INSTRUCTION MANUAL FOR JFDTD2D/3D (v2.00)

Jeffrey M. McMahon
Department of Chemistry, Northwestern University, Evanston, IL 60208
Center for Nanoscale Materials, Argonne National Laboratory, Argonne, IL 60439

TABLE OF CONTENTS

I. INTRODUCTION

JFDTD2D/3D are finite-difference time-domain (FDTD) method electromagnetics codes. JFDTD3D is a 3D version that can run in parallel (multiple CPUs) using the MPI library, and JFDTD2D is a serial 2D version. The code is designed to study scattering from isolated structures in open–regions and scattering from or transmission through periodic structures (e.g. arrays of particles or modulations on films). For all simulations, the incident field (by default) is a Gaussian damped sinusoidal pulse that is normal incident into the computational domain. Schematic diagrams of computational domains for 2D (JFDTD2D) and 3D (JFDTD3D) simulations are shown in Figs. 1 and 2. For the quick instructions on how to perform a simulation see Section II B.

II. SIMULATION

A. System Requirements

- C++ compiler and linker (e.g. g++) - JSCIENCE library (download from http://www.thecomputationalphysicist.com) - (3D) Some version of MPI (must be able to find mpi.h) - (3D) MPI capable C++ compiler and linker (e.g. mpicxx)

B. Quick Instructions

Running a simulation consists of 4 steps:

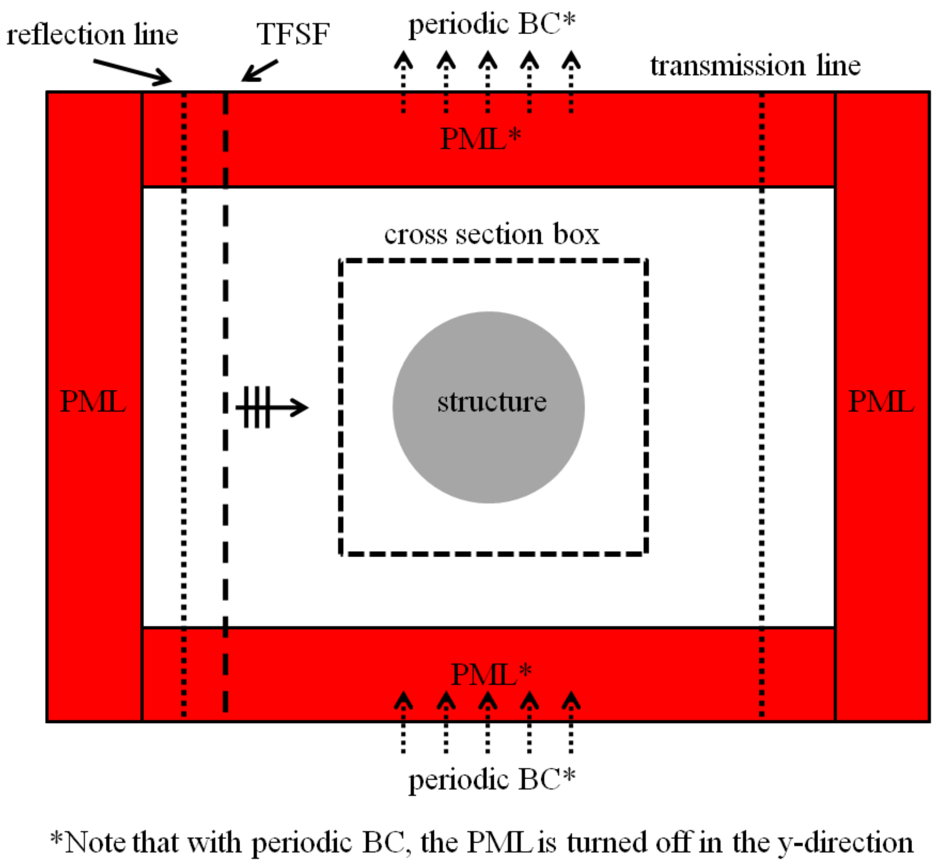

FIG. 1: Schematic diagram of computational domain for JFDTD2D.

1. Define the custom structure (and materials) (see Section III) – in the file "structure.cpp".
2. (Re)compile JSCIENCE (if any files were changed) and JFDTD2D/3D (see Section II C).
3. Set the parameters (see Section IV) – in the file "parameters".
4. Run the simulation (see Section II D).

C. Compilation

1. Linux/Unix

1. Download the JSCIENCE package (e.g. jscience.tar) and extract it. You should be left with a folder /jscience.

2. Compile the JSCIENCE library by going into the folder and using the command (¿make clean) followed by (¿make CC=g++) – or any other C++ (or MPI C++) compiler you

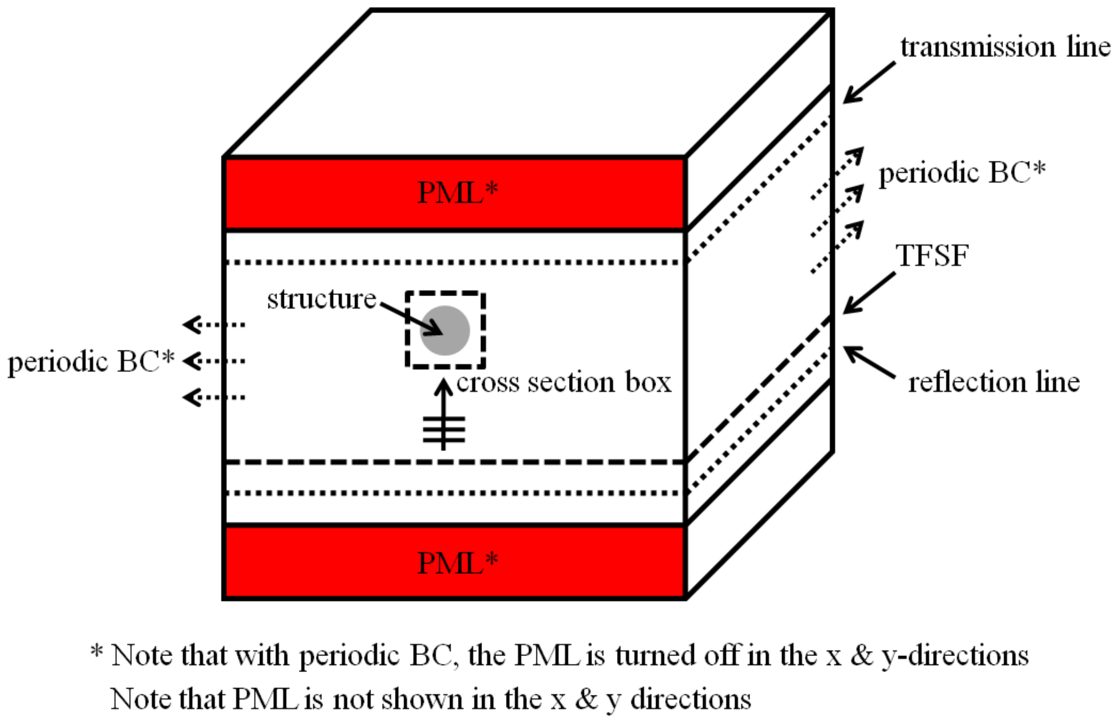
transmission line
periodic BC*
TFSF
reflection line
PML*
structure
cross section box
periodic BC*
PML*
* Note that with periodic BC, the PML is turned off in the x & y-directions
Note that PML is not shown in the x & y directions

FIG. 2: Schematic diagram of computational domain for JFDTD3D.

wish to use. This will compile all of the files in the folder into object (.o) files.
3. You probably already have the JFDTD package downloaded (since you are reading this manual), if not download it (e.g. jfdtd2d.tar) and extract it. You should be left with a folder /jfdtd2d or /jfdtd3d.
4. Go into the JFDTD directory and open the Makefile. Make sure the path to JSCIENCE (from your current location) is correct. Then exit.
5. Make JFDTD using the command (¿make clean_results) followed by (¿make clean) followed by (¿make CC=icc CCL=icpc) or any other C++ (or MPI C++) compiler and linker, respectively. If all goes well, you should be left with the executable jfdtd2d or jfdtd3d.

2. Windows

Currently, there is no official support for Windows. However, the compilation should be straightforward. First read the instructions for Linux to get an idea of the compilation procedure. Then in Windows you should (roughly speaking) compile each of the JSCIENCE files individually into .o files. Then you should compile the files in JFDTD into .o files.

Then link the appropriate files together. (To see what files should be compiled, and how they should be linked, see the Makefiles in both JSCIENCE and JFDTD).

If you get the programs running on Windows, and have any helpful hints, please let me know at jeffrey-mcmahon@northwestern.edu and I can add it to future versions of this manual.

D. Running

1. JFDTD2D

Running JFDTD2D is accomplished by simply running the executable jfdtd2d. From the command line use (¿.jfdtd2d).

2. JFDTD3D

Running JFDTD3D must be accomplished using mpirun on the executable jfdtd2d. From the command line use (¿mpirun -np # jfdtd3d), where # is the number of processors using to run JFDTD3D (this number must be equal to the number specified in "parameters").

III. STRUCTURE

The default structure is vacuum. To define a custom structure you must modify the file "structure.cpp" in the JFDTD2D/3D directory (see Section III A). Note that a recompilation of JFDTD will be necessary after this step. For structures composed of frequency dependent materials (e.g. Ag or Au), you must also modify the file "jmaterials.cpp" in the JSCIENCE directory (see Section III B) (unless the desired material has already been defined by default – such as Ag and Au). Note that after this a recompilation of both JSCIENCE and JFDTD will be necessary.

A. Structure Geometry

In the JFDTD2D/3D directory there is a file called "structure.cpp". Inside this file there is a single subroutine "get_structure()" which takes in the parameters (xpos, ypos, zpos, mat_num, mat_eps, mat_mu). To specify a structure, you must write an algorithm that specifies the material (using the variables mat_num, mat_eps, and mat_mu – see Section III B) at a general position in the computational domain (xpos, ypos, zpos). For example, suppose we are trying to model a sphere of radius r centered at (x0, y0, z0). The C++ pseudocode for this to model this would be:

```
if (sqrt( (xpos - x0)*(xpos - x0) + ... ) < r ) { then assign sphere material }
```

(Be careful to place the structure appropriately so that it does not interfere with other portions of the computational domain, e.g. PML, see Figs. 1 and 2.)

B. Structure Material

The material is specified using the variables *mat_num*, *mat_eps*, and *mat_mu* that are sent to "get_structure()" – see below. Some useful constants from JSCIENCE to know are "MU0" and "EPS0", which are the permeability (μ) and permittivity (ε) values of the vacuum. ω dependent dielectric models are defined in the file "jmaterials.cpp" in JSCIENCE – see below.

mat_mu is the material permeability. For nonmagnetic materials this should be set (or left at) MU0 (the vacuum value). (Note that it is not currently possible to have a ω dependent μ). *mat_num* and *mat_eps* specify the material ε. *mat_num* is a flag that specifies if the material ω dependent.

1. ω Independent Materials

For no ω dependence, *mat_num* should be set to 99. In this case, (constant) ε is set using *mat_eps* (e.g. 2.25*EPS0 for glass – i.e. a refractive index of 1.5).

Returning to our example of a sphere of radius r centered at (x0, y0, z0), for a glass sphere, the the C++ pseudocode is:

```
mat_mu = MU0;
if (sqrt( (xpos - x0)*(xpos - x0) + ... ) < r ) { mat_num = 99; mat_eps = 2.25*EPS0; }
```

2. ω *Dependent Materials*

For ω dependet materials, the dielectric model in Eq. (zzz) is used and implemented with the FDTD method as described in Refs.[?] . These dielectric models are defined in the file "jmaterials.h" and "jmaterials.cpp" in the JSCIENCE package (probably the only file you would ever need to modify yourself in JSCIENCE).

Returning to our example of a sphere of radius r centered at (x0, y0, z0), for a Au sphere (assuming this material number is 0), the the C++ pseudocode is:

```
mat_mu = MU0;
    if (sqrt( (xpos - x0)*(xpos - x0) + ... ) < r ) { mat_num = 0; mat_eps =
EPS0*d2l_inf[mat_num]; }
```

IV. PARAMETERS

The parameters for a simulation are set entirely in the file "parameters" in the main JFDTD directory. (Note that it is not necessary to recompile anything after changing the "parameters" file.) A brief definition of all the parameters can be found in the glossary; Section V.

A. Simulation / Quick Instructions

The following instructions should be give you a brief overview of how to do this. (Note that a lot of the default parameters will probably suit 90% of your simulations, but read all instructions anyway.) These steps are all that is necessary to run a JFDTD simulation. Of course, you will also want to obtain some output from your simulation, and this is discussed in Section IV B.

1) Set up the main computational domain using *grid_xsize*, *grid_ysize*, and *grid_zsize* to set the size and *grid_dx*, *grid_dy*, and *grid_dz* to set the grid spacing. Often you will know the appropriate grid sizes to use, but the grid–spacings may take some trial and error. (You want to use the smallest grid sizes with largest grid spacings spacings necessary to converge the results).

At this point, also set periodicitiy using *ixperiodic*, *iyperiodic*, *izperiodic*.

(In 3D) Finally, (and based on the grid–sizes and grid–spacing) set the desired number of processors using *mpi_nxprocs*, *mpi_nyprocs*, and *mpi_nzprocs*. [This will depend on the machine(s) that you are running the simulation on, and will also take some trial and error.]
2) (The default values of parameters in this step will probably suit 90% of your simulations, so do not change them unless you know what you are doing.) Next setup the CPML using the parameters *cpml_layers*, *cpml_epsr*, *cpml_mur*, *cpml_kappamax*, *cpml_sigmamax_coeff*, *cpml_alphamax*, *cpml_m*, and *cpml_ma*. Note that PML is defined by the number of layers (*cpml_layers*), and so will comprise a portion of the computational domain of size *cpml_layers*grid_dz* (in the z–direction, for example).
3) (The default values of parameters in this step will probably suit 90% of your simulations, so do not change them unless you know what you are doing.) Set the simulated time for your simulation using *time_total* and *time_courant_factor*. Note that the default value of *time_total* = 150e-15 s should be plenty of time to give accurate Fourier–transformed fields, and *time_courant_factor* should *never* be changed from 0.95.
4) (Some of the default values of parameters in this step will probably suit 90% of your simulations, so do not change them unless you know what you are doing.) Set up the incident field using *source_intensity*, *source_wavelength*, *source_gauss_width*, *source_gauss_center*. Default values of *source_wavelength* = 600.0e-9 and *source_gauss_width* = 0.4e-15 give frequency content of the incident field from 1 to 6 eV. (Note that *source_intensity* does not really affect the simulation in any way, as it is used for normalization anyway.)

(This you probably may want to change.) (In 3D) Set the incident field polarization using *src_ieypol*, *src_iexpol*, *src_i45pol*, *src_icircpol*.

(This you definately need to change.) Set the total field / scattered field (TF / SF) line (the position where the incident field originates from – loosely speaking) using (in 3D) *tfsf_zpos* or (in 2D) *tfsf_xpos*. This line must lie outside the PML, and as a general rule try to placeit at least 3 grid points away.

B. Output

There are 3 main types of output that can be calculated / displayed. These are field plots (e.g. Fourier–transformed field profiles, etc), optical cross–sections (absorption, scattering, and extinction), and transmission spectra (transmission, reflection, and absorption) – which is used primarily for film calculations (e.g. hole arrays). All output is written to the /output/ directory in JFDTD.

C. Field Output

To set up field outputs the parameters *noutput*, *output_nplanes*, (3D) *output_format[]*, *output_field[]*, *ft_wavelengths[]*, (3D) *output_plane[]*, and (3D) *output_plane_midpos[]* are used in the file "parameters". The first parameter *noutput* specifies the number of outputs (of each type of field) per simulation (e.g. if this is set to 3, then the desired fields will be output at times 1/3, 2/3, and 3/3 of the simulation). The parameter *output_nplanes* specifies the number of different types of fields you want to output per simulation (e.g. Ex, Ey, ...). In 3D the parameter *output_format[]* the format of the output (*LEAVE THIS AT 1 – GNUPLOT).

For each type of field that you want to output the following information must be specified (these parameters are arrays, so you specify field 1 in the position [1], 2 in [2] ...). *output_field[]* specifies the type of field (e.g. E^2, Ex ...). If the field is a frequency dependent field (which is obtained by Fourier–transforming) the paramater *ft_wavelengths[]* specifies the wavelength of transform. In 3D the plane (xy, xz, or yz) must be specified by using *output_plane[]* and *output_plane_midpos[]*. *output_plane[]* specifies the constant plane (e.g. this would be z for an xy–plane) and *output_plane_midpos[]* specifies the position of this constant (e.g. the value of z). For multiple planes, it is important to specify the entire set of information in order. For example, for 2 planes:

```
noutput = 3;
output_nplanes = 2;
output_format[1] = 1;
output_field[1] = 5;
ft_wavelengths[1] = 500.0e-9;
output_plane[1] = 2;
output_plane_midpos[1] = 200.0e-9;
output_format[2] = 1;
output_field[2] = 2;
```

ft_wavelengths[2] = 500.0e-9;
output_plane[2] = 1;
output_plane_midpos[2] = 400.0e-9;

- Fields set for output with (output_nplanes, etc) are output in the format: e_field.a.b.c.d (3D) or e_field.a.b (2D) where a is the plane number (which goes from 1 to output_nplanes), b is the output number (which goes from 1 to noutput), and c and d are processor ranks. (3D) These files need to be "stitched" together by running filecombine (in the same directory as e_field.*). filecombine was compiled when using make. This program uses field_stitch.dat to stitch the files together.

- Transmission, reflection, and absorption spectra are output as spect_transm.dat, spect_refl.dat, and spect_abs.dat.

- Processor information is output in simfile.*

Cross sections are output in scatcs.dat, abscs.dat, extcs.dat.

V. PARAMETERS GLOSSARY

A. Simulation Parameters

(3D) *mpi_nxprocs*, *mpi_nyprocs*, *mpi_nzprocs*: Number of processes in each dimension. Each of these should be even, especially is syncronous MPI sends are used, otherwise a lockup could occur.

(3D) *isendtype*: MPI communication type. *Do not change*

(3D) *ixperiodic*, *iyperiodic*, *izperiodic*; (2D) iperiodic: Periodicity in each directions. Note that you cannot actually turn off the periodicity, if these flags are set to 0 then PML is placed along the sides (3D) or top and bottom (2D) of the simulation box to simulate a non-periodic system. *Do not turn z-periodicity off*

grid_xsize, *grid_ysize*, (3D) *grid_zsize*: Grid sizes.

grid_dx, *grid_dy*, (3D) *grid_dz*: Grid spacings.

cpml_layers: Layers of CPML. 15-20 layers should be more than enough for most simulations.

cpml_epsr, *cpml_mur*: PML material.

cpml_kappamax, *cpml_sigmamax_coeff*, *cpml_alphamax*: CPML parameters. The default values should be optimum for most simulations.

cpml_m, *cpml_ma*: Polynomial grading of PML. The default values should be optimum for most simulations.

time_total: Simulation time. 100e-15 s is a good length of time to get accurate Fourier transformed fields.

time_courant_factor: Multiplication factor for the maximum stable time step. This factor must be ¡= 1 to be guarenteed a stable simulation (I have never had it become unstable at 0.95). The default values should be optimum for most simulations.

source_intensity: Source intensity.

(3D) *src_ieypol*, *src_iexpol*, *src_i45pol*, *src_icircpol*: Source polarization.

source_wavelength: Source wavelength of the sinusoidal part of the source.

source_gauss_width, *source_gauss_center*: Gaussian parameters of source. The center of the pulse should be at a time so that is approximately 0 at t = 0 and t = time_total.

(3D) *tfsf_zpos*; (2D) *tfsf_xpos*: Total-field / scattered-field z(x)-position.

B. Output Parameters

isimfile: Flag to ouput processor simulation info. *LEAVE OFF FOR NOW*

noutput: Number of field outputs.

output_nplanes: Number of different types of output planes.

(3D) *output_format[]*: Format of output, e.g. gnuplot or OpenDX.

output_field[]: Types of output planes.

ft_wavelengths[]: Wavelengths of Fourier-transformed fields.

(3D) *output_plane[]*: What plane (x, y, or z constant) to output if the format is gnuplot.

(3D) *output_plane_midpos[]*: Position of plane (x, y, or z constant) to output if the format is gnuplot.

iscat_calc: Flag to calculate cross sections.

scat_xcenter, *scat_ycenter*, and (3D) *scat_zcenter*: Center of box used to calculate cross section.

scat_radius: Size (e.g. in x the box goes from scat_xcenter-scat_radius to scat_xcenter+scat_radius) of cross section box.

itransm: Flag to calculate transmission spectrum.

spect_transm_pos, *spect_refl_pos*: z(x)–Positions to calculate the transmission and reflection spectra.

spect_npts, *spect_minwave*, and *spect_maxwave*: Wavelength range of transmission and reflection spectra.

VI. TROUBLESHOOTING / COMMON ISSUES

1. Program fails to begin and returns the error "killed": This error occurs when the machine(s) you are attempting to perform the simulation run out of memory. There are a few ways to rectify this simulation. The first way is to decrease *spect_npts* (if you are using this feature). Increasing *spect_npts* greatly increases the required memory. The second (if possible) is to increase the number of processors that your are running on. Finally, (if possible) decrease the grid sizes, or increase the grid spacing.

VII. CHANGELOG

A. Version 1.0 – Released xxx

• Initial launch of JFDTD2D/3D

B. Version 2.0 – Released xxx

• Structure definition moved from main .cpp file to separate file "structure.cpp" for definition
• Simulation parameters moved from main .cpp file to separate file "paramaters" for definition

C. Version 3.0 – Not released yet

• First full manual for provided
• Nonlocal (spatial dispersion) capability added
• Maxwell Stress Tensor (MST) capability added to JFDTD3D
• VTK / Paraview output format provided
• Error with isimfile and segmentation fault in JFDTD3D fixed