it does let us do is simplify the problem considerably, and hence save on computational cost.

The reason for this is that seismic properties in the Earth vary much more slowly laterally (i.e. in latitude and longitude) than they do radially (i.e in and out toward and from the core). This means that the wavefield is much smoother (less complex) laterally, and is therefore simpler to compute. Of course, the lateral gradients in density and sound speed are not unimportant - so we need to account for them somehow - but equally they are unlikely to cause as much of a challenge from a computational perspective.

In mathematics, it is often common to represent complex functions by series - for example, the Taylor Series of sin(x) can be truncated and used as an approximation. In 3D seismology, we can do a Fourier expansion of the wavefield in the azimuthal direction, and choose how many terms we have in our Fourier series depending on how complex the azimuthal wavefield is.

This is how AxiSEM3D works: it turns out a Fourier order of 0 is sufficient for a radially symmetric (‘1D’) seismic profile with an implosive/explosive source, 1 is sufficient for a dipole, and 2 for a quadropole (second rank moment tensor). If we want to have a non-1D model, such as Crust 1.0, we simply increase the Fourier order. How much to increase it by is a non-trivial question and depends on the simulation in question, but a few hundred to a few thousand is common in simulations I’ve run.

3.3 Meshes

* See here details of the mesher used by AxiSEM3D, called ‘salvus mesher lite’: [Salvus](https://gitlab.com/Salvus/SalvusMeshLite)
* The installation page for AxiSEM3D here also contains an excellent section on the mesher (written by yours truly!): <https://github.com/kuangdai/AxiSEM-3D/wiki/mesher>

In general, you have a choice between cartesian (box-shaped) meshes, and D-shaped (semi-circular) meshes. These obviously solve for the same physics; and you can actually adapt the cartesian simulation to account for the curvature of the Earth’s surface (to move from local to regional scales) or reduce the D-shaped mesh to a wedge-shaped mesh (to move from global to regional scales).

It’s worth bearing in mind that unless you use a full D-shaped (i.e. 180 degrees/full semi-circle) mesh, you need to think about the ‘false’ boundaries that you impose at the edges of your box or wedge. There’s more on this in the mesher and the Haindl et al paper, but in short you can use an absorbing or reflecting boundary. In the former case you have a few options depending on what you need, the latter is far simpler but will also contaminate your seismograms with reflections from the edge of the box if you let the simulation run on for too long.

3.3 Physical Constraints

When designing AxiSEM3D simulations, or running them, it’s worth bearing in mind what thel limitations of the code are and what they might mean for this project. I’ll briefly detail those relevant here:

*3.3.1 Boundary conditions*

These need to be consistent along the outer surface. What this means is that you can’t have arbitrarily alternating patches of solid and fluid boundaries (ocean/land), or alternating bits of absorbing and reflecting - at least at the moment. Instead, your boundary condition needs to be the same across the length of any azimuth.

Note that I’ve worded this rather carefully: it should, in theory, be possible to create a ‘part-ocean’ Earth, where the ocean occupies a ‘trench’ in a ring around the planet (imagine excavating everything between 10 degrees N and 20 degrees N to a depth of 3km, and filling it with water). This might be useful if you’re interested in things like quakes at subduction zones or mid ocean ridges - but you’d have to edit the mesh file manually in python, and I’ve not tried this. It should not be too difficult though. If you try this take care with the geometry: you will need to rotate your crustal models and source-receiver pair to represent the angles correctly.

If this last paragraph seems unnecessarily complicated, you can (and probably might want to?) stick to using an all-solid or all-fluid boundary condition. If you need to, you can account for the weight of the water column in a non-uniform way across the whole surface using the ‘ocean load approximation’ (see manual).

*3.3.2 Discontinuities*

AxiSEM3D allows you to insert any discontinuity you might want into your ‘base’ (1D) structural model: the moho, the CMB, 410/660km, the seafloor, etc. You can also unduate these boundaries (see manual/Leng 2019/Fernando 2020) to represent the variation in depth of a particular boundary with location.

However, this undulation remains subject to a very important constraint - the undulated configuration must remain homeomorphic to the original, non-undulated one. This is a fancy way of saying that all boundaries must remain smooth and well-defined - the Moho cannot jump from 8km depth to 15km depth at a plate boundary (for example), unless you smooth this transition out across some number of elements. Similarly, you cannot have two mohos at different depths beneath a particular point on the surface (as you might arguably have in reality at a subduction zone).

This is not normally important on a global scale, as you would not resolve a subduction zone finely enough for these details to be important. In the case of this project, it might be important - and you may have to think carefully about how to incorporate the correct moho (or seafloor) configuration.

*3.3.3 More on undulation*

In regard to the above point, it’s also worth briefly touching upon some challenges associated with undulating boundaries in the cases where they are strongly varying, such as those we will consider here.

Particle relabelling (the process through which boundaries are undulated) always increases the computational cost of simulations - the main reason for this is the extra floating-point computations which need to be undertaken. Another potential reason for the added cost is a decrease in the minimum element size. For example, if you have a uniform 8km depth moho, and deform it such that in places it is only 5km beneath the surface, remember that elements cannot be added or removed, only stretched or shrunk. As the elements get smaller, the minimum timestep required for stability will also decrease.

The timestep in AxiSEM3D is globally set by the lowest timestep in the whole model. This means that if the element with the smallest global timestep shrinks, the global timestep also goes down - and the overall cost of the whole simulation increases by at least the same fraction.

Adding in multiple undulating boundaries (e.g. a seafloor and a moho) normally further increases the computational cost - but not always. If the smallest timestep is set in a particular single element (which will be the case if you also have a 3D structural model overlaid, like crust 1.0), and that element ends up being stretched because the moho moves down and the seafloor goes up, this will actually decrease the cost. Whether or not this will happen in your particular case is impossible to determine - it very much depends on the models, undulations, and periods you are using - but it is worth thinking about!

One potential problem that you should bear in mind is that discontinuities can never cross. If you don’t get any unexpected errors related to the undulations you can ignore this point, but if you do, it’s worth a check. I suspect that the issue is most prominent in trenches, where the seafloor dips steeply toward the moho’s position. In reality the two never cross, but when the code interpolates crustal and bathymetry model they may artificially do so - especially if you have rescaled the bathymetry to ensure that all land areas are underwater for the purposes of an AxiSEM3D simulation. If this is an issue, you can either just change the bathymetry rescaling, or possibly go to a higher frequency (more elements means a finer sampling of the models and ought solve the issue).

We will come back to some of the challenges associated with debugging the implementation of undulating interfaces later, but for now it’s sufficient to remember that the deformation of each element from the spherical configuration to the 3D one can be described using the transformation’s Jacobian. Things get unstable when the Jacobian becomes negative or otherwise badly behaved; physically this corresponds to elements doing things that they should not be doing (becoming overly stretched, crossing into other elements, etc).

3.4 Cost and resolution

In any seismic simulation, the resolution (seismic period) that you choose is obviously crucial. Higher frequencies are more computationally intensive, though if you choose a smaller domain you can of course go to a higher frequency for the same cost.

In general, simulation cost in AxiSEM3D scales with roughly the third power of the frequency: i.e. increasing to 5s period from 10s will increase the cost by a factor of 23, or 8.

One positive is that AxiSEM3D’s efficiency over other codes grows as you get to higher frequencies, as most comparable method (e.g. SPECFEM3D) scale roughly with the fourth power of the frequency. The extra computational burden of slow velocity layers (including an ocean) is also less as a fraction of total run time at higher frequencies - in short, although the overall cost is higher, so is the efficiency.

3.5 Should I be using AxiSEM3D?

The question of which modelling software you ought to be using probably has no definite or finite-length answer. However, the common pieces of software uses for our purposes include SPECFEM, AxiSEM3D (which superseeds the original AxiSEM, which I would not recommend using), and Salvus. There are of course others, but I don’t really know much about them.

The advantage of AxiSEM3D over the other two methods is its ability to undertake calculations at much higher frequencies, but this also comes with limitations on how complex a scenario you are able to simulate.

If you are interested in the global 5s resolution wavefield in the presence of strong off-plane scattering, AxiSEM3D can do this where the other two could not, at least not with any reasonable resource usage.

If, on the other hand, you are interested in local-scale wavefields in coastal regions with mixed surface patches of ocean and land, AxiSEM3D is absolutely not the correct code to be using. Before undertaking any synthetic studies, it is worth deciding upon what functionality is important to you, and choosing accordingly.

Other factors worth bearing in mind is that in AxiSEM3D creating a mesh is trivially easy but creating a 3D model to superimpose on the mesh can be extremely challenging to do or indeed to debug. AxiSEM3D and SPECFEM also have the advantage that they are free and open-source, but therefore do not come with dedicated technical support. Of the three, AxiSEM3D is by far the least user-friendly, currently lacking a comprehensive release paper or manual.

**4. Installation**

The AxiSEM3D installation page ([here](https://github.com/kuangdai/AxiSEM-3D/wiki/Installation)) may be challenging to follow if you do not have experience in installing codes on clusters before. Either way, you are likely to run into issues when installing for the first time, and the HPC helpdesk is a good resource if you get stuck.

If everything works, steps 4-6 should be very fast (i.e. a few minutes in total). If things seem to be taking ages (other than downloading new packages via conda), something might well be amiss.

4.1 Dependencies via module

The first step is to install the dependencies of AxiSEM3D - that is, other stand-alone libraries which the code relies on.

BEWARE: this document might not be totally up to date with the latest versions of packages, or what is or is not installed on ARC. See <https://arcdocs.leeds.ac.uk/software/start.html> for information on how to check what is installed, or swap things in and out if you make a mistake, etc.

ALSO NOTE: if you simply type the commands in red below into the command line, you will need to re-do this every time you log into the HPC. We’ll cover creating a script to do this further down.

These are: Eigen, Boost, FFTW, Metis, and NetCDF. FFTW, Boost, and NetCDF are already installed on the ARC system.

You can use the ‘module’ package manager to load them, by typing on the command line:

module load fftw

module load boost

module load netcdf

4.2 Dependencies via conda

The other packages are not (or at least were not) installed on the HPC system when I started this project, so you’ll need to install them via another package management tool, called conda.

To do this you need to first load conda:

module load anaconda

The first thing to do is to create an appropriate conda environment (basically a place where you can install the needed packages without installing them for everyone). You can find out more about environments here: <https://arcdocs.leeds.ac.uk/software/compilers/anaconda.html> but in short you want to type:

conda create axisem3d\_environment

Which you will then activate via:

source activate axisem3d\_environment

Here, you can install packages that AxiSEM3D needs, without risking conflicts over versions or paths with other programs that you might run. If you don’t do this step, you’ll end up trying to install the packages for all ARC users which isn’t allowed.

The final step in this part is to actually install the packages.

conda install -c conda-forge eigen

conda install -c anaconda metis

You will probably be prompted for a y/n prompt at each stage, say ‘y’ to this. If you run into issues with installing eigen or running the rest of the code, see the note on the AxiSEM3D page about eigen versions.

In order to make sure that the code knows where the conda-installed versions of eigen and metis are, we’re going to create a new variable to make things simpler.

export conda\_path = “$HOME/.conda/envs/axisem3d”

4.3. Compilers and MPI

The final thing to do is to make sure that you are using a suitable C++ compiler and MPI version. Trying to manually match these up so that they work with each other and with the rest of the code is effectively impossible, so use module to do the hard work for you.

module load cmake/3.15.1

module switch intel gnu/8.3.0

The first line loads cmake, which is the software that we’ll use to build the code into something that we can run.

The second line switches out the default intel compiler for something that will work with AxiSEM3D, which is the gnu compiler.

4.4 Installing AxiSEM3D

The next step is to install AxiSEM3D itself. For this, follow the instructions on the installation page: <https://github.com/kuangdai/AxiSEM-3D/wiki/Installation> and note that on Kuangdai’s page, there are two version of the code (AxiSEM3D, the old one; and AxiSEM-3D, the new one). You want the new one.

In the rest of this document, I will assume that you’re using a file structure identical to that on the install page, that is that you have a folder on your ARC home drive somewhere called ‘AxiSEM3D\_2020’, in which there are three sub-folders: AxiSEM3D (the code you downloaded from github), build (which is where we are going to put the executable) and dependencies (where the dependencies that you installed will be if you installed them using git rather than conda or module).

**5. Configuration**

Next, we’re going to configure the code - where it will check that everything that it needs is installed in the right place.

Move into the ‘build’ subfolder and type the following:

cmake -Dcxx=mpicxx -Deigen=$AxiSEM3D\_WORK\_DIR/dependencies/eigen-stable -Dboost=$AxiSEM3D\_WORK\_DIR/dependencies/boost\_1\_77\_0 -Dfftw=$FFTW\_HOME -Dmetis=$conda\_path -Dnetcdf=$conda\_path ../AxiSEM3D/SOLVER/

*It is highly likely that the command that you will need to type will be slightly different, unless you happened to install all the packages the same way that I did. Fortunately, due to my own poor computer skills, I’ve used all three possible ways of calling the modules in one go - so you can see each of them in action!*

*IN AN IDEAL WORLD, where you followed the instructions in Sec. 4. correctly (unlike me), you should be able to use $conda\_path for eigen, and possibly even boost, as well.*

Let’s run through what the different parts of this command actually mean, so that you know what you might need to change.

cmake: tells the HPC that we are calling the cmake module which we’ve already loaded.

-D: the ‘variable’ flag for cmake, which tells it that what comes after the -D is a variable name (AxiSEM3D will look for this variable name specifically, so make sure they match!)

mpicxx: is the MPI (message parsing interface) compiler for programs written in C++. MPI is a central part of making programs parallel (more on this later).

../AxiSEM3D/SOLVER: this is where the actual lines of code are stored, and I’m linking to it.

For eigen and boost, I’m giving the path to the dependencies folder that I’ve copied across using git or scp. The path you link to is the topmost folder that you download.

For FFTW, I’m linking to the place that the FFTW module is stored on the HPC.

For Metis and NetCDF, I’m linking to the conda path where conda stalls modules using the variable we defined earlier.

**6. Compile**

Thankfully, this stage is relatively short! You just need to actually compile the code, and then we can run it.

make -j

You should now see the compile progress printed to screen, it should end with ‘linking Cxx executable axisem3d’. If you see this, it’s worked - congratulations!

**7. Automating package loading**

Every time you run the code in a new login window, you need to reload some of the dependencies, like fftw and netcdf.

The simplest thing is to create a script that you can run, and just re-run that each time to save time. Open a new text file called ‘run\_axisem3d.sh’ or something, on the HPC you can do this just as you would on your desktop using vim:

vi run\_axisem3d.sh

i will let you type

:wq will save and quit back to the command line.

In the text file, simply list what needs to be done. Mine looks something like:

module load anaconda

export conda\_path=’$HOME/.conda/envs/axisem3d”

module load fftw

module load netcdf

module load boost

module load cmake/3.15.1

module swap intel gnu/8.3.0

export AxiSEM3D\_WORK\_DIR=$HOME/AxiSEM3D\_2020

Make the run\_axisem3d.sh file an executable (only need to do this once) by typing:

chmod +x run\_axisem3d.sh

And then run it first thing every time you log into the HPC using

. run\_axisem3d.sh

**8. Running the code**

8.1 A first run

Before starting to play with input files, it’s worth checking to make sure that everything runs as you expect. To do this, you’ll need to give the code some input files.

I suggest starting with a low-period global simulation, as they’re the simplest to understand and work with. A 50s mesh is provided in one of the template folders, move it via:

cp -r ../AxisEM3D/examples/01\_spherical\_Earth\_PREM\_50s/input/ ./

You should then be able to run the code! We will use mpirun to split the computational load across multiple processors in general. For the purposes of this first run, I suggest using mpirun with only one process:

mpirun -np 1 ./axisem3d

Where the -np flag indicates the number of processes to spawn.

You should see a print-out of the progress going through the time loop, before a final message confirming the simulation length, the number of timesteps, and the total wall-clock time. The last number is the most useful for charging (budgets, grant proposals etc) so remember that the total computational cost is the number of processors multiplied by the runtime.

8.2 Requesting a session in interactive mode

Before you start running more complex jobs and testing 3D inputs, you should request an interactive session on the HPC to avoid clogging up the login nodes. Type:

qrsh -l h\_rt=0:10:0,h\_vmem=1G -pe smp 2

To break this command down,

qrsh -l tells the login node that you want to start an interactive session

h\_rt denotes the runtime that you want the interactive session for, in the form h:m:s

h\_vmem specifies the memory per core

-qe smp 2 requests 2 cores with shared memory.

Bear in mind that you’re not going to run 5s resolution simulations on 2 cores, this is just for testing purposes. Equally, don’t request too short a run-time if you’re doing something in-depth, as when the session ends you might lose what you were working on!

You will also need to re-load the packages that you loaded in the original session, and you’ll have been moved back to your home directory. So,

. run\_axisem.sh

After which you can move to the test directory and run multi-core operations:

mpirun -np 2 ./axisem3d

8.3 Running batch jobs

The vast majority of jobs that you submit will end up being batch jobs, i.e. something that is sent to the HPC scheduler and slotted in to the architecture when there’s space and time for it to run.

It’s worth reading over the limitations here, and making sure that you don’t break them accidentally: <https://arcdocs.leeds.ac.uk/usage/batchjob.html> e.g. if you run out of memory 90% of the way through a job, you have to re-schedule it in to finish and waste a lot of resources.

To submit a batch job, you need to create a submission script which can then be submitted to the scheduler. Start by creating the script, which you can call whatever you like, e.g:

vi submission\_script.sh

And then add the following lines (comments in pink, don’t copy those!)):

#$ -V -cwd Run in the current environment and in the current directory

#$ -l h\_rt=1:0:0 1 hour runtime

#$ -l h\_vmem=2G 2GB memory per core

#$ -pe ib 48 48 core job (with no limitations on nodes, unlike smp)

#$ -m be send me an email when starting and when done

. ../../run\_axisem.sh load packages needed

mpirun -np 48 ./axisem3d run using mpi with 48 processes

Note that the lines that begin with #$ are commented out in the script, but they are read by the scheduler. The last two lines actually run the executable.

To actually submit this job, first make it an executable (you only need to do this the first time after you create it):

chmod +x submission\_script.sh

and then

qsub submission\_script.sh

It’s worth looking up the syntax for checking on jobs, the cluster state, and how to delete jobs when you invariably make mistakes, but for these purposes you can get by with:

qstat -u XXX

Where XXX is your username. If you see ‘r’ that means that the job is running, whilst ‘q’ means that the job is waiting to be run.

You will see a job number given next to the job. *I strongly suggest writing down this number somewhere*, with a short description of what it corresponds to - it makes managing the data easier when you inevitably get confused about which simulation is which!

Note that unlike in interactive mode, there’s no need to restrict yourself to the -pe smp flag, which requires the memory to be on a single node (and thus limits you to 40 cores), you can use -pe ib which lets the memory be split across multiple nodes as enabled by MPI. The h\_vmem argument is per core, not per node.

8.4 Output

The advantage of running in interactive mode when checking new runs is that you can immediately see if something has worked or not: if the code gets to the point where the time loop is beginning to run, the models have loaded correctly.

If things error out before then, you can check the error message to try and figure out what’s gone wrong. It’s also worth having a quick scan of each section in the output and making sure that they look roughly like that you’re expecting: do the mesh periods and timesteps look reasonable, is the computed Fourier order what you expected it to be, etc?

If you run a job in batch mode, the output should be saved to a file with a name something like run\_script.sh.eY where Y is the job number that the job came with.

If you open the folder ‘output’ when the job (either batch or interactive) is complete, you should see four sub-folders. The only one of interest is likely to be ‘stations’.

What exists in this folder will depend on what you put in the input options for inparam.output.yaml.

Assuming that you’ve not changed any of the default input options, you should just have a series of files, arranged by network sub-folder. For each station, three columns corresponding to the displacements should be saved as .ascii files in the convention ‘network.name.ascii’. There should be another file called data\_time.ascii which contains the timesteps for the seismogram readings, and list\_channel.info which confirms which coordinates the data are saved in.

For a first check, you should just be able to download the stations folder and open the ascii files in a text editor. Your outputs should look sensible: e.g. the displacements in all three columns are zero to begin with, and then increase to values on the order of a few millimetres to a few nanometres or thereabouts. If the displacements are on the order of metres, or there’s displacement from time 0, something has gone wrong.

You can also plot the data using a program of your choice, for example python (which I would suggest). The ascii files are easy to read, and you should be able to see whether they make sense or not via inspection.

In general, ascii files are a very inefficient way to save your data and I would not recommend them if you have more than a few dozen stations. The alternatives, netcdf files, are much more compact - but they’re less intuitive and harder to work with so unless you are committing to using AxiSEM3D for an extended period of time, it’s probably not worth it.

**9. Changing the input files (1D, global models)**

In order to explore problems of geophysical interest, you are obviously going to need to change the input files! There are many, many options available to you and most of them don’t need to be edited. Let’s cover some of the main ones (the AxiSEM3D manual has a bit more detail on some of these, too). If you open the input folder, you will see the following files/folders:

inparam.advanced.yaml

inparam.model.yaml

inparam.nr.yaml

inparam.output.yaml

inparam.source.yaml

GSN.txt (also USArray.txt, but you can delete the latter)

Note that the .yaml suffix just denotes the file type - the files are human-readable.

9.1 inparam.advanced.yaml

It is unlikely that you will need to change anything in this file, as the options are quite technical and likely of limited interest to the casual user. The one possible exception is max\_num\_time\_steps which you can set to 1 or 10 or something similar if you want to make sure that the code will add least run through a small number of timesteps without becoming unstable.

9.2 inparam.model.yaml

This is probably the file that you will spend the most time editing. exodus\_mesh: should be the path to the mesh that you want to use, ending in “.e”.

For the purposes of this project you are unlikely to need to change the geodesy inputs, nor the absorbing boundary inputs (unless you start using mesh ‘chunks’, more on these later).

Attenuation does what it says on the tin: setting it to CG4 or Full will enable a simple linear attenuation model.

9.3 inparam.nr.yaml

This file is only really relevant for when you start including 3D models. In 1D, in a limited set of cases, and if you are really pushed for computational resources, you can make a tiny saving by keeping type\_Nr as CONSTANT and then constant: to 3 (if you’re only using a point force/dipole source) or 1 if you’re using a monopolar (isotropic pressure) source.

9.4 inparam.output.yaml

Here, you need to set the station(s) that you want the displacement or velocity recorded at. list\_of\_station\_groups can take as many networks (groups of stations) as you want.

I tend to keep global\_seismic\_network\_GSN as it’s a useful set of stations all around the world with good data, for testing purposes. If you’re doing something specific though, like array beamforming, you will want to add your own networks too.

station\_file is the name of the text file within the /input folder that contains the station coordinates. Subsequent options like horizontal\_x1\_x2 and vertical\_x3 set what the columns in the station\_file text file correspond to. For the GSN, these are LATITUDE\_LONGITUDE and DEPTH (so in your file GSN.txt, the columns are name, network, latitude, longitude, elevation, depth - note that elevation isn’t used in AxiSEM3D but it’s here so that you can swap files with SPECFEM).

You also need to set the output coordinates using coordinate\_frame and channels. I suggest that you leave these as RTZ and [U] (where [U] = [U1, U2, U3]) for our purposes.

The big choice that you need to make is whether to save your outputs as text files (ASCII\_STATION or ASCII\_CHANNEL depending on what you’re doing), or netcdf files (NETCDF). NETCDF if a lot more efficient, but a little harder to use, especially if you’re not used to the file type before. ASCII has the advantage that you can just open the output and look at whether it makes sense (if the ground velocity is 10e10 m/s after 10 seconds, it’s probably wrong…) but it’s a lot less space-efficient.

In the processing examples I’ll stick to ASCII for ease of use but if you’re doing more complex runs or lots of them I’d suggest using NETCDF.

9.5 inparam.source.yaml

This file doesn’t change whether you’re working in 1D or 3D, so it’s important to get to grips with it.

time\_axis does pretty much what it says on the tin. The default courant\_number of 1.0 is in my experience very unlikely to be stable: start with something like 0.6 and increase once you know it’s working.

list\_of\_sources is also important. The key that you use is arbitrary - name it what you like, I normally use the CMT Catalogue designator for the earthquake in question.

For these sorts of computations, it’s worth sticking to latitude\_longitude, remembering that the north and east is positive. depth is in metres - this is important!

The ellipticity correction is easy to change and makes little difference unless you need very exact arrival times.

We’ll come back to depth\_below\_solid\_surface and undulated\_geometry in the 3D models sedition.

You are most likely to use mechanism: type: MOMENT\_TENSOR, in which case you should input the data as the components of the moment tensor in the order given by CMTSolution. Remember that CMT uses the non-SI unit of dyne-cm, so you need to add a correction factor by setting unit to 1e-7 to turn this into N-m.

For most purposes (seismogram generation) you can do everything involving the source-time function after the simulation is complete in post-processing. In this case, set half\_duration to 0.0 and the code will make the width of the STF something like 5x the timestep (much less than the mesh period, i.e. a delta function). Note that in a model like this this is as good as you’re going to get - the half-duration cannot be exactly 0.00000, but the difference is miniscule and irrelevant. Leave class\_name as GaussianSTF and use\_derivative\_integral as either ERF or GAUSSIAN depending on whether you want displacement or velocity output. We’ll come back to how to process the output in a later section.

If you are doing something like wavefield visualisation, then you need to set the STF before you do the simulation. In that case, you can set the half-duration and decay factor as you want, or use NetCDF\_STF or STREAM\_STF (netcdf vs. ascii - more on this later) to read in a particular source-time function that you want to use, for example one downloaded from SCARDEC.

A general note of caution: although you can use any STF you want for seismogram generation, and then de-convolve it out before re-convolving with a new STF, I find that this is rarely a sensible option as it ends up producing numerical artefacts. If you’re not sure what STF to use, use a delta function (as described above) and deal with it later.

**10. 3D models part I - creating and understanding models**

Changing to use fully three-dimensional models (i.e. more realistic models of the Earth) is where things start to get really complicated. There’s not always an obvious reason why one setup works when another does not, nor is there a clear manual written elsewhere for how to design the 3D model files themselves.

It’s worth reminding yourself of the limitations of the code discussed in Sec. 3 - so that you don’t try and create a model which is impossible to implement.

There are two parts to using the full 3D functionality in the code: the input files and the model files themselves. We’ll start with the model files.

10.1 Introduction

There are two kinds of model files in AxiSEM3D: geometric and volumetric.

*10.1.1 Geometric models - introduction*

Geometric models are those that take a specific boundary (discontinuity), and deform/undulate it’s no longer spherically symmetric with respect to the planet’s centre. The discontinuities you are most likely to undulate are probably the surface or seafloor, the Moho, and the CMB; though in principle you can create a boundary wherever you like and undulate it.

If you like, you can think of a geometric model as taking a 2D surface embedded within the Earth and moving it inward our outward radially, with the degree of stretching or shrinking depending on where on the globe you are.

As such, the geometric model files are not too complicated, as they consist of three coordinates: a series of latitude and longitude points (together forming a geometric grid), and a third variable defining the position of the boundary at each point on the grid. The third variable is either a depth or a radius (which you use depends on the boundary: radius makes more sense for the surface, depth more sense for the Moho in my mind). In every case, the depth and radius are given with respect to the boundary’s depth or radius in the 1D case.

As an illustrative but pointless example, if you created a 3D geometric model file for the Earth’s surface but wanted the surface to be totally flat, you’d have a series of latitude and longitude points (-90 to 90 and -180 to 180) respectively, and the third variable would be an array of zeroes (meaning no change in either direction from the surface position in the 1D case).

Obviously the number of depth/radius points in your 3D model must match the number of latitude/longitude gridpoints you have. If you sample latitude and longitude each at a spacing of 1 degree, that gives you 180 x 360 gridpoints, each of which must have a value for the surface elevation or depth. You can choose the spacing that you want: 1 degree is probably fine for most things in this project.

Note that the undulations on a particular boundary are given as depth/radii with respect to the boundary in question itself. This is entirely unclear from the input files or the Leng (2019) paper, but it’s important to bear in mind. This means that an undulation of +1km on either the Moho or the surface is given in your model file as +1000 at that particular point, not 6372000, 6372, or anything else. In the input files you set which interface this is in reference to, e.g. 6371km radius (or 0km depth) for the surface.

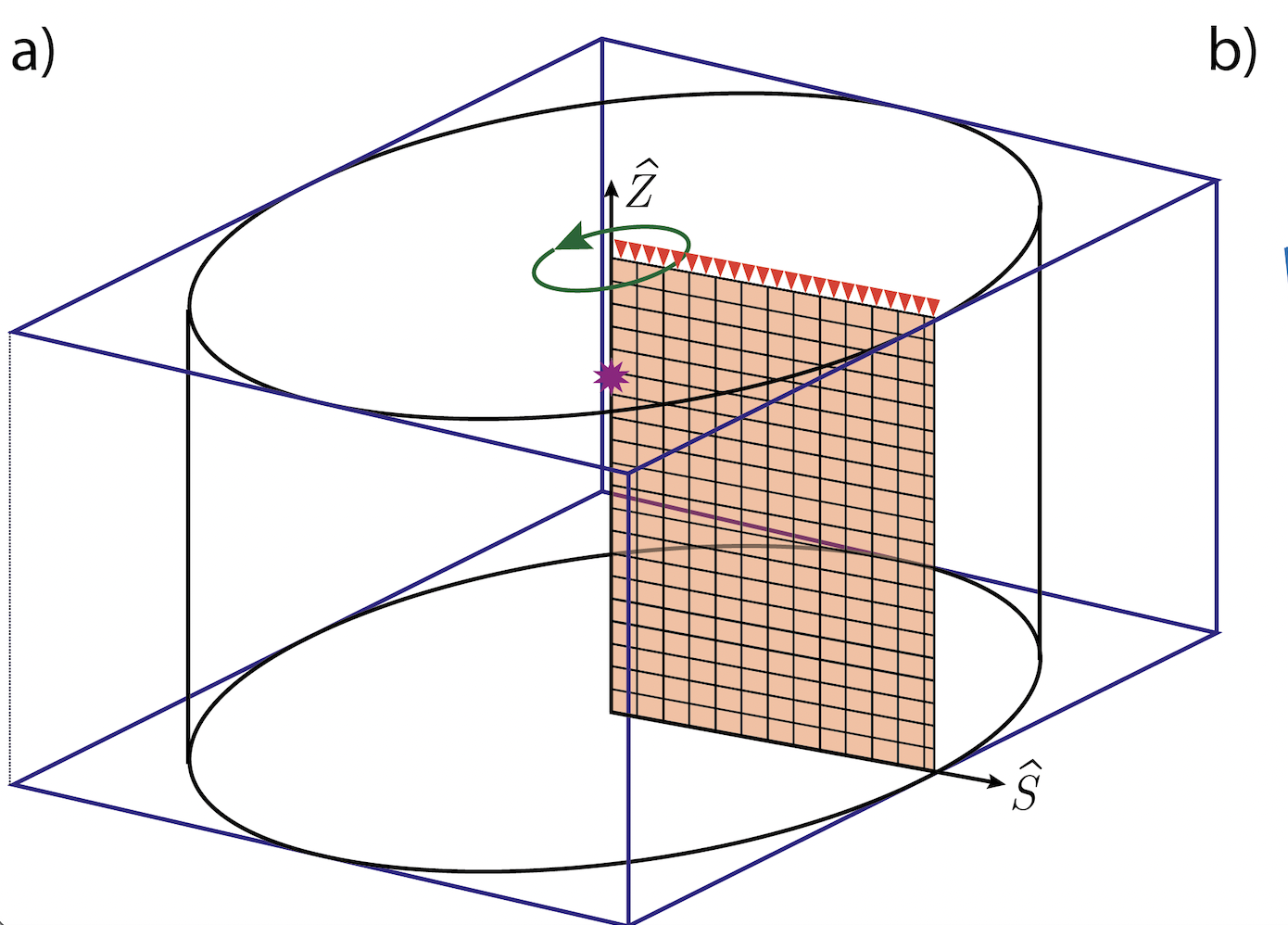
*10.1.2 Volumetric models - introduction*

Volumetric models are those which change the seismic properties of mesh elements not at interfaces: for example, decreasing vs in a particular region to represent a region of partial melt.

In AxiSEM3D, you can alter three parameters at any point in the mesh: vp, vs, and density. The volumetric models are like the geometric ones in structure, but rather than there being two spatial dimensions there are three (latitude, longitude, and radius/depth). There are also up to three other variables, though you need not change all of vp, vs, and density if you do not wish to.

*This section contributed by Will Eaton (Princeton University)*

Volumetric models require the production of a NetCDF dataset file (.nc) which can be done using your favourite coding language such as MATLAB or Python. It is however important to remember that these two languages save multidimensional arrays in column-major and row-major formats respectively, by default, but more on that later. If you are unfamiliar with netcdf files I would recommend reading: https://towardsdatascience.com/create-netcdf-files-with-python-1d86829127dd



AxiSEM incorporates 3D models by rotating the 1D mesh around its axis, forming a cylinder (or sphere if using a D-shaped mesh). Any parts of the 3D model that are within that cylinder will then be incorporated into the overall 3D model that axisem uses. The figure above shows a mesh rotating around a source-centered axis. The 3D input model is shown in blue. Any part of this input model that is within the cylinder (black) is sampled. So, how do we create this 3D model shown in the blue box?

The netcdf file holding the 3D model needs to contain 6 variables, which contain information about: x, y, z/depth, vp, vs, rho - though you do not need to change all of these from the 1D values if you do not want to.

Note that the names of these variables within the .nc file **do not** matter. This is because you will specify what you have named them using the variables nc\_variables (for x, y, z) and the variable factor for each of the properties VP, VS, and RHO. As long as the names are consistent, then all is good!

The .nc file will also need to have 3 **‘dimensions’**. These need to be the dimension of the x, y and z array variables, i.e. the variable has a defined and specific value at each point in the grid. Each of the **variables** x, y, and z (or depth) have a single dimension. Each of the variables Vp, Vs and Rho will have three dimensions (corresponding to the dimensions x, y, z). To see this in action, you can load the .nc file provided in Example 3 *(see template folders in main AxiSEM-3D github)*, and load it into python using:

Import netCDF4 as nc

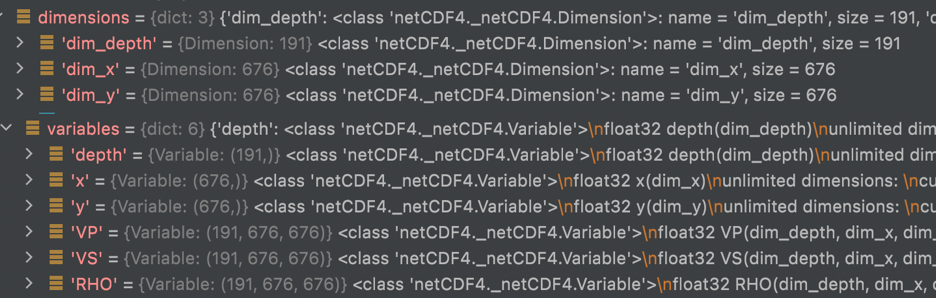
data = nc.Dataset(“path/to/file/SEG\_C3\_SOLID.nc”)

print(data)

In doing this, you will see the 6 variables in which x, y, and depth are 1D; while VP, VS and RHO are 3D. The x, y, z variables are simply an array of coordinates for the limits of your model, e.g with x going from -5062.5 to + 5062.5 km with 676 elements. You can make these using a normal linspace function. To print the actual data values of a variable “x” held in the dataset called “data” you can use:

print(data[“x”][:])

Equivalently, the Vp, Vs and Rho variables hold a 3D array (e.g. a numpy array) with corresponding values at each of those coordinate points in x, y, z space.



10.2 Creating new geometric models

The base structure for global scale, 3D geometric models should be included in the GitHub repository associated with this project. But in short I’ve created two new python files (etop01.py and moho.py) which can be used to create and edit topographic surface and moho geometric models.

*10.2.2 ETOP01:*

* ETOP01 is a global topographic/bathymetric model that is sampled at 1-arcminute resolution. If you are technically minded, I have used the grid-centred version of the model with ice included.
* 1 degree sampling is very much overkill for this project, and is not feasible due to the amount of memory to store a model of this size. It should be downsampled to something more like 1 degree. If you want a more accurate determinant of how to downsample it, you can calculate the minimum resolved seismic wavelength at a particular period, and use a sampling which is at least as fine as this. Remember that as the period of the simulations you are running decreases (i.e. as the frequency increases) the waves become sensitive to smaller and smaller spatial structures, so you can get away with less downsampling.
* The strongest gradients in ETOP01 (by quite some margin) are off the west coast of South America, where the Andes meet the coast and the subduction trench. This can potentially be a problem: strong gradients can make for challenges in interpolation. We’ll come back to this later, but you can get around some of it by applying some smoothing.

*10.2.3 Crust 1.0 Moho:*

* The Crust 1.0 Moho is sampled at one degree intervals, and generally you will not need to resample this as the frequency changes. Certainly you cannot get to anything less than 1 degree resolution, as this is as good as the data gets!
* Again, some of the strongest gradients in Crust 1.0’s Moho are at the western edge of South America. Obviously this is not simply a coincidence - there’s a good physical reason for it! Unfortunately it does mean that the Moho and the seafloor/Earth’s surface end up being quite close to each other. This both shrinks elements (making timesteps small, and cost expensive), and risks the boundaries interpolating into each other (which stops the interfaces being conformal, and breaks the code).

10.3 Using the supplied python scripts

The python scripts available in the GitHub repository are able to write your Moho and topographic models to a format that will be read by AxiSEM3D. I assume some familiarity with python and that you are able to run the functions given and are aware of how they pass outputs to one another using *return*.

You are not obliged to use these scripts by any means, but they are designed such that the file structures, variable names, and sampling all match the template input files provided. If you have your own model which is in a similar gridded format and you want to convert it into something you can use, you should just be able to read in the filenames and use the same functions in the python scripts to write the output. It’s always worth re-reading in a new model that you’ve created and then plotting it to make sure that it’s doing what you think it’s doing.

*10.3.1 Crust 1.0 Moho*

Let’s start with the Crust 1.0 Moho script as that’s somewhat simpler to understand. It’s called “moho.py” and takes as input a grid file (.xyz) called ‘depthtomoho.xyz’ which is a list of latitudes, longitudes, and moho depths at each pair therein.

The first function, load\_moho\_data, takes the input file and uses a series of numpy functions to split it, for simplicity I’ve only actually kept the 3rd column ([2] in python) which is the depths, and re-created the latitude and longitude from scratch. Make sure that the coordinates are sorted in ascending order (-90 to 90, -180 to 180) or else the code will not like it.

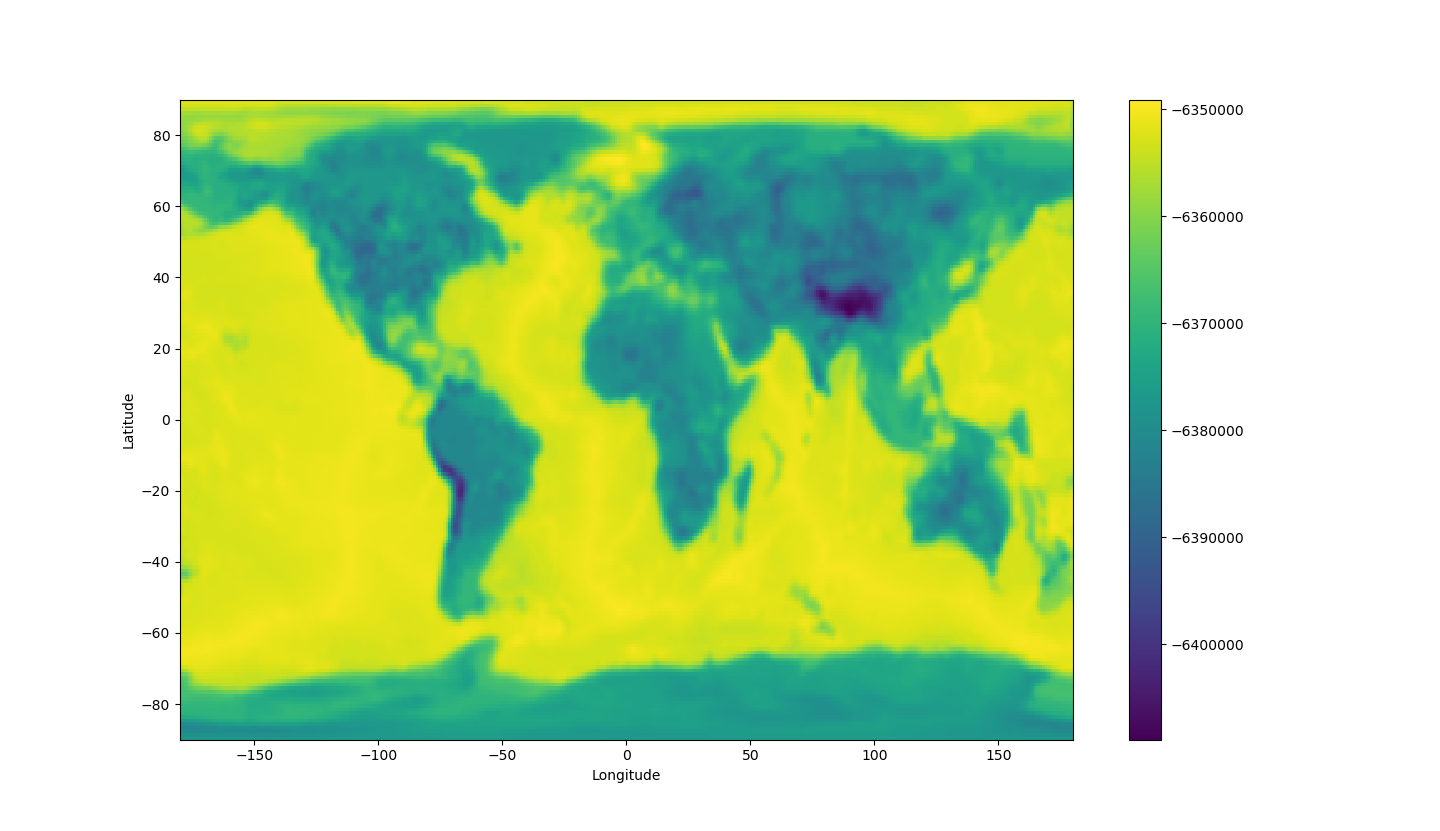
The second function, write\_data\_to\_netcdf\_file, writes the inputs of the above function to a netcdf file. If you’ve not come across netcdf files before it’s worth looking them up, they are basically a highly efficient way of storing data. Unlike ascii files you don’t need to worry about pairing up the points correctly, within each file is a header called ‘latitude’ (similarly for the other variables) which contains the points you’ve created.

There are a few stages to writing your new model file into netcdf format. First you need to create the file using Dataset (a method in the python netCDF4 module), and then create the three dimensions. You can specify the length of each dimension (latitude being 180 points, longitude being 360), or set this to 0 to allow them to grow as needed.

Then, you need to create the variable spaces to go with the dimensions you’ve created, and specify that the values will be floating point numbers (‘d’). Finally, you add the actual points into the variables fields. Make sure that the units match, add corrections for metres to kilometres or vice versa if needed.

Note that if at any point you get errors about not being able to open a file, that’s probably because it’s already open. You just need to close it before opening it again.

Finally, read\_data\_from\_netcdf\_file allows you to just re-load, and if you want to, plot (set plot=True) the saved netcdf file to check that it looks like what you think it should look like.



**Fig 1:** Crust 1.0 Moho undulation as a function of latitude and longitude. This plot shows the model with 1 degree smoothing, sampled at 1 degree. For more details on the radial coordinates used, see below. Note that even with 1 degree smoothing, there are still strong gradients in Moho depth off the west coast of South America. I suspect that this area (or somewhere like it elsewhere in the world) is the cause of the issues.

*10.3.2 ETOP01:*

Using the ETOP01 script is slightly more complicated than using the Moho one, because you need to downsample (and possibly also average or smooth) the topography from the raw file that you download for the reasons that we already discussed.

The functions in etop01.py are similar to those in moho.py but with a few of additions. When you read in the raw model, there are two things that you can do to it:

1. Average: average across n datapoints to produce a new topographic model
2. Filter: smooth the gradients in the new topographic model out using a gaussian filter

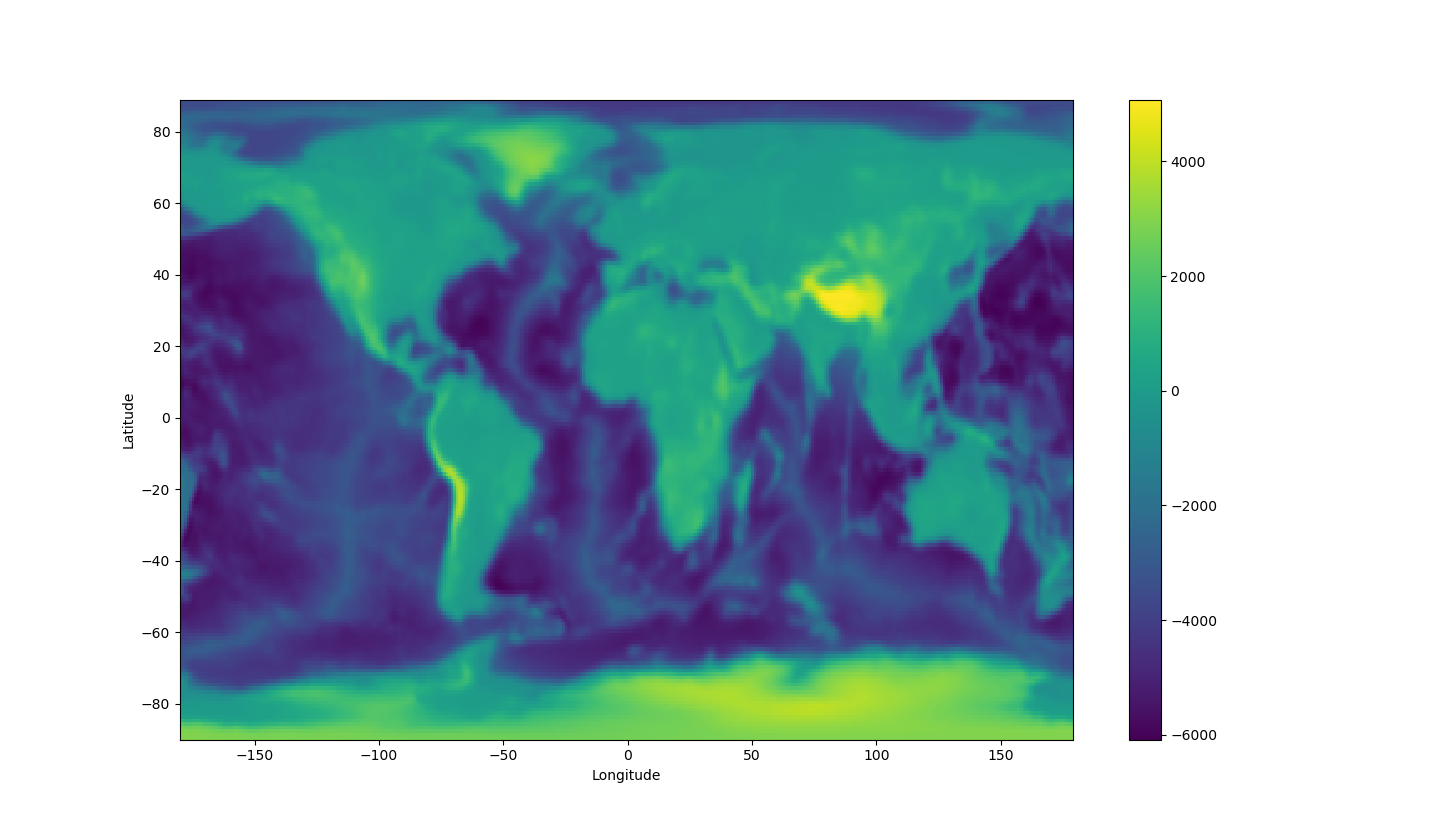
If this seems confusing, consider that 1) basically means that you take a set of n by n points in the original 1-arcminute resolution model and represent their average as a single point in your new topographic model that AxiSEM3D is going to use. Choosing how big n is allows you to choose how fine your new model is (larger n = coarser model because a larger area is being represented by a single new elevation point). Once you have your new model you can further smooth the gradients in it by applying a 2D filter (like a blurring filter on Instagram).

The default sampling that I’ve chosen for the new 3D model is 1 degree, to match what’s used for the Moho. If you want to change this it’s relatively straightforward, just edit the relevant functions in etop01.py.

Rather than just simply choosing the midpoint of every 1 x 1 degree cell you instead take the arithmetic mean of each 1 x 1 arcminute cell within it to create your new elevation point, as per method 1). This naturally averages out extremes in your new model, and makes it less likely that the Jacobian will be badly behaved. I can get this to work for around 10s period global runs, but not 5s. This is probably because some of the gradients are still too large to be stably interpolated by AxiSEM3D. As noted before I suspect that the issue is off the coast of South America, where a trench and an altiplano are found in close proximity, but have not tested this. If you’re doing regional-scale runs away from this area that might solve the issue, but in the more general case, you need to use 3): a filter.

Starting with 1) - an average across each 1 x 1 degree cell - the copy of ETOP01 provided in these files is then filtered using 2): a 2D 1 degree standard deviation Gaussian. In effect, this treats the output array from the averaging as an image and applies a slight blurring filter; or technically a 2D convolution with a Gaussian kernel.

This simply smooths out some of the largest gradients in averaged array. You can choose how much smoothing happens by changing the value of sigma in filter\_data. The default is 1 degree, reducing this toward zero makes the Gaussian kernel steeper and narrower and reduces the degree of blurring (in the limit of a zero standard deviation, the original image is returned). Similarly if you still get an error, you can increase the standard deviation which smooths out even more of the gradients. Note that you don’t need to pass the units of sigma to the code, it does this in terms of sampling points (i.e. ‘1’ ends up being 1 degree because there are 180 x 360 points, i.e. one per degree). If you do edit the python script to change the sampling, make sure the units of sigma are also changed to match.

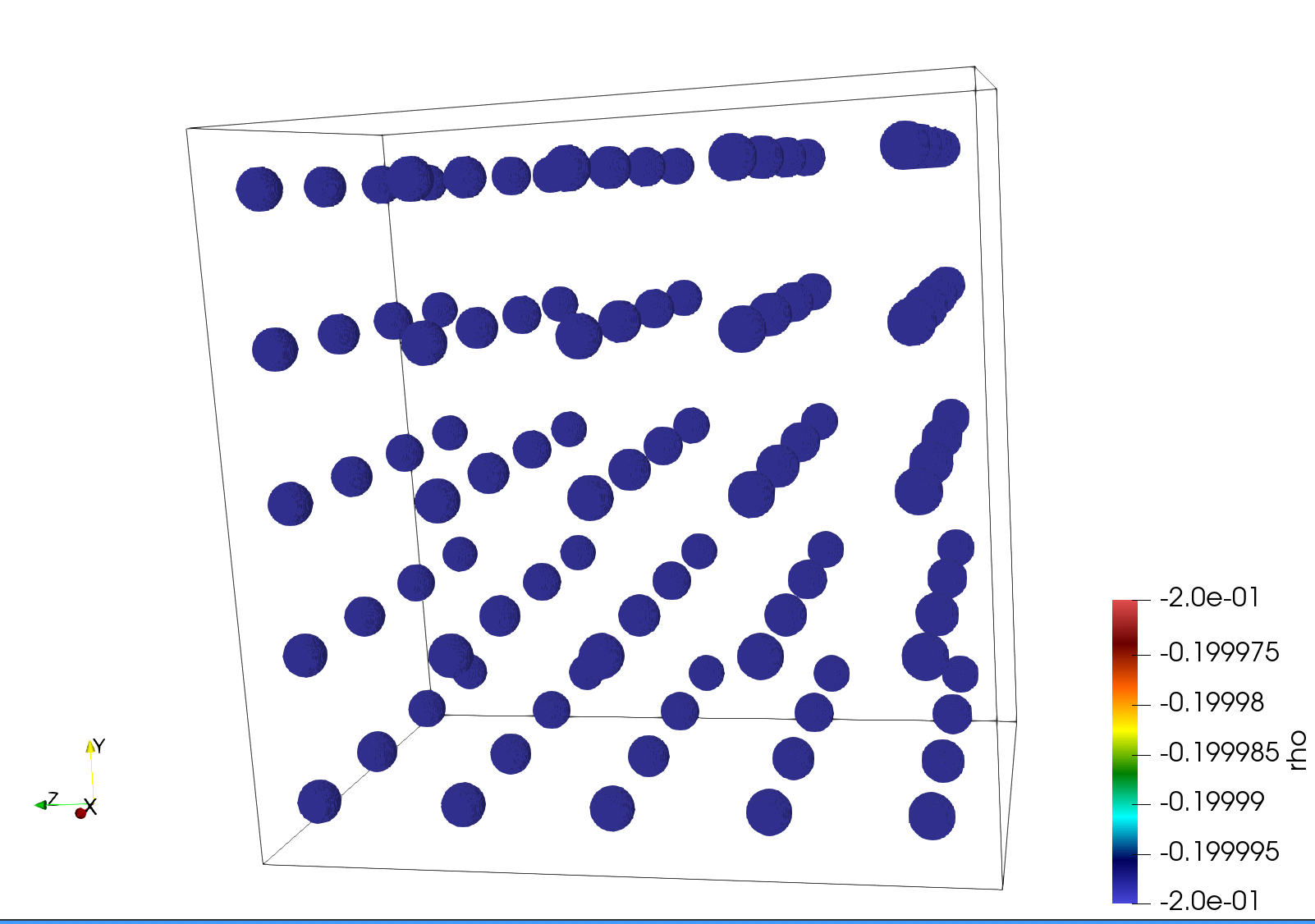


**Fig. 2:** ETOP01, sampled at 1-degree intervals and and smoothed using a 1 degree sigma Gaussian. Again, note that the strongest gradients in topography are at the same locations as the strongest gradients in Moho undulation, i.e. at the western edge of South America.

Note that there are potentially some issues associated with switching from grid-centred to cell-centred systems at the poles - see the moho.py file for some comments on this topic.

10.4 Creating new volumetric models

There are a couple of python scripts for producing 3D models in which you can inject spheres at certain points, so that your 3D model may look like something shown in the figure below. This is available at<https://github.com/willeaton101/axisem3d-blobs> - even if the blobs are of no interest/use it may be worth checking out the write\_NetCDF.py script in that repo, which shows how I take some 3D numpy arrays and make an .nc file from them.



The actual values that you use in your 3D arrays can be used by axis in a number of ways:

1. absolute values - the value in .nc is what is used by axis
2. Ref 1D - a perturbation relative to the 1D model (e.g. 0.2 is 20 %)
3. Ref 3D - perturbation relative to the 3D model
4. Ref\_perturb - perturbation relative to the current perturbation at that point

Most of the variables in the inparam.model.yaml file are quite self explanatory for the 3D case. The ones to draw attention to are:

*data\_rank* - this is where you need to be careful in respect to python vs Matlab, and tell Axisem what order your coordinates are in

*nv\_variables* - tells axisem the name of x, y and z variables that you used in the .nc file - e.g. this could be ‘X’, or ‘x’, or ‘x\_arr’ etc…

**11. 3D models part II - usage**

Once you’ve created a new 3D model and saved it as a netcdf file, you need to upload it to the /input folder. This section focusses in geometric models, mostly because I’ve not yet figured out how to create and use new volumetric models.

Then, there are two input files that you need to alter in order to get the model to be read in and represented properly: inparam.model.yaml and inparam.nr.yaml.

11.1 inparam.model.yaml

The portion of the inparam.model.yaml file that needs to change from the 1D case is the “3D Models” section. Under the header list\_of\_3D models, you can add as many as you want. Let’s start with etopo1, which you can put as the key in the first indented line. If you are doing checks and want to run the code without the model being read in, but don’t want to rewrite the whole input file, just toggle ‘activated’ to false.

class\_name for all 3D geometric models will be StructuredGrid3D. nc\_data\_file is also just the name of the 3D model file that you’ve created and saved in the /input folder.

For coordinates, this project will probably use LATITUDE\_LONGITUDE and either DEPTH or RADIUS. Just make sure you get the latter correct: etop01 is defined as an elevation with respect to some mean sea level whereas the Moho file I’m using is defined as a depth with respect to sea level.

Again, you can use the ellipticity correction if you want to but it is unlikely to make an enormous difference. If you’ve used the provided python scripts, you should not need to change the data\_rank, length\_unit, angle\_unit, or factor flags from those I’ve given ([0,1], m, degree, and 1.0 respectively).

The things that you will need to change are as follows. Close and careful attention is needed when changing these, as mistakes can be very challenging to debug. The order that I go through these is not the order that they appear in the input file, but I think it’s more logical.

* nc\_variables and nc\_var: these are the variable names used in the netcdf file. If you’ve kept the defaults, the first one should be [latitude, longitude] and the second one should be depth for moho and elevation for etop01. Note that these are just the labels that we used in our netcdf file, they can be called anything and just because you set ‘depth’ or ‘radius’ as the name doesn’t mean that the code will treat it as such unless you also set the flags below.
* vertical should be DEPTH for the Moho and RADIUS for topography. Make doubly sure this is consistent, if you have files which you think are the wrong way round and don’t want to remake them, you can always swap factor from 1.0 to -1.0.
* undulation\_range: interface should be the depth or radius of the interface you’re undulating, as defined in the background 1D model. If you’re using depth as your variable (as I used for the Moho), then this should be e.g. -24400.0, as the Moho is 24km below the surface in PREM. If you’re using radius as your variable, as I do for the surface topography, this should be 6371000.0. Remember that you will define the undulations as elevations **above** the boundary in consideration. So, at a subduction trench where the Moho is at 7km below the surface, you would define the gridpoint for the Moho’s depth as +17400 (-7000 + 24400 = 17400m). Similarly, in Tibet where the Moho is at 80km depth, the gridpoint would be -55600 (-80000 + 24400 = -55600m).
* depth\_below\_solid\_surface is an important flag. Lots of models in seismology (e.g. PREM) do not include the ocean, so if you add one on top, you need to either add one right on top (making the total radius 6374km), or remove 3km of rock first (to keep the radius as 6368km). The latter is more physically meaningful but might be harder, depending on the model you’re working with. This flag allows you to define whether your interface coordinates are with respect to the seafloor (true) or the ocean surface (false). You might also want to think about whether, when you read in a 3D model for sub-surface features, the depths should be defined with respect to the solid surface in the 1D case, or the 3D case - as the position of the surface may have moved considerably in between these.
* undulation\_range: min\_max is the final variable to think about here. This is hard to visualise, but in effect it tells the code in which range it’s permissible to stretch or shrink elements to accommodate your deformed boundary. For example, if you set the Moho to be at 24km and then undulate it, elements on either side need to deform to accommodate the undulated boundary. min and max define the lower and upper edges of the deformation zone, which should be at least twice as large as the undulation range. For example, in Crust 1.0 the Moho depth varies between 75km and 7km, so doubling the deepest depth and halving the shallowest one in this range would give you min\_max = [3500.0, 150000.0]. Again, just make sure that the signs are consistent and correct: 3500.0m is actually a shallower depth (i.e. the minimum deformed depth), rather than the maximum! Either way the interface needs to be contained within the bounds you’ve set, else the undulation will not work!

11.2 inparam.nu.yaml

*11.2.1 Fourier series introduction*

This input file, which we did not touch upon earlier, is used to set the azimuthal (i.e. variation in longitude) parameters of the simulation. You will recall that AxiSEM3D works by taking a 2D mesh and representing the third dimension through a pseudospectral (Fourier) expansion. The order of this expansion determines the complexity of the entire simulation in the azimuthal direction. Note that this does not just affect the degree to which 3D models are accurately represented, lower Fourier order expansions also reduce the azimuthal complexity of the seismic wavefield that you are going to record.

To understand this another way, the Fourier expansion in AxiSEM3D is akin to the representation of any normal mathematical function in terms of sines and cosines. The more terms you include, the more accurate the function’s approximation is. For an arbitrary function, an infinite number of terms is needed for an exact solution - but we can make do with a smaller (finite) number of terms Nr if we’re happy for the solution to only be accurate to within a given degree.

How much smaller than infinity the number of terms/Fourier order Nr can be depends on a number of different factors:

* The complexity of the 3D model that you add in to AxiSEM3D. If you have a very long-wavelength model with relatively smooth variations, like SEMUCB, you need far fewer terms than if you have something that’s strongly varying and densely sampled, like ETOP01. For a 1D model, you never need more than Nr = 5.
* The complexity of the wavefield that you want to represent: in general, you’ll only add in a complex 3D models if you want to see what impact they have on synthetic seismograms. However, if you wanted for some reason to use a complex 3D model but were not interested in the fine detail of the wavefield, you could get away with a lower Fourier order.
* The complexity of the source: as noted before, if you’re only using a 1D model you can save a small amount of time by reducing the Fourier order needed to represent the source term. A second rank moment tensor (i.e. something from CMT) requires Nr = 5, a point force requires Nr = 3, an isotropic/monopolar pressure term requires Nr = 1. For most 3D models the value of Nr needed to capture their variations will be far larger than 5, so there’s no saving to be made even if you only use a point force or pressure term.

Remember that the overall resolution of the simulation (i.e. the smallest scale that you can resolve meaningfully) is still limited by the mesh resolution, so increasing the Fourier order too much becomes pointless as you are not sensitive to these small spatial scales anyway. For this reason you should leave bound\_Nr\_by\_inplane = true.

*11.2.2 Choosing input parameters*

You will probably be wondering by now, ‘how small a Fourier order can I get away with?’ - the answer is of course that this depends on how accurate a simulation you want, and how much computational resource you have available - a higher Fourier order increases both runtime and memory demands. There are four options that you can choose in AxiSEM3D for how the Fourier order is calculated, which may help with this question. These are, type\_Nr = :

* CONSTANT: here, the same Nr is used throughout the mesh. As mentioned previously, for a 1D model with a moment tensor 5 is both necessary and sufficient.
* ANALYTICAL: this is like constant, but with slightly finer user-control, in that you can choose how Nr varies through the interior of the planet. As an example, if you wish to investigate the effects of crustal structure on surface waves, you might want a high expansion in the mantle, but not in the core. This is thus a finer level of control than in CONSTANT, which can save you both time and memory.
* STRUCTURED allows you to specify even finer degrees of control, by setting the fourier order on individual elements. You can thus save even more time and resource with respect to ANALYTICAL, but a more detailed knowledge of how to create the appropriate grid of points is needed.
* POINTWISE allows you to go even further by having the code try and optimise the variation of Nr through the mesh. See below for more details.

Within ANALYTICAL, you can specify the variation of Nr with depth that you wish to use within the depth\_dependent\_AxiSEM3D\_default option. Using the options control\_depths and Nr\_at\_control\_depths you can make a pointwise specification of how Nr varies, remembering to give your depths in metres.

For example, control\_depths: [0., 6371.] and Nr\_at\_control\_depths: [100, 100] would set Nr = 100 across the whole of the Earth’s radius. Changing Nr\_at\_control\_depths to [100, 1000] would set Nr to 100 at the surface, and 1000 at the planet’s centre, with linear interpolation in between. You can add as many control points as you want into these arrays. For example, if you wanted to look at high-resolution crustal structure, you might do something like:

control\_depths: [0, 50e3, 200e3, 6371e3] and Nr\_at\_control\_depths = [1000, 1000, 100, 10] which would give you Nr = 1000 in the top 50km, decreasing to 100 in a linear fashion by 200km depth, and then linearly again from 200km to the centre.

*11.2.3 Wavefield scanning*

Wavefield scanning is an advanced feature of AxiSEM3D which I’ve not ever actually used before. It involves the code trying to ‘learn’ the most optimal Nr profile it can, using past runs as a guide. In a way, this lets the code do all the optimisation described in the steps above itself, such that the runtime and memory requirements are minimised.

This method works by analysing the frequency spectrum at all the interpolation points in the mesh. For a given point, if the energy above a given frequency is below some threshold value (which you can set), the code determines that frequencies above this point are not contributing to the overall solution, and the Fourier series can thus be truncated. In subsequent runs, the value of Nr at these points can therefore be reduced as they are determined to have a reduced azimuthal wavefield complexity.

If you want to try this, you can set enable\_scanning to true and type\_Nr = POINTWISE. Follow the instructions in the input file to make sure that you match the filenames and copy the relevant files across.

**12 Common Errors**

It is probably worth covering some of the common errors that may occur in your simulations. I will not cover ‘trivial errors’ (i.e. forgetting to upload the correct files or putting them in the wrong place), but the following might come in useful.

In general, errors will be one of three types: preloop, runtime, or result. Preloop errors occur because the code has been unable to initialise properly, for example because something has gone wrong with one of the inputs. Runtime errors occur within the time loop, and will generally cause the code to crash. Result errors are where the result you get out does not make any physical sense. Some errors are also common to preloop and runtime and may occur during either.

You can sometimes diagnose errors by checking either the .e or the .o files that appear in your code directory. For example, if the script you use to submit the batch job is called run\_script.sh, you should get two files that form in the same directory as the submission script called run\_script.sh.eX and run\_script.sh.oX where X is the job ID of that particular job. The .o file contains code-specific information, i.e. what would have been printed to the terminal. You can check that the output matches what you were expecting, and see where the code stops working. The .e file contains system information and may highlight what sort of error it was.

If you are using the -m be flag in your submission script, the ‘complete’ email might also tell you whether it was a qsub related issue (out of time, out of memory) or something else.

12.1 Either preloop or runtime errors

* Out of time: Probably the most common error will be an out-of-time error, where the AxiSEM3D simulation does not finish within the time window that you specified in the scheduler using h\_rt. There are two ways to address this without changing the simulation: increase the runtime, or increase the number of cores being used. If you cannot do this, you can: reduce the record length, increase the mesh period, or decrease the Fourier order.
* Out-of-memory: Sometimes it can be hard to diagnose out of memory errors, as the code may just crash, but the complete email might tell you that the scheduler killed the job as it exceeded the memory requirements (or words to that effect). The simplest option is to increase the memory allocated per node (change h\_vmem). Note that it’s not quite clear on the ARC4 pages, but the -pe smp flag confines you to a single node, whereas -pe ib allows you to split across multiple nodes, so use the latter. The h\_vmem is **per core** so the total memory you’re requesting is the -pe ib value multiplied by the h\_vmem value. The total memory per node is 192GB (or 768GB on a large memory node), so you can probably ask for 10GB for a 40 core job (400GB total), or 1GB for a 400 core job (400GB total also), but not 10GB for a 400 (4TB total) core job. Alternatively, you can switch to the larger memory nodes, increase the number of cores that the load is split across, or change the simulation itself (lower Fourier order, lower mesh resolution, etc). Note that decreasing the simulation length is unlikely to change the memory requirements of a particular run.

12.2 Preloop errors

* Jacobian distorted: This is a very common error which is due to the way that the 3D model is being applied. As discussed previously, it basically means that there is an error in the interpolation somewhere. There are a few options: 1) decrease the degree of undulation (which you are unlikely to want to do much of in reality) using some sort of filtering as done in the supplied python scripts, 2) or increase the min\_max bounds in inparam.nr.yaml (allowing the deformation to be accommodated across more elements), or 3) change the mesh resolution (this could go either way, as more elements resolves more gradients but also allows more elements to accommodate the deformation). Changing the GLL order in the SOLVER/CMakeLists.txt might also work (from say 4 to 6), but no guarantees here as I’ve not actually tried it. To do a test and make sure that it’s an issue with the file itself,

12.3 Runtime errors

* Code blew up: the code should recalculate the necessary timestep dt to ensure stability regardless of how many 3D models you add onto the 1D base model. However, this does not always work, and sometimes the code will become unstable. The simplest option here is to adjust the courant\_number in inparam.source.yaml, from its default of 1.0 to something more conservative (e.g. 0.6). Note that a decrease in the courant number gives you a roughly linear increase in the runtime, so avoid being over-zealous in reducing it. If you’ve reduced this below around 0.3 and the simulation is still unstable, it may be an issue with how the 3D model is being read in, and you should try checking that instead. If you need to diagnose an instability that you can’t get around, go to inparam.advanced.yaml and make sure that stability\_interval is set to 1.0 (meaning that instabilities are checked at every timestep). If that doesn’t work, go further down the file and set max\_num\_time\_steps=1 (or 10). This will limit your runs to a single (or a few) timestep(s), and you can make sure that things are working properly without having to ask for longer runtimes; thus reducing the waiting time in the ARC queue.

12.3 Result errors

By result errors I mean things that when you plot them up, look wrong - obviously it’s something that’s gone wrong in the simulation, but you may have got to the end of the time loop without it being obvious what the issue is.

* No signal present in seismograms: the simplest explanation would be that you’ve not run the simulations for long enough for any of the energy to arrive at the stations you’re looking at, but it could also be an issue with the source. Check inparam.source.yaml, if nothing looks obviously wrong you can put a station at the exact position of the source and check that the source’s shape is what you expect. If you’ve used a delta function source it should be something with a very steep peak.
* Amplitudes are excessive: it is possible for the runs to be unstable without triggering the instability checker. First, check that the units of your source term are what you expect - though this is a linear scaling so it will just give you amplitudes that are too high, but seismogram-like wiggles. If you’re getting exponential growth or exponentially growing oscillations, try reducing the courant\_number in inparam.source.yaml.
* Seismograms do not match benchmarks: if for some reason you end up benchmarking AxiSEM3D against another code, like SPECFEM or YSpec, you need to be even more careful to get exact agreement between them. Bear in mind that the choice of decay\_factor in inparam.source.yaml makes a big difference, and is not the same in all codes. Also, AxiSEM3D has no gravity, whereas other codes do. Similarly, AxiSEM3D’s ellipticity correction may be written slightly differently to other codes, so check that. Finally, the attenuation bands that you’re using need to match, see inparam.model.yaml for a few more details and links to the relevant papers.

**13 Plotting and processing data**

Perhaps the most important part of the entire simulation process is actually plotting and processing the data that you’ve generated!

In this section I’ll assume that you have a series of ascii (text) files containing the displacements recorded at the stations that you’re interested in. As noted previously, netcdf files are much more efficient and I would recommend them if you’re doing much more than generating basic seismograms. Also, if you’re doing wavefield visualisation, definitely don’t try and do this using ascii!

13.1 Ascii file contents

The inparam.source.yaml and inparam.output.yaml define the contents of the ascii files that we’re going to plot. I will assume that you’re familiar with the files based on the contents of Sec. 9 of this document, and have a file for each station containing three components of displacement; as set using the format and channels options in inparam.source.yaml.

Rather than saving the time series (i.e. the list of timesteps) within each file, this is stored separately in a file called data\_time.ascii. This should be a single column of floating point numbers, each of which is a different point in the time series. The first one is probably slightly negative, as AxiSEM3D ramps-up the source time function before the zero time index on the simulation. The final entry should be roughly what you set the record length to be in inparam.output.yaml.

If you’ve not changed the station list, you should have a series of other files. For example, in the folder global\_seismic\_network\_GSN, you should have a series of stations, for example IU.TEIG.ascii. These are your station files, and the number of columns in each (and the order) will depend on the channels options you set in inparam.output.yaml. If you used [U], you should end up with three components, in the order R, T, Z if you used coordinate\_frame = RTZ.

13.2 Software for working with output files

There are obviously no fundamental limitations on how you can manipulate and use output files, you could even print them out and plot the points by hand if you want! **My strong recommendation would be to use Python, however -** even if you think you’re more comfortable with MATLAB, the open-source seismogram processing functionality exists only in Python.

Within Python, you can probably get by with ObsPy, NumPy, and SciPy. I presume that you already have NumPy and SciPy installed, you can find all the documentation needed to work with ObsPy here: <https://docs.obspy.org> which includes tutorials, documentation, and worked examples.

It’s worth running through some of the first few tutorials, to get you used to the language used: streams, channels, Stats, UTCDateTime objects, and the like.

13.3 Reading in seismograms

As per the included Python scripts, I think the simplest way to read in the seismograms is:

* Use numpy.loadtxt to read them in
* Use numpy.hsplit to split the columns into separate arrays, and then concatenate into a single array
* Create a Stats object which describes the sampling rate (note that you can make things work without this last step, but then you don’t have a proper time axis to do temporal calculations like timestepping, filtering, etc and things are much more likely to go wrong).
* Finally, use Trace to combine the temporal data with the spatial data.

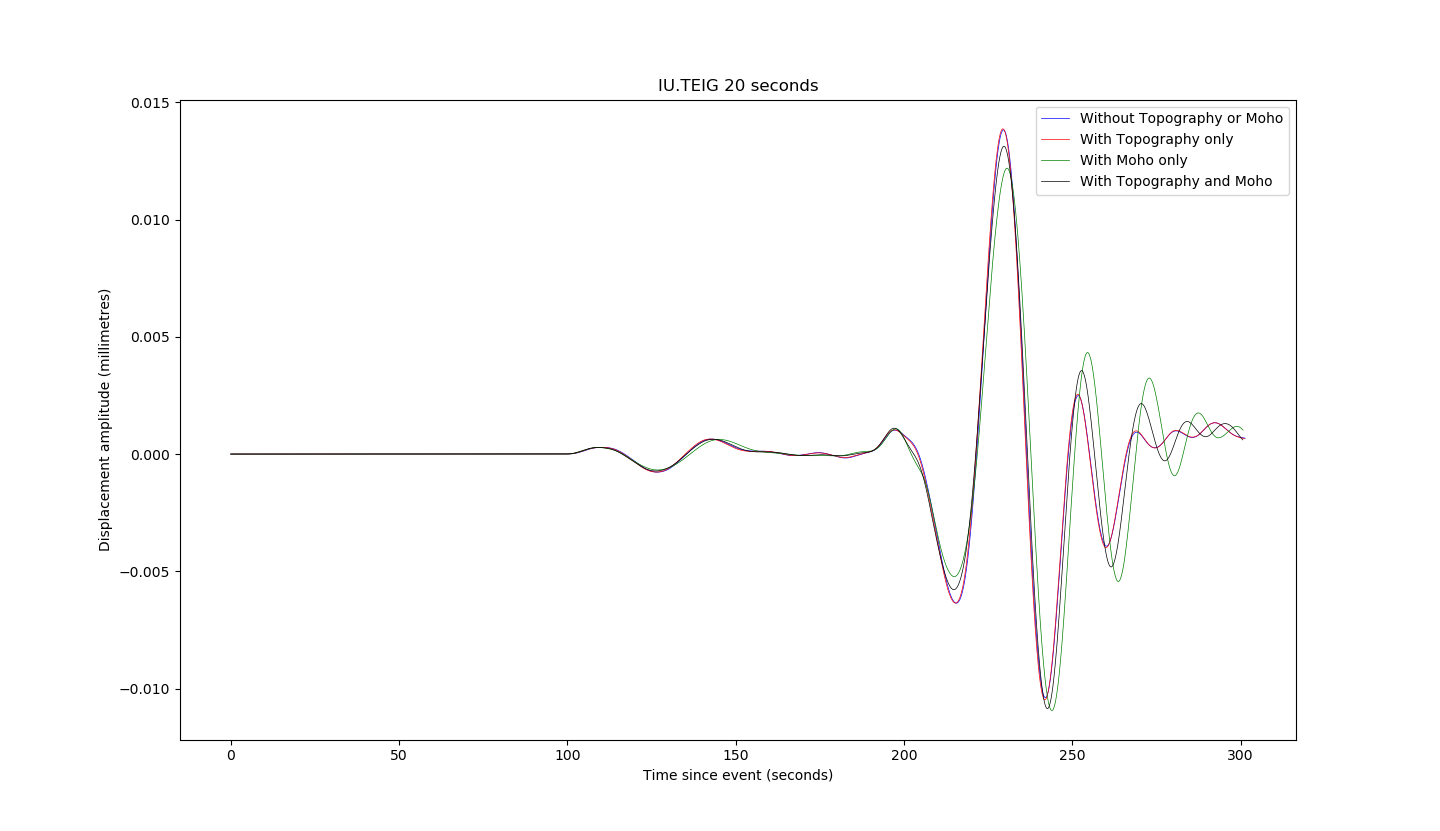
If you then just type the name of the trace, e.g. axisem3d\_trace, it should return a Trace object which includes both the seismogram and the relevant displacements. You can combine multiple Traces into a single Stream if you so wish, but I generally don’t bother.

13.4 Plotting seismograms

Once you’ve created a Trace object, plotting the seismogram is trivial. You can either use the built-in method that Trace objects have, but I prefer to use matplotlib.pyplot as you get much more functionality and user-control.

To do this, inside the plt.plot command you just reference the Trace’s time points and data points separately, using trace.times() and trace.data. Note that the data points are not called (i.e. no parentheses) but the time points are.

You can edit the figures using the same functionality you would for any other matplotlib plot, for example by setting labels, axes, linestyles, etc. You can also write relatively simple routines for doing things like plotting record sections (codes not included for simplicity - email me if you want them).



**Fig. 3:** Some exemplar seismograms. Note that the Moho undulations had to be smoothed to 10% of their original amplitude to make this run work, but some differences are still appreciable.

13.5 Filtering and convolution

If you used a delta function source, you need to either filter or convolve the Green’s function to return a result that is free of ‘noise’ at above the mesh frequency. It’s worth thinking about the subtle differences between ObsPy’s filter routines and SciPy’s convolve ones.

In the broadest sense, these are of course identical, as filtering is equivalent to a multiplication with a kernel in the frequency domain, as is convolution. However, the way in which these are implemented can vary, and you also need to think about how convolution ends up being more complex in a numerical sense (with finite windows) than it is in the simplest, infinite-domain analytical sense.

*13.5.1 Butterworth bandpass filters*

The simplest kind of filter that you might want to use is a Butterworth-bandpass filter, whose functionality is included in ObsPy at obspy.signal.filter.bandpass. This is as close to a ‘rectangular’ filter as you’ll want to get, for technical reasons simply removing all frequencies not between your two limits tends to give you numerical artefacts. A Butterworth-bandpass filter has steep, but not vertical sides. You can also choose the filter order (corners), and need to specify (default four) corner frequencies. In certain cases I’ve also found that running the filter both ways across the data (zerophase=True) can help.

For synthetic data, which effectively contain no noise other than that above the mesh period, the Butterworth-bandpass filter is a good choice. You can cut out the high-frequency noise by simply setting freqmax to the inverse of the mesh period, and you can remove low-frequency energy below 100s by setting freqmin to 0.01. This is worth doing because strictly speaking, frequencies below 100s (i.e. 0.01Hz) are not simulated properly in AxiSEM3D because we have no ellipticity, rotation, or gravity.

If you were using actual data, with a non-white noise content, another filter might be more appropriate: for example a log-gabor filter, which doesn’t come installed in ObsPy (I can also send it to you though).

*13.5.2. Convolution with a Gaussian function*

Convolution with a Gaussian using SciPy ends up being a bit more challenging than you might think - on a qualitative level, it’s just equivalent to swapping the frequency-domain representation of the Butterworth-bandpass filter for the frequency-domain representation of a Gaussian, which is also conveniently another Gaussian.

However, if you use scipy.signal.convolve, you need to make sure that the convolution parameters of the Gaussian (or indeed whatever other source-time function with appropriate frequency content you might wish to use) are set correctly. This includes matching the sampling rates (you can use ObsPy’s Lanczos interpolation to do this), and ensuring that the convolution mode is set to valid - to avoid padding out the seismogram with fake data points at either end.

*13.5.3 Other operations and commutivity*

In an analytical sense, all these operations should commute and it should not matter which order you do them in. In a numerical sense, this does not always appear to be the case (though this may just be due to poor processing on my part). So if something doesn’t look right, try playing around with the order of operations.

I’ve also found that applying a taper to the seismograms can help, as if the end of a Green’s function trace includes a large non-zero displacement when the simulation ends, sometimes the convolution can produce a seismogram with an overall non-zero slope (i.e. trending to negative values over time), which is of course nonsensical.

**14 Project status**

14.1 What’s included

If you are interested in this document from the point of view of investigating the simulation of subduction interfaces at slow-slip locations, it’s worth covering where we are at the current time. Thus far, I have created (and include in this distribution)

* 3D geometric crustal models for the surface topography and Moho topography, based on ETOP01 and Crust 1.0 respectively
* The python scripts needed to read in the raw files (a gridded NetCDF file and a .xyz file respectively)
* Python functions included in these scripts to write the above 3D model files into a NetCDF file structured in a way that AxiSEM3D can read it, with options for resampling, filtering, and smoothing
* Input files for the ‘maximum functionality’ scenario related to the above, i.e. including both surface and Moho topography on a global scale
* This manual! Which is distributed in PDF format but is available as Google Doc for anyone who would like to contribute.

14.2 What’s not included

Unfortunately, I have not actually been able to make the ‘maximum functionality’ scenario work properly, largely because the combination of surface and Moho topography appears to be causing an instability at periods above 50s; whilst the Moho alone appears to fail at around 20s.

The error message indicates that this is due to the Jacobian being negative, I am not surprised to see this error (which likely indicates that the mesh is over-deformed). However, I must admit figuring out how to solve it is beyond me, as only with extreme smoothing (all Moho topography reduced in amplitude by 90%, for example) does the issue appear to be resolved.

It is also not clear to me whether moving to higher frequencies (i.e. more like 5s) would actually make the problem worse or not: on the one hand, the deformation is being accommodated by more elements in the mesh, but on the other hand stronger gradients are being sampled which may prove difficult to interpolate.

*On this note, I am really rather out of ideas as to what to try next.*

**15 Outlook**

For the sake of future users, it is worth being frank about the functionality included here and what is likely to change over time. As of the time of writing (December 2021), AxiSEM3D is no longer being developed, at least not on any systematic and planned level. This means that the above issues might well not be rectified. Of course, this is subject to change as researchers come and go, and perhaps someone else will pick up this approach, but it is worth bearing in mind that there is limited computational support available for debugging at the present time.