HMCMT2D:

An open-source package for 2D Bayesian MT inversion and uncertainty quantification using the Hamiltonian Monte Carlo method

Ronghua Peng, Bo Han
School of Geophysics and Geomatics
China University of Geosciences, Wuhan

Version 1.0

April, 2024

Contents

1 Introduction

This document describes how to use the Julia[1] software package **HMCMT2D** perform Hamiltonian Monte Carlo (HMC) inversion for 2D MT data. Here's a list of features currently supported in **HMCMT2D**:

- Statistical analysis of posterior probability distribution about model parameters (e.g., mean, median, and mode)
- Parallel HMC samplings

[1] http://julialang.org/

2 Installation

2.1 Installation of Julia

The package is compatible with Julia v1.0 or later versions (Current version v1.10.* is recommended).

2.1.1 Windows systems

Go to the Julia download page to download the Windows command line version (.exe) and install it.

2.1.2 Linux systems

Although Julia is a cross-platform language, we strongly recommend to run **HMCMT** under Linux rather than Windows. This is because some of the third-party packages utilized by **HMCMT** such as **Dipole1D** are more straightforward to compile under Linux. There are three ways to install Julia on Linux:

- **Using precompiled binaries (recommended)**. Go to the Julia download page to download the generic Linux binaries (.tar.gz file). Then make sure that the Julia executable is visible for your system. To do this, first extract the .tar.gz file to a folder on your computer. Then you can either add Julias bin folder to your system PATH environment variable, or create a symbolic link to julia inside a folder which is on your system PATH, for example, by using the following command:

  ```
  sudo ln -s /path/to/julia/bin/julia /usr/local/bin/julia
  ```

- **Compiling from source**. Assume that Git has been installed already, then we can grab the Julia sources from GitHub by using the following command:

  ```
  git clone git://github.com/JuliaLang/julia.git
  ```

  This will download the Julia source code into a julia directory in the current folder. The Julia building process needs the GNU compilation tools g++, gfortran, and m4, so make sure

that you have installed them. Now go to the Julia folder and start the compilation process as follows:

```
cd julia
make
```

- **Using PPA for Ubuntu Linux**. Particularly, for Ubuntu systems (Version 12.04 or later), there is a Personal Package Archive (PPA) for Julia that makes the installation painless. All you need to do to get the stable version is to issue the following commands in a terminal session:

```
sudo add-apt-repository ppa:staticfloat/juliareleases
sudo add-apt-repository ppa:staticfloat/julia-deps
sudo apt-get update
sudo apt-get install julia
```

After a successful installation, Julia can be started by double-clicking the Julia executable (on Windows) or typing "`julia`" from the command line (on Linux). Following is an illustration of Julia's command line environment (the so-called REPL):

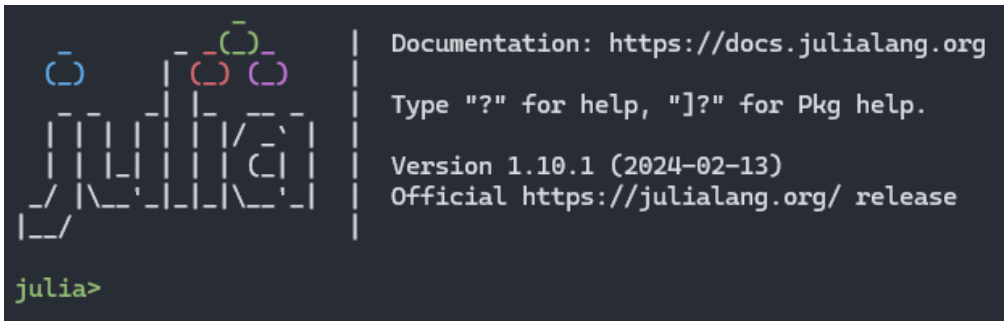

Figure 1: An illustration of Julia REPL.

2.2 Installation of the HMCMT2D package

2.2.1 Setting up the package environment

HMCMT2D depends on several external packages (the so-called dependencies) which are not shipped with the package .zip file. These dependencies can be automatically resolved by activating and instantiating the package environment through Julia's package manager (Pkg). Suppose that the HMCMT package is placed at `home/username/code` on Linux or at `D:\code` on Windows, you can type the following command from the Julia REPL to go to the package directory:

```
julia> cd("/home/username/code/HMCMT")
```

on Linux, or

```
julia> cd("D:\\code\\HMCMT")
```

on Windows. Then press] from the Julia REPL you will enter the Pkg REPL which looks like

```
(v1.10) pkg>
```

indicating that you are currently in the environment named "v1.10", Julia 1.10's default environment. To switch to the package environment, just `activate` the current directory:

```
(v1.10) pkg> activate .
```

you will get:

```
(HMCMT) pkg>
```

indicating that you are in the environment "HMCMT". The environment will not be well-configured until you `instantiate` it:

```
(HMCMT) pkg> instantiate
```

By doing so the dependencies listed in `Project.toml` and `Manifest.toml` can be automatically downloaded and installed.
To get back to Julia REPL from Pkg REPL, press `backspace` or `^C`. Then you need to add these packages to the current environment by typing the following command:

```
julia>
]
(@v1.10) pkg> add BenchmarkTools BinDeps DistributedArrays
Distributions Primes Revise Distributed LinearAlgebra Printf
Random SparseArrays Statistics Test
```

2.2.2 Special Note on MUMPS Integration

Due to the specialized use of the MUMPS package within our environment and the official version's limitations, particularly on Windows systems, we provide a custom-compiled MUMPS package. This ensures compatibility and enhances performance for our users.

- **Custom MUMPS Package:** Initially based on an early version of the MUMPS package for Julia, we have incorporated the latest enhancements into our provided package to ensure seamless integration and superior performance.
- **Windows Compatibility:** Recognizing the challenges of running MUMPS on Windows, we have compiled a Windows-specific version of MUMPS.dll using MSYS2 and gfortran. This enables Windows users to enjoy the same level of efficiency and reliability as their Linux counterparts.
- **Linux Compatibility:** For Linux users, we also provide a Linux-specific version of MUMPS, ensuring compatibility and optimal performance across all platforms. By using our custom MUMPS package, users should install Intel oneAPI MKL to ensure compatibility with the MUMPS package.
- **MUMPS Binaries:** All MUMPS binaries (Linux and Windows) are included in the **./lib** directory, and users can easily integrate them into their Julia environment to leverage the full capabilities of MUMPS.

3 Running the code

3.1 Running single HMC sampling

- **First**, you need to let the **HMCMT** package to be "loaded" by the current Julia environment. This is done by adding the parent directory of the package directory to `LOAD_PATH`, a global environment variable of Julia. For example, the HMCMT package is placed at `home/username/code` on Linux or at `D:\\code` on Windows, then type the following command from the Julia REPL:

  ```
  julia> push!(LOAD_PATH,"/home/username/code")
  ```

  on Linux, or

  ```
  julia> push!(LOAD_PATH,"D:\\code")
  ```

  on Windows.

- **Then**, Then go to the package directory, for example: `cd /home/username/code/HMCMT`, and activate the **HMCMT** environment (please refer to the section **"Setting up the package environment"**).

- **Finally**, go to the directory where the running script loaded, and run the script by typing the following command (for example) from the Julia REPL:

  ```
  julia> include("runHMCScript.jl")
  ```

3.2 Running parallel HMC sampling

To perform parallel HMC sampling, call the parallel HMC sampling function **parallelHMCsampling** instead of the single HMC sampling **runHMC** (please refer to the `paraHMCScript.jl` scripts within the `examples` directory).

- **First**, you need to launch multiple worker processes by either starting Julia like:

  ```
  shell> julia -p 4
  ```

or adding processes within Julia (recommended) like:

```
julia> addprocs(4)
```

- **Then** you need to let the **HMCMT** package to be "loaded" on all processes by following command from the Julia REPL:

```
julia> @everywhere push!(LOAD_PATH,"/home/username/code")
```

- **Finally**, go to the directory where the running script loaded, and run the script by typing the following command (for example) from the Julia REPL:

```
julia> include("paraHMCScript.jl")
```

3.3 Writing Your Own Running Script

- In the **./examples** directory, we have supplied well-documented scripts for both serial and parallel HMC inversion process, named **runHMCScript.jl** or **paraHMCScript.jl**, respectively. These scripts serve as a foundation upon which users can build. We encourage users to customize these scripts according to their unique research needs, promoting both flexibility and adaptability across a range of geophysical inversion and uncertainty quantification projects.

- Adhering to these guidelines and leveraging the custom resources we provide will enable users to fully harness the capabilities of HMCMT for their specific geophysical inversion and uncertainty quantification endeavors.

4 Files required by the package

There are three files required to run the **HMCMT** package: the datafile, modelfile and the startup file. Each file is described in detail in the following sections.

4.1 MT data file format

The data file describes survey information. For current version, the file are designed for MT surveys. The MT data file contains information about MT measurements. Following is an example of MT data file:

```
Format:              MT2DData_1.0
#Description:       Data file generated by MATLAB at 29-Sep-2022 23:37:27
#Comments:          COPROD2 field data
Receiver Location (m):    20
#     Y     Z
        0.00         0.00
    12600.00         0.00
...

   209700.00         0.00
   230800.00         0.00
Frequencies (Hz):    12
    1.09866e-03
    2.92997e-03
...

    3.59971e+01
    9.59693e+01
DataType:  Impedance
DataComp:     2
ZXY
```

```
ZYX
Data Block:     470
# FreqNo.  RxNo.   DataType    RealValue     ImagValue      Error
    1       1        1    4.301004e-04    3.124862e-04    6.692749e-05
    1       1        2   -5.675216e-04   -4.795928e-04    6.496247e-05
    1       2        1    5.215230e-04    3.570942e-04    3.547178e-05
    1       2        2   -5.317268e-04   -4.541376e-04    2.381504e-05
...

    12      18        1    5.888307e-02    8.164339e-02    3.342972e-03
    12      18        2   -5.143789e-02   -7.158342e-02    1.318502e-02
```

The following is an brief explanation of the keywords of the MT data file.

- **Receiver Location** block lists the number of receivers and then their y, z (meters) locations.
- **Frequencies**: followed by the number of frequencies, and the frequency values (in Hz) are listed below it.
- **DataType**: specify the data types used for inversion. For 2D MT inversion, the allowed **DataType**s are the MT impedance components ZXY and ZYX.
- **Data Block**: followed by the number of data points, the maximum value of which is $N_{freq} * N_{type} * N_{Receiver}$. A table of data parameters is given below it. The first column of the data table is the frequency index for each datum, the second column gives the data type index. The last two columns give the corresponding data value and associated standard error.

4.2 MT model file format

The start model file used for define the model mesh and grid sizes in Cartesian coordinate system. And the initial conductivity of each model parameters can be given any random number. Here is an example model file:

```
#Format:            EM2DModelFile
```

```
#Description:      Model file generated by MATLAB at 02-Oct-2022 10:21:44
#Comments:         The 2D model used
# cell size at y direction
Ny:   30
  100.00   100.00   100.00   100.00   100.00   100.00   100.00   100.00
  100.00   100.00   100.00   100.00   200.00   200.00   200.00   200.00
  ...

# cell size for air layer
NAIR: 8
  50.00   100.00   200.00   400.00   800.00   1600.00   3200.00   6400.00
# cell size at z direction excluding air and seawater layers
Nz:   30
  100.00   100.00   100.00   100.00   100.00   100.00   100.00   100.00
  100.00   100.00   100.00   100.00   200.00   200.00   200.00   200.00
  ...

Resistivity Type: Resistivity
Model Type: Linear
1.00000e-02    1.00000e-02    1.00000e-02    1.00000e-02    1.00000e-02
1.00000e-02    1.00000e-02    1.00000e-02    1.00000e-02    1.00000e-02
1.00000e-02    1.00000e-02    1.00000e-02    1.00000e-02    1.00000e-02
...

Origin (m):   1024000.00         0.00
```

- **Ny** block lists cell size in y direction
- **NAIR** block lists cell size for air layer.
- **Nz** block lists cell size in z direction excluding air layer.
- **Resistivity Type** specifies model parameter type in the model file.

- **conductivity** or **resistivity** can be chosen, it will be immediately converted into conductivity by the program.
- **Model Type** defines model parameter in linear or nature logarithmic space. It will be immediately converted into linear space. The following is the list of model parameter, note that model parameter along y direction changes fastes, then the z direction.
- **Origin** block specifies the origins of model used, which are the distances from southmost, westmost, and uppermost of the model, respectively. Note that the origin of model MUST be consistent with the origin of data.

4.3 Startup File

The startup file describes the options that control the HMC inversion. It is composed of a list of option keywords and corresponding values. Following is an example of the startup file:

```
datafile:        coprod2data.dat
modelfile:       coprod2.mod
burninsamples:   100
totalsamples:    10000
resistivity:    0.1 1e4 0.05
timeinterval:   0.015
timestep:       6 10
smoothparameter: 1.0
```

The following is an brief explanation of the keywords of the startup file.

- **datafile**: followed by a string that indicates the file name of the data file used in the inversion.
- **modelfile**: followed by a string that indicates the file name of the model file used in the inversion.
- **burninsamples**: number of samples for burn-in period used in the estimate of the posterior probability distribution.
- **totalsamples**: number of total samples the Markov chains will produce.

- **resistivity**: followed by three real values. The first two indicates the lower and upper bounds for layer resistivities (in Ωm), the last one is the standard deviation (in percent) when perturbing the value of the resistivity of each model parameter.
- **timeinterval**: specifies a uniform distribution range with two integer values, from which the number of integration steps L is sampled at each Hamiltonian Monte Carlo (HMC) step.
- **timestep**: a real value that specifies the size of each time step in the leapfrog integrator.
- **smoothparameter**: a real value that indicates the variance of the model gradient, serving as a regularization parameter to control model complexity.