II. AxiSEM3D – Installation guide

Benjamin Fernando[1,*], Jonathan Wolf[2], Kuangdai Leng[3], Tarje Nissen-Meyer[4], Will Eaton[5], Andrew Walker[7], Tim Craig[6], Jack Muir[7], Ceri Nunn[8], Maureen D. Long[2]

Abstract

*Corresponding author

Email address: `bfernan9@jh.edu` (Benjamin Fernando)

[1]Department of Earth and Planetary Sciences, Johns Hopkins University, Baltimore, MD, USA

[2]Department of Earth and Planetary Sciences, Yale University, New Haven, CT, USA

[3]Rutherford Appleton Laboratories, UK

[4]Department of Mathematics and Statistics, University of Exeter, UK

[5]Department of Geosciences, Princeton University, NJ, USA

[6]School of Earth and Environment, University of Leeds, UK

[7]Department of Earth Sciences, University of Oxford, UK

[8]NASA Jet Propulsion Laboratory, California Institute of Technology, CA, USA

This document was designed following a research project with Dr Tim Craig at the University of Leeds. It describes installation of AxiSEM3D on common high-performance computing architectures.

This document should be read after you have perused the relevant papers, and the ‘I. AxiSEM3D – User guide’ document. There are additional resources available in the source code comments, and on the GitHub wiki.

This document is designed to assist in **installing** and **running** AxiSEM3D and associated codes, rather **using** them.

How to cite the AxiSEM3D manual:

---

1. Introduction

1.1. Languages

The AxiSEM3D code (Leng et al., 2016; 2019) is written in the **C++** programming language. Various sub-routines involve Fortran, and some of the pre- and post-processing codes are supplied in Python or Matlab. Input files are human-readable text files, and no particular programming languages are required to understand and use them.

The casual user will not need to edit the source code, and as such, no familiarity with C++ is required. You may see references elsewhere to the 'old' AxiSEM code (Nissen-Meyer et al., 2014), which is written in Fortran. You do not need to be familiar with either old AxiSEM or Fortran to run AxiSEM3D.

1.2. Understanding inputs and outputs

The file formats associated with inputs and outputs are (at least compared to the rest of the code) simple. Unless you choose otherwise, you should be able to read these as plain text.

For viewing and editing input files, you may wish to consider using an editor such as **VSCode** or **Kate** *according to Jonathan, Kate is amazing; no one else has ever heard of her*, which will render the text in the files in an easier-to-read way (for example, keys and values in different colours). Else, readers such as vim or gedit are also fine.

At the simplest level, you can save outputs as ASCII files and plot them using something as simple as Excel. As discussed in the user guide, non-human readable formats are more space efficient, but do require some basic familiarity with Python or Matlab (or equivalent) to read and plot.

If you are not biased either way, we suggest doing post-processing in **Python**, since you can then make use of the enormous functionality offered by the **ObsPy** (Beyreuther et al., 2010) package.

1.3. AxiSEM3D architecture

Upon downloading the source code from Github (more on this later), it is worth taking a brief look at it, even if you do not plan on doing any coding in C++ yourself.

You will see that the main body of the code is divided into two sections: the **core** and the **preloop**.

1.3.1. The Preloop

The preloop involves everything that only needs to be done once in a simulation before it starts – e.g., dividing up the mesh between different nodes. The code will also perform a series of checks in the preloop, for example checking that the Jacobian for each element (used to translate between the physical configuration and the reference configuration used for numerical integration) is positive.

1.3.2. The Core

The core of the code contains the routines for executing the main time loop, that is, solving the equations of motion numerically at each timestep.

AxiSEM3D is designed to be highly parallel, meaning that the workload in the time loop is efficiently divided between a number of different computational nodes and their constituent processors. The communication overheads between different nodes, and the time required to assemble the whole solution at the end of the simulation, are designed to be small compared to the overall runtime of the code.

1.4. Computing architectures

The vocabulary surrounding computing architectures can be difficult to get to grips with at first. Here, we will briefly discuss the relationship between processors and nodes, as applied to AxiSEM3D.

1.4.1. On your machine

When running an AxiSEM3D simulation on a local machine (e.g., your laptop), you generally only need to set the number of **processes** that you want to run. In short, a process is a parallel ‘thread’ of operations that is executed simultaneously (‘in parallel’) to other processes. Most of the time, this should be the same as the number of processors that your machine has, though some machines support more advanced operations such as hyperthreading. Generally, creating more processes than you have processors starts to reduce efficiency again.

1.4.2. On an HPC architecture

On **high-performance computing** (HPC) architectures, i.e. **clusters and supercomputers**, you will probably need to be a bit more specific about how you want the code to execute. In general, you will need to specify the number of nodes, the number of processors, and the runtime.

Most HPC systems have a standard number of processors per node, say 64 or 128. This means that you can have up to this many independent threads running on a single shared-memory unit. Each node will also have a fixed amount of memory (around 256 or 512 GB often) which all the processors will have to share between them.

Sometimes, you will also have the option to request **high-memory nodes**, which can be more efficient if you are doing memory-intensive computations (specifically, things like wavefield visualisation which have a heavy input/output load). These high-memory nodes can let you run a higher number of processors-per-node without getting an Out Of Memory (OOM) error than the equivalent standard nodes; unused processors on a node are left dormant and hence the simulation is less efficient if this occurs.

Note that most HPC systems have separate **login and compute nodes**. It is important to make sure that you only run AxiSEM3D on the compute nodes, as these have the required power and memory to execute simulations. Running on the login nodes will likely result in error from a lack of permissions, slowing down the machine for all users, or even crashing it.

2. Getting the AxiSEM3D code

In this section, we will try to be as thorough as possible about how to download, install, and eventually use it. If you find this explanation too basic feel free to skip ahead, we are just trying to be as thorough as possible.

2.1. System requirements

The current system requirements for installing AxiSEM3D can be found here: `https://github.com/AxiSEMunity/AxiSEM3D/wiki/installation`. They probably look something like this (we do not include the version numbers here, because they are liable to change and the Github is updated more often than this manual!)

- A Unix-like operation system (Linux, MacOS). If you are using Windows, you might need a Unix dual-boot, etc. Your university or institution may have Linux systems available too,
- A C++ compiler supporting C++17 standard and cmake,
- Package management tools, conda (Anaconda or Miniconda) is ideal but you can use whatever you like,
- wget and git,
- The message parsing interface MPI (a serial build can be made but it is not really useful - there is little point in running AxiSEM3D in serial except for developer tests).

2.2. The mesher

This page (`https://github.com/AxiSEMunity/AxiSEM3D/wiki/installation`) also includes the installation instructions for the mesher (**Salvus Mesh Lite**), which is a stand-alone package that is not distributed along with the rest of the AxiSEM3D distribution.

There are more specific instructions on using the mesher on this page: `https://github.com/AxiSEMunity/AxiSEM3D/wiki/mesher`. These instructions were written by the same group as this manual, so we do not repeat them here.

You can find more information about the Salvus family of packages here `https://gitlab.com/swp_ethz/public/SalvusMeshLite`.

2.3. Dependencies

AxiSEM3D relies on a number of external packages. These generally execute specific tasks within the code (e.g. the Fourier Transform) that there is no need for AxiSEM3D to have a bespoke version of.

Unfortunately, AxiSEM3D dependencies can be a little tricky to install. The current list of dependencies is given in Table 1.

3. Installing AxiSEM3D

AxiSEM3D is a compiled code. This means that it is not enough just to download it into one place, you must also build and compile it. This will

Package	**Role**	**Version**
Eigen	Linear algebra	3.3.9
Boost	C++ Routines	1.73.0
FFTW	Fast Fourier transforms (single and double precision needed)	3.3.8
Metis	Mesh partitioning	5.1.0
NetCDF	Input/output. Parallel build useful but not strictly necessary for basic runs.	4.4.1

Table 1: Dependencies for AxiSEM3D and current version suitable for AxiSEM3D. Check the Wiki page for the latest updates to the list of dependencies!

produce a **binary**, and the binary is what you actually 'run' to produce simulations.

3.1. Creating a Conda environment

As discussed earlier, you can use any package management program that you want. We suggest using **Conda** (either the full Anaconda distribution, or the lite Miniconda distribution), and these instructions assume you are doing the this.

The first step is to create a new Conda environment. This is a key stage that allows you to help manage dependency clashes, and so on.

```
conda create -n axisem3d python=3.8 eigen boost fftw metis netcdf4
```

Will create a conda environment called **axisem3d** which has the above packages installed.

Each time you open a new shell or log in anew to the HPC, you will need to activate the environment (if you do not do this, you will often see it suggested that certain packages are missing). Type:

```
conda activate axisem3d
```
.

3.1.1. Troubleshooting

Ideally, this stage will have worked without error. In reality, it is probably struggling to find one or more modules. There are then three options, which we suggest doing in this order:

1. Use Conda to install the package if it does not already seem to be there,

2. Point to the package's location in later stages (assuming it is actually installed, as checked by typing something like module avail XXX),
3. Ask your HPC admin for help.

3.2. Downloading from Github

AxiSEM3D is distributed via Github: `https://github.com/AxiSEMunity/AxiSEM3D`. You will see that there are multiple branches of the code, because different people are working on developing different aspects of it. You should probably stick to the **Master** branch in the first instance.

If you need to, you can download from this repository and move the folders to the location that you want to install AxiSEM3D in, but the easiest thing to do is to use **git clone**.

The advantage of using git is that you can easily update the code when changes are pushed to the master branch; and if you end up making edits to the source code you can create a pull request to merge them back as proposed changes to the Master branch as well.

From the command line, navigate to the location you want the code to live in, and type:

```
mkdir -p axisem3d_root && cd $_
[ ! -d ./AxiSEM3D ] && \
git clone https://github.com/AxiSEMunity/AxiSEM3D.git AxiSEM3D
git -C AxiSEM3D pull
```

The first line creates a root directory, and a sub-folder called AxiSEM3D if it does not yet exist. It then clones the code into that location.

3.3. Building the Code

The next stage is to **build** the code. Because the build folder gets emptied every time that you make the code, don't put anything else in there. Type:

```
mkdir -p build && cd $_
```

3.4. Configuration

In this stage, we tell the code where all of its dependencies are. This stage can also be a little fraught, so follow the instructions carefully. Type:

```
export conda_path=$(dirname $(dirname $(which conda)))/envs/axisem3d
```

This line sets a variable called **conda path** which is where all of the packages that we loaded via Conda can be found.

Then, we remove everything in the build folder and tell cmake that each of the dependencies can be found at the Conda path. Type:

```
rm -rf ./* && cmake -Dcxx=mpicxx \
-Deigen=$conda_path \
-Dboost=$conda_path \
-Dfftw=$conda_path \
-Dmetis=$conda_path \
-Dnetcdf=$conda_path \
../AxiSEM3D/SOLVER/
```

The first line is a **cmake** command that tells the code to use the **mpicxx** compiler wrapper. The last line tells the code that the actual source code is located in the **SOLVER** folder of AxiSEM3D.

If this did not work (probably because you could not load the dependencies through Conda), you will have to set the paths differently. To do this, change the **conda path** variable in the relevant line. For example, if you know that the Boost library can be found at path **\users\libraries\boost**, you would set this in place of the Conda path, e.g.

```
-Dboost=\users\libraries\boost
```

You might need to adjust where in that folder you are pointing, e.g., it could be the **lib** or **bin** sub-folders.

If, in the long run, you want to save time and know that the Conda path will not work for certain dependencies, you can hard-code in the **SOLVER \CMakeLists.txt**. There are other options that you can set at this point too, e.g., the polynomial order of the solver or the precision (double/single) of the solver. Unless you have a very specific reason (e.g., you are using the solar example in the examples list, and need to mesh huge velocity contrasts), we would not change either of these.

There are example configuration scripts for commonly used HPC architectures on the main GitHub page. If you write your own, feel free to push it there!

3.5. Making the code

This should be the last step before you run the code! You now need to finally make it – this will link together all the different pieces that you have told the code about. To do this, type make -j. You can also add a number after the j (no space) to specify the number of threads to do this on, if you need to. This will speed it up a bit.

Ideally, this should work and produce a binary executable called **./axisem3d**.

4. Testing the code

4.1. Understanding the executable

It is worth briefly touching on what this executable does. If you are not familiar with compiled codes, you can think of it as a command that launches a program (the program being AxiSEM3D). The executable "knows" what's in the source code, but it does not "know" what is in the input files yet.

What we mean by that is that the executable will start to run regardless of whether there is an input file or not, but it will error out if the input file is missing.

4.2. Running the executable

Like most executables, you can type **./axisem3d** directly in the command line, and it should work.

If you get an error saying that you do not have permission, you may need to change the permissions by typing chmod +x ./axisem3d first.

On your local machine, just typing **./axisem3d** should start the program running on a single processor. This is not particularly efficient, but is sufficient for a first test.

4.2.1. Using MPIrun

In order to take advantage of the parallelisation of the code, we need to make use of additional functionality. The **MPI** dependency that you installed or used in the build stage earlier is the key to this.

MPI, or the message-parsing interface, is a package which lets you split the computational load up across different processors. Whilst you do not need to worry about the details, this is not a trivial thing to do – the mesh needs to be chopped up, there are boundaries between different sub-domains, the solution needs to be re-assembled at the end, etc. MPI handles the actual communication between different processes.

To run the process on multiple threads, we just type `mpirun -np X /axisem3d`; where X is the number of processes that you want to run on.

4.3. Running on HPC

On HPC machines, things might work a little differently. On most machines, you should be able to log in and type **`./axisem3d`** on the command line, and the program should start running.

Although this is an acceptable way to make sure that the binary at least works, it is an inappropriate way of running simulations because it makes use of the login nodes on the HPC, rather than the compute nodes. If you start it running and it works, you should cancel it via Ctrl + C and move on to the next stage.

4.3.1. Testing in parallel

Most HPC systems do not allow you to just use resources at will. We assume that you have an account on the system that you are using, and that there are some resources attached to that account.

In order to run in parallel, you need to request a certain number of nodes, and a runtime. On most machines, you also need to tell the system how many processors you want to run on each node. Remember, the total memory of each node is fixed and must be partitioned between all processors, so for testing purposes it is sensible to set the number of computing processors-per-node to the actual number of processors-per-node, as any leftover processors cannot be assigned to other users and are just wasted.

4.3.2. Interactive mode

The best way to test and debug the code on HPC systems is in **interactive mode**. Interactive mode is different to **batch/schedule mode**. In the former, you tell the system that you need X nodes for Y minutes (a sensible test case might be 4 nodes, each with 128 processors for example, for 10 minutes at a time). The system will generally allocate them to you immediately, or after a very short delay.

In the latter, you do the same, but the request can normally be larger (sometimes a lot larger - more on this later). The delay for this many processors to become available can be significant, often hours to days on busy machines for large jobs. So, testing in interactive mode is much faster.

Once you have been assigned your interactive mode session, you will probably need to re-activate your Conda environment, etc. You can do this in the same way that you did it before. You do not need to re-build the code, but if you loaded any modules 'by hand', i.e. using something like module load fftw in the command line you will need to do this again. To save time, we normally use a configuration script to do this (more on this later).

Then, to test in interactive mode, you will need to use **mpirun**, or sometimes an alternative called **aprun**, to run in parallel.

You can do this by typing **mpirun -np X ./axisem3d**, or **aprun -n X ./axisem3d**. Note that X is the number of processes to spawn (in this case, equal to the number of processors that the code will run on), not the number of nodes. The most efficient solution is to set X to the number of processors-per-node multiplied by the number of nodes that you requested in the interactive session.

4.4. Success!

Ideally, you should see something that tells you that the code has begun running (this is very obvious - it will print the words AxiSEM3D ASCII-art. If you do not see this, go to the debugging stages as discussed below.

If you are running one of the default test cases (e.g. PREM for 3600 s at 50 s resolution), it should run pretty quickly. If it is something more complicated, it might error out because you have run out of time (or memory), but that is okay for these testing purposes. If you want to learn more about **outputs**, see the user guide.

5. Configuration scripts

As mentioned above, you will need to do certain steps every time that you log into the HPC from a new session. These steps include things like activating your Conda environment, and so on.

The simplest way to do this is to create a text file which includes all of the steps that you need to undertake, turn it into a script, and run it each time to you log in (if you want to, you could even add it to your **.\bashprofile** and have it execute automatically so that you do not have to worry about it).

We include some exemplar configuration scripts in on the AxiSEM Github page.

6. Submission Scripts

As mentioned, useful-scale simulations in AxiSEM3D need to be run in batch (or scheduled) mode. The batch submission system on each HPC is slightly different, but is typically something like **PBS** or **slurm**.

Because the submission scripts are different on each system, we include a sampling on the Github page. You can adapt the scripts as needed, in general you need to include the usual things like the number of nodes, number of processors-per-node, the maximum job runtime, and the resource budget code. Some systems may also let you set the priority of your runs (which can be very useful when running big simulations, if you set them as the cheapest priority and run them over the weekend!).

7. Debugging

Debugging an installation can be difficult. In this section, we will discuss debugging of a failed installation, build, configure, or make - i.e. not an error that is related to the science that AxiSEM3D is simulating. For more on simulation debugging, see the user guide. *Let's edit this later - once we have tested the install on the latest machine updates and know what likely issues are.*

8. Installations on different HPCs

8.1. ARCHER2 and Anvil

Exemplar configuration scripts for the UK National Supercomputer ARCHER2 and the NSF-funded Anvil HPC at Purdue are included as top-level files in the AxiSEM3D GitHub.

8.2. GRACE cluster at Yale

AxiSEM3D is installed on GRACE as a module. Simply load it via `module load AxiSEM3D/2022Sep13-iomkl-2020b-avx512`. To run it, just use `mpirun axisem3d`.

References

Beyreuther, M., Barsch, R., Krischer, L., Megies, T., Behr, Y., Wassermann, J., 2010. ObsPy: A Python Toolbox for Seismology. Seismological Research Letters 81, 530–533. URL: `https://doi.org/10.1111/10.1785/gssrl.81.3.530`.

Leng, K., Nissen-Meyer, T., van Driel, M., 2016. Efficient global wave propagation adapted to 3-D structural complexity: a pseudospectral/spectral-element approach. Geophysical Journal International 207, 1700–1721. URL: `https://doi.org/10.1093/gji/ggw363`.

Leng, K., Nissen-Meyer, T., van Driel, M., Hosseini, K., Al-Attar, D., 2019. AxiSEM3D: broad-band seismic wavefields in 3-D global earth models with undulating discontinuities. Geophysical Journal International 217, 2125–2146. URL: `https://doi.org/10.1093/gji/ggz092`.

Nissen-Meyer, T., van Driel, M., Stähler, S.C., Hosseini, K., Hempel, S., Auer, L., Colombi, A., Fournier, A., 2014. AxiSEM: broadband 3-D seismic wavefields in axisymmetric media. Solid Earth 5, 425–445. URL: `https://doi.org/10.5194/se-5-425-2014`.