libEMM - A 3D Controlled-source ElectroMagnetic Modeling library

Pengliang Yang
Harbin Institute of Technology, China
Email: ypl.2100@gmail.com

April 27, 2022

Abstract

This is the mannual for `libEMM` - a software to do 3D controlled-source electromagnetic (CSEM) modelling. Because diffusive Maxwell equation, (in original form has to be discretized in time-domain with stringent stability condition and a large number of time steps), has been transformed into wave domain using fictitious transform (Mittet, 2010), `libEMM` can therefore compute the frequency domain CSEM response extremely efficiently because multiple frequencies can be integrated on the fly during time stepping in one go. Thanks to the time-domain formulation, `libEMM` is capable to handle the modelling in the resistivity model of very large size, since the EM fields will be updated at each time step on the same memory unit. The mannual explains how to install, compile and run the code.

Contents

1 Software overview

1.1 Software dependencies

`libEMM` assumes that the following softwares have been installed and placed in a path recognizable by the `Makefile` (you might need to modify a bit to make it working properly).

- FFTW;
- MPI (OpenMPI or MPICH);
- CUDA programming environment if needed.

1.2 Structure

`libEMM` has a simple yet standard structure:

- `src`: the folder includes all source files in C and CUDA programming language (.c and .cu files);
- `include`: the folder includes all the header files (.h files);
- `doc`: the folder stores the mannual for `libEMM`;
- `bin`: the folder is used to store the executable after the compilation;
- `run_1d`: a 3D CSEM modelling example to compare the modelled solution with Dipole1D using a 1D layered resistivity model;
- `run_bathy_2d`: a 3D CSEM modelling example to compare the modelled solution with MARE2DEM using model of 2D geometry;
- `src_nugrid_homogenization`: the folder prepares the resistivity models, the nonuniform grid in x-, y- and z- directions for `run_1d` and `run_bathy_2d` in binary format.

2 Get the code and compile

One must install FFTW and MPI before start to use `libEMM`. On Ubuntu system, one can use

```
sudo apt-get install libfftw3-dev
```

to install FFTW, and then use

```
sudo apt-get install libopenmpi-dev
```

to install OpenMPI. Other MPI implementations such as OpenMPI are also good to use.

Installation of CUDA is optional. If you want to try GPU modelling, you are referred to Nvidia websites on installation issues `https://docs.nvidia.com/cuda/cuda-quick-start-guide/index.html`.

One can check out the code from github:

```
git clone git@github.com:yangpl/libEMM.git
```

You will find the software `libEMM`. Go to the folder `libEMM/src` by

```
cd libEMM;
cd src;
```

Then you can compile the code via

```
make
```

This command will compile the code using `mpicc`. You might need to edit the `Makefile` if the path of FFTW and MPI in your computer/cluster is not at default location. Then, the executable `fdtd` will be generated in the directory `/bin`.

Inside `/src`, to compile the code when CUDA has been installed properly, we type:

```
make GPU=1
```

Again, the executable `emf` will be generated in the directory `/bin`.

3 How to run

Run There is a running template to follow: `run_1d` which runs forward modeling of electromagnetic wave modeling and outputs the simulated EM data in frequency domain as ASCII files `emf_xxxx.txt`.

1. One may need to first prepare the input resistivity files `rho11`, `rho22`, `rho33` in `src_nugrid` by compiling the code and generting them (`make`; `./main`). Copy the generated files into `run_1d`.

2. In `run_1d`, we can create the acquisition files by compiling the code in Fortran:

   ```
   gfortran create_acquisition.f90 -o main;
   ./main
   ```

   One can edit the Fortran source code to create different survey layout sheet.

3. Using the above inputs, we create a shell script `run.sh` with relevant parameters in the following:

   ```
   n1=101
   n2=101
   n3=101
   d1=200 #100m
   d2=200 #100m
   d3=50 #100m


   export OMP_NUM_THREADS=8
   mpirun -n 1 ../bin/fdtd \
          mode=0 \
          fsrc=sources.txt \
          frec=receivers.txt \
          fsrcrec=src_rec_table.txt \
          frho11=rho11 \
          frho22=rho22 \
          frho33=rho33 \
          chsrc=Ex \
          chrec=Ex \
          x1min=-10000 x1max=10000 \
          x2min=-10000 x2max=10000 \
          x3min=0 x3max=5000 \
          n1=$n1 n2=$n2 n3=$n3 \
          d1=$d1 d2=$d2 d3=$d3 \
          nb=12 ne=6 \
          freqs=0.25,0.75,1.25 \
          rd=2
   ```

   One may modify the parameters such as the number of process in parallel.

All the parameters are placed in a text file `run_good_src_rec.sh`, which will be executed in shell by `bash run_good_src_r`

Plot Inside the folder, the ASCII sript `plot_survey_layout.gnu` will plot out the source-receiver acquisition geometry in x-y plane, based on the acquisition files `sources.txt`, `receivers.txt`.

After the modelling, we can plot out the CSEM data for visualization purposes. This is achieved by python script `plot_emdata.py`.

4 Suggestion to your own modelling jobs

Use the two running templates and modify the parameters and the codes to prepare acquisition files and input models, nonuniform grid!

5 Other important details

5.1 Design

The desgin of this `libc_emf` software has been heavily influenced by my personal experience working with `Seismic Unix` package, `Madagascar` package and the development of full waveform inversion code `TOYxDAC_Time` (the software I developed at SEISCOPE consortium in France).

- Since `Seismic Unix` has been used and validated by the geophysicists in more than 3 decades, I feel confident to borrow many routines for my own software development. I grouped some of the key functions in `Seismic Unix` into one file - `/src/cstd.c`, which shares the common macro definitions of some frequently used variables in the header file `/include/cstd.h`.
- I used the idea from `Madagascar` to develop code module, which allows the C code sharing some similarities to Fortran `module`. Many functions with suffix `xxx_init()` and `xxx_close()` serve as constructor and destructor, which makes the C module equivalent to C++ class.
- The parallelization of the code is organized using MPI in a shot/source independent manner, to maximize the scaling of the code to achieve best efficiency as possible. This idea is borrowed from `TOYxDAC_Time` software, based on the physical understanding of the acquisition settings. The downside of this parallelism is that I always assume the number of cores/processors available is larger than the actual number of sources deployed in the simulation. This can be a restriction for the resources in the company cluster where the cluster is too small. Also, the job queuing time might be long in case freely available number of cores is smaller than the number of sources. The use of reciprocal modeling allows to do simulation by switching the source and the receiver, thus helps to mitigate this issue because in practical CSEM acquisition the number of sources is usually much larger than the number of receivers.

5.2 Macros

The following macros are included in `cstd.h`.

- `ABS`: absolute value of the input value;
- `MAX`: the maximum value between two;
- `MIN`: the minimum value between two;
- `NINT`: the nearest integer of the input value;
- `SGN`: the sign of the input value;

These macros are very useful and can be widely used in the code development once the header is included.

5.3 Memory management

The file `alloc.c` provides the interfaces to dynamically allocate multidimensional arrays in C language for different types of data (`char`, `int`, `float`, `double` and `complex`). There are the most frequently used routines for the memory allocation to do scientific computation:

```
...
int *alloc1int (size_t n1);
int *realloc1int (int *v, size_t n1);
int **alloc2int (size_t n1, size_t n2);
int ***alloc3int (size_t n1, size_t n2, size_t n3);
int ****alloc4int (size_t n1, size_t n2, size_t n3, size_t n4);
int *****alloc5int (size_t n1, size_t n2, size_t n3, size_t n4, size_t n5);
...

float *alloc1float (size_t n1);
float *realloc1float (float *v, size_t n1);
float **alloc2float (size_t n1, size_t n2);
float ***alloc3float (size_t n1, size_t n2, size_t n3);
float ****alloc4float (size_t n1, size_t n2, size_t n3, size_t n4);
float *****alloc5float (size_t n1, size_t n2, size_t n3, size_t n4, size_t n5);
float ******alloc6float (size_t n1, size_t n2, size_t n3, size_t n4, size_t n5, size_t n6
...

double *alloc1double (size_t n1);
double *realloc1double (double *v, size_t n1);
double **alloc2double (size_t n1, size_t n2);
double ***alloc3double (size_t n1, size_t n2, size_t n3);
...

double complex *alloc1complex (size_t n1);
double complex *realloc1complex (double complex *v, size_t n1);
double complex **alloc2complex (size_t n1, size_t n2);
double complex ***alloc3complex (size_t n1, size_t n2, size_t n3);
double complex ****alloc4complex (size_t n1, size_t n2, size_t n3, size_t n4);
...
```

Keep in mind that these routines allocates consecutive memory units from the address of the first element `p[0][0][0]` to the last `p[n3-1][n2-1][n1-1]`, which gives the best possible computational efficiency. These not only makes memory allocation very convenient but also allows the use of square brackets for multidimensional arrays in a similar way as static arrays.

As an example, we can allocate a 3D array of size 100*200*300 and initialize it in the following:

```
int n1 = 100;
int n2 = 200;
int n3 = 300;
float ***p;
p = alloc3float(n1, n2, n3);
memset(p[0][0], 0, n1*n2*n3*sizeof(float));
```

Indexing an element of this 3D array is easy:

```
for(i3=0; i3<n3; i3++)
    for(i2=0; i2<n2; i2++)
        for(i1=0; i1<n1; i1++)
            p[i3][i2][i1] = ...;
```

To free the 3D array, we use:

```
free(**p); free(*p); free(p); // or use: free3float(p);
```

5.4 Parameter parsing

The parameter parser is borrowed from `Seismic Unix` and grouped in the file `par.c`. It is very easy to fetch parameters of different types using the routines from this parameter list:

```
void initargs(int argc, char **argv);//initialize the code and store parameter list
int getparint(char *name, int *p);//fetch a parameter using a name of int type
int getparfloat(char *name, float *p);//fetch a parameter using a name of float type
int getpardouble(char *name, double *p);//fetch a parameter using a name of double type
int getnparint(int n, char *name, int *p);//fetch comma separated ints
int getnparfloat(int n, char *name, float *p);//fetch comma separated floats
int getnpardouble(int n, char *name, double *p);//fetch comma separated doubles
int countparval(char *name);//count the number of parameter values based on given name
```

The following shows an example of usage for the above routines:

```
...
if(!getparint("n1", &n1)) n1=100;
/* number of cells in axis-1, nx */
if(!getparfloat("d1", &d1)) d1=100.;
/* grid spacing in 1st dimension, dx */
...
```

Be aware that if no parameter is supplied from parameter list, the default value is set by the code itself to avoid crashing of the simulation.

It is also important to mention that `Madagascar` has the same functionity to manipulate the memory and parse the input parameters, although the routines may be slightly different.

5.5 Read more

There are a number of papers coming out during the development of this software:

- Boost the efficiency of 3D CSEM modelling using graphics processing units (Yang, 2021);
- CSEM modelling using high order FDTD on the nonuniform grid (Yang and Mittet, 2022b,a);
- Efficient 3D CSEM inversion in fictitious wave domain (Yang, 2022)

References

Mittet, R. (2010). High-order finite-difference simulations of marine CSEM surveys using a correspondence principle for wave and diffusion fields. *Geophysics*, 75(1):F33–F50.

Yang, P. (2021). Boost the efficiency of 3D CSEM modelling using graphics processing units. In *82nd EAGE Annual Conference & Exhibition*, volume 2021, pages 1–5. European Association of Geoscientists & Engineers.

Yang, P. (2022). Efficient 3D CSEM inversion in fictitious wave domain. In *83rd EAGE Annual Conference & Exhibition*, volume Accepted. European Association of Geoscientists & Engineers.

Yang, P. and Mittet, R. (2022a). Controlled-source electromagnetics modelling using high order finite-difference time-domain method on a nonuniform grid. *Geophysics*, submitted.

Yang, P. and Mittet, R. (2022b). CSEM modelling using high order FDTD on the nonuniform grid. In *83rd EAGE Annual Conference & Exhibition*, volume Accepted. European Association of Geoscientists & Engineers.