Spring 1-1-2012

Broadband electromagnetic analysis of dispersive, periodic structures for radiometer calibration

Srikumar Sandeep
University of Colorado at Boulder, sandeepsrikumar@yahoo.com

Follow this and additional works at: https://scholar.colorado.edu/ecen_gradetds
Part of the Electromagnetics and Photonics Commons

Recommended Citation
https://scholar.colorado.edu/ecen_gradetds/48

This Dissertation is brought to you for free and open access by Electrical, Computer & Energy Engineering at CU Scholar. It has been accepted for inclusion in Electrical, Computer & Energy Engineering Graduate Theses & Dissertations by an authorized administrator of CU Scholar. For more information, please contact cuscholaradmin@colorado.edu.
Broadband electromagnetic analysis of dispersive, periodic structures for radiometer calibration

by

S. Sandeep

M.S., University of Colorado, Boulder 2011

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Department of Electrical, Energy and Computer engineering

2012
This thesis entitled:
Broadband electromagnetic analysis of dispersive, periodic structures for radiometer calibration
written by S. Sandeep
has been approved for the Department of Electrical, Energy and Computer engineering

Prof. Albin Gasiewski

Prof. Dejan Filipovic

Prof. Edward Kuester

Prof. Bob McLeod

Dr. David Walker

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.
This thesis primarily focuses on the full wave electromagnetic analysis of radiometer calibration targets using doubly dispersive 3D Finite Difference Time Domain (FDTD) formulation. The boundary conditions are set up to solve for doubly periodic structures. The thesis contains very detailed derivation and equations regarding this formulation. One of the novelty in this formulation is the handling of magnetically and electrically dispersive media (usually it is just the electrical dispersion which is incorporated). Using a custom developed code which can be run on a distributed computing system, the reflectivity spectrum of calibration targets of different geometries, coating thicknesses and aspect ratios are analyzed. The results are well validated using commercial simulation softwares and custom Geometric Optics (GO) code. The geometries analyzed include square pyramids, conical pyramids, truncated square pyramids and truncated conical pyramids with spherical top. The coating thicknesses used are 1 mm, 2 mm and 3 mm. The aspect ratios (ratio of base to height) used include 1 : 1, 1 : 2 and 1 : 4. The nominal target structure has 1 : 4 aspect ratio and 2mm coating thickness. The material used for simulation is ECCOSORB MF112. The material properties of other materials such as MF110 and MF114 are listed. It should be remarked that measured material properties are available only in the frequency range [8, 26] GHz and a Debye series extrapolation was used for simulation at frequencies outside this range. Throughout this work 0.5" base was used. Some significant conclusions include the following: 1) 1:4 aspect ratio or better is required to achieve a -50 dB reflectivity or lower 2) Low frequency reflectivity is independent of the target geometry. 3) At high frequencies, the conical target results in better performance when compared to square pyramids (by about 10 dB). 4) The reflectivity spectrum exhibits a general trend of high reflectivity at low frequencies followed by decreasing reflectivity as frequency is increased. There is a reflectivity jump at frequencies where non-specular Floquet modes start
propagating. This is followed by nearly sinusoidal oscillations at high frequencies. 5) Asymptotic

techniques can be used at high frequencies instead of full wave analysis. The plane wave reflectivity
estimated using full wave analysis is an approximate method to calculate brightness temperature as
measured by antenna during radiometer calibration. It assumes two conditions: 1) The calibration
targets have a uniform temperature profile. 2) Antenna is in the far field. These two conditions are
never met in practice. In order to estimate the near field thermal emission, Fluctuation Dissipation
Theorem (FDT) must be used. Dyadic Green Function (DGF) along with FDT can be used to
calculate the thermal emission from simple geometries. Analytical formulations to this end is given
in this thesis.

The rest of the thesis (∼ 50%) contains work related to numerical methods applied to ra-
diative transfer and computational electromagnetics. In the first part, a novel method to calculate
the absorption coefficient, scattering coefficient, backscattering coefficient and phase asymmetry
parameter of a polydispersed distribution of liquid water and ice hydrometeors is presented. The
conventional method of calculating these coefficients can be time consuming, because of the Mie
series summation to calculate Mie coefficients and the numerical quadrature over a distribution
of spheres to calculate the required coefficients. By using spline interpolation on a precomputed
look up table, the calculation procedure can be accelerated. The second part deals with time do-
main analysis of dispersive, periodic structures for oblique plane wave incidence. This is a difficult
problem with only one work available in literature till now. The proposed method uses Laguerre
Marching-In-On-Degree (MoD) where time dependant quantities are expressed as an expansion of
Laguerre basis functions. Using several properties of Laguerre basis functions, the time dependant
problem is converted to a time independent problem in Laguerre basis coefficients. This in turn is
solved using the familiar finite difference format. The novel method was validated with analytical
results for incident angles as large as 75°.
Dedication

To my parents
Acknowledgements

I would like to thank Prof. Albin Gasiewski for providing me an opportunity to do PhD with funding and for his patience. His broad knowledge in several fields (both practical and theoretical) inspired me tremendously. The last four years of grad school made me realize that my knowledge from undergrad textbooks was nothing! (I would have been in the same state of mind, if I continued working in industry). I would also like to thank Dr. David Walker of NIST for the NIST PREP fellowship. I hope the invaluable experience and knowledge which I obtained in last four years will help me in solving more difficult and important problems in electromagnetics and applied mathematics.
Contents

Chapter

1 Introduction 1

2 FDTD formulation 5
   2.1 Piecewise Linear Recursive Convolution 6
   2.2 Uniaxial Perfectly Matched Layer 12
   2.3 Boundary conditions 14
   2.4 Parallel computation 15
   2.5 Microwave characterization of radar absorbing material 17
   2.6 Numerical validation 19

3 Results and Discussion 25
   3.1 Total reflectivity formulation 25
   3.2 Reflectivity spectrum of square pyramid array 27
   3.3 Geometrical optics validation 29
   3.4 Floquet mode reflectivity analysis 30
   3.5 Reflectivity spectrum of conical pyramid array 31
   3.6 Reflectivity spectrum of truncated square pyramid array 33
   3.7 Target base width scaling 34
   3.8 Unscented transformation sensitivity analysis 34
   3.9 Uncertainty in material properties 35
4 Transient analysis of dispersive, periodic structures for obliquely incident plane wave using
Laguerre Marching-On-In Degree (MoD)

4.1 Introduction .................................................. 48
4.2 Laguerre MoD formulation for periodic, dispersive structure ....................... 49
4.3 UPML formulation in Laguerre MoD ........................................ 53
4.4 Numerical validation ............................................. 56
4.5 Conclusions .................................................... 57

5 Near Field Thermal Emission ................................. 60

5.1 Fluctuation Dissipation Theorem .................................... 61
5.2 Dyadic Green’s Function .......................................... 61
5.3 Formulation of stochastic electromagnetic power density ......................... 63
5.4 Cylindrical Vector Wave Functions .................................... 65
  5.4.1 Orthogonality of VWFs .................................... 66
5.5 DGF for dielectric cylinder ......................................... 67

6 Fast Jacobian Mie Library for Terrestrial Hydrometeors ......................... 71

6.1 Introduction .................................................... 72
6.2 Mie theory .................................................... 74
6.3 Look-up table calculation ......................................... 76
6.4 B-spline interpolation ............................................ 79
6.5 Results and discussion .......................................... 82
  6.5.1 Memory and Computational Overhead Reduction ............................ 82
  6.5.2 Radiation Jacobian .......................................... 83
  6.5.3 Computational Savings ....................................... 84
6.5.4 Reconstruction Error Analysis ................................. 84
6.5.5 Radiative Transfer Simulations ............................... 87
6.6 Conclusions ......................................................... 88

7 Conclusions and Future work 99
  7.1 Conclusions ..................................................... 99
  7.2 Future work ..................................................... 102
    7.2.1 Heat Diffusion Equation ................................. 104

Bibliography 106

Appendix

A 3D dispersive FDTD code 113
B Genetic algorithm Debye series fitting code 171
C Periodic surface reflectivity using Geometric Optics (GO) 173
D Truncated pyramid geometry 178
E Laguerre MoD code 181
F ADE (Auxiliary Differential Equation) formulation for $TM_z$ case (Debye media) 192
Tables

Table

2.1  5 pole Debye parameters for MF-112 ............................................ 20
2.2  5 pole Debye parameters for MF-114 ............................................ 21
2.3  5 pole Debye parameters for MF-110 ............................................ 21
Figures

Figure

1.1 Radiometer calibration target ................................................... 3

2.1 FDTD computational engine ...................................................... 6

2.2 FDTD unit cell ................................................................. 15

2.3 Symmetry/Antisymmetry boundary conditions on $x,z$ boundaries ........... 16

2.4 1D Domain Decomposition for parallel computation ........................... 17

2.5 1D Domain Decomposition in the case of 1D FDTD ........................... 17

2.6 MF112 permittivity: Measured data and 5 pole Debye fit .................. 21

2.7 MF112 permeability: Measured data and 5 pole Debye fit .................. 22

2.8 MF110 permittivity: Measured data and 5 pole Debye fit .................. 22

2.9 MF110 permeability: Measured data and 5 pole Debye fit .................. 23

2.10 Reflectivity spectrum of 0.5" base,1:1 pyramid with 1 mm MF112 coating .... 23

2.11 Numerical validation of the code .............................................. 24

3.1 2D cross section of calibration target ....................................... 26

3.2 Reflectivity of pyramidal targets with MF112 coating for various aspect ratios and coating thicknesses in the frequency range [6,26] GHz .............................................. 28

3.3 HFSS model for square pyramid specular mode reflectivity calculation ........ 39

3.4 Reflectivity of pyramidal targets with MF112 coating for various aspect ratios and coating thicknesses 1 mm,2 mm in the frequency range [6,200] GHz ................................. 39
3.5 Reflectivity of pyramidal targets with MF112 coating for various aspect ratios and coating thickness 3 mm in the frequency range [6,200] GHz. GO validation is also shown for each case. ......................................................... 40
3.6 Front view of pyramid showing facets for GO modelling .......................... 40
3.7 Floquet mode reflectivity spectrum for 1:2, 0.5″ base pyramid with 2 mm MF112 coating ................................................................. 41
3.8 Reflectivity spectrum of conical pyramid array with 1 mm of MF112 coating .......... 41
3.9 Reflectivity spectrum of conical pyramid array with 2 mm of MF112 coating .......... 42
3.10 Reflectivity spectrum of conical pyramid array with 3 mm of MF112 coating .......... 42
3.11 Reflectivity spectrum comparison of circular and square pyramids with 2 mm MF112 coating. Periodicity is $\frac{0.5''}{\sqrt{2}}$. ......................................................... 43
3.12 Reflectivity spectrum of truncated radiometer calibration targets .................... 43
3.13 Reflectivity spectrum of truncated, conical pyramid array. Aspect ratio of 1:4 and 2 mm MF112 coating. ................................................................. 44
3.14 Reflectivity spectrum of square pyramid array. Base = 0.2833″,1:4 aspect ratio and 2 mm MF112 coating. ................................................................. 44
3.15 Uncertainty bounds in the reflectivity spectrum of 1:4,2 mm square pyramid target with MF112 coating ................................................................. 45
3.16 Late time instability ................................................................. 45
3.17 Prony extrapolation of scattered field time series ........................................ 46
3.18 Effect of late time instability on reflectivity spectrum .................................... 46
3.19 Convergence study: 0.5″ base,1:4 circular pyramid with 1 mm MF112 coating ........ 47
3.20 Observation plane location : 0.5″ base,1:4 circular pyramid with 1 mm MF112 coating 47

4.1 Transmission spectrum of dispersive slab at 0° angle of incidence ............... 58
4.2 Transmission spectrum of dispersive slab at 45° angle of incidence ............... 58
4.3 Transmission spectrum of dispersive slab at 60° angle of incidence ............... 59
4.4 Transmission spectrum of dispersive slab at 75° angle of incidence .......................... 59

6.1 Logarithmic ratio of the calculation time for B-spline interpolation to that for exact Mie series calculation of polydispersed liquid hydrometeor absorption coefficients. An exponential drop-size distribution, temperature $T = 0^\circ C$ and fractional volume $f = 10^{-6}$ are assumed ............................................................... 90

6.2 Fractional error in reconstructed values of liquid and ice hydrometeor absorption coefficient vs. liquid and ice hydrometeor absorption values. The maximum allowable error bounds for $T_{DB} = 0.01K$ and 0.1 K are shown. ............................................................... 90

6.3 Liquid hydrometeor $\kappa_a$ (dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ .......................... 91

6.4 Ice hydrometeor $\kappa_a$ (dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ............................... 91

6.5 Liquid hydrometeor $\kappa_s$ (dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ............................. 92

6.6 Ice hydrometeor $\kappa_s$ (dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ............................... 92

6.7 Liquid hydrometeor $\kappa_b$ (dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ............................. 93

6.8 Ice hydrometeor $\kappa_b$ (dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ............................... 93

6.9 Liquid hydrometeor $g$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ........................................... 94

6.10 Ice hydrometeor $g$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ .............................................. 94

6.11 Liquid hydrometeor $\frac{d\kappa_a}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ......................... 95

6.12 Ice hydrometeor $\frac{d\kappa_a}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ......................... 95

6.13 Liquid hydrometeor $\frac{d\kappa_s}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ......................... 96

6.14 Ice hydrometeor $\frac{d\kappa_s}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ......................... 96

6.15 Liquid hydrometeor $\frac{d\kappa_b}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ......................... 97

6.16 Ice hydrometeor $\frac{d\kappa_b}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ......................... 97

6.17 Liquid hydrometeor $\frac{dg}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ......................... 98

6.18 Ice hydrometeor $\frac{dg}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$ ......................... 98

7.1 Reflectivity spectrum as shown in [1] .......................................................... 100
D.1  Truncated pyramid geometry .......................................................... 178
D.2  Spherical cap ................................................................. 179
Microwave and millimeter wave remote sensing offer several advantages over infrared and visible observations. This is primarily due to the fact that microwave radiation is transparent to clouds and aerosols, thereby aiding in the accurate determination of temperature and water-vapor profiles. Passive microwave remote sensing of planetary atmospheres and surfaces provides a valuable means of mapping a variety of atmospheric and surface variables with minimal adverse influence from clouds and aerosols. Variables regularly measured and imaged include temperature and water vapor profiles, cloud liquid and ice content, soil moisture, sea surface salinity and wind vector, snow water equivalent, and sea ice extent. Increasingly, these variables are the object of climate studies which seek to link anthropogenic forcings to regional or global changes in one or several of them. Unambiguous determination of changes in any of these variables generally requires absolute radiometric accuracy of a small fraction of a Kelvin over decadal time scales. For example, instrumental biases as small as 0.05K or better out of a total system temperature of 500-1500K or more are required over the typical lifetime of a sounding instrument (several years) in order to unambiguously identify climatological temperature trends of order 0.1C per decade. Such accuracy is difficult to achieve across generations of instruments without traceable and accurate calibration methods. Radiometer calibration is the process by which a relationship is obtained between the output indicator of the radiometer usually voltage or current and the noise temperature at the radiometer input (usually the antenna temperature, when an antenna is connected to the radiometer receiver input) [2].
Radiometer calibration is most practically accomplished by periodic observations of highly emissive thermal references whose physical temperature is used to determine the measured antenna temperature \[2\]. Precise absolute calibration of radiometers for many Earth remote sensing applications requires knowledge of the antenna temperature to ±100 mK \[3\], or even a factor of ten better in the case of atmospheric temperature trend estimation. To achieve such precision - on the order of -40 dB or better - highly stable and emissive thermal microwave references are used for external calibration of spaceborne radiometers. Sensors relying on such external references for calibration include the JAXA AMSR-E \[4\], AMSR-2 \[5\], DMSP SSM/I and SSMIS \[6\], NRL WindSat \[7\], Suomi ATMS, and NASA GMI \[8\]. Reference targets are typically periodic 1-dimensional arrays of wedges or two-dimensional arrays of pyramids constructed of a highly thermally-conductive substrate (e.g., Al or Mg) coated with a thin layer of microwave absorbing material as shown in Fig. 1.1. Microwave absorbing materials are often mixtures of iron particles in an epoxy matrix, and are electrically dispersive in nature. The ideal radiometer calibration reference should be a blackbody radiator with an emissivity of unity at all radiometric bands of interest. However, this is not achieved in practice, in which case the emitted brightness temperature of the reference \(T_B(\theta, \phi)\) becomes

\[
T_B(\theta, \phi) = e(\theta, \phi)T_P + \int_0^{\frac{\pi}{2}} \int_0^{2\pi} (1 - e(\theta', \phi'))T_R(\theta', \phi') d\Omega'
\]

where the emissivity \(e(\theta, \phi)\) is related to reflectivity \(r(\theta, \phi)\) by Kirchoff’s law, \(e(\theta, \phi) = 1 - r(\theta, \phi)\) and \(d\Omega' = \sin\theta' d\theta' d\phi'\). \(T_P\) is the physical temperature of the reference and \(T_R(\theta', \phi')\) is the background temperature. Accordingly, we seek thermally conductive structures with low and well known reflectivities at key microwave sounding and imaging bands often spanning a decade or more in frequency. Since the background radiation field is often difficult to control and predict, accurate estimation of target reflectivity is therefore essential for precise calibration.

In order to efficiently design wideband calibration targets, an accurate electromagnetic and thermal analysis of these structures is essential. Electromagnetic and thermal analysis of one-dimensional periodic wedge shaped structures was studied by Jackson and Gasiewski \[9, 10\].
work used the coupled-wave approach for the electromagnetic field analysis [11]. However the work was not able to be extended to 2-D periodic pyramidal structures due to computational and numerical accuracy limitations. The Extended Boundary Condition (EBC) was used for predicting surface scattering from doubly periodic pyramidal structures in [12]. This method has the disadvantage of being ill-conditioned when the surface corrugation is deep or when the corrugation depth divided by the period is large. The method also is not easily able to handle inhomogenous calibration references which might involve a number of layers or even gradations in material parameters. Geometrical optic techniques have been used to study wedge shaped structures [13]. However, these optical techniques provide accurate results only at sufficiently high frequencies.

In chapter 2, the formulation of Finite Difference Time Domain (FDTD) method for estimating the reflectivity of periodic, dispersive (electric and magnetic dispersion) structures is outlined. The results obtained by the developed code is detailed in chapter 3. In chapter 4, a novel method for transient analysis of electrically dispersive periodic structures for oblique plane wave incidence is
presented. This method is based on Laguerre Marching-On-In-Degree (MoD), which can eliminate the constraint on the temporal discretization interval of the FDTD. Near field thermal emission from a surface with non uniform temperature profile is the subject of chapter 5. The formulations based on Fluctuation Dissipation Theorem (FDT) and Dyadic Green’s Function (DGF) is explained. Chapter 6 deals with a faster way of calculating absorption, scattering and phase asymmetry coefficients of a polydispersed distribution of liquid water or ice hydrometeors. The method uses 3D spline interpolation rather than the conventional Mie series expansions and time consuming numerical quadratures. The technique was found to have significant advantages particularly for large hydrometeors (or alternatively at high frequency atmospheric radiative transfer simulations).
Chapter 2

FDTD formulation

Finite Difference Time Domain (FDTD) is a time domain numerical electromagnetic method introduced by K.S.Yee [14]. Several decades of research had resulted in the method being one of the most powerful and commonly used Computational Electromagnetic (CEM) method. The introduction to this method and its application to wide range of electromagnetic problems can be found in [15, 16, 17]. In this section, the FDTD notation which is required to understand the rest of the chapter is outlined. In FDTD, the spatial domain is discretized into rectangular cubical cells called Yee cells of dimension $\Delta x \times \Delta y \times \Delta z$. Electric field components are located at the edge centers and magnetic field components at the face centers of the Yee cell. Similarly, the time axis is discretized into intervals of $\Delta t$. The electric field components are evaluated at $n \Delta t$ and magnetic field components are evaluated at $(n + 0.5) \Delta t$. The notation used for FDTD difference equations in this work is given by,

\[
\begin{align*}
E_x|^{n}_{i,j,k} & \rightarrow E_x((i + 0.5)\Delta x, j\Delta y, k\Delta z; n\Delta t) \\
E_y|^{n}_{i,j,k} & \rightarrow E_y(i\Delta x, (j + 0.5)\Delta y, k\Delta z; n\Delta t) \\
E_z|^{n}_{i,j,k} & \rightarrow E_z(i\Delta x, j\Delta y, (k + 0.5)\Delta z; n\Delta t) \\
H_x|^{n+0.5}_{i,j,k} & \rightarrow H_x(i\Delta x, (j + 0.5)\Delta y, (k + 0.5)\Delta z; (n + 0.5)\Delta t) \\
H_y|^{n+0.5}_{i,j,k} & \rightarrow H_y((i + 0.5)\Delta x, j\Delta y, (k + 0.5)\Delta z; (n + 0.5)\Delta t) \\
H_z|^{n+0.5}_{i,j,k} & \rightarrow H_z((i + 0.5)\Delta x, (j + 0.5)\Delta y, k\Delta z; (n + 0.5)\Delta t)
\end{align*}
\] (2.1)
where $i, j, k$ are the spatial discretization indices and $n$ is the temporal discretization index. This notation is more helpful in software implementation than using integer and half indices for spatial discretization, such as $E_{x,i+0.5,j,k}^n$, since software array indexing is integer based. The components of the FDTD formulation for this problem is given in Fig. 2.1. Piecewise Linear Recursive Convolution (PLRC) is used to handle dispersive coating material. The problem space is bounded by Uniaxial Perfectly Matched Layers (UPML) on the $y-$ axis boundaries and Periodic Boundary Condition (PBC) is used to simulate an infinite 2D array of pyramids. In order to simulate electrically large structures, parallel computation is used in the form of 1D domain decomposition.

![FDTD computational engine](image)

**Figure 2.1: FDTD computational engine**

### 2.1 Piecewise Linear Recursive Convolution

Temporal dispersion is characterized by frequency dependent electric and magnetic susceptibility as given by,

\[
\varepsilon_r(\omega) = \varepsilon_\infty + \chi_e(\omega) \tag{2.2}
\]

\[
\mu_r(\omega) = \mu_\infty + \chi_m(\omega) \tag{2.3}
\]
where $\chi_e(\omega), \chi_m(\omega)$ are the frequency domain electric and magnetic susceptibility functions, respectively. Depending on the functional form of these susceptibility functions, the material is classified as Debye, Lorentz, Drude media etc. The time domain form of (2.2), (2.3) is obtained by inverse Fourier transform as,

$$\varepsilon_r(t) = \varepsilon_\infty \delta(t) + \chi_e(t) \quad (2.4)$$

$$\mu_r(t) = \mu_\infty \delta(t) + \chi_m(t) \quad (2.5)$$

where $\chi_e(t), \chi_m(t)$ are associated time domain electric and magnetic susceptibility functions. In order to derive the FDTD update equation for the field component $H_x$, we start with the following continuous domain PDE and its discretized version as shown below,

$$\frac{\partial E_z(x, y, z; t)}{\partial y} - \frac{\partial E_y(x, y, z; t)}{\partial z} = - \frac{\partial B_x(x, y, z; t)}{\partial t} \quad (2.6)$$

$$\begin{bmatrix} E_z^{n+1}_{i,j,k} - E_z^n_{i,j,k} \\ E_y^n_{i,j,k+1} - E_y^n_{i,j,k} \end{bmatrix} \frac{\Delta y}{\Delta z} = - \begin{bmatrix} B_x[n+0.5] - B_x[n-0.5] \\ H_x[n] \end{bmatrix} \Delta t \quad (2.7)$$

In time domain, $B_x(t)$ is related to $H_x(t)$ through the convolution integral,

$$B_x(t) = \mu_o [\mu_r(t) \otimes H_x(t)] = \mu_o \mu_\infty H_x(t) + \mu_o \int_0^t \chi_m(\tau) H_x(t - \tau) d\tau \quad (2.8)$$

In the discrete space, the equation (2.8) can be written as,

$$B_x[n+0.5]_{i,j,k} = \mu_o \mu_\infty H_x[n+0.5]_{i,j,k} + \mu_o \int_0^{n+0.5} \chi_m[\tau] H_x[i,j,k] d\tau \quad (2.9)$$

It should be noted that in (2.9), $n\Delta t$ rather than $(n + 0.5)\Delta t$ is used as the upper limit of the integral. This is due to the fact that when $\tau$ ranges from $n\Delta t$ to $(n + 0.5)\Delta t$, the argument of the $H_x$ ranges between 0 and $0.5\Delta t$. In this interval, all the magnetic fields are assumed to be zero. In (2.9), it can be seen that the values of $H_x$ between temporal discretization points is required for integration. This problem is overcome by using linear interpolation in the interval between
two time discretization points. This is the main idea behind PLRC. In order to facilitate linear interpolation, (2.9) can be rewritten in the following summation form.

\[ B^{n+0.5}_{i,j,k} = \mu o \mu_\infty |_{i,j,k} H^{n+0.5}_{x_{i,j,k}} + \mu o \sum_{d=0}^{n-1} \int_{\tau}^{(d+1) \Delta t} \chi_{m|_{i,j,k}} (\tau) H_{x_{i,j,k}} \left[(n + 0.5) \Delta t - \tau\right] d\tau \] (2.10)

In the interval \( \tau \in [d \Delta t, (d + 1) \Delta t] \), \( H_{x_{i,j,k}} \left[(n + 0.5) \Delta t - \tau\right] \) can be approximated using a piece-wise linear form as follows,

\[ H_{x_{i,j,k}} \left[(n + 0.5) \Delta t - \tau\right] = H_{x_{i,j,k}}^{n-d+0.5} + \left( \frac{\tau - d \Delta t}{\Delta t} \right) \left[H_{x_{i,j,k}}^{n-d-0.5} - H_{x_{i,j,k}}^{n-d+0.5}\right] \] (2.11)

By applying (2.11) in (2.10), an expression for \( B^{n+0.5}_{i,j,k} \) can be obtained as,

\[ B^{n+0.5}_{i,j,k} = \mu o \mu_\infty H_{x_{i,j,k}}^{n+0.5} + \mu o \sum_{d=0}^{n-1} X_{m|_{i,j,k}} H_{x_{i,j,k}}^{n-d+0.5} + \xi_{m|_{i,j,k}} \left[H_{x_{i,j,k}}^{n-d-0.5} - H_{x_{i,j,k}}^{n-d+0.5}\right] \] (2.12)

where we have used the following coefficients,

\[ X_{m|_{i,j,k}}^d = \int_{\Delta t}^{(d+1) \Delta t} \chi_{m|_{i,j,k}} (\tau) d\tau \]

\[ \xi_{m|_{i,j,k}}^d = \frac{1}{\Delta t} \int_{\Delta t}^{(d+1) \Delta t} (\tau - d \Delta t) \chi_{m|_{i,j,k}} (\tau) d\tau \] (2.13)

By applying (2.12) and time shifted version of (2.12) to (2.7), the update equation for \( H_{x_{i,j,k}}^{n+0.5} \) is derived as,

\[ H_{x_{i,j,k}}^{n+0.5} = \left(\frac{\Delta t}{\mu o K1|_{i,j,k}}\right) \left[ \frac{E_{y_{i,j,k+1}}^n - E_{y_{i,j,k}}^n}{\Delta z} + E_{z_{i,j,k+1}}^n - E_{z_{i,j,k}}^n \right] + \frac{K2|_{i,j,k}}{K1|_{i,j,k}} H_{x_{i,j,k}}^{n-0.5} + \frac{\Psi H_{x_{i,j,k}}^{n-0.5}}{K1|_{i,j,k}} \]

\[ K1|_{i,j,k} = \mu_\infty|_{i,j,k} + X_{m|_{i,j,k}}^0 - \xi_{m|_{i,j,k}}^0 \]

\[ K2|_{i,j,k} = \mu_\infty|_{i,j,k} - \xi_{m|_{i,j,k}}^0 \] (2.14)

\( \Psi H_{x_{i,j,k}}^{n-0.5} \) is called the recursive accumulator and is given by

\[ \Psi H_{x_{i,j,k}}^{n-0.5} = \sum_{d=0}^{n-2} \Delta X_{m|_{i,j,k}}^d H_{x_{i,j,k}}^{n-d-0.5} + \Delta \xi_{m|_{i,j,k}}^d \left[H_{x_{i,j,k}}^{n-d-1.5} - H_{x_{i,j,k}}^{n-d-0.5}\right] \] (2.15)
\[\Delta X_m^{|d|}_{i,j,k} = X_m^{|d|}_{i,j,k} - X_m^{|d+1|}_{i,j,k}\]  
(2.16)

\[\Delta \xi_m^{|d|}_{i,j,k} = \xi_m^{|d|}_{i,j,k} - \xi_m^{|d+1|}_{i,j,k}\]  
(2.17)

From (2.15), it can be seen that for calculating \(H_x|_{i,j,k}^{n-0.5}\), all the previous \(H_x|_{i,j,k}\) field components are required. However in FDTD simulations only the latest previous field components are stored due to enormous memory requirements. This difficulty is overcome by recursively calculating \(\Psi H_x|_{i,j,k}^{n-0.5}\).

\[
\Psi H_x|_{i,j,k}^{n+0.5} = \Psi H_x|_{i,j,k}^{n-0.5} + \Delta X_m^{|0|}_{i,j,k} H_x|_{i,j,k}^{n+0.5} + \Delta \xi_m^{|0|}_{i,j,k} \left( H_x|_{i,j,k}^{n-0.5} - H_x|_{i,j,k}^{n+0.5} \right) \\
+ \sum_{d=0}^{n-2} \left( \Delta X_m^{|d+1|}_{i,j,k} - \Delta X_m^{|d|}_{i,j,k} \right) H_x|_{i,j,k}^{n-d-0.5} \\
+ \left( \Delta \xi_m^{|d+1|}_{i,j,k} - \Delta \xi_m^{|d|}_{i,j,k} \right) \left( H_x|_{i,j,k}^{n-d-1.5} - H_x|_{i,j,k}^{n-d-0.5} \right) \\
= \Delta X_m^{|0|}_{i,j,k} H_x|_{i,j,k}^{n+0.5} + \Delta \xi_m^{|0|}_{i,j,k} \left( H_x|_{i,j,k}^{n-0.5} - H_x|_{i,j,k}^{n+0.5} \right) \\
+ \sum_{d=0}^{n-2} \Delta X_m^{|d+1|}_{i,j,k} H_x|_{i,j,k}^{n-d-0.5} + \Delta \xi_m^{|d+1|}_{i,j,k} \left( H_x|_{i,j,k}^{n-d-1.5} - H_x|_{i,j,k}^{n-d-0.5} \right)
\]  
(2.18)

For a multipole Debye model with \(P\) poles, \(\chi_m(\omega)\) and \(\chi_m(t)\) are given by

\[
\chi_m(\omega) = \sum_{p=1}^{P} \frac{\Delta \mu_p}{1 + j\omega \tau_p}; \quad \chi_m(t) = \sum_{p=1}^{P} \frac{\Delta \mu_p e^{-\frac{t}{\tau_p}}}{\tau_p}
\]  
(2.19)

By substituting (2.19) in (2.13), expressions for \(X_m^{|d|}_{i,j,k}\) and \(\xi_m^{|d|}_{i,j,k}\) are obtained.

\[
X_m^{|d|}_{i,j,k} = \sum_{p=1}^{P} X_m^{|d|}_{p,i,j,k}
\]

\[
X_m^{|d|}_{p,i,j,k} = \Delta \mu_p e^{-\frac{\Delta t}{\tau_p}} \left[ 1 - e^{-\frac{\Delta t}{\tau_p}} \right]
\]  
(2.20)

\[
\xi_m^{|d|}_{i,j,k} = \sum_{p=1}^{P} \xi_m^{|d|}_{p,i,j,k}
\]

\[
\xi_m^{|d|}_{p,i,j,k} = \Delta \mu_p e^{-\frac{\Delta t}{\tau_p}} \left[ \frac{\tau_p}{\Delta t} - \left( \frac{\tau_p}{\Delta t} + 1 \right) e^{-\frac{\Delta t}{\tau_p}} \right]
\]  
(2.21)

Using (2.20) and (2.21), it can be shown that

\[
X_m^{|d+1|}_{p,i,j,k} = e^{-\frac{\Delta t}{\tau_p}} X_m^{|d+1|}_{p,i,j,k}
\]

\[
\xi_m^{|d+1|}_{p,i,j,k} = e^{-\frac{\Delta t}{\tau_p}} \xi_m^{|d+1|}_{p,i,j,k}
\]  
(2.22)
\[
\Delta X^p_{m|_{i,j,k}}^{d+1} = e^{-\frac{\Delta t}{\tau_p}} \Delta X^p_{m|_{i,j,k}}^{d}
\]
\[
\Delta \xi^p_{m|_{i,j,k}}^{d+1} = e^{-\frac{\Delta t}{\tau_p}} \Delta \xi^p_{m|_{i,j,k}}^{d}
\]

(2.23)

For a multipole Debye media, \(\Psi_{Hx|_{i,j,k}}^{n+0.5}\) should be written as a summation of terms corresponding to each pole.

\[
\Psi_{Hx|_{i,j,k}}^{n+0.5} = \sum_{p=1}^{P} \Psi_{Hx,p|_{i,j,k}}^{n+0.5}
\]
\[
\Psi_{Hx,p|_{i,j,k}}^{n+0.5} = \Delta X^p_{m|_{i,j,k}}^{0} H_x^{n+0.5} + \Delta \xi^p_{m|_{i,j,k}}^{0} \left( H_x^{n-0.5} - H_x^{n+0.5} \right)
\]
\[
+ \sum_{d=0}^{n-2} \Delta X^p_{m|_{i,j,k}}^{d+1} H_x^{n-d-0.5} + \Delta \xi^p_{m|_{i,j,k}}^{d+1} \left( H_x^{n-d-1.5} - H_x^{n-d-0.5} \right)
\]

(2.24)

By substituting (2.23) in (2.24), a recursive expression for \(\Psi_{Hx,p|_{i,j,k}}^{n+0.5}\) is obtained.

\[
\Psi_{Hx,p|_{i,j,k}}^{n+0.5} = \Delta X^p_{m|_{i,j,k}}^{0} H_x^{n+0.5} + \Delta \xi^p_{m|_{i,j,k}}^{0} \left( H_x^{n-0.5} - H_x^{n+0.5} \right)
\]
\[
+ e^{-\frac{\Delta t}{\tau_p}} \Psi_{Hx,p|_{i,j,k}}^{n-0.5}
\]

(2.25)

The same procedure can be repeated for the remaining 5 field components. The FDTD-PLRC update equations for electrically and magnetically dispersive multipole Debye media in 3D domain is given by (2.26)-(2.31).
\[ E_{y|i,j,k}^{n+1} = \left( \frac{\Delta t}{\varepsilon_0 K^3|_{i,j,k}} \right) \left[ H_{y|i,j,k}^{n+0.5} - H_{x|i,j,k-1}^{n+0.5} \frac{\Delta z}{\Delta x} - H_{z|i,j,k}^{n+0.5} - H_{z|i-1,j,k}^{n+0.5} \frac{\Delta z}{\Delta y} \right] \\
+ \left( \frac{K4|_{i,j,k}}{K3|_{i,j,k}} \right) E_{y|i,j,k}^{n} + \Psi E_{y|i,j,k}^{n} \\
\Psi E_{y|i,j,k}^{n} = \sum_{p=1}^{P} \Psi E_{y,p|i,j,k}^{n} \\
\Psi E_{y,p|i,j,k}^{n} = \sum_{d=0}^{n-1} \Delta x^{p|d}_{e|i,j,k} E_{y|i,j,k}^{n-d} + \Delta \epsilon^{p|d}_{e|i,j,k} \left( E_{y|i,j,k}^{n-d-1} - E_{y|i,j,k}^{n-d} \right) \\
\Psi E_{y,p|i,j,k}^{n+1} = \Delta x^{p|0}_{e|i,j,k} E_{y|i,j,k}^{n+1} + \Delta \epsilon^{p|0}_{e|i,j,k} \left( E_{y|i,j,k}^{n} - E_{y|i,j,k}^{n+1} \right) \\
+ e^{-\frac{\Delta t}{r}} \Psi E_{y,p|i,j,k}^{n+1} \tag{2.27} \]

\[ E_{z|i,j,k}^{n+1} = \left( \frac{\Delta t}{\varepsilon_0 K^3|_{i,j,k}} \right) \left[ H_{y|i,j,k}^{n+0.5} - H_{y|i-1,j,k}^{n+0.5} \frac{\Delta x}{\Delta y} - H_{x|i,j,k}^{n+0.5} - H_{x|i,j-1,k}^{n+0.5} \frac{\Delta x}{\Delta z} \right] \\
+ \left( \frac{K4|_{i,j,k}}{K3|_{i,j,k}} \right) E_{z|i,j,k}^{n} + \Psi E_{z|i,j,k}^{n} \\
\Psi E_{z|i,j,k}^{n} = \sum_{p=1}^{P} \Psi E_{z,p|i,j,k}^{n} \\
\Psi E_{z,p|i,j,k}^{n} = \sum_{d=0}^{n-1} \Delta x^{p|d}_{e|i,j,k} E_{z|i,j,k}^{n-d} + \Delta \epsilon^{p|d}_{e|i,j,k} \left( E_{z|i,j,k}^{n-d-1} - E_{z|i,j,k}^{n-d} \right) \\
\Psi E_{z,p|i,j,k}^{n+1} = \Delta x^{p|0}_{e|i,j,k} E_{z|i,j,k}^{n+1} + \Delta \epsilon^{p|0}_{e|i,j,k} \left( E_{z|i,j,k}^{n} - E_{z|i,j,k}^{n+1} \right) \\
+ e^{-\frac{\Delta t}{r}} \Psi E_{z,p|i,j,k}^{n+1} \tag{2.28} \]

\[ H_{x|i,j,k}^{n+0.5} = \left( \frac{\Delta t}{\mu_0 K^2|_{i,j,k}} \right) \left[ E_{y|i,j,k+1}^{n} - E_{y|i,j,k}^{n} \frac{\Delta z}{\Delta y} - E_{z|i,j+1,k}^{n} - E_{z|i,j,k}^{n} \frac{\Delta z}{\Delta x} \right] \\
+ \left( \frac{K2|_{i,j,k}}{K1|_{i,j,k}} \right) H_{x|i,j,k}^{n-0.5} + \Psi H_{x|i,j,k}^{n-0.5} \\
K1|_{i,j,k} = \mu_0|_{i,j,k} + X_{m|_{i,j,k}}^{0} - \xi_{m|_{i,j,k}}^{0} \\
K2|_{i,j,k} = \mu_0|_{i,j,k} - \xi_{m|_{i,j,k}}^{0} \\
\Psi H_{x,p|i,j,k}^{n-0.5} = \sum_{p=1}^{P} \Psi H_{x,p|i,j,k}^{n-0.5} \\
\Psi H_{x,p|i,j,k}^{n-0.5} = \sum_{d=0}^{n-2} \Delta X^{p|d}_{m|i,j,k} H_{x|i,j,k}^{n-d-0.5} + \Delta \xi^{p|d}_{m|i,j,k} \left( H_{x|i,j,k}^{n-d-1.5} - H_{x|i,j,k}^{n-d-0.5} \right) \\
\Psi H_{x,p|i,j,k}^{n+0.5} = \Delta X^{p|0}_{m|i,j,k} H_{x|i,j,k}^{n+0.5} + \Delta \xi^{p|0}_{m|i,j,k} \left( H_{x|i,j,k}^{n-0.5} - H_{x|i,j,k}^{n+0.5} \right) \\
+ e^{-\frac{\Delta t}{r}} \Psi H_{x,p|i,j,k}^{n+0.5} \tag{2.29} \]
in this work is that the UPML absorbing slabs are placed only on the
FDTD update equations and its software implementation can be found in [18, 15]. The difference
wave can be of arbitrary frequency, incident angle and polarization. The derivation of UPML-
capable of absorbing electromagnetic waves incident on it with minimal reflections. The incident

2.2 Uniaxial Perfectly Matched Layer

The Uniaxial Perfectly Matched Layer (UPML) is an uniaxial anisotropic media that is
capable of absorbing electromagnetic waves incident on it with minimal reflections. The incident
wave can be of arbitrary frequency, incident angle and polarization. The derivation of UPML-
FDTD update equations and its software implementation can be found in [18, 15]. The difference
in this work is that the UPML absorbing slabs are placed only on the y-axis boundaries. The

\[
H_y|_{i,j,k}^{n+0.5} = \left( \frac{\Delta t}{\mu_0 K_1|_{i,j,k}} \right) \left[ \frac{E_x|_{i+1,j,k}^n - E_x|_{i,j,k}^n}{\Delta x} - \frac{E_z|_{i,j,k}^n - E_z|_{i-1,j,k}^n}{\Delta z} \right] \\
+ \left( \frac{K_2|_{i,j,k}}{K_1|_{i,j,k}} \right) H_y|_{i,j,k}^{n-0.5} + \frac{\Psi H_y|_{i,j,k}^{n-0.5}}{K_1|_{i,j,k}}
\]

\[
\Psi H_y|_{i,j,k}^{n-0.5} = \sum_{p=1}^P \Psi H_y|_{i,j,k}^{p,n-0.5}
\]

\[
\Psi H_y|_{i,j,k}^{p,n-0.5} = \sum_{d=0}^{n-2} \Delta X_p^d m_i,j,k H_y|_{i,j,k}^{n-d-0.5} + \Delta \xi_p^d m_i,j,k (H_y|_{i,j,k}^{n-d-1.5} - H_y|_{i,j,k}^{n-d-0.5})
\]

\[
\Psi H_y|_{i,j,k}^{p,n+0.5} = \Delta X_p^0 m_i,j,k H_y|_{i,j,k}^{n+0.5} + \Delta \xi_p^0 m_i,j,k (H_y|_{i,j,k}^{n+0.5} - H_y|_{i,j,k}^{n+0.5})
\]  

\[ + e^{-\frac{\Delta t}{\tau p}} \Psi H_y|_{i,j,k}^{p,n-0.5} \]  

\[
H_z|_{i,j,k}^{n+0.5} = \left( \frac{\Delta t}{\mu_0 K_1|_{i,j,k}} \right) \left[ \frac{E_x|_{i,j+1,k}^n - E_x|_{i,j,k}^n}{\Delta y} - \frac{E_y|_{i,j+1,k}^n - E_y|_{i,j,k}^n}{\Delta y} \right] \\
+ \left( \frac{K_2|_{i,j,k}}{K_1|_{i,j,k}} \right) H_z|_{i,j,k}^{n-0.5} + \frac{\Psi H_z|_{i,j,k}^{n-0.5}}{K_1|_{i,j,k}}
\]

\[
\Psi H_z|_{i,j,k}^{n-0.5} = \sum_{p=1}^P \Psi H_z|_{i,j,k}^{p,n-0.5}
\]

\[
\Psi H_z|_{i,j,k}^{p,n-0.5} = \sum_{d=0}^{n-2} \Delta X_p^d m_i,j,k H_z|_{i,j,k}^{n-d-0.5} + \Delta \xi_p^d m_i,j,k (H_z|_{i,j,k}^{n-d-1.5} - H_z|_{i,j,k}^{n-d-0.5})
\]

\[
\Psi H_z|_{i,j,k}^{p,n+0.5} = \Delta X_p^0 m_i,j,k H_z|_{i,j,k}^{n+0.5} + \Delta \xi_p^0 m_i,j,k (H_z|_{i,j,k}^{n+0.5} - H_z|_{i,j,k}^{n+0.5})
\]  

\[ + e^{-\frac{\Delta t}{\tau p}} \Psi H_z|_{i,j,k}^{p,n-0.5} \]  

2.2 Uniaxial Perfectly Matched Layer

The Uniaxial Perfectly Matched Layer (UPML) is an uniaxial anisotropic media that is
UPML update equations for the $y$-axis slabs are given by (2.32)-(2.43).

\[
D_{x|i+0.5,j,k}^{n+1} = C1Dx.D_{z|i+0.5,j,k} + C2Dx \left( \frac{H_{z|i+0.5,j+0.5,k}^{n+0.5} - H_{z|i+0.5,j+0.5,k-0.5}^{n+0.5}}{\Delta y} \right. \\
- \frac{H_{y|i+0.5,j,k+0.5}^{n+0.5} - H_{y|i+0.5,j,k-0.5}^{n+0.5}}{\Delta z} \right)
\]

\[
C1Dx = \frac{2\varepsilon_o - \sigma_y \Delta t}{2\varepsilon_o + \sigma_y \Delta t} ; \quad C2Dx = \frac{2\varepsilon_o \Delta t}{2\varepsilon_o + \sigma_y \Delta t} 
\] (2.32)

\[
D_{y|i,j+0.5,k}^{n+1} = D_{y|i,j+0.5,k}^n + \Delta t \left( \frac{H_{x|i,j+0.5,k+0.5}^{n+0.5} - H_{x|i,j+0.5,k-0.5}^{n+0.5}}{\Delta z} \right. \\
- \frac{H_{z|i,j+0.5,k}^{n+0.5} - H_{z|i,j+0.5,k-0.5}^{n+0.5}}{\Delta x} \right)
\]

\[
D_{z|i,j,k+0.5}^{n+1} = D_{z|i,j,k+0.5}^n + \Delta t \left( \frac{H_{y|i,j+0.5,k+0.5}^{n+0.5} - H_{y|i,j+0.5,k-0.5}^{n+0.5}}{\Delta x} \right. \\
- \frac{H_{x|i,j+0.5,k}^{n+0.5} - H_{x|i,j+0.5,k-0.5}^{n+0.5}}{\Delta y} \right)
\]

\[
E_{x|i+0.5,j,k}^{n+1} = E_{x|i+0.5,j,k}^n + \frac{D_{x|i+0.5,j,k}^{n+1} - D_{x|i+0.5,j,k}^n}{\varepsilon_o} 
\] (2.35)

\[
E_{y|i,j+0.5,k}^{n+1} = E_{y|i,j+0.5,k}^n + C2Ey.D_{y|i,j+0.5,k}^{n+1} - C3Ey.D_{y|i,j+0.5,k}^n 
\]

\[
C2Ey = \frac{2\varepsilon_o + \sigma_y \Delta t}{2\varepsilon_o^2} ; \quad C3Ey = \frac{2\varepsilon_o - \sigma_y \Delta t}{2\varepsilon_o^2} 
\] (2.36)

\[
E_{z|i,j+0.5,k}^{n+1} = C1Ez.E_{z|i,j+0.5,k}^n + C2Ez.(D_{z|i,j+0.5,k}^{n+1} - D_{z|i,j+0.5,k}^n) 
\]

\[
C1Ez = \frac{2\varepsilon_o - \sigma_y \Delta t}{2\varepsilon_o + \sigma_y \Delta t} ; \quad C3Ez = \frac{2\varepsilon_o}{\varepsilon_o(2\varepsilon_o + \sigma_y \Delta t)} 
\] (2.37)

\[
B_{x|i,j+0.5,k+0.5}^{n+1} = C1Bx.B_{x|i,j+0.5,k+0.5}^n \left( \frac{E_{z|i,j+1,k+0.5}^{n+1} - E_{z|i,j,k+0.5}^{n+1}}{\Delta y} \right. \\
- \frac{E_{y|i,j+0.5,k+1}^{n+1} - E_{y|i,j+0.5,k}^{n+1}}{\Delta z} \right)
\]

\[
C1Bx = \frac{2\varepsilon_o - \sigma_y \Delta t}{2\varepsilon_o + \sigma_y \Delta t} ; \quad C2Bx = \frac{2\varepsilon_o \Delta t}{2\varepsilon_o + \sigma_y \Delta t} 
\] (2.38)
\begin{align}
B_y|_{i+0.5,j,k+0.5}^{n+1} &= B_y|_{i+0.5,j,k+0.5}^{n+0.5} - \Delta t \left( \frac{E_x|_{i+0.5,j,k+0.5}^{n+1} - E_x|_{i+0.5,j,k}^{n+1}}{\Delta z} ight) \\
&\quad - \frac{E_z|_{i+0.5,j,k+0.5}^{n+1} - E_z|_{i+0.5,j,k}^{n+1}}{\Delta x} \\
B_z|_{i+0.5,j,k+0.5}^{n+1} &= B_z|_{i+0.5,j,k+0.5}^{n+0.5} - \Delta t \left( \frac{E_y|_{i+0.5,j,k+0.5}^{n+1} - E_y|_{i+0.5,j,k}^{n+1}}{\Delta x} ight) \\
&\quad - \frac{E_x|_{i+0.5,j,k+0.5}^{n+1} - E_x|_{i+0.5,j,k}^{n+1}}{\Delta y} \\
H_z|_{i+0.5,j,k+0.5}^{n+1} &= H_z|_{i+0.5,j,k+0.5}^{n+0.5} + \frac{1}{\mu_o} \left( B_x|_{i+0.5,j,k+0.5}^{n+1} - B_x|_{i+0.5,j,k+0.5}^{n+0.5} \right) \\
H_y|_{i+0.5,j,k+0.5}^{n+1} &= H_y|_{i+0.5,j,k+0.5}^{n+0.5} + C2Hy |_{i+0.5,j,k+0.5}^{n+1} - C3Hy |_{i+0.5,j,k+0.5}^{n+0.5} \\
C2Hy &= \frac{2\varepsilon_o + \sigma_y \Delta t}{2\mu_o \varepsilon_o} ; C3Hy = \frac{2\varepsilon_o - \sigma_y \Delta t}{2\mu_o \varepsilon_o} \\
H_z|_{i+0.5,j,k+0.5}^{n+1} &= C1Hz |_{i+0.5,j,k+0.5}^{n+1} + C2Hz \left( B_z|_{i+0.5,j,k+0.5}^{n+1} - B_z|_{i+0.5,j,k+0.5}^{n+0.5} \right) \\
C1Hz &= \frac{2\varepsilon_o - \sigma_y \Delta t}{2\varepsilon_o + \sigma_y \Delta t} ; C2Hz = \frac{2\varepsilon_o}{\mu_o(2\varepsilon_o + \sigma_y \Delta t)} 
\end{align}

Polynomial grading is used to define \( \sigma_y \) in the above equations and is given by

\begin{equation}
\sigma_y = \sigma_o (y/d)^m
\end{equation}

where \( m = 3.5 \), \( \sigma_o = \frac{16(m+1)}{2pd} \) and \( d \) is the length of the PML. \( y \) is the distance of the field component from the problem space UPML boundary. For instance, when using (2.44) in calculating \( C2Ey \), \( y \) should be the distance of \( E_y \) component from problem space UPML boundary. In this work, the number of Yee cells in each UPML slab is 12 (i.e. \( d = 12\Delta y \)). The UPML-main grid interface has to be carefully programmed. These two regions have two different formulations. The main grid PLRC field components are \( \vec{E}, \vec{H}, \vec{B}, \vec{D} \).

### 2.3 Boundary conditions

Periodic boundary conditions (PBC) are imposed on the \( x \) and \( z \) boundaries to simulate a 2D periodic array of pyramidal structures. In this work, incident plane wave is assumed to be normal
to the structure (i.e. wave vector is parallel to $y$-axis). This simplifies the implementation of the PBC tremendously. PBC for obliquely incident plane wave requires a more complex formulation known as the Split Field Method (SFM) [19]. SFM combined with dispersive FDTD can be even more complex. It is presumed that the immediate results obtained from a normal incidence case outweighs oblique incidence case. Therefore normal incidence PBC is preferred in this work. The FDTD unit cell bounded by PBC’s and UPML is shown in Fig. 2.2. In that figure, the red object is the scatterer, which is the coated pyramid in our case. PLRC update equations are used in the problem space. Symmetry and antisymmetry boundary conditions can further be used in our problem to reduce the computational burden by a factor of 4 [20]. These conditions are shown in Fig. 2.3. Fig. 2.3 is the front view of the pyramidal structure, where the dashed line represent the pyramidal edges. In the case of normal incidence on a doubly periodic structure each quadrant is indistinguishable. For an incident wave lineary polarized along $z$ direction, PEC and PMC boundaries as shown in Fig. 2.3 is used to simulate the pyramidal quadrant. We note that this boundary condition can only be applied for the case of a normally incident plane wave.

2.4 Parallel computation

FDTD simulation of this particular problem is computationally expensive for two reasons: 1) the structure is electrically large, thus requiring a large number of Yee cells for accurate simulation.
For instance, the nominal calibration target is a pyramid with base equal to $0.5''$ and height $2''$. In addition to this, there is a 2 mm coating and the reflectivity spectrum is required at frequency as high as 200 GHz. 2) memory usage is considerably increased due to the dispersive formulation which results in a large number of precomputed coefficients. This is more pronounced in this problem where material is both magnetically and electrically dispersive. In order to overcome these problems the code was parallelized using a one dimensional Domain Decomposition (DDM) technique [21]. The FDTD computational domain is divided into slabs along the $y$-axis direction as shown in Fig. 2.4. Each slab was handled on a separate processor. The surface fields at the interface between two slabs are passed through adjacent processors using Message Passing Interface (MPI) commands [22, 23]. The code was developed and executed on 200 cores of NOAA HPCS (High Performance Computing System) distributed computing system. A simple 1D DDM applied to 1D FDTD is depicted in Fig. 2.5. At time instant $n$, the $E_x$ field components need to be calculated. $E_x$ at $y_3$ belongs to core $j$ and $E_x$ at $y_1$ belongs to core $j + 1$. For core $j$ to calculate the $E_z$ at position $y_3$, core $j - 1$ passes the $H_z$ at position $y_1$ and time $n - 0.5$ to core $j$. Similarly, core $j$ passes $H_z$ value at position $y_2$ and time $n - 0.5$ to core $j + 1$. At time instant $n + 0.5$, the $H_z$ field components
need to be updated. In order to calculate \( H_z \) at \( y_1 \), core \( j \) passes the value of \( E_z \) at position \( y_3 \) and time \( n \) to core \( j - 1 \). Similarly, core \( j + 1 \) passes the value of \( E_z \) at position \( y_4 \) and time \( n \) to core \( j \). A C++ code of roughly 5000 lines was developed with the above mentioned formulations and boundary conditions. The main code (excluding postprocessing codes, library files etc.) is listed in appendix A.

### 2.5 Microwave characterization of radar absorbing material

The microwave absorbent material used in radiometer calibration references is usually a ferrous-doped epoxy resin composite. These materials consist of ferrous inclusions such as carbonyl iron powder in a dielectric epoxy resin matrix. Commonly used material designations are MF-110, MF-112, MF-114, MF-116 and MF-117 [24]. Since FDTD is a time domain method, the complex permittivity and permeability (along with DC conductivity) of these materials needs to be known as a function of frequency along the entire frequency axis. Inverse Fourier transformation can
then be applied to the frequency domain data to obtain the time domain electric and magnetic susceptibility functions, which are needed in FDTD modeling. The complex permittivity and permeability of microwave absorbent materials in the range 8 GHz to 26 GHz is reported in [24]. Waveguide based Transmission/Reflection (T/R) method is used to measure these quantities. In this work, we have used these measurements for material characterization.

The time domain electric and magnetic susceptibility functions are the impulse responses for the linear time-invariant system with electric/magnetic field intensities as input and polarization as the output. Hence the time domain susceptibility functions have to be causal. In order to ensure causality in the time domain, the transform in the frequency domain, i.e. complex susceptibilities, has to satisfy the Kramers - Kronig relation [25]. In short, the real and imaginary parts of the complex susceptibilities are required to be Hilbert-transform pairs. Moreover the complex susceptibilities should be Hermitian functions in order to assure a real time domain signal. Debye functions satisfy the above mentioned conditions and can be used for approximating frequency characteristics of composites for time domain electromagnetic modeling. The measured data is curve fitted to a series of Debye functions. The curve fitting procedure can be carried out using global optimization techniques such as Genetic Algorithm (GA) or simulated annealing [26].

GA is a robust, stochastic optimization method developed using the concepts of natural selection and evolution. GA is particularly suited for solving complex optimization problems, where traditional optimization methods fail [27]. In GA, a set of trial solutions termed as population is evaluated using a suitable fitness function. Each potential solution in the population is called a chromosome, which essentially consists of a string of variables (or genes). The fitness function will be a function of these variables. This function will evaluate the fitness of each solution in the current generation of population. The chromosomes for the next generation population are selected from the fittest current generation chromosomes. The algorithm is continued until the fitness function of a solution is below some prescribed value. The complex permittivity can be expressed as a series
19

of Debye terms as

\[ \varepsilon_r(\omega) = \varepsilon_\infty + \sum_{i=1}^{N} \frac{\Delta \varepsilon_i}{1 + j\omega \tau_i} \]  

(2.45)

where \( N \) denotes the number of Debye poles and \( \Delta \varepsilon_i, \tau_i \) represent the parameter for the \( i^{th} \) pole.

The fitness function \( J \left( \varepsilon_\infty, \{\Delta \varepsilon_i\}_{i=1}^{N}, \{\tau_i\}_{i=1}^{N} \right) \) can be written as

\[ J \left( \varepsilon_\infty, \{\Delta \varepsilon_i\}_{i=1}^{N}, \{\tau_i\}_{i=1}^{N} \right) = \sum_{m=1}^{M} \left| \varepsilon_{\text{r,meas}}(\omega_m) - \varepsilon_\infty - \sum_{i=1}^{N} \frac{\Delta \varepsilon_i}{1 + j\omega_m \tau_i} \right|^2 \]  

(2.46)

where \( M \) is the total number of measurement points, \( \varepsilon_{\text{r,meas}}(\omega_m) \) is the measured value of complex relative permittivity at the frequency \( \omega_m \). The value of \( M \) can be increased by using spline interpolation between measured frequency points. MATLAB® GA toolbox can be used to perform GA optimization with the fitness function given by (2.46). The GA code for complex permittivity fitting is listed in appendix B. The initial values for the Debye parameters should be selected reasonably for faster convergence. For instance, the relaxation time \( \tau_i \) can be bounded in the interval \([10^{-12}, 10^{-4}]\). A 5 pole Debye fit of the complex permittivity and permeability of MF112 and MF110 is shown in Figs. 2.6, 2.7 and Figs. 2.8, 2.9 respectively. The Debye parameters for MF112, MF114 and MF110 are listed in tables 2.1, 2.2 and 2.3 respectively. It should be remarked that the values of Debye coefficients listed in these tables are just mathematical quantities and do not have any physical significance. Moreover a comparison of FDTD results using a 4-pole and 5-pole Debye fit can be seen in Fig. 2.10. This was done for 0.5” base, 1:1 pyramid with 1 mm of MF112 coating. From the figure, it can be seen that there is not much difference between the two results, even though the 4 pole and 5 pole Debye fit parameters vary significantly.

2.6 Numerical validation

The numerical accuracy of the developed code was validated by estimating the transmission coefficient of an infinite slab of MF112 material. The slab is infinite along the \( x, z \) - directions and has a finite width \( d \) (m) along the \( y \)-direction. If a plane TEM wave is incident normally on one
side of the slab, the ratio of the magnitude of the transmitted wave (wave on the other side of
the slab) to incident wave as a function of frequency is the transmission coefficient, $|S_{21}(f)|$. The
analytical expression for $|S_{21}(f)|$ can be shown to be

$$|S_{21}(f)| = \left| \frac{4\eta_o\eta_s(f)}{e^{+jkd[\eta_s(f) + \eta_o]^2} - e^{-jkd[\eta_s(f) - \eta_o]^2}} \right|$$

$$\eta_s(f) = \eta_o \sqrt{\varepsilon_{rs}(f) \mu_{rs}(f)}$$

$$k_s(f) = \frac{2\pi f}{c} \sqrt{\varepsilon_{rs}(f) \mu_{rs}(f)}$$

(2.47)

where $\eta_o$ is the intrinsic impedance of free space and $\varepsilon_{rs}(f), \mu_{rs}(f)$ are the relative complex permittivity and permeability of the slab medium. The 5 pole Debye series fit was used for MF112 material in these simulations. The simulation was carried out by exciting a plane TEM wave with a Gaussian time signature. In the first step of the simulation, the incident field (i.e. electric field) is sampled at a suitable observation plane without the presence of an infinite slab. The transmitted field is sampled at this same observation point. A Fast Fourier Transform (FFT) is applied to these time series to obtain the frequency domain electric field intensity. The ratio of the transmitted electric field intensity to the incident field in the frequency domain gives $S_{21}(f)$. Transmission spectrum of a 3.1 mm MF112 slab obtained by the code is compared with analytical results in Fig. 2.11.

Table 2.1: 5 pole Debye parameters for MF-112

<table>
<thead>
<tr>
<th>$\varepsilon_{\infty}$=5.36759e+000</th>
<th>$\mu_{\infty}$=6.09930e-001</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta \varepsilon_i$</td>
<td>$\tau_i$</td>
</tr>
<tr>
<td>2.49845e-010</td>
<td>9.76769e-004</td>
</tr>
<tr>
<td>7.50170e-009</td>
<td>9.93680e-004</td>
</tr>
<tr>
<td>2.87363e-009</td>
<td>9.95832e-004</td>
</tr>
<tr>
<td>3.19969e-001</td>
<td>1.85652e-011</td>
</tr>
<tr>
<td>1.94940e-008</td>
<td>8.74442e-004</td>
</tr>
</tbody>
</table>
Table 2.2: 5 pole Debye parameters for MF-114

<table>
<thead>
<tr>
<th>$\varepsilon_\infty$=1.00102e+001</th>
<th>$\mu_\infty$=6.66467e-001</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta \varepsilon_i$</td>
<td>$\tau_i$</td>
</tr>
<tr>
<td>2.77091e+001</td>
<td>2.71277e-009</td>
</tr>
<tr>
<td>2.99984e+001</td>
<td>8.59746e-008</td>
</tr>
<tr>
<td>2.99384e+001</td>
<td>2.81365e-008</td>
</tr>
<tr>
<td>2.99613e+001</td>
<td>5.71671e-008</td>
</tr>
<tr>
<td>2.99977e+001</td>
<td>1.44155e-007</td>
</tr>
</tbody>
</table>

Figure 2.6: MF112 permittivity : Measured data and 5 pole Debye fit

Table 2.3: 5 pole Debye parameters for MF-110

<table>
<thead>
<tr>
<th>$\varepsilon_\infty$=3.91208e+000</th>
<th>$\mu_\infty$=8.70924e-001</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta \varepsilon_i$</td>
<td>$\tau_i$</td>
</tr>
<tr>
<td>3.37157e-010</td>
<td>9.77380e-004</td>
</tr>
<tr>
<td>2.88869e-010</td>
<td>9.73578e-004</td>
</tr>
<tr>
<td>1.21663e-010</td>
<td>6.78805e-004</td>
</tr>
<tr>
<td>1.63012e-001</td>
<td>1.99686e-011</td>
</tr>
</tbody>
</table>
Figure 2.7: MF112 permeability: Measured data and 5 pole Debye fit

Figure 2.8: MF110 permittivity: Measured data and 5 pole Debye fit
Figure 2.9: MF110 permeability: Measured data and 5 pole Debye fit

Figure 2.10: Reflectivity spectrum of 0.5" base,1:1 pyramid with 1 mm MF112 coating
Figure 2.11: Numerical validation of the code
Chapter 3

Results and Discussion

This chapter contains the results obtained from the FDTD code. Results are validated using commercial HFSS software and custom Geometric Optics (GO) code. Commonly used radiometer calibration references, or targets, are finite arrays of square metallic pyramids with a thin coating of microwave absorbing material. Such structures are designed to absorb incident plane waves across a broad range of frequencies, although in practice appear electrically flat and specular at low enough frequencies where the depth $h \ll \lambda$ and exhibit geometrical optics reflection at high enough frequencies where the period $\Lambda \gg \lambda$. Depending on the ratio of height to depth, coating thickness, and specific geometry of the pyramids a broad absorbing band in between these extremes can be tailored. The target geometries studied include square pyramids, circular pyramids and truncated square pyramids. The cross-sectional view of either the square or circular pyramid is given in Fig. 3.1. In order to assess the impact of material property variations on the emissivity spectrum a means of determining the uncertainty on the calculated target reflectivity is developed based on a procedure using the unscented transform [28]. Using this procedure the propagation of error from material parameters is efficiently tracked through the model to the resulting reflectivity product. Stability and convergence issues are also discussed.

3.1 Total reflectivity formulation

Consider a TEM plane wave propagating along $+y$ direction incident on the pyramidal structure. The wave is linearly polarized along $z$-direction. In order to get the reflectivity, two simu-
lations are performed: incident field simulation (without the scattering structure) and total field simulation (with the scattering structure). The incident electric field intensity can be denoted by 
\[ E_i^z(y; t) = g(t - \frac{y}{c}) \], where \( g(t - \frac{y}{c}) \) represents a transient travelling wave propagating in \( +y \) direction. In our study, we used a modulated Gaussian pulse excitation. However \( g(.) \) can be any other energy signal. Let \( y = 0 \) be the observation plane, i.e. the plane where the field values are sampled. Therefore the sampled field is \( E_i^z(y = 0; t) = g(t) \). By defining the Fourier transform, 
\[ FT\{g(t)\} = G(f) = |G(f)| e^{j\theta(f)} \], the electric field intensity vector can be represented as
\[ E_i^z(y; t) = 2 \int_{-\infty}^{\infty} |G(f)| \cos \left( 2\pi ft - 2\pi f \frac{y}{c} + \theta(f) \right) df \] (3.1)

In deriving (22), we have used the time shift property of Fourier transforms, and Hermitian conjugate symmetry of Fourier transforms of real valued signals. The total energy incident normally on an area \( \frac{A^2}{4} \) (m²), \( E_i^i(J) \) is related to the incident time instantaneous power density \( S_y^i(y = 0; t) = \frac{1}{\eta_0} E_i^{i2}(y = 0; t) \) (Wm⁻²) by
\[ E_i = \frac{A^2}{4} \int_{-\infty}^{\infty} S_y^i(y = 0; t)dt = \frac{A^2}{2\eta_0} \int_0^{\infty} |G(f)|^2 df \] (3.2)

where \( A \) (m) is the periodicity of the pyramidal base in either \( x \) or \( z \) direction. It should be noted that in the incident field simulation, the power density is independent of \( x, z \) directions.
In the total field simulation, the simulation is carried out with exact settings as that of incident field simulation but with the presence of the scattering structure. The scattered field $E_s^s, E_s^z, H_s^x, H_s^z$ is obtained by subtracting the incident field from the total field. The total energy scattered back through the area $\frac{\Lambda^2}{4}$, $E^s$ is given by,

$$E^s = -\int_0^\frac{\Lambda}{2} \int_0^\frac{\Lambda}{2} \int_{-\infty}^{\infty} E_s^s(x, z; t) H_s^z(x, z; t) |_{y=0}$$

$$-E_x^s(x, z; t) H_x^z(x, z; t) |_{y=0} dt dx dz$$  \hspace{1cm} (3.3)

After some mathematical manipulations $E^s$ can be expressed as,

$$E^s = \int_0^\infty E^s_f(f) df$$

$$E_f^s = 2 \int_0^\frac{\Lambda}{2} \int_0^\frac{\Lambda}{2} Re\{E_x^s(x, z; f) H_z^s(x, z; f)\}$$

$$-Re\{E_z^s(x, z; f) H_x^s(x, z; f)\} dx dz$$  \hspace{1cm} (3.4)

The ratio of the reflected energy to the incident energy as a function of frequency, reflectivity $r(f)$ can be obtained as,

$$r(f) = \frac{E_f^s(f)}{2\eta_o |G(f)|^2}$$  \hspace{1cm} (3.5)

### 3.2 Reflectivity spectrum of square pyramid array

The reflectivity spectrum of square pyramidal targets in the frequency range $[6, 26]$ GHz is shown in Fig. 3.2. The pyramid base is $\Lambda = 0.5''$ and the aspect ratios $(\frac{\Lambda}{d})$ used are $1 : 1, 1 : 2$ and $1 : 4$. The coating material is MF112 and the thicknesses used are $1$ mm, $2$ mm and $3$ mm. The simulations were performed with yee cell size of $\frac{\lambda_{min}}{140}$, where $\lambda_{min}$ corresponds to the free space wavelength at $26$ GHz. The FDTD results are compared with results obtained from the commercial FEM software HFSS (see Fig. 3.3). There is good agreement between the FDTD and HFSS results until $23$ GHz, thereby confirming the validity of the code and post-processing. HFSS simulations were carried out using wave port excitation, therefore it fails to take into account the higher order Floquet modes other than the fundamental specular mode. For $\Lambda = 0.5''$, $23.6$ GHz is the frequency at which the first non-specular Floquet mode starts propagating. It can be noticed that in all the
curves, there is a reflectivity jump at around this frequency. Generally, the 1:4 targets show low reflectivity in the range less than -40 dB. The reflectivity spectrum is characterized by an increase in reflectivity at lower frequencies. This behavior can be attributed to the fact that at lower frequencies, wavelength will be larger thereby the effective thickness of the coating will be reduced considerably. The black vertical line in Fig. 3.2 at 23.6 GHz highlights the reflectivity jump due to the first non-specular Floquet mode. The reflectivity spectrum till 26 GHz as shown in Fig. 3.2 is obtained using measured values of permittivity and permeability. The Debye series model detailed in the last chapter can be used to extrapolate the values of $\varepsilon'_r, \varepsilon''_r, \mu'_r, \mu''_r$ for frequencies greater than 26 GHz. The reflectivity spectrum of square pyramid array in the frequency range [6, 200] GHz for 1 mm, 2 mm coating cases are depicted in Fig. 3.4 and for 3 mm case in Fig. 3.5. For 3 mm case, Geometric Optics (GO) based reflectivity spectrum at high frequencies is also shown in Fig. 3.5.

Figure 3.2: Reflectivity of pyramidal targets with MF112 coating for various aspect ratios and coating thicknesses in the frequency range [6, 26] GHz.
3.3 Geometrical optics validation

Geometrical optics (GO) based ray tracing was used to evaluate the reflectivities of the targets at high frequencies. The normally incident rays are assumed to have an electric field intensity magnitude of unity. The incident rays will undergo multiple reflections between the pyamidal surfaces before emerging out of the structure. Each reflection will result in an attenuation of the ray magnitude. The ray reflection at the pyramidal surface was taken into account by using Fresnel reflection coefficients for either parallel or perpendicular polarization (depending on the polarization of electric field and which surface the ray strikes). The normal vector to the pyramidal surface depends on the facet on which the ray strikes. In Fig. 3.6, for the case of facet 1, the unit normal vector $\hat{n}$ is given by,

$$\hat{n} = \frac{-\hat{x} + \frac{\Lambda}{2h} \hat{z}}{\sqrt{1 + \frac{\Lambda^2}{4h^2}}}$$  \hspace{1cm} (3.6)

where $h$ is the height of the pyramid. Once $\hat{n}$ is known for the point of reflection, the reflection vector $\vec{r}$ (i.e. vector along the direction of reflected ray) can be calculated as follows.

$$\vec{r} = \hat{i} - 2(\hat{n} \cdot \hat{i})\hat{n}$$  \hspace{1cm} (3.7)

where $\hat{i}$ is the unit incident vector. Fresnel coefficients are computed for PEC backed MF112 coating. Code for GO reflectivity estimation is listed in appendix C. The GO results are plotted in Fig. 3.5 for the 3 mm coating case. There is good agreement between the FDTD and GO results for frequencies greater than $\sim 120$ GHz. Hence it can be concluded that to evaluate the pyramid reflectivity at frequencies greater than $\sim 120$ GHz, GO can be used instead of the computationally expensive dispersive FDTD. The GO method can also be used to estimate the reflectivities at oblique angles. In Figs. 3.4 and 3.5, it can be seen that all the plots exhibit a decaying sinusoidal spectrum at high frequencies. The amplitude of this oscillation is smaller for thicker coatings.
3.4 Floquet mode reflectivity analysis

The frequency domain scattered electric field $\vec{E}_{sc}$ in the case of the doubly periodic scattering structure can be written as

$$\vec{E}_{sc} = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} \vec{E}_{m,n} e^{-jk^m_x x - jk^m_z z} e^{-jk^n_y y}$$

where $(m,n)$ denote the Floquet mode indices and $\vec{E}_{m,n}$ is the corresponding Floquet mode amplitude. The Floquet mode denoted by $(m,n)$ propagates only when $k^m_y$ is real, thereby resulting in a cutoff frequency for the mode [29]. The cutoff frequency is given by $f_{m,n} = \frac{c}{\Lambda} \sqrt{m^2 + n^2}$. The Floquet mode amplitudes $\vec{E}_{m,n}$ can be calculated by performing the inverse Fourier series of (3.8).

A similar procedure can be used to obtain the Floquet mode magnetic field amplitudes $\vec{H}_{m,n}$. Using $\vec{E}_{m,n}$ and $\vec{H}_{m,n}$, the Floquet mode power density can be calculated. The $-y$ component of the Floquet mode power density and the incident power density can be used to calculate the Floquet mode reflectivity $r_{m,n}(f)$. The $r_{m,n}(f)$ spectrum for a pyramid with base $\Lambda = 0.5''$, $1:2$ aspect ratio and with 2 mm MF112 coating is shown in figure Fig. 3.7. In the figure, $r_{tot}$ represents the total reflectivity estimated using the scattered fields (i.e. fields without performing Floquet decomposition). $\Sigma r_{m,n}$ denotes the sum of reflectivities for indices $-7 < m, n < 7$. $r_{0,0}$ is the specular mode reflectivity with cutoff frequency 0 GHz. It can be seen in Fig. 3.7, that this is the only mode that contributes to the total reflectivity till the onset of the first nonspecular mode at 23.6 GHz. At 23.6 GHz four modes will become propagating modes carrying reflected power at oblique angles. These mode reflectivities are $r_{0,1}, r_{0,-1}, r_{1,0}$ and $r_{-1,0}$. The sum of these identical reflectivities (except for the direction of propagation) is shown as $4 \times r_{01}$ in Fig. 3.7.
3.5 Reflectivity spectrum of conical pyramid array

It is of interest to examine structures other than square pyramids, including circular pyramids and truncated square pyramids (i.e., frustra). Such novel structures admit to (respectively) either reduced edge diffraction and more uniform thermal profiles. Accordingly, periodic cone structures and truncated square pyramids are examined for their emissivity spectrum in comparison to regular square pyramids. The reflectivity spectrum of doubly periodic, conical pyramid array is estimated using a previously validated dispersive 3D FDTD code [30]. The pyramid is coated with Emerson-Cummings MF112 trademark material of varying normal thickness $t$ (i.e. thickness along the normal to the cone surface). The base diameter of the cone is $0.5''$ and the ratio of the base diameter to the height of the pyramid is varied as 1:1, 1:2 and 1:4. The center-to-center periodicity of the array is $\Lambda = \frac{0.5''}{\sqrt{2}}$. The reflectivity spectrum in the frequency range $[6, 200]$ GHz is depicted in Figs. (3.8)-(3.10) for $t = 1$ mm, 2 mm and 3 mm coating thickness cases. Further validation of these curves is performed in a manner similar to the procedure in [30]. HFSS is used to compute the normal reflectivity up to 33 GHz, where the first non-specular Floquet mode begins to propagate. The Geometric optics (GO) based high frequency reflectivity approximation is also shown for each case. There is good agreement of $\sim 2 - 3$ dB or better between HFSS and FDTD in most of the cases except for 1:4 pyramids with 1 mm and 3 mm coating thickness. The intercomparison of FDTD with GO results at high frequency is not as good as that of the square pyramid results in [30]. Nevertheless, the GO captures the general reflectivity trend at high frequencies, thus lending credibility to the full wave FDTD solution. It should be remarked that the permittivity and permeability of the MF112 material is available only within the frequency range $[6, 26]$ GHz [30]. Therefore, similar to [30], an extrapolation to higher frequencies using a Debye series was used. An electromagnetic analysis of circular pyramids is reported in [31], but analyzed from the standpoint of the bistatic scattering pattern rather than emissivity spectrum.

A comparison can be made between the calculated reflectivity spectrum of conical pyramids and that of square pyramids as shown in figure 3.11. The periodicity of both the circular and square
pyramid array is $0.5'' \times 0.707$. Only the cases for $t = 2$ mm are compared. For the 1:2 aspect ratio square pyramid, the mean reflectivity in the frequency range $[100, 200]$ GHz is -31.96 dB. On the other hand, for the 1:2.2 mm conical pyramid case, the mean high frequency reflectivity is -41.86 dB. The mean high frequency reflectivity is estimated over the frequency range where there is an oscillation in the reflectivity spectrum. A similar conclusion can be reached for 1:4 aspect ratio case where the mean reflectivity for square pyramids is -51.48 dB and that of conical pyramids is -64.76 dB. Another noticeable feature is that for conical pyramids, the high frequency reflectivity oscillations are more damped when compared to square pyramids. The standard deviation (std) of $r$ (dB) in the high frequency range for 1:2 square pyramid is 3.76 dB, while for 1:2 circular pyramid it is 2.58 dB. Similarly for 1:4 square pyramid, std is 2.44 dB and for 1:4 circular pyramid it is 1.67 dB. The damping is believed to be due to the broadening of resonant geometrical optics features that would otherwise be caused by specular reflections from flat pyramidal facets. This damping could be a favorable attribute for the construction of broadband targets. At the lowest frequency of 6 GHz the square pyramid reflectivity is lower than that of circular pyramids by $\sim 2$ dB. Other than these differences, the general reflectivity trends depend more on the coating thickness and aspect ratio than on the pyramid cross-sectional geometry. A solid conclusion that can be reached from figures (3.8)-(3.10) and results in [30] is that irrespective of pyramid cross-section, an aspect ratio of 1:4 or better is essential to achieve broadband reflectivity of -50 dB or less.

In order to ensure the correctness of the FDTD results, a convergence study was carried out for the 0.5'' base, 1:4, 1 mm MF112 coated circular pyramid by reducing the FDTD grid size in successive simulations. It was found that any grid size less than $\frac{\lambda}{42}$, where $\lambda$ corresponds to the wavelength in vacuum at 200 GHz, gives accurate results. The results are shown in Fig. 3.19. In addition to this, the distance of the observation plane [30] from the tip of the pyramid was varied and (as expected) the results did not show any difference (Fig. 3.20). Indeed, the observation plane is the virtual plane where the scattered fields are sampled in order to evaluate the reflectivity. Since the scatterer is doubly periodic, the reflectivity should be invariant with respect to the location of this plane. Therefore the only conclusion that can be reached is that for 1:4 circular pyramid cases,
the HFSS results are wrong.

### 3.6 Reflectivity spectrum of truncated square pyramid array

Due to the preponderance of absorption that occurs near the tips of the pyramids along with the sharp thermal gradient at the tip, it is desirable to truncate the tip so as to lower the location of the emission maximum into a region of more uniform temperature. Such truncation can also occur as the result of tip breakage during the manufacturing process, where sharp tips of composite absorber material can be difficult to mold. Accordingly, the reflectivity spectrum of a truncated pyramidal array is of interest. In order to study the effect of truncation a nominal square pyramidal calibration target array with base of $\Lambda = 0.5''$ and 1:4 aspect ratio was simulated. The pyramid is coated with MF112 absorbent material of thickness, $t = 2$ mm. Figure 3.12 shows the broadband reflectivity in the frequency range $[6, 200]$ GHz for various truncation percentages. The truncation percentage is the height of the pyramid which is removed from the top portion of the untruncated full pyramid as a percentage of the total height of untruncated pyramid. For the 1:4 case with $t = 2$ mm the total height of the untruncated pyramid is $2'' + 16.12$ mm = 66.92 mm. Therefore, a 5% truncation results in removing a pyramidal tip of height 3.34 mm from the untruncated pyramid. The results are validated at high frequencies using GO. Overall there is good agreement between the FDTD and GO solutions in the range $\sim 100 - 200$ GHz, except for the 1% truncation case where the plateau of the truncated pyramid is small enough so as to behave as a Rayleigh scatterer. In this case the GO model overestimates the scattering from the plateau, leading to an overestimation in reflectivity by up to several dB.

The results in figure (3.12) illustrate the importance of the sharpness of the pyramidal structures. A mere 2% or more truncation results in reflectivity values higher than -50 dB for frequencies greater than $\sim 70$ GHz. However, truncation does not affect the low frequency reflectivity performance. The reason for this is that the truncated tops behave as Rayleigh scatterer at low enough frequencies. The Rayleigh scattering dependence of $f^4a^6$, where $f$ is frequency and $a$ is the plateau size favors stronger scattering by the truncated tips only at the higher frequencies. We note that in
no case in Figure 3.12 is the truncation large enough to expose the metallic pyramidal core, although
the proximity of the core tip to the incident wave contributes to the tip scattering. Another geom-

etry investigated was the truncated pyramid with spherical top. The geometric details are given in
appendix D. It should be noted that spherical top can be fixed only on truncated conical pyramids.
For the case of 1:4, 2 mm conical pyramid, the truncation percentage has to be 23.7228%, for the
spherical top to have radius of curvature equal to 2 mm. The reflectivity spectrum of truncated,
conical pyramid array of aspect ratio 1:4 and 2 mm MF112 coating is shown in Fig. 3.13.

3.7 Target base width scaling

For ISSASI (International Space Station Atmospheric Sounding of Ice) and terahertz at-
mospheric sounding, the following radiometric frequency bands are of interest: 110 – 118 GHz,
spectrum of square pyramid array of base width 0.5′′, aspect ratio 1:4 and MF112 coating of 2
mm is given in Fig. 3.4. By scaling the target (i.e. decreasing the dimensions of the nominal
pyramid array), we expect to get same reflectivity spectrum at high frequencies. Moreover, FDTD
simulation of the nominal target (0.5′′, aspect ratio 1:4) is not possible at frequencies higher than
200 GHz, due to the unreasonable memory requirements. In Fig. 3.4, a reflectivity minima of
-67 dB can be noticed at 64.6 GHz. In order to shift this minima to 114 GHz (i.e. mid point of
the required 110-118 GHz), the base width is scaled as follows, 0.5′′ x 64.6/114 = 0.2833′′. The
reflectivity spectrum of this scaled pyramid is shown in Fig. 3.14. From the figure, it can be seen
that the minima had moved to 114 GHz. In the FDTD simulation, the Debye series extrapolation
of permittivity and permeability using measured data in the frequency range [8, 26] GHz is used.
This will not be an accurate model in this high frequency range.

3.8 Unscented transformation sensitivity analysis

The unscented transformation (UT) can be used to find the statistical moments of a nonlinear
transformation of a random vector. While increasingly used in extended Kalman filtering, it is
particularly useful when there is no analytical functional form for the transformation, but the values of the function can be calculated at discrete points in the random vector sample space \([32\). Let 

\[ y = h(\bar{x}), \bar{x} \in \mathbb{R}^n \]

be a function of the \(n\)-dimensional random vector \(\bar{x}\). The mean vector \(\bar{\mu}_x\) and autocovariance matrix \(C_x\) of \(\bar{x}\) are given by

\[
\bar{\mu}_x = E[\bar{x}]
\]

\[
C_x = E[(\bar{x} - \bar{\mu}_x)(\bar{x} - \bar{\mu}_x)^T]
\]

(3.9)

Using the above statistical information, \(2n\) vectors denoted as sigma points are defined. The sigma points \(\{\bar{s}^{(i)} | i = 1, 2, \cdots, 2n\}\) are calculated using the following expressions.

\[
\bar{s}^{(i)} = \bar{\mu}_x + (\sqrt{nC_x})^T_i ; i = 1, 2, \cdots, n
\]

\[
\bar{s}^{(i+n)} = \bar{\mu}_x - (\sqrt{nC_x})^T_i ; i = 1, 2, \cdots, n
\]

(3.10)

where \((\sqrt{nC_x})^T_i\) is the transpose of the \(i^{th}\) row of the matrix square root of \(nC_x\). An approximate value for the mean \(\mu_y\) and variance \(\sigma_y^2\) of the function \(y = h(\bar{x})\) can be calculated as follows.

\[
\mu_y \approx \frac{1}{2n} \sum_{i=1}^{2n} h(s^i)
\]

\[
\sigma_y^2 \approx \frac{1}{2n} \sum_{i=1}^{2n} (h(s^i) - \mu_y)^2
\]

(3.11)

The advantage of the UT is that it can be used to obtain approximate values for statistical moments without resorting to time consuming and often impractical Monte Carlo simulations.

### 3.9 Uncertainty in material properties

The uncertainty in the real and imaginary parts of the permittivity and permeability of the coating material arises due to two factors: (1) dielectric measurement errors and (2) variations in the composite material’s properties from batch to batch. In order to study the effect of this material property variation on the target reflectivity, other parameters such as target geometry, target dimensions, coating thickness etc are assumed to be constant. In that case, the target reflectivity \(r(\text{dB})\) is a nonlinear transformation of the four dimensional random vector \(\bar{x}\).

\[
r(\text{dB}) = h(\bar{x}) ; \bar{x} = [\varepsilon_r, \varepsilon_\prime, \mu_r, \mu'_r]^T
\]

(3.12)
where $\varepsilon'_r, \varepsilon''_r, \mu'_r, \mu''_r$ are the properties of the dispersive coating material.

In order to perform either a Monte Carlo (MC) simulation or UT, it is assumed that $\bar{x}$ is an Independent and Identically Distributed (IID) $n$-dimensional Gaussian random vector. Each component of the vector is a Gaussian random variable, which is statistically independent of the remaining elements of the vector. The mean of $\bar{x}$ as a function of frequency is obtained from [24]. Since the random vector $\bar{x}$ is IID, the covariance matrix will be a diagonal matrix, with each diagonal element being the variance of the individual random variable. In order to obtain the variance, it is assumed that the uncertainty bound found in the measurement [24] is equal to 3 times the standard deviation. For instance at 16 GHz, the constitutive properties of MF112 material are given by: $\varepsilon'_r = 5.582, \varepsilon''_r = 0.147, \mu'_r = 1.106, \mu''_r = 0.347$. The standard deviations of $\varepsilon'_r, \varepsilon''_r, \mu'_r, \mu''_r$ are given by 0.36/3, 0.4/3, 0.05/3, 0.15/3, respectively. The covariance matrix can be obtained by squaring these quantities and placing them in the diagonal elements of the matrix. Hence the covariance matrix at 16 GHz is given by $C_x = \text{diag}(0.0144, 0.0178, 0.0003, 0.0025)$, and independent errors are assumed.

For comparison purposes the mean vector and standard deviation information were also used to drive a MC simulation based upon a large set of Gaussian random variables generated for each material property at 16 GHz. HFSS software was used for the UT and MC calculations, with 3000 random MC vector samples generated. The target simulated was the nominal 0.5” base,1:4 square pyramid with 2 mm coating. The mean reflectivity and standard deviation obtained from MC analysis are -46.8864 dB and 0.8972 dB, respectively, at 16 GHz. The mean reflectivity and standard deviation obtained by UT simulation are -46.1964 dB and 0.7026 dB respectively. However, the MC simulation of 3000 sample points in HFSS required $\sim$ 27 hours whereas the UT simulation required only 7 minutes. Further confirmation of the utility of the UT method was found by repeating the same validation at 12 GHz. The MC results at this frequency are -45.3929 dB $\pm$ 1.1526 dB and UT results are -46.8732 dB $\pm$ 1.1477 dB. The reflectivity spectrum along with the uncertainty bounds in the frequency range [8.5, 18] GHz, at an interval of 0.125 GHz is shown in figure 3.15. In electromagnetics, the UT had been applied to uncertainty in antenna design and other related
electromagnetic compatibility problems [28, 33]. As far as we know, this is the first time that the UT has been applied to study the effect of uncertainty in dielectric properties.

### 3.10 Stability and convergence

In order to ensure stability in 3D FDTD, the Courant Friedrich Levy (CFL) conditions states that $\Delta t \leq \frac{\Delta x}{c\sqrt{3}}$. However even after satisfying this condition, instabilities could arise if the simulation is run for a large number of time steps, as shown in Fig. 3.16. It was found that this late time instability is more profound for 1:1 structures. But for 1:4 structures, large number of time steps did not result in instability. Due to the late time instability and excessive computational time for large number of simulation steps, the simulation should be stopped after reasonable number of steps. If this procedure do not capture all the scattered energy, Prony’s extrapolation [34] can be used extrapolate the scattered field time series at each pixel. An example plot of Prony’s extrapolation is shown in Fig. 3.17. However it was found that if the simulation is stopped after all the major scattered pulses are sampled, the application of Prony’s extrapolation did not made any difference. The effect of late instability on the reflectivity spectrum of 0.5”, 1:1, 3 mm MF112 coated square pyramid array is shown in Fig. 3.18. In the figure, if 12000 time steps are used for the simulation, the scattered time series is corrupted resulting in erroneous $r$(dB). Some of the simulation settings used are given below.

**Geometry:** 1:1, 0.5” pyramid with 3 mm MF112 coating

**Source:** Gaussian plane wave with bandwidth 26 GHz

**Discretization:** $ds = \frac{3e8}{26e9/120} = 9.615e-5$ m , $dt = ds/3e8/2 = 0.1602$ ps

**Simulation steps:** $T = 9000$

**Geometry:** 1:4, 0.5” pyramid with 3 mm MF112 coating

**Source:** Diff Gaussian plane wave with bandwidth [6,200] GHz

**Discretization:** $ds = \frac{3e8}{200e9/28} = 5.3571e-5$ m , $dt = ds/3e8/2 = 89.286$ fs

**Simulation steps:** $T = 16100$ or 18000 steps

For 1:4 structures, a discretization better than 1/48 of the wavelength in free space at 200 GHz is
not possible even with 200 cores of HPCS. The number of simulation steps used will not matter, if the number of steps is greater than 18000 or so. The issue of late time instability matters only for 1:1 structures.

3.11 Conclusions

The reflectivity of square pyramids, conical pyramids and truncated square pyramids with MF112 coatings has been studied over a broad range of frequencies commonly used in microwave radiometry. It is concluded that, all other things equal, conical pyramids could be slightly more suitable than square pyramids for constructing broadband targets. However, the reflectivity spectrum is more strongly dependent on pyramidal aspect ratio rather than the specific pyramid cross-sectional geometry. A base to height ratio of 1:4 or better is required to achieve normal reflectivity of less than $\sim -50$ dB for frequencies from $\sim 20 - 200$ GHz for $\Lambda = 0.5''$ pyramids. The study also explored the effect of sharpness of the pyramidal tips and the use of the unscented transformation to understand the effect of material property uncertainties. Future work includes estimating near field thermal emission from calibration targets and analysis of finite arrays of coated pyramids [35].
Figure 3.3: HFSS model for square pyramid specular mode reflectivity calculation

Figure 3.4: Reflectivity of pyramidal targets with MF112 coating for various aspect ratios and coating thicknesses 1 mm, 2 mm in the frequency range [6, 200] GHz
Figure 3.5: Reflectivity of pyramidal targets with MF112 coating for various aspect ratios and coating thickness 3 mm in the frequency range [6,200] GHz. GO validation is also shown for each case.

Figure 3.6: Front view of pyramid showing facets for GO modelling
Figure 3.7: Floquet mode reflectivity spectrum for 1:2, 0.5” base pyramid with 2 mm MF112 coating

Figure 3.8: Reflectivity spectrum of conical pyramid array with 1 mm of MF112 coating
Figure 3.9: Reflectivity spectrum of conical pyramid array with 2 mm of MF112 coating

Figure 3.10: Reflectivity spectrum of conical pyramid array with 3 mm of MF112 coating
Figure 3.11: Reflectivity spectrum comparison of circular and square pyramids with 2 mm MF112 coating. Periodicity is $\frac{0.57}{\sqrt{2}}$.

Figure 3.12: Reflectivity spectrum of truncated radiometer calibration targets
Figure 3.13: Reflectivity spectrum of truncated, conical pyramid array. Aspect ratio of 1:4 and 2 mm MF112 coating.

Figure 3.14: Reflectivity spectrum of square pyramid array. Base = 0.2833”, 1:4 aspect ratio and 2 mm MF112 coating.
Figure 3.15: Uncertainty bounds in the reflectivity spectrum of 1:4,2 mm square pyramid target with MF112 coating

Figure 3.16: Late time instability
Figure 3.17: Prony extrapolation of scattered field time series

Figure 3.18: Effect of late time instability on reflectivity spectrum
Figure 3.19: Convergence study: 0.5" base, 1:4 circular pyramid with 1 mm MF112 coating

Figure 3.20: Observation plane location: 0.5" base, 1:4 circular pyramid with 1 mm MF112 coating
Chapter 4

Transient analysis of dispersive, periodic structures for obliquely incident plane wave using Laguerre Marching-On-In Degree (MoD)

4.1 Introduction

Time domain analysis of dispersive, periodic structures in the case of oblique plane wave incidence is studied in this work. Laguerre Marching-on-in-Degree (MoD) is used for this purpose. Dispersive, periodic structures are important in several applications such as photonic crystals [36], electromagnetic band-gap materials [37] and frequency selective surfaces [38]. Time domain numerical methods are attractive in the sense that a wideband response can be obtained by a single simulation. A FDTD based method for the same problem can be found in [39]. In [39], split-field FDTD [19] with a new set of variables is used. However, it should be remarked that this technique is unstable for incident angles greater than 75°. Conventional dispersive FDTD is limited to dispersive models such as Debye, Lorentz and Drude. On the other hand, by virtue of the Laguerre decomposition, the presented method can simulate general dispersive media. It should also be remarked that, commercial time domain software such as CST Microwave Studio can not perform oblique time domain analysis of periodic structures.

In Laguerre MoD, the transient behaviour of electromagnetic fields is expressed in terms of weighted Laguerre polynomials or Laguerre basis functions [40]. Time domain operations such as time derivative and convolution can be handled in this Laguerre transformation. The Laguerre basis functions are convergent to zero as $t \to \infty$, there by ensuring unconditionally stability. In Laguerre-FDTD simulations, the CFL stability condition [15] can be circumvented. This will reduce the
simulation time considerably, particularly in the case of dispersive structures. Conventional FDTD simulation of dispersive media is carried out using Piecewise Linear Recursive Convolution (PLRC) [41]. PLRC can be quite time consuming, especially for 3D multiscale, dispersive structures [42]. Laguerre-FDTD simulation of dispersive media can be found in [43]. An alternative formulation based on the wave equation is reported in [44]. In this work, we use the former method. The use of Laguerre-FDTD for the analysis of periodic structures and oblique plane wave incidence is explained in [45]. However, the method is non-dispersive in nature and had been applied only to PEC structures. Laguerre basis functions along with time domain integral equations were used in [46, 47] to analyze transient scattering. A three dimensional implementation of Laguerre-FDTD is reported in [48].

In this work, an one dimensional periodic, electrically dispersive structure is analyzed when a plane wave is incident on it at any arbitrary angle. 2D TM polarization is assumed and periodicity is along the y-direction. However, it is straightforward to extend the formulations to 2D TE case. In section 2, the Laguerre MoD formulation for periodic, dispersive structure is explained. The Field Transformation Technique (FTM) [49] is used to handle the obliquely incident plane wave on a periodic structure. This is followed by applying the dispersive Laguerre formulation [43] to the field transformed variables. In section 3, the implementation of UPML for the case of dispersive, periodic, Laguerre domain is presented. In section 4, the formulation is numerically validated by estimating the transmission spectrum of a dispersive slab at various angles of incidence and comparing the results with analytical results.

4.2 Laguerre MoD formulation for periodic, dispersive structure

In the case of 2D TM polarization, the material properties and the field variables $E_z(x, y; t)$, $H_x(x, y; t), H_y(x, y; t)$ are independent of $z$ direction. Maxwell’s equations in time domain reduces to the following set of equations
\[
\frac{\partial E_z}{\partial y} = -\mu_0 \frac{\partial H_x}{\partial t} \tag{4.1}
\]
\[
\frac{\partial E_z}{\partial x} = \mu_0 \frac{\partial H_y}{\partial t} \tag{4.2}
\]
\[
\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} = \varepsilon_0 \frac{\partial}{\partial t} \left[ \varepsilon_r(t) \otimes E_z(t) \right] + \sigma E_z \tag{4.3}
\]

where \( \otimes \) denotes convolution integral. In the above equations, spatial dependence of the field variables and constitutive parameters are not explicitly shown. A non-magnetic media is assumed and the permittivity exhibits temporal dispersion. For time domain simulation of periodic structures with oblique plane wave incidence, the Field Transformation Technique (FTM) is used to remove the time shift across the grid. In FTM, \( \{E_z, H_x, H_y\} \) are replaced by new variables \( \{P_z, Q_x, Q_y\} \) according to the relations,

\[
\tilde{P}_z = \tilde{E}_z \exp(jk_0 \sin \phi y) \tag{4.4}
\]
\[
\tilde{Q}_x = \eta_0 \tilde{H}_x \exp(jk_0 \sin \phi y) \tag{4.5}
\]
\[
\tilde{Q}_y = \eta_0 \tilde{H}_y \exp(jk_0 \sin \phi y) \tag{4.6}
\]

where variables with breve accent are in the frequency domain and \( y \) is the direction of periodicity. \( \phi \) is the angle between the incident wave vector and \( x \) axis. By substituting (4.4)-(4.6) in frequency domain versions of (4.1)-(4.3), followed by converting the resultant equations into time domain yields the time domain equations for \( P_z, Q_x, Q_y \).

\[
\begin{align*}
-\frac{\partial P_z}{\partial y} + \frac{\sin \phi}{c} \frac{\partial P_z}{\partial t} &= \frac{1}{c} \frac{\partial Q_x}{\partial t} \tag{4.7} \\
\frac{\partial P_z}{\partial x} &= \frac{1}{c} \frac{\partial Q_y}{\partial t} \tag{4.8} \\
\frac{\partial Q_y}{\partial x} - \frac{\partial Q_x}{\partial y} + \frac{\sin \phi}{c} \frac{\partial Q_x}{\partial t} &= \frac{1}{c} \frac{\partial}{\partial t} \left[ \varepsilon_r(t) \otimes P_z(t) \right] + \sigma \eta_0 P_z \tag{4.9}
\end{align*}
\]

In the Laguerre-MoD formulation, the time dependant quantities are replaced by weighted summation of Laguerre basis functions [40],
\[ F(x, y; t) = \sum_{q=0}^{\infty} F_q(x, y) \phi_q(st) \]  

(4.10)

where \( F(x, y; t) \) can be either \( P_z, Q_x, Q_y \) or \( \varepsilon_r \), and \( \phi_q(st) \) represents the Laguerre basis function. The Laguerre basis function is given by

\[ \phi_q(st) = L_q(st) e^{-\frac{st}{2}} \]  

(4.11)

where \( L_q(st) \) is Laguerre polynomial of degree \( q \) and \( s \) is the time scaling parameter. By applying (4.10) to equations (4.7)-(4.9), followed by Galerkin testing \([40, 43]\] results in expressions for the \( q^{th} \) basis coefficient of the field variables,

\[ Q^q_x = \frac{2c}{s} \frac{\partial P^q_z}{\partial y} + \sin \phi P^q_z + \sum_{p=0}^{q-1} 2 \sin \phi P^p_z - 2Q^p_x \]  

(4.12)

\[ Q^q_y = \frac{2c}{s} \frac{\partial P^q_z}{\partial x} - 2 \sum_{q>0}^{q-1} Q^p_y \]  

(4.13)

\[ P^q_z = \left( \frac{2c}{\varepsilon^0_r + 2c\eta_0\sigma} \right) \left[ \frac{\partial Q^q_y}{\partial x} - \frac{\partial Q^q_x}{\partial y} \right] + \frac{s \sin \phi}{\varepsilon^0_r + 2c\eta_0\sigma} Q^q_x + \sum_{p=0}^{q-1} \sum_{q>0}^{q-1} \frac{2s \sin \phi}{\varepsilon^0_r + 2c\eta_0\sigma} Q^p_x - \varepsilon^0_r + 2c\eta_0\sigma P^p_z \]  

(4.14)

where we have used the following two properties of Laguerre transformation \([43, 44]\],

\[ \frac{\partial P_z}{\partial t} = \sum_{q=0}^{\infty} \left[ \frac{s}{2} P^q_z + \sum_{p=0}^{q-1} sP^p_z \right] \phi_q(st) \]  

(4.15)

\[ \frac{\partial}{\partial t} \left[ \varepsilon_r(t) \otimes P_z(t) \right] = \sum_{q=0}^{\infty} \left[ \frac{P^q_z \varepsilon^0_r}{2} + \sum_{p=0}^{q-1} \frac{P^p_z (\varepsilon^{q-p-1}_r + \varepsilon^{q-p}_r)}{q>0} \right] \phi_q(st) \]  

(4.16)

By substituting (4.12),(4.13) in (4.14), an expression for \( P^q_z \) which is amenable to marching-on-in degree can be obtained,
\[
(s \sin^2 \phi - \varepsilon^0_r - 2c \eta_0 \sigma) P_z^q + \frac{4 c^2}{s} \frac{\partial^2 P_z^q}{\partial x^2} + \frac{4 c^2}{s} \frac{\partial^2 P_z^q}{\partial y^2} - 4 c \sin \phi \frac{\partial P_z^q}{\partial y} = \\
\sum_{p=0}^{q-1} 4 c \frac{\partial Q_y^p}{\partial x} + 4 c \sin \phi \frac{\partial P_z^p}{\partial y} - 4 c \frac{\partial Q_x^p}{\partial y} + \left( \varepsilon_r^q - \varepsilon_r^{q-p-1} - 2 s \sin^2 \phi \right) P_z^p
\]  

(4.17)

The equation (4.17) is discretized using the conventional 2D FDTD TM grid [15] to obtain the discrete equation for software implementation.

\[
\frac{4 c^2}{s \Delta x^2} P_z^{q+1}_{i+1,j} + \frac{4 c^2}{s \Delta x^2} P_z^{q+1}_{i-1,j} + \left( \frac{4 c^2}{s \Delta y^2} - \frac{2 c \sin \phi}{\Delta y} \right) P_z^{q+1}_{i,j+1} + \left( \frac{4 c^2}{s \Delta y^2} + \frac{2 c \sin \phi}{\Delta y} \right) P_z^{q+1}_{i,j-1} + \\
\left( s \sin^2 \phi - \varepsilon^0_r - 2c \eta_0 \sigma_{i,j} - \frac{8 c^2}{s \Delta x^2} - \frac{8 c^2}{s \Delta y^2} \right) P_z^q_{i,j} = \\
\sum_{p=0}^{q-1} 4 c \frac{\Delta x}{\Delta y} \left[ Q_y^p_{i+1,j} - Q_y^p_{i,j} \right] + \frac{2 c \sin \phi}{\Delta y} \left[ P_z^q_{i,j+1} - P_z^q_{i,j-1} \right] - \frac{4 c}{\Delta y} \left[ Q_x^p_{i,j+1} - Q_x^p_{i,j-1} \right] \\
+ \left( \varepsilon_r^q - \varepsilon_r^{q-p-1} \right)_{i,j} - 2 s \sin^2 \phi \right) P_z^p_{i,j} 
\]

(4.18)

where the equation (4.18) along with the UPML boundary condition explained in the next section can be used to build a matrix equation to solve for \( P_z^q \) in the entire 2D domain. In (4.18), \( P_z_{i,j}, Q_x_{i,j} \) and \( Q_y_{i,j} \) are located at the center, bottom edge and left edge of the \((i,j)\)th Yee cell. Since the coefficients of \( P_z^q \) quantities on the left hand side of (4.18) do not change with each MoD step, the matrix can be decomposed using LU decomposition [50] resulting in faster calculation of \( P_z^q \) in each MoD step. For the generation of incident wave, a Total Field/Scattered Field (TF/SF) formulation similar to [45] is used. Once the \( P_z^q \) coefficients are obtained, \( Q_x^q, Q_y^q \) coefficients in the entire domain are obtained by using the discretized versions of (4.12),(4.13) given as follows.

\[
Q_x^q_{i,j} = \left( -\frac{2 c}{s \Delta y} + \frac{\sin \phi}{2} \right) P_z^q_{i,j} + \left( \frac{2 c}{s \Delta y} + \frac{\sin \phi}{2} \right) P_z^{q-1}_{i,j-1} + \sum_{p=0}^{q-1} \sin \phi \left[ P_z^p_{i,j} + P_z^p_{i,j-1} \right] - 2 Q_x^p_{i,j} 
\]

(4.19)
\[ Q_y^{p|l,j} = \frac{2c}{s\Delta x} P_z^{q|l,j} - \frac{2c}{s\Delta x} P_z^{q|l-1,j} - 2 \sum_{p=0}^{q-1} Q_y^{p|l,j} \]  

(4.20)

4.3 UPML formulation in Laguerre MoD

For UPML slabs on the x-axis boundaries [18, 15], the governing equations can be written as

\[
\begin{align*}
\frac{\partial \hat{E}_z}{\partial y} &= -j\omega \mu_os_x^{-1} \hat{H}_x \\
\frac{\partial \hat{E}_z}{\partial x} &= j\omega \mu_os_x \hat{H}_y \\
\frac{\partial \hat{H}_y}{\partial x} - \frac{\partial \hat{H}_z}{\partial y} &= j\omega \epsilon_os_x \hat{E}_z
\end{align*}
\]  

(4.21) \hspace{1cm} (4.22) \hspace{1cm} (4.23)

where \( s_x = 1 + \frac{k}{j\omega \epsilon_o} \). By applying (4.4)-(4.6) to (4.21)-(4.23), the following set of frequency domain equations are obtained.

\[
\begin{align*}
s_x \frac{\partial \tilde{P}_z}{\partial y} - jk_0 \sin \phi s_x \tilde{P}_z &= -\frac{j\omega}{c} \tilde{Q}_x \\
\frac{\partial \tilde{P}_z}{\partial x} &= \frac{j\omega s_x}{c} \tilde{Q}_y \\
\frac{\partial \tilde{Q}_y}{\partial x} - \frac{\partial \tilde{Q}_x}{\partial y} + jk_0 \sin \phi \tilde{Q}_x &= \frac{j\omega s_x}{c} \tilde{P}_z
\end{align*}
\]  

(4.24) \hspace{1cm} (4.25) \hspace{1cm} (4.26)

This is followed by the expansion of \( s_x \) and converting the resultant equations to time domain.

The time domain UPML equations for the field transformed variables are as follows.

\[
\begin{align*}
\varepsilon_o \frac{\partial^2 P_z}{\partial y \partial t} + \sigma_x \frac{\partial P_z}{\partial y} - \varepsilon_o \sin \phi \frac{\partial^2 P_z}{\partial t^2} &= -\frac{\varepsilon_o}{c} \frac{\partial^2 Q_x}{\partial t^2} + \frac{\sigma_x \sin \phi}{c} \frac{\partial P_z}{\partial t} \\
\frac{\partial P_z}{\partial x} &= \frac{1}{c} \frac{\partial Q_y}{\partial t} + \frac{\sigma_x}{c \varepsilon_o} Q_y \\
\frac{\partial Q_y}{\partial x} - \frac{\partial Q_x}{\partial y} + \sin \phi \frac{\partial Q_x}{\partial t} &= \frac{1}{c} \frac{\partial P_z}{\partial t} + \frac{\sigma_x}{c \varepsilon_o} P_z
\end{align*}
\]  

(4.27) \hspace{1cm} (4.28) \hspace{1cm} (4.29)

By applying Laguerre basis function expansion to equations (4.27)-(4.29), we arrive at the spatial differential equations for the Laguerre basis coefficients in the UPML domain.
\[
\left[ \frac{s \varepsilon_o}{2} + \sigma_x \right] \frac{\partial P^q_z}{\partial y} + \sum_{p=0}^{q-1} \left[ \frac{s \varepsilon_o \sin^2 \phi}{4c} - \frac{s \sigma_x \sin \phi}{2c} \right] P^p_z + \sum_{q>0}^{q-1} \left[ \frac{-s^2 \varepsilon_o \sin^2 \phi}{c} - \frac{s \sigma_x \sin \phi}{c} \right] + \sum_{q>0}^{q-1} (q - p) Q^p_y \right] \frac{\partial P^q_z}{\partial y} + \sum_{q>0}^{q-1} \left[ -s^2 \varepsilon_o \sin \phi \left( q^2 - 4c \right) - s \sigma_x \sin \phi c \right] P^q_z = \frac{Q^q_x}{4c} + \sum_{p=0}^{q-1} \sum_{q>0}^{q-1} \left( q - p \right) Q^p_y \]

(4.30)

\[
\frac{\partial P^q_z}{\partial x} = \left[ \frac{s}{2c} + \frac{\sigma_x}{c \varepsilon_o} \right] Q^q_y + \frac{s}{c} \sum_{p=0}^{q-1} Q^p_y
\]

(4.31)

\[
\frac{\partial Q^q_y}{\partial x} - \frac{\partial Q^q_x}{\partial y} + \frac{s \sin \phi}{2c} Q^q_y + \frac{s \sin \phi}{c} \sum_{p=0}^{q-1} Q^p_x = \left[ \frac{s}{2c} + \frac{\sigma_x}{c \varepsilon_o} \right] P^q_z + \frac{s}{c} \sum_{p=0}^{q-1} \sum_{q>0}^{q-1} P^p_z
\]

(4.32)

In the derivation of (4.30) from (4.27), we have used the following second derivative property of Laguerre decomposition.

\[
\frac{\partial^2 P^q_z}{\partial t^2} = s^2 \sum_{q=0}^{\infty} \left[ \frac{1}{4} P^q_z + \sum_{p=0}^{q-1} (q - p) P^p_z \right] \phi_q (st)
\]

(4.33)

The equations (4.30)-(4.32) are discretized on the 2D TMz Yee cell to obtain the discrete equations given by (4.34)-(4.36).

\[
Q^p_x |_{i,j} = A1_{i,j} P^p_z |_{i,j} + A2_{i,j} P^p_z |_{i,j-1} + \sum_{p=0}^{q-1} \left[ A3^{p,q} |_{i,j} P^p_z |_{i,j} + A4^{p,q} |_{i,j} P^p_z |_{i,j-1} - 4 (q - p) Q^p_y |_{i,j} \right]
\]

(4.34)

\[
Q^q_y |_{i,j} = A5_{i,j} P^q_z |_{i,j} - A5_{i,j} P^q_z |_{i,j-1} - \frac{s \Delta x}{c} A5_{i,j} \sum_{p=0}^{q-1} Q^p_y |_{i,j}
\]

(4.35)

\[
P^p_z |_{i,j} = A6_{i,j} Q^p_y |_{i,j+1} - A6_{i,j} Q^p_y |_{i,j} + A7_{i,j} Q^q_x |_{i,j+1} + A8_{i,j} Q^q_x |_{i,j} + \sum_{p=0}^{q-1} s \sin \phi \Delta x A6_{i,j} (Q^p_z |_{i,j} + Q^p_z |_{i,j+1}) - \frac{s \Delta x A6_{i,j}}{c} P^p_z |_{i,j}
\]

(4.36)
\[ A1_{i,j} = \frac{-2c}{s \Delta y} - \frac{4c \sigma_x^Q}{s^2 \varepsilon_o \Delta y} |_{i,j} + 0.5 \sin \phi + \frac{\sigma_x^Q |_{i,j} \sin \phi}{s \varepsilon_o} \] (4.37)

\[ A2_{i,j} = \frac{2c}{s \Delta y} + \frac{4c \sigma_x^Q}{s^2 \varepsilon_o \Delta y} |_{i,j} + 0.5 \sin \phi + \frac{\sigma_x^Q |_{i,j} \sin \phi}{s \varepsilon_o} \] (4.38)

\[ A3^{p,q}_{i,j} = \frac{-4c}{s \Delta y} + 2 \sin \phi (q - p) + \frac{2 \sigma_x^Q |_{i,j} \sin \phi}{s \varepsilon_o} \] (4.39)

\[ A4^{p,q}_{i,j} = \frac{4c}{s \Delta y} + 2 \sin \phi (q - p) + \frac{2 \sigma_x^Q |_{i,j} \sin \phi}{s \varepsilon_o} \] (4.40)

\[ A5_{i,j} = \frac{2c \varepsilon_o}{s \varepsilon_o \Delta x + 2 \Delta x \sigma_x^Q |_{i,j}} \], \quad A6_{i,j} = \frac{2c \varepsilon_o}{s \varepsilon_o \Delta x + 2 \Delta x \sigma_x^P |_{i,j}} \] (4.41)

\[ A7_{i,j} = \frac{(s \sin \phi \Delta x - 4c) \Delta x A6 |_{i,j}}{4c \Delta y}, \quad A8_{i,j} = \frac{(s \sin \phi \Delta x + 4c) \Delta x A6 |_{i,j}}{4c \Delta y} \] (4.42)

It should be noted that in (4.37)-(4.40), \( \sigma_x^Q |_{i,j} \) represents the value of \( \sigma_x \) at the position of \( Q_x \) vector in the Yee cell denoted by indices \((i, j)\). The same holds for \( \sigma_x^P |_{i,j} \) in (4.41).

By substituting (4.34),(4.35) in (4.36), a matrix equation given by (4.43) can be obtained for \( P_z^g \) coefficients.

\[
(A6 |_{i,j} A5 |_{i+1,j}) P_z^q |_{i+1,j} + (-A6 |_{i,j} A5 |_{i+1,j} - A6 |_{i,j} A5 |_{i,j} + A7 |_{i,j} A2 |_{i,j+1} + A8 |_{i,j} A1 |_{i,j} - 1) P_z^p |_{i,j} +

(A6 |_{i,j} A5 |_{i,j}) P_z^q |_{i-1,j} + (A7 |_{i,j} A1 |_{i,j+1}) P_z^q |_{i,j} + (A8 |_{i,j} A2 |_{i,j}) P_z^q |_{i,j-1} = \sum_{p=0}^{q-1} \frac{s \Delta x}{c} A6 |_{i,j} A5 |_{i+1,j}

\frac{Q_y^p |_{i+1,j} - \frac{s \Delta x}{c} A6 |_{i,j} A5 |_{i,j} Q_y^p |_{i,j} - A7 |_{i,j} A3^p |_{i,j+1} + (A7 |_{i,j} A4^p |_{i,j+1} + A8 |_{i,j} A3^p |_{i,j} - \frac{s \Delta x \Delta A6 |_{i,j}}{c}) P_z^p |_{i,j} - A8 |_{i,j} A4^p |_{i,j} P_z^p |_{i,j-1} + \left(4A7 |_{i,j} (q - p) - \frac{s \sin \phi \Delta x \Delta A6 |_{i,j}}{2c}\right) Q_x^p |_{i,j+1} +

\left(4A8 |_{i,j} (q - p) - \frac{s \sin \phi \Delta x \Delta A6 |_{i,j}}{2c}\right) Q_x^p |_{i,j} \] (4.43)

The equations (4.18),(4.43) are used to construct matrix equation for \( P_z \) in the computational domain. Along the \( y \) direction, the periodicity condition of \( P_z^g |_{i,0} = P_z^g |_{i,Y}, \quad Q_y^q |_{i,0} = Q_y^q |_{i,Y} \) is used, where \( Y \) is the number of Yee cells along the \( y \) direction. The code for this work is shown in appendix ???. MATLAB do not have built-in Laguerre polynomial functions. Therefore GSL (GNU Scientific Library) is used to generate the Laguerre coefficients for the source functions.
4.4 Numerical validation

In order to numerically validate the formulations, the transmission spectra $S_{12}(dB)$ of a dispersive slab is estimated at various angles of incidence using this method and the results are compared with analytical results. The slab material is a double pole Debye media. The relative permittivity of the double pole Debye media is given by,

$$
\varepsilon_r(\omega) = \varepsilon_\infty + \sum_{p=1}^{2} \frac{\varepsilon_s,p - \varepsilon_\infty}{1 + j\omega \tau_p} \tag{4.44}
$$

where $\varepsilon_\infty = 3, \varepsilon_{s,1} = 4, \tau_1 = 7 \times 10^{-8}, \varepsilon_{s,2} = 4.5, \tau_2 = 2 \times 10^{-10}$. The Laguerre decomposition coefficients $\varepsilon_q^2$ for (4.44) can be obtained either by using analytical expressions given in [43] or by numerical quadrature of the product of time domain version of (4.44) and the corresponding Laguerre basis function. The spatial discretization is $\Delta x = 24.98 \, \mu m$ and a 15 layer UPML is used. The width of the slab is 100 cells along the x-direction. The number of basis functions, $M = 339$ and the simulation duration is $T_f = 1.3374 \, ns$. The scaling factor is set to $s = M/T_f$. A differentiated Gaussian plane wave is used as excitation.

$$
f(t) = \frac{2(t_o - t)}{\tau^2} \exp \left[ -\frac{(t-t_o)^2}{\tau^2} \right] \tag{4.45}
$$

where $\tau = 4.8274 \times 10^{-12}$ and $t_o = 4\tau$. The angle of incidences used are: $0^\circ, 45^\circ, 60^\circ$ and $75^\circ$. The intercomparison of the MoD results with analytical results are shown in figures (4.1)-(4.4). There is good agreement for incident angles $0^\circ, 45^\circ$ and $60^\circ$. For the $75^\circ$ case, there is a slight mismatch. For most practical problems, the incident angle is almost always less than $\sim 70^\circ$. Since Laguerre MoD uses an entire domain temporal basis function, the accuracy of Laguerre MoD based methods may be inferior to FDTD based methods. However, the validation confirms the correctness of the formulations in both the main grid and UPML, with the added advantage of unconditional stability, less simulation time and ability to handle general dispersive media.
4.5 Conclusions

A novel method for transient electromagnetic analysis of dispersive periodic structures at off-normal angles of incidence is described. By using Laguerre-MoD, unconditional stability is assured. Another contribution of this work is the implementation of UPML in Laguerre MoD using field transformed variables. The developed method can be used to obtain broadband results at incidence angles less than 75°. In comparison to previous FDTD based methods for solving the same problem, this method do not exhibit instability at grazing angles of incidence and can handle general dispersive media. Future work include detailed error analysis and the extension of this method to 3D problems.
Figure 4.1: Transmission spectrum of dispersive slab at 0° angle of incidence

Figure 4.2: Transmission spectrum of dispersive slab at 45° angle of incidence
Figure 4.3: Transmission spectrum of dispersive slab at 60° angle of incidence

Figure 4.4: Transmission spectrum of dispersive slab at 75° angle of incidence
Chapter 5

Near Field Thermal Emission

If the temperature of a medium is not uniform, the emissivity definition defined by is no longer applicable. In order to calculate thermal emission from a medium with nonuniform temperature profile, Fluctuation Dissipation Theorem (FDT) must be used. FDT gives an expression for the correlation between thermal current densities at a point as a function of temperature, frequency and material properties at that point [51, 52]. Using FDT, Dyadic Green’s Function (DGF) can be used to obtain the power density of the fluctuating electromagnetic fields. Thermal emission from simple geometries such as sphere [53], infinite cylinder [54] and layered media [55] had been done mostly for theoretical remote sensing studies. Lately, near field thermal emission using FDT-DGF methodology is applied to radiative heat transfer at nanoscale dimensions [52, 56, 57].

It is not possible to calculate thermal emission from calibration targets of pyramidal geometry using FDT-DGF method, because DGF is not known for such geometries. A stochastic numerical method may be the only solution. The objectives of this chapter are three fold: 1) Thermal emission due to fluctuating magnetic current density sources is not studied well in literature. 2) Calculation of near field thermal emission from simple geometries is essential for the validation of any numerical method, which can for solve thermal emission from arbitrary geometries. A thorough understanding of the theory and some experience is required to do novel work in this area 3) Explore the boundaries between classical radiative transfer theory and electromagnetic wave theory. An approach based on principle of detailed balance is attempted in [10]. This method which may be more simpler than either DGF-FDT or any stochastic numerical method should be revisited once
some solid results are obtained from the FDT-DGF method. Stochastic numerical methods which should be attempted include the Polynomial Chaos based methods such Wiener Chaos Expansion (WCE) [58] or Stochastic Finite Element Method (SFEM) [59]. As a first step, near field thermal emission from an infinite circular cylinder made of MF112 material is calculated using the FDT-DGF method. This is followed by development of a stochastic numerical method for calculation of the near field thermal emission from arbitrarily shaped 2D structures and validating the results with circular cylinder analytical results.

5.1 Fluctuation Dissipation Theorem

FDT for electric and magnetic current densities are given by the following equations.

\[
\langle J_j(\vec{r}, \omega) J_k^*(\vec{r}', \omega') \rangle = \frac{\omega \varepsilon_o \varepsilon'_\tau}{\pi} \Theta(\omega, T) \delta(\omega - \omega') \delta(\vec{r} - \vec{r}') \delta_{jk} \tag{5.1}
\]

\[
\langle M_j(\vec{r}, \omega) M_k^*(\vec{r}', \omega') \rangle = \frac{\omega \mu_o \mu'_\tau}{\pi} \Theta(\omega, T) \delta(\omega - \omega') \delta(\vec{r} - \vec{r}') \delta_{jk} \tag{5.2}
\]

\[
\Theta(\omega, T) = \frac{\hbar \omega}{\exp \left( \frac{\hbar \omega}{k_B T} \right) - 1} \tag{5.3}
\]

where \( j, k \) can be either \( x, y \) or \( z \), \( T(\vec{r}) \) is the temperature in Kelvin, \( k_B \) is the Boltzmann’s constant equivalent to \( 1.38 \times 10^{-23} \text{J/K} \), \( \hbar \) is the reduced Planck constant and other variables have the usual meaning. As per these equations, the current densities are spatially incoherent and orthogonal current density components are uncorrelated (e.g. \( J_x, J_y \)). The electric and magnetic current densities are uncorrelated [60].

\[
\langle J_j(\vec{r}, \omega) M_k^*(\vec{r}', \omega') \rangle = 0 \tag{5.4}
\]

5.2 Dyadic Green’s Function

Dyadic Green Function (DGF) relates a vector source to a vector field. A very clear introduction to DGF is given in first four chapters of [61]. Therefore only important results without proofs are presented in this chapter. However it should be noted that exposure to material in [61] is required to understand this chapter. Let the source electric volume current density be given
by \( \vec{J}(\vec{r}) = \frac{1}{i\omega\mu_o} \left[ \delta(\vec{r} - \vec{r}')\hat{x} + \delta(\vec{r} - \vec{r}')\hat{y} + \delta(\vec{r} - \vec{r}')\hat{z} \right] \). Let \( \vec{G}_{ex}(\vec{r}, \vec{r}') \), \( \vec{G}_{mx}(\vec{r}, \vec{r}') \) be the electric field intensity and magnetic field intensity generated due to the source \( \delta(\vec{r} - \vec{r}')\hat{x} \). Similarly, \( \delta(\vec{r} - \vec{r}')\hat{y} \), \( \delta(\vec{r} - \vec{r}')\hat{z} \) generates \( \vec{G}_{ey}(\vec{r}, \vec{r}') \), \( \vec{G}_{my}(\vec{r}, \vec{r}') \) and \( \vec{G}_{ez}(\vec{r}, \vec{r}') \), \( \vec{G}_{mz}(\vec{r}, \vec{r}') \) respectively. The electric DGF, \( \vec{G}_e(\vec{r}, \vec{r}') \) and magnetic DGF, \( \vec{G}_m(\vec{r}, \vec{r}') \) is then given by the following expressions.

\[
\vec{G}_e(\vec{r}, \vec{r}') = \vec{G}_{ex}(\vec{r}, \vec{r}')\hat{x} + \vec{G}_{ey}(\vec{r}, \vec{r}')\hat{y} + \vec{G}_{ez}(\vec{r}, \vec{r}')\hat{z} \quad (5.5)
\]

\[
\vec{G}_m(\vec{r}, \vec{r}') = i\omega\mu_o \left[ \vec{G}_{mx}(\vec{r}, \vec{r}')\hat{x} + \vec{G}_{my}(\vec{r}, \vec{r}')\hat{y} + \vec{G}_{mz}(\vec{r}, \vec{r}')\hat{z} \right] \quad (5.6)
\]

As per Maxwell’s equations, \( \{\vec{G}_{ex}, \vec{G}_{ey}, \vec{G}_{ez}\} \) are related to \( \{\vec{G}_{mx}, \vec{G}_{my}, \vec{G}_{mz}\} \) through the following equations.

\[
\nabla \times \vec{G}_{ex} = i\omega\mu_o \vec{G}_{mx} \quad (5.7)
\]

\[
\nabla \times \vec{G}_{ey} = i\omega\mu_o \vec{G}_{my} \quad (5.8)
\]

\[
\nabla \times \vec{G}_{ez} = i\omega\mu_o \vec{G}_{mz} \quad (5.9)
\]

\[
\nabla \times \vec{G}_{mx} = \delta(\vec{r} - \vec{r}')\hat{x} - i\omega\varepsilon_o \vec{G}_{ex} \quad (5.10)
\]

\[
\nabla \times \vec{G}_{my} = \delta(\vec{r} - \vec{r}')\hat{y} - i\omega\varepsilon_o \vec{G}_{ey} \quad (5.11)
\]

\[
\nabla \times \vec{G}_{mz} = \delta(\vec{r} - \vec{r}')\hat{z} - i\omega\varepsilon_o \vec{G}_{ez} \quad (5.12)
\]

By using the above equations, a “Maxwell’s equation analogue” for DGFs can be obtained.

\[
\nabla \times \vec{G}_e(\vec{r}; \vec{r}') = \vec{G}_m(\vec{r}; \vec{r}') \quad (5.13)
\]

\[
\nabla \times \vec{G}_m(\vec{r}; \vec{r}') = \nabla \times \left[ \vec{G}_e(\vec{r}; \vec{r}') + k^2 \vec{G}_e(\vec{r}; \vec{r}') \right] \quad (5.14)
\]

\[
\nabla \cdot \vec{G}_e(\vec{r}; \vec{r}') = -\frac{1}{k^2} \nabla \delta(\vec{r} - \vec{r}') \quad (5.15)
\]

\[
\nabla \cdot \vec{G}_m(\vec{r}; \vec{r}') = 0 \quad (5.16)
\]

where \( \vec{I} = \hat{x}\hat{x} + \hat{y}\hat{y} + \hat{z}\hat{z} \) is the unit dyadic. From the above equations, “Helmholtz equation analogue” for DGFs are obtained.

\[
\nabla \times \nabla \times \vec{G}_e - k^2 \vec{G}_e = \vec{I}\delta(\vec{r} - \vec{r}') \quad (5.17)
\]

\[
\nabla \times \nabla \times \vec{G}_m - k^2 \vec{G}_m = \nabla \times \left[ \vec{I}\delta(\vec{r} - \vec{r}') \right] \quad (5.18)
\]
The equations (5.17),(5.17) are quite important in the sense that the derivation of DGF for various simple geometries starts with expressing the dyadic on the right hand side of these equations as an eigen function expansion of Vector Wave Functions (VWF). The electric field intensity $\vec{E}(\vec{r})$ due to an arbitrary source $\vec{J}(\vec{r'})$ can be defined by the principle of superposition.

$$\vec{E}(\vec{r}) = i\omega\mu_o \iiint_V \overline{G}_{eo}(\vec{r};\vec{r'}) \cdot \vec{J}(\vec{r'}) d\vec{v}' \quad (5.19)$$

By applying Maxwell’s equation to (5.19) and using (5.13), the expression for magnetic field intensity is obtained.

$$\vec{H}(\vec{r}) = \iiint_V \overline{G}_{mo}(\vec{r};\vec{r'}) \cdot \vec{J}(\vec{r'}) d\vec{v}' \quad (5.20)$$

where $\overline{G}_{eo}(\vec{r};\vec{r'})$, $\overline{G}_{mo}(\vec{r};\vec{r'})$ are the free space DGFs given by

$$\overline{G}_{eo}(\vec{r};\vec{r'}) = \left[ \frac{1}{k^2} + \frac{\nabla \nabla}{k^2} \right] g_o(\vec{r};\vec{r'}) \quad (5.21)$$

$$\overline{G}_{mo}(\vec{r};\vec{r'}) = \nabla \times \left[ \bar{T}_{g_o}(\vec{r};\vec{r'}) \right] \quad (5.22)$$

g_o(\vec{r};\vec{r'}) is the scalar free space Green’s function, $g_o(\vec{r};\vec{r'}) = \frac{e^{ik|\vec{r}-\vec{r}'|}}{|\vec{r}-\vec{r}'|}$. When magnetic sources are also present, the electric and magnetic field intensities are given as follows.

$$\vec{E}(\vec{r}) = \iiint_V \left[ i\omega\mu_o \overline{G}_{eo}(\vec{r};\vec{r'}) \cdot \vec{J}(\vec{r'}) - \overline{G}_{mo}(\vec{r};\vec{r'}) \cdot \bar{M}(\vec{r'}) \right] d\vec{v}' \quad (5.23)$$

$$\vec{H}(\vec{r}) = \iiint_V \left[ i\omega\varepsilon_o \overline{G}_{eo}(\vec{r};\vec{r'}) \cdot \bar{M}(\vec{r'}) + \overline{G}_{mo}(\vec{r};\vec{r'}) \cdot \vec{J}(\vec{r'}) \right] d\vec{v}' \quad (5.24)$$

### 5.3 Formulation of stochastic electromagnetic power density

The quantity of interest in near field thermal emission problems is the stochastic electromagnetic power density given by the following equation.

$$\langle \vec{S}(\vec{r}) \rangle = \frac{1}{2} \text{Re} \left\{ \langle \vec{E}(\vec{r}) \times \vec{H}^*(\vec{r}) \rangle \right\} \quad (5.25)$$

We will first show how an expression for $\langle \vec{S}_z(\vec{r}) \rangle$ is derived. From (5.23),(5.24) expressions for $E_x(\vec{r}), E_y(\vec{r}), H^x(\vec{r}), H^y(\vec{r})$ relating it to source current densities are obtained.

$$E_x(\vec{r}) = \iiint_V \left[ i\omega\mu_o \sum_{l=x,y,z} G^{x}_{eo}(\vec{r};\vec{r'}) J_l(\vec{r'}) - \sum_{n=x,y,z} G^{xn}_{mo}(\vec{r};\vec{r'}) M_n(\vec{r'}) \right] d\vec{v}' \quad (5.26)$$
\[
E_y(\vec{r}) = \iiint_V \left[ i\omega \mu_o \sum_{l=x,y,z} G_{eo}^{yl}(\vec{r}; \vec{r}') J_l(\vec{r}') - \sum_{n=x,y,z} G_{mo}^{yn}(\vec{r}; \vec{r}') M_n(\vec{r}') \right] dv'
\] (5.27)

\[
H^x_\perp(\vec{r}) = \iiint_V \left[ -i\omega \varepsilon_o \sum_{l=x,y,z} G_{eo}^{xl}(\vec{r}; \vec{r}') M_l^x(\vec{r}') + \sum_{n=x,y,z} G_{mo}^{xn}(\vec{r}; \vec{r}') J_n^x(\vec{r}') \right] dv'
\] (5.28)

\[
H^y_\perp(\vec{r}) = \iiint_V \left[ -i\omega \varepsilon_o \sum_{l=x,y,z} G_{eo}^{yl}(\vec{r}; \vec{r}') M_l^y(\vec{r}') + \sum_{n=x,y,z} G_{mo}^{yn}(\vec{r}; \vec{r}') J_n^y(\vec{r}') \right] dv'
\] (5.29)

\[
\langle E_x(\vec{r})H^y_\perp(\vec{r}) - E_y(\vec{r})H^x_\perp(\vec{r}) \rangle = \sum_{l=x,y,z} \sum_{n=x,y,z} \int_V dv' \int_V dv'' \left[ i\omega \mu_o G_{eo}^{xl}(\vec{r}; \vec{r}') G_{mo}^{yn}(\vec{r}; \vec{r}'') \left( J_l(\vec{r}') J_n^y(\vec{r}'') \right) \right]
\]
\[
+ \left[ i\omega \varepsilon_o G_{mo}^{xn}(\vec{r}; \vec{r}') G_{eo}^{yl}(\vec{r}; \vec{r}'') \left( M_n(\vec{r}') M_l^y(\vec{r}'') \right) \right]
\]
\[
- \left[ i\omega \mu_o G_{eo}^{pl}(\vec{r}; \vec{r}') G_{mo}^{zn}(\vec{r}; \vec{r}'') \left( J_l(\vec{r}') J_n^x(\vec{r}'') \right) \right]
\]
\[- \left[ i\omega \varepsilon_o G_{mo}^{pn}(\vec{r}; \vec{r}') G_{eo}^{xl}(\vec{r}; \vec{r}'') \left( M_n(\vec{r}') M_l^x(\vec{r}'') \right) \right]
\]
(5.30)

In deriving above equation, we have used the property (5.4) resulting in the removal of cross product terms (terms with cross-correlation of electric and magnetic current densities). The components of the DGF matrix are indicated by superscripts along with \( G_{eo} \). For instance, \( G_{eo}^{pl} \) relates the field component \( E_y \) to \( J_x \). By substituting the FDT expressions (5.1),(5.2) in (5.30), the double summation and double volume integral collapses to single summation over the coordinate directions and single volume integral over the source volume. This is attributed to the sifting property of \( \delta(\vec{r}-\vec{r}') \) and \( \delta_{jk} \). The expression for \( \langle S_z(\vec{r}; \omega) \rangle \) is given by equation (5.33). By inspection, expression for \( \langle S_x(\vec{r}; \omega) \rangle \) and \( \langle S_y(\vec{r}; \omega) \rangle \) are also obtained.

\[
\langle S_x(\vec{r}; \omega) \rangle = \frac{1}{2} \text{Re} \left\{ \frac{i k_o^2}{\pi} \sum_{n=x,y,z} \int_V dv' \left[ G_{eo}^{yn}(\vec{r}; \vec{r}') G_{mo}^{zn}(\vec{r}; \vec{r}') - G_{mo}^{zn}(\vec{r}; \vec{r}') G_{eo}^{yn}(\vec{r}; \vec{r}') \right] \right\}
\]
(5.31)

\[
\varepsilon''(\vec{r}') \Theta(\omega, T(\vec{r}')) + \left[ G_{mo}^{zn}(\vec{r}; \vec{r}') G_{eo}^{zn}(\vec{r}; \vec{r}') - G_{mo}^{zn}(\vec{r}; \vec{r}') G_{eo}^{zn}(\vec{r}; \vec{r}') \right] \mu''(\vec{r}') \Theta(\omega, T(\vec{r}'))
\]
(5.32)
\[
(S_z(\vec{r}; \omega)) = \frac{1}{2} \text{Re} \left\{ \frac{i k_o^2}{\pi} \sum_{n=x,y,z} \int d\vec{v}' \left[ G_{eo}^{zn}(\vec{r}; \vec{r}') G_{mo}^{yn*}(\vec{r}; \vec{r}') - G_{eo}^{yn}(\vec{r}; \vec{r}') G_{mo}^{zn*}(\vec{r}; \vec{r}') \right] \right\}
\]

\[
\varepsilon''(\vec{r}') \Theta(\omega, T(\vec{r}')) + \left[ G_{mo}^{zn}(\vec{r}; \vec{r}') G_{eo}^{yn*}(\vec{r}; \vec{r}') - G_{mo}^{yn}(\vec{r}; \vec{r}') G_{eo}^{zn*}(\vec{r}; \vec{r}') \right] \mu''(\vec{r}') \Theta(\omega, T(\vec{r}')) \}
\]

\[
(5.33)
\]

### 5.4 Cylindrical Vector Wave Functions

Vector Wave Functions (VWF) are eigen functions of vector Helmholtz equation [61, 62]. In cylindrical coordinate system, VWFs are given by

\[
\bar{M}_n(k_\rho, k_z; \bar{r}) = \nabla \times \hat{z} \psi_n(k_\rho, k_z; \bar{r}) \quad (5.34)
\]

\[
\bar{N}_n(k_\rho, k_z; \bar{r}) = \frac{1}{k} \nabla \times \nabla \times \hat{z} \psi_n(k_\rho, k_z; \bar{r}) \quad (5.35)
\]

\[
\bar{L}_n(k_\rho, k_z; \bar{r}) = \nabla \psi_n(k_\rho, k_z; \bar{r}) \quad (5.36)
\]

where \(\psi_n(k_\rho, k_z; \bar{r})\) is a solution to the scalar wave equation.

\[
\nabla^2 \psi_n + k^2 \psi_n = 0 ; \quad k = \sqrt{k_z^2 + k_\rho^2} \quad (5.37)
\]

One possible solution to (5.37) is as follows.

\[
\psi_n(k_\rho, k_z; \bar{r}) = J_n(k_\rho \rho) e^{+in\phi} e^{+ik_z z} \quad (5.38)
\]

where \(J_n(k_\rho \rho)\) is Bessel function of first kind. Instead of Bessel function of first kind, Neumann function and Hankel functions can also be solutions. The orthogonality condition for \(\psi_n(k_\rho, k_z; \bar{r})\) given by (5.38) can be derived [62, 63] and is given by

\[
\int_0^\infty \int_0^{2\pi} \int_{-\infty}^\infty \psi_n(k_\rho, k_z; \bar{r}) \psi_{n'}(-k'_\rho, -k'_z; \bar{r}) \rho dp \rho d\phi d\phi = (2\pi)^2 \delta_{nn'} \delta(k_z - k'_z) \frac{\delta(k_\rho - k'_\rho)}{k_\rho} \quad (5.39)
\]

\(\bar{M}_n(k_\rho, k_z; \bar{r}), \bar{N}_n(k_\rho, k_z; \bar{r})\) and \(\bar{L}_n(k_\rho, k_z; \bar{r})\) satisfies the vector Helmholtz equation given by

\[
\nabla \times \nabla \times \vec{F} - k^2 \vec{F} = 0 \quad (5.40)
\]

where \(\vec{F}\) can be either \(\bar{M}_n(k_\rho, k_z; \bar{r}), \bar{N}_n(k_\rho, k_z; \bar{r})\) or \(\bar{L}_n(k_\rho, k_z; \bar{r})\). \(\bar{M}, \bar{N}\) are solenoidal VWFs and \(\bar{L}\) is an irrotational VWF. If \(\bar{M}\) is electric (magnetic) field, \(\bar{N}\) is magnetic (electric) field. By
substituting (5.38) in (5.34) and (5.35), expressions for \( \tilde{M}_n(k_p, k_z; \bar{r}) \), \( \tilde{N}_n(k_p, k_z; \bar{r}) \) are obtained.

\[
\tilde{M}_n(k_p, k_z; \bar{r}) = \left[ \frac{i}{\rho} J_n(k_p \rho) \hat{\rho} - k_p J'_n(k_p \rho) \hat{\phi} \right] e^{i \rho \phi} e^{i k_z z} 
\]

(5.41)

\[
\tilde{N}_n(k_p, k_z; \bar{r}) = \frac{1}{k} \left[ i k z k_p J'_n(k_p \rho) \hat{\rho} - \frac{n k_z}{\rho} J_n(k_p \rho) \hat{\phi} + k_p^2 J_n(k_p \rho) \hat{z} \right] e^{i \rho \phi} e^{i k_z z} 
\]

(5.42)

where \( J'_n(k_p \rho) = \frac{d J_n(k_p \rho)}{d \rho} \). Since VWFs are eigen functions of vector wave equation, an arbitrary electric field satisfying vector wave equation can be expressed using eigen function expansion.

\[
\tilde{E}(\bar{r}) = \sum_{n=-\infty}^{+\infty} \int_{-\infty}^{\infty} dk_z \int_{0}^{\infty} dk_p \left[ a_n(k_p, k_z) \tilde{M}_n(k_p, k_z; \bar{r}) + b_n(k_p, k_z) \tilde{N}_n(k_p, k_z; \bar{r}) + c_n(k_p, k_z) \tilde{L}_n(k_p, k_z; \bar{r}) \right] 
\]

(5.43)

the expansion coefficients is obtained by taking the inner product of \( \tilde{E}(\bar{r}) \) with the corresponding VWF and using the orthogonality properties of VWF.

### 5.4.1 Orthogonality of VWFs

The orthogonality properties of VWFs are derived in several books [64, 62, 61, 63]. Since we will not be using \( \tilde{L}_n \), only the orthogonality of \( \tilde{M}_n, \tilde{N}_n \) are stated here.

\[
\int_{-\infty}^{\infty} \int_{0}^{2\pi} \int_{0}^{\infty} \tilde{M}_n(k_p, k_z; \bar{r}) \tilde{N}_n'(k'_p, k'_z; \bar{r}) \rho d \rho d \phi d z = 0 
\]

(5.44)

\[
\int_{-\infty}^{\infty} \int_{0}^{2\pi} \int_{0}^{\infty} \tilde{M}_n(k_p, k_z; \bar{r}) \tilde{M}_{-n'}(-k'_p, -k'_z; \bar{r}) \rho d \rho d \phi d z = \left(2\pi\right)^2 k_p \delta_{nn'} \delta(k_z - k'_z) \delta(k_p - k'_p) 
\]

(5.45)

\[
\int_{-\infty}^{\infty} \int_{0}^{2\pi} \int_{0}^{\infty} \tilde{N}_n(k_p, k_z; \bar{r}) \tilde{N}_{-n'}(-k'_p, -k'_z; \bar{r}) \rho d \rho d \phi d z = \left(2\pi\right)^2 k_p \delta_{nn'} \delta(k_z - k'_z) \delta(k_p - k'_p) 
\]

(5.46)

\[
\int_{-\infty}^{\infty} \int_{0}^{2\pi} \int_{0}^{\infty} \tilde{N}_n(k_p, k_z; \bar{r}) \tilde{N}_n'(-k'_p, -k'_z; \bar{r}) \rho d \rho d \phi d z = \left(2\pi\right)^2 k_p \delta_{nn'} \delta(k_z - k'_z) \delta(k_p - k'_p) 
\]

(5.47)

\[
\int_{-\infty}^{\infty} \int_{0}^{2\pi} \int_{0}^{\infty} \tilde{N}_n(k_p, k_z; \bar{r}) \tilde{N}_n'(-k'_p, -k'_z; \bar{r}) \rho d \rho d \phi d z = \left(2\pi\right)^2 k_p \delta_{nn'} \delta(k_z - k'_z) \delta(k_p - k'_p) 
\]

(5.48)
The above three properties can be derived by substituting (5.41), (5.42) in the left hand side of the above properties and using the following properties of Bessel function of first kind.

\[
J_{-n}(x) = (-1)^n J_n(x) \tag{5.49}
\]

\[
J_n(-x) = (-1)^n J_n(x) \tag{5.50}
\]

\[
J_{-n}(-x) = J_n(x) \tag{5.51}
\]

\[
\int_0^\infty J_n(k_\rho \rho) J_n(k_\rho' \rho) \rho \, d\rho = \frac{\delta(k_\rho - k_\rho')}{k_\rho} \tag{5.52}
\]

\[
J_n(x) = \frac{x}{2n} [J_{n-1}(x) + J_{n+1}(x)] \tag{5.53}
\]

\[
\frac{dJ_n}{dx} = \frac{1}{2} [J_{n-1}(x) - J_{n+1}(x)] \tag{5.54}
\]

### 5.5 DGF for dielectric cylinder

The $\overline{G}_{e\omega}$ in cylindrical coordinate system is given by the following two equations.

\[
\overline{G}_{e\omega}(\vec{r}; \vec{r}') = \frac{i}{8\pi} \int_{-\infty}^\infty dk_z \sum_{n=-\infty}^{+\infty} \frac{(-1)^n}{k_\rho^2} \left[ M_n(k_\rho, k_z; \vec{r}) \tilde{M}_{-n}(k_\rho, -k_z; \vec{r}') + \tilde{N}_n(k_\rho, k_z; \vec{r}) \tilde{N}_{-n}(k_\rho, -k_z; \vec{r}') \right] ; \rho > \rho' \tag{5.55}
\]

\[
\overline{G}_{e\omega}(\vec{r}; \vec{r}') = \frac{i}{8\pi} \int_{-\infty}^\infty dk_z \sum_{n=-\infty}^{+\infty} \frac{(-1)^n}{k_\rho^2} \left[ \tilde{M}_n(k_\rho, k_z; \vec{r}) \tilde{M}_{-n}(k_\rho, -k_z; \vec{r}') + \tilde{N}_n(k_\rho, k_z; \vec{r}) \tilde{N}_{-n}(k_\rho, -k_z; \vec{r}') \right] ; \rho < \rho' \tag{5.56}
\]

Superscript (1) denotes that Hankel function of first kind should be used instead of Bessel function of first kind in the definition of the VWFs. For the case of $\rho > \rho'$, the integrand in (5.55) can be written as the following matrix.

\[
\begin{bmatrix}
(M_{n,\rho}^{(1)} M_{-n,\rho}' + N_{n,\rho}^{(1)} N_{-n,\rho}' \hat{\rho} \hat{\rho}) & (M_{n,\rho}' M_{-n,\rho}^{(1)} + N_{n,\rho}^{(1)} N_{-n,\rho}' \hat{\rho} \hat{\phi}) & N_{n,\rho}^{(1)} N_{-n,\rho}' \hat{\rho} \hat{\phi} \\ (M_{n,\rho}' M_{-n,\rho}^{(1)} + N_{n,\rho}^{(1)} N_{-n,\rho}' \hat{\rho} \hat{\rho}) & (M_{n,\rho}' M_{-n,\rho}^{(1)} + N_{n,\rho}^{(1)} N_{-n,\rho}' \hat{\rho} \hat{\phi}) & N_{n,\rho}^{(1)} N_{-n,\rho}' \hat{\rho} \hat{\phi} \\ N_{n,\rho}^{(1)} N_{-n,\rho}' \hat{\rho} \hat{\rho} & N_{n,\rho}^{(1)} N_{-n,\rho}' \hat{\rho} \hat{\phi} & N_{n,\rho}^{(1)} N_{-n,\rho}' \hat{\rho} \hat{\phi}
\end{bmatrix} \tag{5.57}
\]

where $M_{n,\rho}^{(1)}$ represents the $\rho$ component of $\tilde{M}_n^{(1)}(k_\rho, k_z; \vec{r})$, $N_{-n,\rho}'$ represents the $\phi$ component of $\tilde{N}_{-n}(k_\rho, -k_z; \vec{r}')$. 
DGF for dielectric cylinder when source is outside the cylinder is given in [61]. But for thermal emission, we need the DGF when source is inside cylinder. The results for such a DGF is given in [54, 65]. However without deriving it from basic principles, it is not possible to extend it or even apply it for our problem. Let the free space outside a dielectric cylinder of radius $a$ be noted by region 1 and the region inside the cylinder be region 2. By the method of scattering superposition [61]

$$\mathbf{G}_{e}^{(22)}(\mathbf{r}; \mathbf{r}') = \mathbf{G}_{eo}(\mathbf{r}; \mathbf{r}') + \mathbf{G}_{es}^{(22)}(\mathbf{r}; \mathbf{r}'); \quad \rho \leq a$$  \hspace{1cm} (5.58)

$$\mathbf{G}_{e}^{(12)}(\mathbf{r}; \mathbf{r}') = \mathbf{G}_{es}^{(12)}(\mathbf{r}; \mathbf{r}'); \quad \rho > a$$  \hspace{1cm} (5.59)

where the first number in the superscript denotes the media in which the field point $\mathbf{r}$ is located and the second number in the superscript denotes the media in which the source point $\mathbf{r}'$ is located.

Since in our case, the source is always inside the cylinder (i.e. region 2), the second number of the superscript is always 2. $\mathbf{G}_{eo}(\mathbf{r}; \mathbf{r}')$ is given by equations (5.55) and (5.56). The expressions for $\mathbf{G}_{es}^{(22)}(\mathbf{r}; \mathbf{r}')$ and $\mathbf{G}_{es}^{(12)}(\mathbf{r}; \mathbf{r}')$ are as follows.

$$\mathbf{G}_{es}^{(22)}(\mathbf{r}; \mathbf{r}') = \frac{i}{8\pi} \int \frac{dkz}{2\pi} \int_{-\infty}^{+\infty} d\eta \sum_{n=-\infty}^{+\infty} \frac{(-1)^n}{\eta^2} \left[ a(n,k_z)\tilde{M}_n(\eta,k_z;\mathbf{r})\tilde{M}_{-n}(\eta,-k_z;\mathbf{r}') + b(n,k_z)\tilde{N}_n(\eta,k_z;\mathbf{r})\tilde{M}_{-n}(\eta,-k_z;\mathbf{r}') + c(n,k_z)\tilde{M}_n(\eta,k_z;\mathbf{r})\tilde{N}_{-n}(\eta,-k_z;\mathbf{r}') + d(n,k_z)\tilde{N}_n(\eta,k_z;\mathbf{r})\tilde{N}_{-n}(\eta,-k_z;\mathbf{r}') \right]$$  \hspace{1cm} (5.60)

$$\mathbf{G}_{es}^{(12)}(\mathbf{r}; \mathbf{r}') = \frac{i}{8\pi} \int \frac{dkz}{2\pi} \int_{-\infty}^{+\infty} d\eta \sum_{n=-\infty}^{+\infty} \frac{(-1)^n}{\eta^2} \left[ e(n,k_z)\tilde{M}_n^{(1)}(\eta,k_z;\mathbf{r})\tilde{M}_{-n}(\eta,-k_z;\mathbf{r}') + f(n,k_z)\tilde{N}_n^{(1)}(\eta,k_z;\mathbf{r})\tilde{M}_{-n}(\eta,-k_z;\mathbf{r}') + g(n,k_z)\tilde{M}_n^{(1)}(\eta,k_z;\mathbf{r})\tilde{N}_{-n}(\eta,-k_z;\mathbf{r}') + h(n,k_z)\tilde{N}_n^{(1)}(\eta,k_z;\mathbf{r})\tilde{N}_{-n}(\eta,-k_z;\mathbf{r}') \right]$$  \hspace{1cm} (5.61)

where $\eta = \sqrt{k_1^2 - k_z^2}, \xi = \sqrt{k_2^2 - k_z^2}$. The unknown coefficients $\{a, b, c, d, e, f, g, h\}$ are obtained by using the boundary conditions for electric DGF.

$$\hat{\rho} \times \left[ \mathbf{G}_{es}^{(12)}(\mathbf{r}; \mathbf{r}') \right] = \hat{\rho} \times \left[ \mathbf{G}_{eo}(\mathbf{r}; \mathbf{r}') + \mathbf{G}_{es}^{(22)}(\mathbf{r}; \mathbf{r}') \right]$$  \hspace{1cm} (5.62)
\[
\hat{\rho} \times \left[ \frac{\nabla \times \overline{G}^{(12)}_{es}(\vec{r}; \vec{r}')}}{\mu_1} \right] = \hat{\rho} \times \left[ \frac{\nabla \times \overline{G}_{eo}(\vec{r}; \vec{r}')} + \nabla \times \overline{G}^{(12)}_{es}(\vec{r}; \vec{r}')}{\mu_2} \right]
\]
(5.63)

These boundary conditions are applied at the boundary of the cylinder i.e. \( \vec{r}' = (\rho = a, \phi, z) \). When using the above boundary conditions, the expression for \( \overline{G}_{eo}(\vec{r}; \vec{r}') \) when \( \rho > \rho' \) should be used. This is because all the sources are inside the boundary. The expression is repeated below.

\[
\overline{G}_{eo}(\vec{r}; \vec{r}') = \frac{i}{8\pi} \int_{-\infty}^{\infty} dk_z \sum_{n=-\infty}^{+\infty} \frac{(-1)^n}{\xi^2} \left[ M_n^{(1)}(\xi, k_z; \vec{r})M_{-n}(\xi, -k_z; \vec{r}') + \bar{N}_n^{(1)}(\xi, k_z; \vec{r})\bar{N}_{-n}(\xi, -k_z; \vec{r}') \right] ; \rho > \rho'
\]
(5.64)

The curl of the DGF can be obtained in a straightforward manner using the fact that the VWFs \( M, N \) is proportional to the curl of \( N, M \) respectively. For instance, \( \nabla \times \overline{G}^{(12)}_{es}(\vec{r}; \vec{r}') \) is given by the following equation.

\[
\nabla \times \overline{G}^{(12)}_{es}(\vec{r}; \vec{r}') = \frac{ik_1}{8\pi} \int_{-\infty}^{+\infty} dk_z \sum_{n=-\infty}^{+\infty} \frac{(-1)^n}{\eta^2} \left[ (n, k_z)\bar{N}_n^{(1)}(\eta, k_z; \vec{r})\bar{M}_{-n}(\eta, -k_z; \vec{r}') + f(n, k_z)M_n^{(1)}(\eta, k_z; \vec{r})M_{-n}(\eta, -k_z; \vec{r}') + \right.
\]

\[
g(n, k_z)\bar{N}_n^{(1)}(\eta, k_z; \vec{r})\bar{N}_{-n}(\eta, -k_z; \vec{r}') + h(n, k_z)\bar{M}_n^{(1)}(\eta, k_z; \vec{r})\bar{N}_{-n}(\eta, -k_z; \vec{r}') \right]
\]
(5.65)

By comparing the above equation with (5.61), the presence of \( k_1 \) and the swapping of \( M, N \) with \( N, M \) can be seen. The coefficients can be obtained by solving the system of equations \( Ax = b \) obtained by applying the boundary conditions.

\[
A = \begin{bmatrix}
-J_n'(\xi a) & -\frac{nk - j_n(\xi a)}{\omega \sigma} & 0 & 0 & H_n^{(1)'\prime}(\eta a) & \frac{nk - j_n(\xi a)}{\omega \sigma} H_n^{(1)'\prime}(\eta a) & 0 & 0 \\
0 & -J_n(\xi a) & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -J_n'(\xi a) & -\frac{nk - j_n(\xi a) - j_n'(\xi a)}{\omega \sigma} & 0 & 0 & H_n^{(1)'\prime}(\eta a) & \frac{nk - j_n(\xi a) - j_n'(\xi a)}{\omega \sigma} H_n^{(1)'\prime}(\eta a) \\
0 & 0 & 0 & -\frac{j_n(\xi a)}{\omega \sigma} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -\frac{j_n(\xi a)}{\omega \sigma} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\frac{j_n(\xi a)}{\omega \sigma} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{j_n(\xi a)}{\omega \sigma} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]
(5.66)
This is similar to the matrix equation given in [61], but that is for the case of source outside the cylinder.
Chapter 6

Fast Jacobian Mie Library for Terrestrial Hydrometeors

This work presents an approach for the fast, accurate computation of several useful Mie-based parameters for homogenous, spherical, liquid water and ice hydrometeor distributions over a wide range of frequencies, mean hydrometeor diameters, and physical temperatures as occur in the terrestrial atmosphere. The absorption coefficient, scattering coefficient, backscattering coefficient, and phase asymmetry parameters are cast into functions of three independent variables: frequency, temperature and mean diameter. An exponential drop size distribution with a constant fractional volume of $10^{-6}$ is used to model polydispersed hydrometeors. The ranges used for frequency, temperature and mean diameter are $[1, 1000]$ GHz, $[-50, +50]$°C and $[0.002, 20]$ mm respectively. The functions are then sampled on a logarithmic grid. Trivariate cubic spline interpolation using non uniform B-splines is then used to efficiently represent these three dimensional functions in a compact library. By using this method, four important criteria are achieved: 1) fast random computability of any of these parameters given the values of frequency, temperature and mean diameter, 2) minimal memory usage by storage of only B-spline coefficients, 3) representation of parameters using well behaved functional forms amenable to analytical differentiation for evaluation of Jacobians or alternatively for higher accuracy, B-spline coefficients calculated using true Jacobian values can be used, and 4) negligibly small and bounded error over the entire domain of the library.

These procedures results in considerable acceleration of microwave radiative transfer simulations across a broad frequency spectrum, as demonstrated in calculations for both scattering and non-scattering atmospheres. The methods discussed can also be applied to other geophysical problems.
requiring rapid calculation of series-based functions of several independent variables, where the function evaluation is a time consuming process, and maximum error bounds are critical.

6.1 Introduction

Numerical radiative transfer (RT) calculations are essential for understanding and assimilating brightness temperature measurements made at various microwave frequencies and geographical locations. RT model inversions performed using these measurements can yield vertical temperature and water vapor density profiles of the atmosphere, which can be used in numerical weather prediction and climate forecasting [66, 67, 68]. Alternately, brightness temperature measurements can also be directly assimilated into numerical weather models [69]. Efficient numerical weather forecasting applications require particularly fast scattering-based radiative transfer (RT) simulations. One of the processes that impart a high computational burden for hydrometeor laden atmospheres is the calculation of hydrometeor absorption and scattering coefficients and the phase asymmetry parameter. The excessive computational overhead is a result of the nested summations required in the calculation of the Mie efficiencies and the subsequent numerical integration of the Mie efficiencies over the hydrometeor drop size distribution. The large number of times that such calculations are required in radiative transfer modeling suggests that library look-up techniques can provide significant computational efficiencies provided that library error can be bounded. Reflectivity modeling for meteorological weather radar is similarly well understood [70]. In the Rayleigh regime, the backscattering coefficient is related to the reflectivity factor $Z$, which in turn depends on the precipitation rate. As a part of this work, fast libraries for liquid and ice hydrometeor backscattering coefficients are also developed.

RT theory describes the interaction of radiation with matter by taking into the consideration the effects of the atmospheric absorption, emission and scattering due to cloud, fog, snow etc on electromagnetic radiation. It has been thoroughly discussed in numerous references [71, 72, 73]. One of the first steps in RT modeling is the computation of the absorption vector, extinction matrix and phase matrix at each point within the medium of interest [66]. For plane parallel atmospheric
models, this step entails calculations at each of many (typically 50-100) vertical levels of the atmosphere. The atmospheric medium consists of gaseous absorbing constituents such as oxygen, water vapor, ozone, and nitrogen. In addition to these gases the atmosphere may also contain suspended or falling liquid or frozen water particles in the troposphere and lower stratosphere. Hydrometeors can absorb, emit, and scatter radiation at microwave frequencies. Hydrometeors can be either rain drops, ice crystals, snowflakes, graupel or hail. They are in general nonspherical in shape. For instance, rain drops are slightly oblate shaped [74, 75]. Ice crystals are either hexagonal or irregular in shape. However for small hydrometeor dimensions, spherical drop assumption is accurate. In this work, we have used the assumption of modelling hydrometeors as either liquid water or solid ice spheres. Furthermore, in this work only homogenous liquid water or ice hydrometeors are considered. In reality, many hydrometeors are a multiphase mixture of air, ice and water [76, 77]. A future extension of this work will deal with the extinction coefficients of nonspherical hydrometeors. T-matrix method instead of Lorentz-Mie theory need to be used for this purpose [78]. The problem of applying an iterative numerical radiative transfer model to hydrometeor - laden atmospheres has been addressed in [66], within which hydrometeors are modeled as dielectric spheres. The analytical solution to the absorption and scattering of electromagnetic waves by a sphere of arbitrary radius, permittivity and permeability was provided by Mie [79, 64, 25], from whose theory the absorption efficiency, scattering efficiency, backscattering efficiency, and phase function asymmetry of a single dielectric sphere can be calculated. However, expressions for the above parameters are in the form of infinite summations, where each term contains spherical Bessel functions requiring software evaluation. Accordingly, doubly nested summations are required to evaluate either the cross-sectional efficiencies or the phase function asymmetry for a single hydrometeor. Owing to hydrometeors of varying radii, clouds are modeled by a polydispersed distribution of hydrometeors. The hydrometeor coefficients of a polydispersed cloud is obtained by the integration of the corresponding efficiencies multiplied by the sphere cross-sectional area over the entire hydrometeor distribution. Numerical quadrature is used to perform this integration, resulting in a third level of nesting. Due to the need for accuracy, the resulting numerical integration often requires many
tens of thousands of elementary calculations for each atmospheric spatial point. Depending on the hydrometeor mean electrical size, this computational burden is comparable to or greater than that of the vertical RT quadrature itself.

This paper presents an efficient approach based on cubic B-spline interpolation for fast, accurate calculation of the Mie hydrometeor coefficients and phase asymmetry parameter for exponential size distributions of terrestrial hydrometeors. B-splines are standard for representing curves and surfaces in computer graphics and computer aided design [80, 81]. Recently, they have been used for volume reconstruction of arbitrarily dimensioned data. For the present Mie application, lookup tables (LUT) for the absorption coefficient, scattering coefficient, backscattering coefficient, and phase matrix asymmetry parameter for both liquid and ice hydrometeor distributions are generated. These LUTs are three dimensional arrays coding the values of these products as a function of the three independent variables of temperature, average hydrometeor size, and frequency over a wide range of these parameters. In total, eight LUTs corresponding to four different products for two types of hydrometeors (liquid and frozen) are generated. Spline interpolation is applied on any of these eight LUTs to evaluate the corresponding products. The method results in considerable acceleration of microwave radiative transfer simulations across a broad frequency spectrum and fast calculation of radar reflectivity for weather radar data assimilation. We demonstrate the effectiveness of this approach by comparing radiative transfer simulations using the B-spline library to the conventional series approach within scattering and non-scattering atmospheres. These inter-comparisons confirm acceptable reconstruction accuracy for most relevant problems in terrestrial radiative transfer and radar studies.

6.2 Mie theory

The electromagnetic absorption and scattering properties of a single sphere are described by the absorption efficiency $\eta_a$, scattering efficiency $\eta_s$ and backscattering efficiency $\eta_b$ [25, 2]. Let $S_i$ (Wm$^{-2}$) be the average power density of the electromagnetic wave incident on the sphere, and $P_a$ (W), $P_s$ (W) and $P_b$ (W) be the absorbed, scattered and backscattered power respectively. $P_a$, $P_s$
and \( P_b \) are related to \( S_i \) through the absorption cross section \( \sigma_a \) (m\(^2\)), scattering cross section \( \sigma_s \) (m\(^2\)) and backscattering cross section \( \sigma_b \) (m\(^2\)). They are given by the following relations,

\[
\sigma_a = \frac{P_a}{S_i} ; \quad \sigma_s = \frac{P_s}{S_i} ; \quad \sigma_b = \frac{P_b}{S_i}
\] (6.1)

The absorption efficiency \( \eta_a \), scattering efficiency \( \eta_s \) and backscattering efficiency \( \eta_b \) are defined as the ratio of the corresponding electromagnetic cross-sections to the physical cross-section of a sphere of radius \( a \),

\[
\eta_a = \frac{\sigma_a}{\pi a^2} ; \quad \eta_s = \frac{\sigma_s}{\pi a^2} ; \quad \eta_b = \frac{\sigma_b}{\pi a^2}
\] (6.2)

The extinction cross-section \( \sigma_e \) and extinction efficiency \( \eta_e \) are similarly given by the sum of the corresponding absorption and scattering quantities,

\[
\sigma_e = \sigma_a + \sigma_s ; \quad \eta_e = \eta_a + \eta_s
\] (6.3)

The \( \eta_e, \eta_s \) and \( \eta_b \) for a homogenous dielectric sphere of arbitrary radius and refractive index is given by Mie theory as,

\[
\eta_e(x, m) = \frac{2}{x^2} \sum_{n=1}^{\infty} (2n + 1) \text{Re} \{a_n + b_n\}
\] (6.4)

\[
\eta_s(x, m) = \frac{2}{x^2} \sum_{n=1}^{\infty} (2n + 1) \{|a_n|^2 + |b_n|^2\}
\] (6.5)

\[
\eta_b(x, m) = \frac{1}{x^2} \sum_{n=1}^{\infty} (-1)^n (2n + 1) (a_n - b_n)
\] (6.6)

In (4-6), \( x \) is the size parameter given by \( x \equiv \frac{2\pi a}{\lambda} \) and \( a_n, b_n \) are the Mie coefficients, which are generally complex and calculated as follows,

\[
a_n = \frac{m^2 j_n(mx)xj_n(x) - j_n(x)mxj_n(mx)}{m^2 j_n(mx)xh_n^1(x) - h_n^1(x)mxj_n(mx)}
\] (6.7)

\[
b_n = \frac{j_n(mx)xj_n(x) - j_n(x)mxj_n(mx)}{j_n(mx)xh_n^1(x) - h_n^1(x)mxj_n(mx)}
\] (6.8)

where \( j_n(x), h_n^1(x) \) are spherical Bessel and Hankel functions of first kind and order \( n \) respectively.

In the above equations \( m(f, T) \equiv \frac{n_s(f, T)}{n_b(f, T)} \) is the relative complex refractive index of the material of the sphere with respect to the background medium, \( n_s \) is the refractive index of the sphere and
$n_b$ is the refractive index of the background medium (usually air). The refractive indices $n_s, n_b$ are both functions of frequency $f$ and temperature $T$. In these general expressions both the sphere and background are assumed to be non-magnetic. In practice the summations in (7-8) are truncated at a finite number of terms. The maximum number of terms is commonly taken to be the next integer closest to $x + 4x^{3/4} + 2$ [25]. The phase function asymmetry parameter $G$ defines the relative proportion of energy scattered in the forward versus backward directions [66]. The Mie phase function asymmetry for a single spherical scatterer is,

$$G = \frac{4}{x^2 \eta_s(x, m)} \sum_{n=1}^{\infty} \left[ \frac{n (n+2)}{n+1} Re \{a_n^* a_{n+1} + b_n^* b_{n+1} \} \right]$$

For $G = 1$, there is only scattering in the forward direction and $G > 0$ denotes a preferential scattering in the forward versus backward direction. Similarly, for $G = -1$ there is scattering only in the backward direction and $G < 0$ denotes a preferential scattering backwards. In (4-6,9), the value of the size parameter $x$ determines the number of significant terms in the series, and as a result the scattering regime of the particle. For electrically small spheres with $x \ll 1$, a single term is usually sufficient. This is the Rayleigh scattering approximation [82]. For electrically large spheres with $x \gg 1$ geometrical optics, physical optics, or ray tracing, approximations can often be used effectively. Nonetheless, in this work the truncated Mie summations are exclusively used without further approximation.

### 6.3 Look-up table calculation

The radius of hydrometeors varies from $\sim 1\mu m$ for haze, fog, and small cloud particles up to about $\sim 10 \ mm$ for large frozen particles such as graupel, snow and small hail. The statistical characterization of the hydrometeor particle size for a polydispersed cloud is given by a drop size distribution, $n(D)$ [2, 66]. A general functional form of $n(D)$ (mm$^{-1}$m$^{-3}$) is the modified gamma distribution given by,

$$n(D) = N_0 (\Lambda D)^P e^{-(\Lambda D)^Q}$$

(6.10)
where \( n(D)\, dD \) is the number of particles per unit volume with diameters in the range \([D, D + dD]\) (mm) and \( N_0 \) (mm\(^{-1}\)m\(^{-3}\)), \( \Lambda \) (mm\(^{-1}\)), \( P \) and \( Q \) are distribution parameters. By considering \( n(D) \) given by (10) as a scaled probability density function, closed form expressions for important statistical quantities such as mean diameter, diameter variance, particle number density, and fractional volume can be derived [2, 66]. For example, the mean diameter \( \langle D \rangle \) (mm) and fractional volume of water \( f_v \) are given by,

\[
\langle D \rangle = \frac{1}{\Lambda} \frac{\Gamma \left( \frac{P+2}{Q} \right)}{\Gamma \left( \frac{P+1}{Q} \right)}
\]

\[
f_v = \frac{10^{-9} \pi N_0}{6 \Lambda^4 Q} \Gamma \left( \frac{P + 4}{Q} \right)
\]

where \( \Gamma(.) \) is the gamma function. The aggregate absorption, scattering and backscattering coefficients in (Npm\(^{-1}\)) can be computed as,

\[
\kappa_\alpha (f,T) = \frac{\pi}{4} \int_0^\infty D^2 n(D) \eta_\alpha \left( \frac{\pi D}{x}, m(f,T) \right) dD
\]

where \( \alpha = a, s, b \). Similarly, the aggregate phase function asymmetry \( g(f,T) \) for a polydispersion of spherical hydrometeors is given by,

\[
g (f,T) = \frac{\int_0^\infty D^2 n(D) \eta_s \left( \frac{\pi D}{x}, m \right) G \left( \frac{\pi D}{x}, m \right) dD}{\int_0^\infty D^2 n(D) \eta_s \left( \frac{\pi D}{x}, m \right) dD}
\]

Most of the commonly used drop size distributions for precipitation such as Marshall - Palmer (MP), Laws and Parsons, and Sekhon - Srivastava (SS) are of the exponential form [2, 66]. Accordingly, to avoid the problem of cataloguing a four-fold infinity of size distributions we limit this study to exponential distributions for which \( P = 0 \) and \( Q = 1 \). Exponential distribution can be used to model major hydrometeor particles such as rain, snow, graupel and hail [83]. In this case \( \langle D \rangle = \frac{1}{\Lambda} \). Although the upper limit of integration in \((13,14)\) is infinite, the practical upper limit on the drop diameter is about 6 mm [84]. However, we have used a conservative upper limit of about 15 mean particle diameters due to the exponential decay factor. By \( n = 15 \), it is noted that the relative error in the integral becomes bounded by \( 9.6 \times 10^{-4} \) times the real value, viz.:

\[
\frac{\int_0^{n\langle D \rangle} D^2 e^{-\frac{D^2}{\langle D \rangle^2}} dD}{\int_0^\infty D^2 e^{-\frac{D^2}{\langle D \rangle^2}} dD} = 1 - e^{-n} \left[ 0.5n^2 + n + 1 \right]
\]
A simple empirical sensitivity analysis was performed to confirm the above mentioned upper limit. For 1000 random values of $(f, \langle D \rangle, T)$, $\kappa_b$ was calculated for liquid hydrometeors for values of $n$ ranging from 5 to 25 in steps of 5. The relative error percentage was calculated for each increment of $n$. The averages of these error percentages tend to converge to a very small value with each increment. The average percentage errors for the increments 5-10, 10-15, 15-20, 20-25 are 14.5%, 0.31%, 0.023% and 0.021% respectively. This confirms that for $n > 15$, there is hardly any contribution to the coefficient value. In this work, we have used mean diameters ranging from 2 $\mu$m to 20 mm to cover the range of terrestrial hydrometeors encountered in realistic precipitation models. The constant $N_0$ for an exponential drop size distribution is a function of $\langle D \rangle$, and can be calculated using equations (11,12). The average fractional volume of liquid or frozen precipitation rarely exceeds $5 \times 10^{-6}$. Since $N_0$ is directly proportional to fractional volume, $f_v$, a constant value $f_v = 10^{-6}$, corresponding to 1 gm$^{-3}$ of liquid water was used in this work. In this case, $N_0 = \frac{10^{6}}{\pi(D)^4}$, where $\langle D \rangle$ is in mm. The coefficients for a different value of fractional volume can be obtained by scaling the results according to the actual density of hydrometeors. Therefore, equations (13,14) reduce to,

$$
\kappa_a (f, \langle D \rangle, T) \approx \frac{\pi 10^{-6}}{4} \int_0^{15\langle D \rangle} D^2 N_0(\langle D \rangle) e^{-\frac{D}{\langle D \rangle}} \eta_a \left( \frac{\pi D}{\lambda}, m \right) dD \quad (6.16)
$$

$$
g (f, \langle D \rangle, T) \approx \frac{\int_0^{15\langle D \rangle} D^2 N_0(\langle D \rangle) e^{-\frac{D}{\langle D \rangle}} \eta_s \left( \frac{\pi D}{\lambda}, m \right) dD}{\int_0^{15\langle D \rangle} D^2 N_0(\langle D \rangle) e^{-\frac{D}{\langle D \rangle}} \eta_a \left( \frac{\pi D}{\lambda}, m \right) dD} \quad (6.17)
$$

where $D$ is in mm and $\kappa_a (f, \langle D \rangle, T)$ is in Npm$^{-1}$. The above products are functions of three independent variables: frequency, temperature, and the mean diameter of the polydispersion. Since the Mie efficiencies are a function of the refractive index of the sphere, the complex dielectric constants of liquid water and ice as a function of frequency and temperature are also implicitly required.

Each product LUT is a three dimensional array that stores the values of $\kappa_a, \kappa_s, \kappa_b, g$ at discrete values of the independent variable triplet $(f, \langle D \rangle, T)$. The ranges used for each independent variable are: $f \in [1, 1000]$ GHz, $\langle D \rangle \in [0.002, 20]$ mm and $T \in [-50, +50]^\circ$C. For ice hydromete-
ors, \( T \in [-50, 0] ^\circ C \) is the temperature range used. Due to the large range of mean diameter and frequency, a logarithmic sampling grid was used for these two variables. Ray’s model was used for the dielectric constant of pure water at microwave frequencies [85]. The dielectric constant of ice at microwave frequencies was obtained from Warren [86]. Linear interpolation was used to obtain values of the dielectric constant at frequencies that are not listed in [86]. The Mie efficiencies are calculated using routines provided by Maetzler [87]. These routines are based on Mie formulations in [25], but stable for size parameters up to \( \sim 1000 \). Once the Mie efficiencies \( (\eta_a, \eta_s, \eta_b, G) \) and \( N_0 \) were calculated, adaptive Simpson’s quadrature was used for the numerical evaluation of the distribution integrals (16,17). The surface plots of hydrometeor absorption, scattering, backscattering coefficients and phase asymmetry parameter for both liquid and frozen hydrometeors at \( T = 0 ^\circ C \) is shown in Fig. (6.6)-(6.6). These plots can be considered as slices from the product LUTs at a constant temperature.

### 6.4 B-spline interpolation

Spline functions are piecewise polynomials on subintervals that are joined together with prescribed continuity conditions [80]. A spline function of order \( k \) consists of piecewise polynomials of maximum degree \( k - 1 \). Therefore a spline of order 1 is composed of piecewise constants, a spline of order 2 consists of piecewise linear polynomials, etc. A univariate spline function \( S(x) \) can be represented in the piecewise polynomial form as,

\[
S(x) = P_i(x) ; x \in [\xi_i, \xi_{i+1})
\]  

(6.18)

where \( \{\xi_i\}_{i=1}^{l+1} \) are a strictly increasing sequence of \( l + 1 \) break (or tie) points of \( S(x) \). A spline function of order \( k \) has the following continuity conditions at its breaks.

\[
P_i(\xi_{i+1}) = P_{i+1}(\xi_{i+1}) ; i \in [1, l - 1]
\]

\[
P_i^{(n)}(\xi_{i+1}) = P_{i+1}^{(n)}(\xi_{i+1}) ; i \in [1, l - 1] ; n = 1, 2, \ldots k - 2
\]  

(6.19)
where \( P_i^{(n)}(x) \) stands for the \( n^{\text{th}} \) derivative of \( P_i(x) \). The spline interpolation problem can be stated as follows: given the values of a function at break points, \( \{g(\xi_i)\}_{i=1}^{l+1} \), the spline function \( S(x) \) needs to be determined such that \( S(\xi_i) = g(\xi_i) ; i \in [1, l+1] \). Splines of order 4 (cubic splines) are twice continuously differentiable at the break points and most commonly used in practice. For a cubic spline, each polynomial section would be of degree 3, i.e.

\[
P_i(x) = c_{i3}(x - \xi_i)^3 + c_{i2}(x - \xi_i)^2 + c_{i1}(x - \xi_i) + c_{i0}.
\]

Given the values of the function \( g \) at the break points, the continuity conditions, and boundary conditions (usually the values of the first and second derivatives of \( S(x) \) at the first and last break points), the coefficients of each piecewise polynomial can be uniquely determined. Simple tridiagonal linear systems of equations are solved to obtain \( \{c_{ij}; i \in [1, l], j \in [0, 3]\} \).

An alternate representation of the spline function is known as the B-form representation \([80, 81, 88]\). This representation is based on the Curry-Schoenberg theorem, which is stated as follows: All the spline functions of order \( k \) and break sequence \( \{\xi_i\}_{i=1}^{l+1} \) form a linear space denoted by \( \Pi_{k,\xi} \). To represent a function using B-splines we choose a strictly non-decreasing sequence of points \( \{\tau_i\}_{i=1}^{l+k+1} \) called the knot sequence. The knot sequences need not coincide with the break points. In the B-form representation, the spline function is expressed as a weighted superposition of basis functions called B-splines (or, basis splines). For example, the B-spline of order 1 is given by,

\[
B_1^1(x) = \begin{cases} 
1 & ; \ x \in [\tau_i, \tau_{i+1}) \\
0 & ; \ \text{otherwise} 
\end{cases}
\]  

(6.20)

B-splines of higher order can be determined from lower order B-splines using the Cox-de Boor recurrence relation \([80]\),

\[
B_k^k(x) = \left( \frac{x - \tau_i}{\tau_{i+k-1} - \tau_i} \right) B_{i}^{k-1}(x) \\
+ \left( \frac{\tau_{i+k} - x}{\tau_{i+k} - \tau_{i+1}} \right) B_{i+1}^{k-1}(x)
\]  

(6.21)

Using the Cox-de Boor relation, the B-spline of order 4 (or, cubic B-spline) can be obtained and is given by (6.25). The B-splines \( B_1^k, B_2^k, \ldots, B_{l+1}^k \) defined on the knot sequence \( \{\tau_i\}_{i=1}^{l+k+1} \) form the basis for the spline space, \( \Pi_{k,\xi} \). It can be shown that the B-form representation of the spline
function $S(x)$ of order $k$ is given by,

$$S(x) = \sum_{i=1}^{l+1} \alpha_i B^k_i (x); x \in [\xi_i, \xi_{i+1})$$  \hspace{1cm} (6.22)$$

where $\{\alpha_i\}_{i=1}^{l+1}$ are the B-spline coefficients. Importantly, B-splines have local support. Hence the $i^{th}$ B-spline of order $k$, $B^k_i (x) = 0; x \notin [\tau_i, \tau_{i+k})$. For example, cubic B-splines have support between five successive knots. The local support of B-splines results in computational advantages for the B-form representation over the piecewise polynomial representation of a spline function. A good choice of knot sequence $\{\tau_i\}_{i=1}^{l+5}$ for cubic B-spline interpolation can be determined from the breaks $\{\xi_i\}_{i=1}^{l+1}$ as follows,

$$\tau_1 = \tau_2 = \tau_3 = \tau_4 = \xi_1$$

$$\tau_{i+5} = \tau_{i+4} = \tau_{i+3} = \tau_{i+2} = \xi_{i+1}$$

$$\tau_i = \frac{\xi_{i-3} + \xi_{i-2} + \xi_{i-1}}{3}; \ 5 \leq i \leq l + 1$$  \hspace{1cm} (6.23)$$

Extension of spline function theory to multiple variables can be obtained by tensor product construction of univariate B-splines. In this case a trivariate cubic spline function $F(x, y, z)$ is represented in B-form as,

$$F(x, y, z) = \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \sum_{m=1}^{N_z} \alpha_{ijm} B^4_i (x) B^4_j (y) B^4_m (z)$$  \hspace{1cm} (6.24)$$

In this work, the three dimensional function $F$ can be any of the following eight functions: absorption coefficient, scattering coefficient, backscattering coefficient, or phase asymmetry for liquid water and ice hydrometeors, with the independent variables being frequency, temperature
and mean diameter,

\[
F(f, T, \langle D \rangle) = \sum_{i=1}^{N_f} \sum_{j=1}^{N_T} \sum_{m=1}^{N_{\langle D \rangle}} \alpha_{ijm} B_i^4(f) B_j^4(T) B_m^4(\langle D \rangle) \tag{6.26}
\]

The \( \alpha_{ijm} \) in (26) are the stored Mie B-spline coefficients, and MATLAB Spline toolbox was used for the calculation of these coefficients. The generation of B-spline coefficients is a one-time process that does not need to be repeated. The code for reconstructing the Mie products is a straightforward implementation of (26).

6.5 Results and discussion

In this section we discuss the five main aspects of this work - memory and overhead reduction, radiation Jacobian, reconstruction error, computational savings, and radiative transfer simulations. The simulations and related software codes were written in the MATLAB environment.

6.5.1 Memory and Computational Overhead Reduction

To generate the Mie LUTs the mean diameter and frequency ranges were logarithmically gridded with 200 and 60 points, respectively, and the temperature range was gridded linearly with an interval of \( 2.5^\circ \text{C} \). The eight normalized Mie products were sampled at these grid points and stored as eight LUTs. The total memory usage of these LUTs is \( \sim 10.6 \text{ MB} \). The product LUTs are subsequently used for spline interpolation (i.e., piecewise polynomial spline interpolation). The eight product LUTs were converted to eight corresponding B-spline LUTs which contain the B-spline coefficients. From (26), it can be inferred that the size of B-spline LUTs is nearly the same as the size of corresponding product LUTs. The disadvantage of using piecewise polynomial spline interpolation is that the values stored in the product LUTs need to be converted to the piecewise spline polynomial coefficients. This procedure is not localized to a small subset of intervals, thus resulting in the required calculation of all the polynomial coefficients for each interpolation. Hence there is a fixed overhead time required for the evaluation of all the piecewise polynomial coefficients irrespective of the number of product values that need to be interpolated. In the trivariate case, if
the number of values in an individual product LUT is $N$, the total number of piecewise polynomial coefficients will be $4^3 N = 64N$. Therefore in our case, assuming one product LUT is $\sim 1.25$ MB in size, a RAM usage of 80 MB will be required when performing the piecewise polynomial spline interpolation. This high memory usage also limits the possibility of storing the piecewise polynomial coefficients as LUTs. The fixed overhead and high memory usage thus obviates many of the advantages of conventional splines.

In contrast, B-form interpolation is both localized to a small subset of intervals, requires a large fixed overhead time to calculate the $\alpha_{ijm}$ coefficients, but does not require the temporary storage of large number of spline coefficients. However, when sufficiently large numbers of products needs to be evaluated by interpolation, the spline interpolation has an advantage over B-spline method. This computational advantage is due to the fact that for the spline interpolation, once all the piecewise polynomial coefficients are calculated, the evaluation of these piecewise functions is extremely rapid compared to the three nested summations required for B-splines, as given by (26).

6.5.2 Radiation Jacobian

The radiation Jacobian is defined as the derivative of the radiance field with respect to any electromagnetic parameter of the atmosphere [89]. Therefore, the derivative of any coefficient with respect to (for example) the mean diameter can be evaluated by differentiation of (6.16), (6.17) using Leibnitz rule for differentiation under integral sign. Surface plots of the radiation Jacobian of different Mie products for ice and liquid hydrometeors at $T = 0^\circ C$ calculated using analytical differentiation are shown in Fig. (6.11)-(6.18). One possible way to evaluate Jacobian is facilitated by analytical differentiation of the cubic B-spline. Therefore, the derivative of any coefficient with respect to (for example) the mean diameter can possibly be evaluated as follows,

$$
\frac{dF(f,T,\langle D \rangle)}{d\langle D \rangle} = \sum_{i=1}^{N_f} \sum_{j=1}^{N_T} \sum_{m=1}^{N_{\langle D \rangle}} \alpha_{ijm} B_i^f(f) B_j^T(T) \frac{dB_m^4(\langle D \rangle)}{d\langle D \rangle}.
$$

(6.27)

However, this procedure cannot produce accurate estimates consistently. This is because the B-spline coefficients were calculated using the product coefficients. A better approach is to generate
B-spline coefficients using analytically calculated Jacobian values in the \((f, <D>, T)\) grid and then use these B-spline coefficients, \(\beta_{ijm}\) in (6.26). Error analysis was performed by comparing the B-spline evaluated coefficient values with analytically obtained values for 20000 random \((f, <D>, T)\) points. This was done for \(\kappa_a, \kappa_s\) for both liquid and ice hydrometeors. The mean fractional error in all the four cases were of the order of \(10^{-4}\). The maximum fractional error was less than 0.1 for all the cases except liquid \(\kappa_a\) for which it was 1.8. However, the method given by (27), can be used in a 2D case, where we are only concerned about radiometric frequencies, thereby reducing the 3D LUT to a 2D LUT and hence allowing for much finer discretization along the \(\langle D\rangle\) and \(T\) axes.

6.5.3 Computational Savings

In this section, the improvement in computational speed using the fast Mie library is discussed. In Fig. 6.1, the logarithmic ratio of the B-spline interpolation time to the exact calculation time using the full Mie series is illustrated for liquid hydrometeor absorption at \(T = 0^\circ\text{C}\). This time ratio will be nearly the same for all other products and at any temperature. From the Fig. 6.1, it can be seen that there is considerable speedup when the hydrometeors are large in diameter compared to the wavelength. In this case, the time consuming step of integration over the drop-size distribution is obviated. Usually radiative transfer simulations are performed at the channel frequencies of the remote sensing instrument. In such a case, the B-spline coefficients for only the instrument’s channel frequencies need be stored. This reduced storage necessitates only two dimensional interpolation, which is faster than the three dimensional interpolation. Equivalently, for the same amount of memory, in the two dimensional case a finer gridding can be used to provide extreme accuracy.

6.5.4 Reconstruction Error Analysis

In this section, we derive an expression for the maximum error that can be allowed for the reconstructed values of Mie products, and use this to determine the overall library accuracy. Any error in the absorption or scattering coefficient will result in an associated error in the computed
brightness temperature. By using the condition, that the maximum allowed brightness temperature error is bounded by a prescribed value we can derive expressions for the maximum error bound on the coefficients. We use the approach in [90] to derive the absorption coefficient incremental weighting function (IWF) for downwelling radiation. The IWF describes the relationship between infinitesimal variations in any atmospheric parameter and the downwelling brightness temperature. The downwelling brightness temperature observed at height \( h \) and zenith angle \( \theta \) for a non-scattering plane-parallel atmosphere is,

\[
T_{DB} (h, \theta, \nu) = T_B^h (T_{CB}, \nu) \exp \left[ - \int_h^\infty \kappa_\alpha^0 (z) \sec \theta dz \right]
+ \int_h^\infty T_B^h (T (z), \nu) \sec \theta \kappa_\alpha^0 (z) \exp \left[ - \int_h^z \kappa_\alpha^0 (z') \sec \theta dz' \right] dz
\]  

(6.28)

In (28), \( \nu \) denotes the frequency and \( T_{CB}, \kappa_\alpha^0 (z), T (z) \) and \( T_B^h (.) \) represents the cosmic background temperature, absorption coefficient as a function of altitude, physical temperature as a function of altitude and blackbody brightness temperature function respectively. The downwelling brightness temperature is due to the cosmic background radiation and the thermal emission, both attenuated by the atmospheric absorption. If \( \delta \kappa_\alpha (z) \) is the variation in the absorption coefficient profile, the corresponding variation in the observed downwelling brightness temperature, \( T_{DB} (h, \theta, \nu) \) denoted by \( \delta T_{DB} \) is given by,

\[
\delta T_{DB} = \int_h^\infty W_{\kappa_\alpha}^\downarrow (z, \theta, \nu) \delta \kappa_\alpha (z) dz
\]  

(6.29)

where \( W_{\kappa_\alpha}^\downarrow (z, \theta, \nu) \) is the absorption IWF for downwelling brightness temperature. An approximate expression for \( W_{\kappa_\alpha}^\downarrow (z, \theta, \nu) \) is derived by substituting \( \kappa_\alpha^0 (z) + \delta \kappa_\alpha (z) \) in (28) and subtracting the
orginal expression for $T_{DB}(h, \theta, \nu),$

$$W_{\kappa_a}^\downarrow(z, \theta, \nu) = -T_B^h(T_{CB}, \nu) \sec \theta \exp \left[-\int_h^\infty \kappa_a^o(z) \sec \theta dz \right]$$

$$+ T_B^h(T(z), \nu) \sec \theta \exp \left[-\int_h^z \kappa_a^o(\tilde{z}) \sec \theta d\tilde{z} \right]$$

$$- \sec^2 \theta \int_z^\infty T_B^h(T(\tilde{z}'), \nu) \kappa_a^o(\tilde{z}') d\tilde{z}'$$

$$\exp \left[ -\int_h^\infty \kappa_a^o(\tilde{z}) \sec \theta d\tilde{z} \right]$$

By applying the following assumptions to (30), the IWF can be simplified: 1) the observation point is assumed to be on the ground i.e., $h = 0$. 2) $\kappa_a(z) = 0; z > H$, where $H = 8 \text{ km}$ is the atmospheric scale height. 3) $\kappa_a^o(z) = \kappa_a^o$ i.e. a constant atmospheric absorption coefficient profile is assumed, and 4) the thermodynamic temperature, $T(z) = T$ is constant. The simplified IWF then becomes,

$$W_{\kappa_a}^\downarrow(z, \theta, \nu) = \sec \theta \exp \left[-\kappa_a^o \sec \theta H \right]$$

$$\left( T_B^h(T, \nu) - T_B^h(T_{CB}, \nu) \right)$$

The simplified expression for $\delta T_{DB}$ is obtained by substituting (31) in (29),

$$\delta T_{DB} = \delta \kappa_a H \sec \theta \exp \left[-\kappa_a^o \sec \theta H \right]$$

$$\left( T_B^h(T, \nu) - T_B^h(T_{CB}, \nu) \right)$$

Since we are concerned about the maximum possible value of the brightness temperature deviation, $|\delta T_{DB}|_{max}$, (32) is maximized by choosing an extreme atmospheric temperature of $T = 325 \text{K}$. In this case, $|\delta T_{DB}|_{max}$ becomes,

$$|\delta T_{DB}|_{max} = 2.56 \times 10^6 \delta \kappa_a \sec \theta \exp \left[-\kappa_a^o \sec \theta H \right]$$

Constraining the maximum brightness temperature error by 10 mK yields,

$$|\delta T_{DB}|_{max} \leq 0.01 K \Rightarrow \frac{\delta \kappa_a}{\kappa_a^o} \leq \frac{3.9 \times 10^{-9}}{\kappa_a^o \sec \theta \exp \left[-\kappa_a^o \sec \theta H \right]}$$
For a constant $\kappa_a^0$, the expression on the right side of (34) can be minimized with respect to $\theta$, thus resulting in an expression for maximum allowable fractional error in the reconstructed value of the absorption coefficient,

$$\left| \frac{\delta \kappa_a}{\kappa_a^0} \right|_{\text{max}} = \begin{cases} 8.48 \times 10^{-5} & ; \ k_a^0 < \frac{1}{\pi} \\ \frac{3.9 \times 10^{-5} \exp \kappa_a^0 H}{\kappa_a^0} & ; \ k_a^0 > \frac{1}{\pi} \end{cases}$$  (6.35)

It should be noted that the assumptions we have used to derive this expression are conservative, and not normally encountered in practice.

The model in (35) can be used to evaluate the accuracy of the spline reconstructed absorption coefficient values. In Fig. 6.2, the fractional error in reconstructed values of liquid and ice hydrometeor absorption are plotted against the corresponding absorption coefficients for 2000 random values of $(f, <D>, T)$ and fractional volume. The maximum allowable fractional error in the absorption coefficient for brightness temperature errors less than 0.01 K and 0.1 K are also plotted. It can be seen that for the prescribed level of library discretization the liquid absorption coefficient fractional error is always well below the 0.01 K error bound. However the ice absorption coefficient fractional error is higher than the water absorption coefficient, but still smaller than $\sim 0.1K$. This error is attributed to the ice dielectric model that was used in this work [86], which has discontinuities in its derivatives. The reconstruction error for ice hydrometeors can be reduced further by either finer sampling or by smoothing the ice dielectric constant model before calculating the ice extinction coefficients.

### 6.5.5 Radiative Transfer Simulations

In this section, the results of RT simulations performed to verify the accuracy of the developed library is presented. These results confirm the applicability of the library for RT computations. The atmosphere used for the RT simulations was the 1976 version of U.S. Standard Atmosphere, with water vapor density in each level corresponding to 50% of the saturation vapor density at the level pressure and temperature. Surface reflectivity is computed at a particular frequency by assuming an ocean surface with salinity 3.5% and using the Ray model [85] and Klein and Swift
model [91] for dielectric constant and conductivity respectively. The base atmosphere was modified by inserting liquid and ice hydrometeors over $1 - 10$ km [66]. The simulations were carried out in the frequency range $[1, 1000]$ GHz in steps of $1$ GHz.

In the first simulation, multiple scattering was ignored by setting the extinction coefficient of each level to the sum of absorption coefficient and scattering coefficient. The relative difference in the top of the atmosphere brightness temperature for a downward looking radiometer was obtained as a function of frequency. The relative difference corresponds to the difference between the cases when real Mie series value and spline interpolated values of extinction coefficients were used. The maximum and the mean value for this relative difference are $1.15 \times 10^{-3}\%$ and $3 \times 10^{-5}\%$ respectively.

In the second RT simulation, multiple scattering was taken into account by using Henyey-Greenstein phase function and a scattering based RT model [66]. The maximum and mean value of the relative difference are $9.72 \times 10^{-2}\%$ and $7.21 \times 10^{-3}\%$ respectively.

6.6 Conclusions

In this work, the absorption coefficient, scattering coefficient, backscattering coefficient, and phase asymmetry parameter of both liquid and ice spherical, homogenous hydrometeors as a function of frequency, temperature and mean diameter of the hydrometeor distribution is represented in a piecewise functional form using trivariate cubic B-splines. By using this method, it was possible to achieve significant computation time reduction for calculating the extinction parameters and phase asymmetry parameter, especially for large hydrometeors and at high microwave frequencies. Furthermore, the reconstruction error that is caused by the spline interpolation is negligible enough to preclude any adverse impact on the accuracy of radiative transfer simulations for most relevant terrestrial applications. The memory requirement for this fast library is around 10.6 MB for all eight products. The library further supports evaluation of the radiation Jacobian by either the rapid analytical differentiation of the B-spline basis functions or for higher accuracy, B-spline coefficients can be calculated by using true Jacobian values. The reconstruction time, memory usage can be further decreased with improvement in accuracy by storing the B-spline coefficients only
at frequencies of interest where radiative transfer simulation is required. The cubic B-spline based approximation method can be applied to other geophysical problems, where function evaluation is a time consuming process and reconstruction accuracy is critical. Future work will include applying these techniques to nonspherical or even multi-phase hydrometeors.

Acknowledgment

The authors are grateful to Prof. Gregory Beylkin of University of Colorado, Boulder for valuable suggestions regarding B-splines and Prof. Christian Maetzler for providing the code to calculate Mie efficiencies. The authors would also like to thank Dr. Bjorn Lambrigsten of NASA Jet Propulsion Laboratory for his support of this study. This work was funded by award number 1358415 from the NASA Jet Propulsion Laboratory.
Figure 6.1: Logarithmic ratio of the calculation time for B-spline interpolation to that for exact Mie series calculation of polydispersed liquid hydrometeor absorption coefficients. An exponential drop-size distribution, temperature $T = 0^\circ C$ and fractional volume $f = 10^{-6}$ are assumed.

Figure 6.2: Fractional error in reconstructed values of liquid and ice hydrometeor absorption coefficient vs. liquid and ice hydrometeor absorption values. The maximum allowable error bounds for $T_{DB} = 0.01 K$ and 0.1 K are shown.
Figure 6.3: Liquid hydrometeor $\kappa_a$(dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$

Figure 6.4: Ice hydrometeor $\kappa_a$(dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$
Figure 6.5: Liquid hydrometeor $\kappa_s$(dB/km) vs. ($f, \langle D \rangle$) at $T = 0^\circ C$

Figure 6.6: Ice hydrometeor $\kappa_s$(dB/km) vs. ($f, \langle D \rangle$) at $T = 0^\circ C$
Figure 6.7: Liquid hydrometeor $\kappa_b$(dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$

Figure 6.8: Ice hydrometeor $\kappa_b$(dB/km) vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$
Figure 6.9: Liquid hydrometeor $g$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$

Figure 6.10: Ice hydrometeor $g$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$
Figure 6.11: Liquid hydrometeor $\frac{d\rho_a}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ$C

Figure 6.12: Ice hydrometeor $\frac{d\rho_a}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ$C
Figure 6.13: Liquid hydrometeor $\frac{d\kappa}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$

Figure 6.14: Ice hydrometeor $\frac{d\kappa}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$
Figure 6.15: Liquid hydrometeor $\frac{d\kappa_b}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$

Figure 6.16: Ice hydrometeor $\frac{d\kappa_b}{d\langle D \rangle}$ vs. $(f, \langle D \rangle)$ at $T = 0^\circ C$
Figure 6.17: Liquid hydrometeor \( \frac{dq}{d(D)} \) vs. \((f, \langle D \rangle)\) at \(T = 0^\circ C\)

Figure 6.18: Ice hydrometeor \( \frac{dq}{d(D)} \) vs. \((f, \langle D \rangle)\) at \(T = 0^\circ C\)
Chapter 7

Conclusions and Future work

7.1 Conclusions

This thesis is a “hybrid thesis” rather than a monolithic one. The central theme of this thesis is the full wave electromagnetic analysis of radiometer calibration targets. The originality of the work comes from the fact that a doubly dispersive 3D FDTD code capable of being run on a distributed processor system was developed from scratch. The developed code was well validated and reflectivity spectrum of a wide variety of calibration target geometries, aspect ratios and coating thicknesses were obtained by using the custom code. The reflectivity spectrum was validated using HFSS commercial software and asymptotic GO technique. This resulted in the first full wave electromagnetic analysis of calibration targets published in a peer reviewed journal. This work supercedes the previous work of [10], where only 2D wedge shaped structures were analyzed. The 2D targets were made of homogenous material and the variation of constitutive parameters as a function of frequency was not taken into account. In this work the aluminum core is also taken into account and measured values of $\varepsilon(f), \mu(f)$ in the range $[8, 26]$ GHz are used. For the results outside this frequency range, a Debye series extrapolation is used. From a conceptual point of view, the reflectivity spectrum of all the cases exhibit a general trend. At low frequencies, the wavelength is large resulting in the corrugated surface appearing “flat” to incident wave. Hence the reflectivity is quite high, because it is similar to a plane wave incident on a thin dispersive coating on a PEC ground plane. This occurs at frequencies where the Rayleigh criterion for smooth surface is satisfied (i.e. $f < \frac{c}{8H}$, where $H$ is the pyramid height). This is the reason why the low frequency
reflectivity is essentially same for different target cross-sections (square, circular, truncated etc). As the frequency of the incident plane wave is increased, there is a sharp increase in reflectivity at the point where the first non-specular Floquet mode starts propagating. At high frequencies, the GO behavior dominates and the reflectivity oscillations arise from the standing wave in the coating. Further contributions include analysis of alternate structures such as conical targets, truncated pyramids and truncated conical pyramids with spherical top.

The near field thermal emission from calibration targets (which is measured by the antennas during radiometer calibration) is a difficult problem from a numerical/analytical perspective. Some guidelines along this direction is given in the thesis in chapter 5 and this chapter. Such a work if done could supercede this thesis. The two other contributions in this thesis (\(\sim 50\%\)) are given in chapters 4 and 6.

In order to compare the quality of this work with the similar studies done in China, we refer the reader to few conference papers [1, 92]. A reflectivity spectrum copied from [1] is shown in Fig. 7.1. These works [1, 92] are flawed due to several reasons as listed below.

![Reflectivity spectrum as shown in [1]](image)

**Figure 7.1:** Reflectivity spectrum as shown in [1]

1. The authors claim to have used FDTD with subgridding method to analyse this problem.

   But from the paper, it is obvious that they have used some commercial software package
(most likely CST Microwave Studio). In order to make readers believe that they have used subgridding, they have copied and pasted the details of subgridding from [93, 15] into their conference papers. Also the figure depicting subgridding is copied and pasted. Subgridding as given in [93, 15] is a very complex procedure which includes 3D spatial interpolation and time interpolation between successive FDTD updation. It is not trivial to implement it in a 3D environment. Moreover, the subgridding in [93, 15] is done for boundary surfaces parallel to the coordinate planes, which is not in the case for pyramids.

(2) FDTD is a time domain method, but they have not mentioned anything about dispersive FDTD which is essential for handling the dispersive coating. A table is given showing the electrical properties of the CR110 and CR112 material. But the data is given in a piecewise constant format. Since the sources used in FDTD simulations are of finite duration, the frequency domain content of the signals will not be confined to a single frequency. Hence dispersive FDTD is required for dispersive materials. Non-dispersive FDTD can be used only if the constitutive parameters do not vary across the frequency spectrum of the source signal.

(3) The results shown in these papers are not validated with an independent method, which is essential to confirm the accuracy of the results. An erroneous feature that can be observed in Fig. 7.1 is the reflectivity going above 0 dB for the pyramids with no coating.

(4) In Fig. 7.1, it is obvious that they have solved the problem at discrete frequencies (shown by markers) and joined the discrete results with straight lines. This is characteristic of frequency domain methods such as FEM and is contradictory to the authors claim of using FDTD. The shapes of these curves are of a “zig-zag” nature. However, in our work, the reflectivity spectrum for various aspect ratios and coating thicknesses exhibit a general trend.
7.2 Future work

From a numerical/theoretical perspective, the future work in this problem include the following investigations.

(1) Measurement of complex relative permittivity and permeability of ECCOSORB™ materials such as MF-110, MF-112, MF-114 and MF-124 for frequencies greater than 26 GHz. The Debye series based extrapolation is not a reliable method, considering the fact these materials are composite materials.

(2) Measurement of specific heat capacity and conductivity of ECCOSORB™ materials. The ECCOSORB MF series material datasheet gives the following data: the density $\rho = 1.6 \times 10^3 - 4.9 \times 10^3$ Kg m$^{-3}$ and thermal conductivity $k \approx 1.2552$ W m$^{-1}$ K$^{-1}$.

(3) Use of commercial CEM softwares such as HFSS, FEKO, CST and COMSOL. These softwares may be more faster than custom dispersive FDTD code and may give oblique plane wave reflectivity. However the 3D FDTD code developed as part of thesis is a very reliable analysis and validation tool, which can be used along with these standard CEM softwares. The importance of the custom code is manifested in estimating the reflectivity spectrum of 1:4 aspect ratio conical pyramids, where there was considerable discrepancy between the FDTD and HFSS results. By performing a convergence study, the correctness of the FDTD results were confirmed.

(4) Analysis of finite array of square pyramids should be attempted. In this thesis, PEC/PMC boundary conditions are used to simulate an infinite array, which is only an approximation to the real scenario. One possible technique to handle finite array is to use Characteristic Basis Function Method (CBFM) which can solve electrically large scattering problems [94, 95, 96]. CBFM is a relatively new technique that reduces the number of unknowns and hence the size of MoM matrix. It is a “physics based method”, where MoM basis functions are specially constructed to fit the problem geometry by incorporating the physics of the
problem into their generation. However this should be attempted only after verifying the size of the largest finite array FEKO can simulate (preferably on a multicore system), thus making sure whether FEKO can satisfy our requirements. Typically, MoM is used for solving scattering from moderate-size objects in terms of the wavelength.

(5) Use of COMSOL multiphysics software for solving the heat equation for the periodic square pyramid array with suitable thermal boundary conditions.

(6) Development of a stochastic numerical method to estimate near field thermal emission from structures of arbitrary geometry and temperature profile. Temperature profile is obtained by the COMSOL heat equation solution. The numerical method may be either WCE (Wiener Chaos Expansion) or a weighting function based method [10]. Even if such a method is developed, the accuracy of estimating the near field brightness temperature is dependent on how accurately we know the pointwise temperature distribution inside the target. The uncertainty in our knowledge of the temperature distribution affects the accuracy of the estimated brightness temperature.

(7) One of the difficulty in using the FDT-DGF method outlined in chapter 5 for near field thermal emission is the fact that DGF can be derived only for certain simple geometries. Obtaining DGF from FDTD is a not a practical task and had never been done before, because each FDTD simulation will give just the 3 components among the 9 components of the DGF for fixed source and observation position!. The number of simulations can be reduced by coding the sources. Due to the linearity of Maxwell’s equations and assuming a nonlinear media, sources of different frequencies do not result in harmonic generation. Hence we can use sources of different frequencies at different source points. For example, assume the observation point \( \mathbf{r} \) is fixed. \( n \) sources are defined by \( S_n(\mathbf{r}_n^i; f_n) \), where \( \mathbf{r}_n^i \) is the source location and \( f_n \) its frequency. Let the sources be directed along \( x \) direction. By recording the vector field time series at observation point \( \mathbf{r} \) and performing a spectral estimation procedure, the three DGF components \( \left\{ G^{xx}_e(\mathbf{r};\mathbf{r}_n^i; f_n), G^{yx}_e(\mathbf{r};\mathbf{r}_n^i; f_n), G^{zx}_e(\mathbf{r};\mathbf{r}_n^i; f_n) \right\} \) for various \( \mathbf{r}_n^i \) at \( f_n \)
can be obtained. Suppose we want to find \( \{ G_{xx}^e(\vec{r},\vec{r}'; f_n), G_{yx}^e(\vec{r},\vec{r}'; f_n), G_{zx}^e(\vec{r},\vec{r}'; f_l) \} \), where \( f_l = \{ f_1, f_2, \cdots, f_n \} \) using FDTD without source coding will take \( n^2 \) simulations, one the other hand by using source coding it will take \( n \) simulations. The obvious bottleneck is that when have reduced the number of simulations by \( n \), the DGF components obtained are at different frequencies given by list \( f_l \). Furthermore, large number of sources in FDTD (without any frequency overlap) may be difficult. Gaussian modulated sinusoids have minimum overlap if their center frequencies are sufficiently apart.

### 7.2.1 Heat Diffusion Equation

The heat diffusion equation is given by [97]

\[
\frac{\partial}{\partial x} \left( k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( k \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( k \frac{\partial T}{\partial z} \right) + \dot{q} = \rho c_p \frac{\partial T}{\partial t} \tag{7.1}
\]

where \( T(\vec{r},t) \) is the temperature, \( k(\vec{r}) \) (Wm\(^{-1}\)K\(^{-1}\)) is the thermal conductivity, \( \rho(\vec{r}) \) is the density and \( c_p(\vec{r}) \) is the specific heat capacity of the material. \( \dot{q}(\vec{r}) \) (Wm\(^{-3}\)) is the heat rate per unit volume (heat source). Assuming steady state conditions and no internal heat source, the heat diffusion equation simplifies to the following equation.

\[
\frac{\partial}{\partial x} \left( k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left( k \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left( k \frac{\partial T}{\partial z} \right) = 0 \tag{7.2}
\]

The steady state temperature in the calibration target array is obtained by numerically solving the above equation subjected to any of the following boundary conditions [97].

1. **Constant surface temperature**: Boundary surface is held at a constant temperature, \( T_s \)

2. **Finite heat flux**: \(-k \frac{\partial T}{\partial n} \big|_s = q''_s\)

3. **Adiabatic or insulated surface**: \( \frac{\partial T}{\partial n} \big|_s = 0 \)

4. **Convection boundary condition**: \(-k \frac{\partial T}{\partial n} \big|_s = h [T_\infty - T_s] \)

5. **Periodic boundary condition**
where $h$ is the convection heat transfer coefficient. The value of $h$ depends on number of factors such as surface geometry, nature of fluid motion etc. Its exact value will not be known. For free convection from a surface to a gaseous fluid, $h \in [2, 25] \text{ Wm}^{-2}\text{K}^{-1}$ [97]. $T_\infty$ is the temperature of the fluid in contact with the surface. $\frac{dT}{dn}$ is the normal derivative of temperature on the boundary surface.
Bibliography


W. ECKHARDT. First and second fluctuation-dissipation-theorem in electromagnetic fluctuation theory. OPTICS COMMUNICATIONS, 41(5), may 1982.


Appendix A

3D dispersive FDTD code

/* FDTD code for broadband electromagnetic analysis of electric and magnetic dispersive periodic structures
 - 3D
 - UPML on y-axis boundaries
 - PBC/SBC on x,z-axis boundaries
 - Electric,Magnetic Dispersive
 - Parallel FDTD

1) E,D vectors on yee cell edge centres and H,B vectors on Yee cell face centres.
Ex[i + 0.5][j][k], Ey[i][j + 0.5][k], Ez[i][j][k + 0.5]
Dx[i + 0.5][j][k], Dy[i][j + 0.5][k], Dz[i][j][k + 0.5]
Hx[i][j + 0.5][k + 0.5], Hy[i + 0.5][j][k + 0.5], Hz[i + 0.5][j][k + 0.5]
Bx[i][j + 0.5][k + 0.5], By[i + 0.5][j][k + 0.5], Bz[i + 0.5][j + 0.5][k]

2) UPML field components : Ex,Ey,Ez,Dx,Dy,Dz,Hx,Hy,Hz,Bx ,By,Bz
Main grid field components : Ex,Ey,Ez,Hx,Hy,Hz

3) UPML slabs just on ymin and ymax
PBC/SBC on xmin,xmax,zmin,zmax

4) C1 constants defined only for UPML
K1 constants defined only for maingrid
RA only for maingrid

5) Field components on UPML-problem space boundary are calculated with UPML update equations

6) PML0 - PML on ymin boundary (rank 1)
PML1 - PML on ymax boundary (rank 2)
*/

#include "mpi.h"
#include "iostream.h"
#include "math.h"
#include "stdlib.h"
#include "fstream.h"
#include "Defs.h"
#include "Matrix.h"
#include "AuxFuncs.h"
#include "complex.h"
#include "fft.h"

#define PL 12 //Number of PML layers.
int PSX; //Number of yee cells in problem space along x-direction
int PSY; //Number of yee cells in problem space along y-direction
int PSZ; //Number of yee cells in problem space along z-direction
int XCELLS; //Total number of yee cells along x-direction
int YCELLS; // Total number of yee cells along y-direction
int ZCELLS; // Total number of yee cells along z-direction
int OP;
int SP = PL + 25;
double dx, dy, dz;

// Switch between PBC or SBC
const bool SBC = true;
// Switch between incident field and total field simulation
bool INCSIM = true;
// Source excitation type
const int source = DIFFGAUSSIAN;
bool POSTPROC = false;

// Structure dimension
// mp : base/height of the pyramid (PBC)
// dbase : Number of yee cells for the base
// basewidth in cm
// aircells : Number of air cells behind the metal base
double mp = 0.25;
const uint dbase = 4;
uint coatthck = 1; // Thickness in mm
uint coating;
double basewidth = 1.27;
uint pyrtopyidx;

// Parallelization
int yeePerProc;
int NProcs;

// Matrices
MatPtr3 Dx, Dy, Dz, Bx, By, Ez, Ex, Ey, Hx, Hy, Hz;
MatPtr3 Dxo, Dyo, Dzo;
MatPtr3 Bxo, Byo, Bzo;
MatPtr3 C1Dx, C2Dx;
MatPtr3 C1Bx, C2Bx;
MatPtr3 C2Ey, C3Ey, C1Ex, C2Ex;
MatPtr3 C2Hy, C3Hy, C1Hz, C2Hz;
MatPtr3 K1Ex, K2Ex, K1Ey, K2Ey, K1Ez, K2Ez;
MatPtr3 K1Hx, K2Hx, K1Hy, K2Hy, K1Hz, K2Hz;
MatPtr4 RAEx, RAEy, RAEx, RAHy, RAHz;

// Material matrix
UintMat MediaFullDomain;

// Function prototypes
void CreateAllMatrices(int rank);
void DeleteAllMatrices(int rank);
void FillUpdateEqnConstantMatrices(double dx, double sigmao, double d, double m, double dt, int rank);
void InitMaterialMatrix();
uint AddMetalPyr(double dx, uint material);
uint AddCoating(double dx, uint material);
void ConvToSBC(int rank);
double GetEpsinf(int xidx, int yidx, int zidx);
double GetSigma(int xidx, int yidx, int zidx);
double GetMuinf(int xidx, int yidx, int zidx);
double GetXmDebye(int xidx, int yidx, int zidx, int p, double dt, uint pole);
double GetXeDebye(int xidx, int yidx, int zidx, int p, double dt, uint pole);
double GetXmDebye(int xidx, int yidx, int zidx, int p, double dt, uint pole);
double GetXeDebye(int xidx, int yidx, int zidx, int p, double dt, uint pole);
double GetTaumu(uint media, uint pole);
double GetTaueps(uint media, uint pole);
double Round(double);
inline bool IsDispersive(uint media) { return (media == MF112 || media == MF110 || media == MF114); }
void TimeAlignHfield(double* field, uint sz);
void TimeAlignHfield(complex* field, uint sz);
double GetFieldVal(uint xidx, uint yidx, uint zidx, uint fieldComp);

int main(int argc, char* argv[]) {
   // Input argument to switch between incident and total field
   // simulation
   if(argc > 1) {
      if(!strcmp(argv[1],"t") || !strcmp(argv[1],"T"))
         INCSIM = false;
      else if(!strcmp(argv[1],"i") || !strcmp(argv[1],"I"))
         INCSIM = true;
      else if(!strcmp(argv[1],"p") || !strcmp(argv[1],"P")) {
         POSTPROC = true;
         INCSIM = false;
      }
   }

   coatthck = strtod(argv[2],\'\0\');
   mp = strtod(argv[3],\'\0\');

   // Output frequencies
   int NumofOutFreq = 389;
   double OutFreqs[389];
   int outfreqcnt = 0;
   for(double outfr = 6; outfr <= 200; outfr += 0.5) {
      OutFreqs[outfreqcnt++] = outfr*1e9;
   }

   // Initialize MPI
   int rank;
   MPI_Status status;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
   MPI_Comm_size(MPI_COMM_WORLD,&NProcs);
   T = 19000;
   i,j,k,n,pidx;
   ii, jj, kk;
   mpidx;
   double time = 0;
   int SPrank, SPidx;
   int OPrank, OPidx;
   RAExtot, Exo, RAEztot, Ez0, RAExytot, Eyo,
   RAHxtot, Hxo, RAHytot, Hz0, RAHztot;
   double DXeEx0, DXieEx0, CEx, DXeEz0, DXieEz0,
   CEz, DXeEy0, DXieEy0, CEy;
   bool dispcond;

   ofstream outEz, outEy, outEx;
   ofstream outHx, outHy, outHz;
   ofstream outMedia;

   // Initializations
   int invC = 2; spd = 28;

   C = 1/(double)(invC);
if(source == GAUSSIAN) {
    f = 26e9;
    tau = sqrt(2.3)/pi/f;
    to = 4*tau;
    dx = c/f/spd;
    dy = dx;dz = dx;
    dt = dz*C/c;
}
else if(source == MODGAUSSIAN) {
    //fc - center frequency
    //Df - BW
    Df = 153e9;
    fc = 147e9;
    tau = 2/pi/(Df);
    dx = c/(fc + Df)/spd;
    dy = dx;dz = dx;
    dt = dz*C/c;
    to = 4*tau;
}
else if(source == EXPRAMPSINE) {
    fs = 30e9;
    dx = c/fs/spd;
    dy = dx;dz = dx;
    dt = dz*C/c;
}
else if(source == DIFFGAUSSIAN) {
    fl = 6e9;
    fu = 200e9;
    Tb = (1/pi)*sqrt(log(fu/fl)/(fu*fu - fl*fl));
    to = 3.2*Tb;
    dx = c/200e9/spd;
    dy = dx;dz = dx;
    dt = dz*C/c;
}

//Number of cells for coating
coating = Round(sqrt(1+pow(2/mp,2))*coatthck*1e-3/dy);
//Determine the number of cells for the base (full base)
PSX = Round(basewidth*1e-2/dx);
//Make PSX even
if((PSX%2))
    PSX = PSX + 1;
PSZ = PSX;
//Rounded up basewidth (cm)
basewidth = PSX*dx*100;
XCELLS = PSX;ZCELLS = PSZ;
PSY = Round((basewidth/sp)*1e-2/dx) + coating + 200 + dbase + aircells;
//Number of yee cells per processor in problem space
yeePerProc = (int)ceil(double(PSY)/double(NProcs-3));
YCELLS = PSY + (2 * PL);

// Polynomial grading (UPML)
double m = 3.5;
double sigmao = -log(exp(-16))*(m+1)/(2*120*pi*dx*PL);
double d = PL*dx;
if(!POSTPROC) {
    CreateAllMatrices(rank);
    InitMaterialMatrix();
    pyrtopyidx = AddMetalPyr(dx,AL);
    pyrtopyidx = AddCoating(dx,MF112);
    FillUpdateEqnConstantMatrices(dx,sigmao,d,m,dt,rank);
    OP = pyrtopyidx - 150;
    //OP = PL+10;
if(INCSIM) {
    //Remove the scatterer for incident field simulation
    InitMaterialMatrix();
}
FillUpdateEqnConstantMatrices(dx, sigmao, d, m, dt, rank);
}
if(SBC) {
    if(!POSTPROC)
        ConvToSBC(rank);
    XCELLS = Round(XCELLS/2);
    ZCELLS = Round(ZCELLS/2);
}

//rank 0 is not used
//Find rank and rankidx for SP, OP
int flag = 0;
while(!flag && !POSTPROC) {
    for(int ridx = 1; ridx <= NProcs - 1; ridx++) {
        int ymin, ymax;
        if(ridx == 1) {
            ymin = 0; ymax = PL;
        } else if(ridx == 2) {
            ymin = YCELLS + 1 - PL; ymax = YCELLS + 1;
        } else if(ridx == NProcs - 1) {
            ymin = PL + (ridx - 3)*yeePerProc + 1;
            ymax = PL + (ridx - 3)*yeePerProc + yeePerProc - 1;
        } else {
            ymin = PL + (ridx - 3)*yeePerProc + 1;
            ymax = PL + (ridx - 3)*yeePerProc + yeePerProc;
        }
        if(SP >= ymin && SP <= ymax) {
            SPrank = ridx;
            SPidx = SP - ymin;
        }
        if(OP >= ymin && OP <= ymax) {
            OPrank = ridx;
            OPidx = OP - ymin;
        }
    }
    //NProcs - 1 ignored for this case
    if(OPidx == yeePerProc - 1) {
        OP = OP - 1;
        flag = 0;
    } else {
        flag = 1;
    }
}
//Output files
if(rank == OPrank && !POSTPROC) {
    if(!INCSIM) {
        outEz.open("/pan1/projects/awmra/Ez.dat", ios::out);
        outEx.open("/pan1/projects/awmra/Ex.dat", ios::out);
        outEy.open("/pan1/projects/awmra/Ey.dat", ios::out);
        outHx.open("/pan1/projects/awmra/Hx.dat", ios::out);
        outHz.open("/pan1/projects/awmra/Hz.dat", ios::out);
        outHy.open("/pan1/projects/awmra/Hy.dat", ios::out);
        outMedia.open("/pan1/projects/awmra/Media.dat", ios::out);
    } else {
        outEz.open("/pan1/projects/awmra/Ezinc.dat", ios::out);
    }
}
outHx.open("/pan1/projects/awmra/Hxinc.dat",ios::out);
}
}
//Write to Media.txt
/*for(i = 0; i < XCELLS;i++) {
 for(j = 0; j < YCELLS;j++) {
 for(k = 0; k < ZCELLS;k++) {
 outMedia<<MediaFullDomain[i][j][k]<<endl;
 }
 }
 }
*/
if(rank == 0 && !POSTPROC) {
 cout<<"--------------FDTD 3D simulation--------------------\n";
 if(INCSIM)
  cout<<"Incident field simulation"<<endl;
 else
  cout<<"Total field simulation"<<endl;
 if(source == EXPRAMPSINE)
   cout<<"Sinusoidal excitation frequency = "<<fs/1e9<<" GHz"<<endl;
 cout<<"ds = lambda/#spc" = "<<dx*1000<<" mm"<<endl;
 cout<<"dt = "<<dt*1e12<<" ps"<<endl;
 cout<<"Number of Yee cells along x/z direction = "<<XCELLS<<endl;
 cout<<"Number of Yee cells along y direction = "<<YCELLS<<endl;
 cout<<endl;
 cout<<"Pyramid base width = "<<basewidth<<" cm"<<endl;
 cout<<"Pyramid height = "<<basewidth/mp<<" cm"<<endl;
 cout<<endl;
 if(SBC) cout<<"Boundary condition : PEC/PMC"<<endl;
 else cout<<"Boundary condition : Periodic Boundary Condition (PBC)"<<endl;
 if(source == GAUSSIAN) {
  cout<<"Source excitation : Gaussian"<<endl;
  cout<<"tau/dt = "<<tau/dt<<endl;
  cout<<"to/dt = "<<to/dt<<endl;
 }
 else if(source == MODGAUSSIAN) {
  cout<<"Source excitation : Mod Gaussian"<<endl;
  cout<<"tau/dt = "<<tau/dt<<endl;
  cout<<"to/dt = "<<to/dt<<endl;
 }
}

//-----FDTD update loop starts here------------------------
//Temp variables to hold previous field values and field differential values.
double dHx, dHy, dHz; 
double dEx, dEy, dEz;

//Boundary field matrices
MatPtr2 Hzb, Hxb, Ezb, Exb, Ezb1, Ezb;
goto pproc;
}

//Create boundary field matrices
if(rank == 1) {
  CreateMatrix(Hzb,XCELLS,ZCELLS + 1);
  CreateMatrix(Hxb,XCELLS + 1,ZCELLS);
}
if(rank == 2) {
  CreateMatrix(Hzb,XCELLS,ZCELLS + 1);
  CreateMatrix(Hxb,XCELLS + 1,ZCELLS);
}
if(rank >= 3 && rank <= NProcs - 2) {
  CreateMatrix(Hzb,XCELLS,ZCELLS + 1);
  CreateMatrix(Hxb,XCELLS + 1,ZCELLS);
  CreateMatrix(Ezb,XCELLS,ZCELLS + 1);
  CreateMatrix(Exb,XCELLS,ZCELLS + 1);
}
if(rank == NProcs - 1) {
  CreateMatrix(Ezb,XCELLS + 1,ZCELLS);
  CreateMatrix(Ezb1,XCELLS + 1,ZCELLS);
  CreateMatrix(Exb,XCELLS,ZCELLS + 1);
  CreateMatrix(Exb1,XCELLS,ZCELLS + 1);
}

//PML0 (rank1)
if(rank == 1) {
  for(n = 0;n < T;n++) {
    cout<<n<<endl;
    // Dx update (Dx at n)
    for(i = 0;i < XCELLS;i++) {
      for(j = 0;j < PL + 1;j++) {
        for(k = 0;k < ZCELLS + 1;k++) {
          // PEC
          if(j == 0) {
            Dx[i][j][k] = 0;
          }
          // PBC on zm
          else if(k == 0) {
            if(!SBC) {
              dHz = (Hzb[i][k] - Hz[i][j-1][k])/dy;
            }
            else {
              dHz = (Hz[i][j][k] - Hz[i][j-1][k])/dy;
            }
            dHy = (Hy[i][j][k] - Hy[i][j][ZCELLS-1])/dz;
            Dx[i][j][k] = ClDx[i][j][k]*Dx[i][j][k] + C2Dx[i][j][k]*(dHz - dHy);
          }
          // PBC on zmax
          else if(k == ZCELLS) {
            if(!SBC) {
              Dx[i][j][k] = Dx[i][j][0];
            }
            else {
              Dx[i][j][k] = 0;
            }
          }
          else {
          }
        }
      }
    }
  }
}

//PML0 (rank1)
if(rank == 1) {
  for(n = 0;n < T;n++) {
    cout<<n<<endl;
    // Dx update (Dx at n)
    for(i = 0;i < XCELLS;i++) {
      for(j = 0;j < PL + 1;j++) {
        for(k = 0;k < ZCELLS + 1;k++) {
          // PEC
          if(j == 0) {
            Dx[i][j][k] = 0;
          }
          // PBC on zm
          else if(k == 0) {
            if(!SBC) {
              dHz = (Hzb[i][k] - Hz[i][j-1][k])/dy;
            }
            else {
              dHz = (Hz[i][j][k] - Hz[i][j-1][k])/dy;
            }
            dHy = (Hy[i][j][k] - Hy[i][j][ZCELLS-1])/dz;
            Dx[i][j][k] = ClDx[i][j][k]*Dx[i][j][k] + C2Dx[i][j][k]*(dHz - dHy);
          }
          // PBC on zmax
          else if(k == ZCELLS) {
            if(!SBC) {
              Dx[i][j][k] = Dx[i][j][0];
            }
            else {
              Dx[i][j][k] = 0;
            }
          }
          else {
          }
        }
      }
    }
  }
}
if (j == PL)
    dHz = (Hzb[i][k] - Hz[i][j-1][k])/dy;
else
    dHz = (Hz[i][j][k] - Hz[i][j-1][k])/dy;

Dx[i][j][k] = C1Dx[i][j][k]*Dx[i][j][k] + C2Dx[i][j][k]*(dHz - dHy);

//Ex update (Ex at n)
for (i = 0; i < XCELLS; i++) {
    for (j = 0; j < PL + 1; j++) {
        for (k = 0; k < ZCELLS + 1; k++) {
            Ex[i][j][k] = Ex[i][j][k] + (1/epso)*(Dx[i][j][k] - Dxo[i][j][k]);
        }
    }
}

//Dy update at n
for (i = 0; i < XCELLS + 1; i++) {
    for (j = 0; j < PL; j++) {
        for (k = 0; k < ZCELLS + 1; k++) {
            //Dyo = Dy at n - 1
            Dy[i][j][k] = Dy[i][j][k];

            // BC on corners
            if (i == 0 && k == 0) {
                if (SBC) {
                    dHz = (Hz[i][j][k] - Hz[XCELLS-1][j][k])/dx;
                    dHx = (Hx[i][j][k] - Hx[i][j][ZCELLS-1])/dz;
                    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
                }
                else {
                    Dy[i][j][k] = 0;
                }
            }

            // PBC on corners
            else if ((i == 0 && k == ZCELLS) || (i == XCELLS && k == ZCELLS)) {
                if (SBC) {
                    Dy[i][j][k] = Dy[0][j][0];
                }
                else {
                    Dy[i][j][k] = 0;
                }
            }

            // BC on xmin
            else if (i == 0) {
                if (SBC) {
                    dHz = (Hz[i][j][k] - Hz[XCELLS-1][j][k])/dx;
                    dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
                    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
                }
                else {
                    dHz = 2*(Hz[i][j][k])/dx;
                    dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
                    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
                }
            }

            // BC on xmax
            else if (i == XCELLS) {
                if (SBC) {
                    dHz = (Hz[i][j][k] - Hz[XCELLS-1][j][k])/dx;
                    dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
                    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
                }
                else {
                    dHz = 2*(Hz[i][j][k])/dx;
                    dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
                    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
                }
            }
        }
    }
}
if(!SBC) {
  Dy[i][j][k] = Dy[0][j][k];
} 
else {
  dHz = -2*(Hz[XCELLS-1][j][k])/dx;
  dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;

  Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
}

// BC on zmin
else if(k == 0) {
  if(!SBC) {
    dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx;
    dHx = (Hx[i][j][k] - Hx[i][j][ZCELLS-1])/dz;

    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
  } 
  else {
    Dy[i][j][k] = 0;
  }
}
// BC on zmax
else if(k == ZCELLS) {
  if(!SBC) {
    Dy[i][j][k] = Dy[i][j][0];
  } 
  else {
    Dy[i][j][k] = 0;
  }
}
else {
  dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
  dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx;

  Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
}

// Ey update (Ey at n)
for(j = 0;j < PL;j++) {
  for(i = 0;i < XCELLS + 1;i++) {
    for(k = 0;k < ZCELLS + 1;k++) {
      Ey[i][j][k] = Ey[i][j][k] + C2Ey[i][j][k]*Dy[i][j][k]
                 - C3Ey[i][j][k]*Dyo[i][j][k];
    }
  }
}

// Dz update
for(i = 0;i < XCELLS + 1;i++) {
  for(j = 0;j < (PL + 1);j++) {
    for(k = 0;k < ZCELLS;k++) {
      // Dzo = Dz at n - 1
      Dzo[i][j][k] = Dz[i][j][k];

      // PEC
      if(j == 0) {
        Dz[i][j][k] = 0;
      }
      // BC on xmin
      else if(i == 0) {
        if(!SBC) {
          dRy = (Ry[i][j][k] - Ry[XCELLS-1][j][k])/dx;
        }
      }
    }
  }
}
if (j == PL)
    dHx = (Hxb[i][k] - Hx[i][j-1][k])/dy;
else
    dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;

Dz[i][j][k] = Dz[i][j][k] + dt*(dHy - dHx);
}
else {
    dHy = 2*(Hy[i][j][k])/dx;
    if (j == PL)
        dHx = (Hxb[i][k] - Hx[i][j-1][k])/dy;
    else
        dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;

    Dz[i][j][k] = Dz[i][j][k] + dt*(dHy - dHx);
}
}
// BC on xmax
else if (i == XCELLS) {
    if (!SBC) {
        Dz[i][j][k] = Dz[0][j][k];
    }
    else {
        dHy = -2*(Hy[XCELLS-1][j][k])/dx;
        if (j == PL)
            dHx = (Hxb[i][k] - Hx[i][j-1][k])/dy;
        else
            dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;

        Dz[i][j][k] = Dz[i][j][k] + dt*(dHy - dHx);
    }
}
else {
    dHy = (Hy[i][j][k] - Hy[i-1][j][k])/dx;
    if (j == PL)
        dHx = (Hxb[i][k] - Hx[i][j-1][k])/dy;
    else
        dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;

    Dz[i][j][k] = Dz[i][j][k] + dt*(dHy - dHx);
}
}
// Ez update (Ez at n)
for (j = 0; j < PL + 1; j++) {
    for (i = 0; i < XCELLS + 1; i++) {
        for (k = 0; k < ZCELLS; k++) {
            Ez[i][j][k] = C1Ez[i][j][k] * Ez[i][j][k] + C2Ez[i][j][j] * (Dz[i][j][k] - Dzo[i][j][k]);
        }
    }
}

// Send Ex,Ez to rank 3 proc
for (mpidx = 0; mpidx < XCELLS; mpidx++) {
    MPI_Send(&Ex[mpidx][PL][0], ZCELLS + 1, MPI_DOUBLE, rank + 2, 1*10+3*1+0, MPI_COMM_WORLD);
}
for (mpidx = 0; mpidx < XCELLS + 1; mpidx++) {
    MPI_Send(&Ez[mpidx][PL][0], ZCELLS, MPI_DOUBLE, rank + 2, 1*10+3*1+1, MPI_COMM_WORLD);
}
// B,H updates
    time = time + 0.5*dt;
//Bx update (Bx at n + 0.5)
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < PL; j++) {
        for(k = 0; k < ZCELLS; k++) {

            //Bxo = Bx at n - 0.5
            Bxo[i][j][k] = Bx[i][j][k];

            dEz = (Ez[i][j+1][k] - Ez[i][j][k])/dy;
            dEy = (Ey[i][j][k+1] - Ey[i][j][k])/dz;

            Bx[i][j][k] = C1Bx[i][j][k]*Bx[i][j][k] - C2Bx[i][j][k]*(dEz - dEy);
        }
    }
}

//Hx update (Hx at n + 0.5)
for(j = 0; j < PL; j++) {
    for(i = 0; i < XCELLS + 1; i++) {
        for(k = 0; k < ZCELLS; k++) {
            if(i == XCELLS && !SBC) {Hx[i][j][k] = Hx[0][j][k];}
            else {
                Hx[i][j][k] = Hx[i][j][k] + (1/muo)*(Bx[i][j][k] - Bxo[i][j][k]);
            }
        }
    }
}

//By update
for(i = 0; i < XCELLS; i++) {
    for(j = 0; j < PL + 1; j++) {
        for(k = 0; k < ZCELLS; k++) {

            //Byo = By at n - 0.5
            Byo[i][j][k] = By[i][j][k];

            dEx = (Ex[i][j][k+1] - Ex[i][j][k])/dz;
            dEz = (Ez[i+1][j][k] - Ez[i][j][k])/dx;

            By[i][j][k] = By[i][j][k] - dt*(dEx - dEz);
        }
    }
}

//Hy update
for(i = 0; i < XCELLS; i++) {
    for(j = 0; j < PL + 1; j++) {
        for(k = 0; k < ZCELLS; k++) {

            Hy[i][j][k] = Hy[i][j][k] + C2Hy[i][j][k]*By[i][j][k] -
                          C3Hy[i][j][k]*Byo[i][j][k];
        }
    }
}

//Bz update (Bz at n + 0.5)
for(i = 0; i < XCELLS; i++) {
    for(j = 0; j < PL; j++) {
        for(k = 0; k < ZCELLS + 1; k++) {

            //Bzo = Bz at n - 0.5
            Bzo[i][j][k] = Bz[i][j][k];

            dEy = (Ey[i+1][j][k] - Ey[i][j][k])/dx;
            dEx = (Ex[i][j+1][k] - Ex[i][j][k])/dy;

            Bz[i][j][k] = C1Bz[i][j][k]*Bz[i][j][k] - C2Bz[i][j][k]*(dEx - dEy);
        }
    }
}
//Hz update (Hz at n + 0.5)
for(i = 0; i < XCELLS; i++) {
    for(j = 0; j < PL; j++) {
        for(k = 0; k < ZCELLS + 1; k++) {
            Hz[i][j][k] = C1Hz[i][j][k]*Hz[i][j][k] + C2Hz[i][j][k]*
            *(Bz[i][j][k] - Bzo[i][j][k]);
        }
    }
}

for(mpidx = 0; mpidx < XCELLS * 1; mpidx++) {
    MPI_Recv(&Hxb[mpidx][0], ZCELLS, MPI_DOUBLE, 3, 3*10 + 1*1 + 0, MPI_COMM_WORLD,
    &status);
}

for(mpidx = 0; mpidx < XCELLS; mpidx++) {
    MPI_Recv(&Hzb[mpidx][0], ZCELLS + 1, MPI_DOUBLE, 3, 3*10 + 1*1 + 1, MPI_COMM_WORLD,
    &status);
}

time = time + 0.5*dt;
}

//PML1 (rank2)
if(rank == 2) {
    for(n = 0; n < T; n++) {
        // Dx update (Dx at n)
        for(i = 0; i < XCELLS; i++) {
            for(j = 0; j < (PL + 1); j++) {
                for(k = 0; k < ZCELLS + 1; k++) {
                    //Dxo = Dx at n - 1
                    Dxo[i][j][k] = Dx[i][j][k];
                    // PEC
                    if(j == PL) {
                        Dx[i][j][k] = 0;
                    }
                    // PBC on zmin
                    else if(k == 0) {
                        if(!SBC) {
                            if(j == 0)
                                dHz = (Hz[i][j][k] - Hzb[i][k])/dy;
                            else
                                dHz = (Hz[i][j][k] - Hz[i][j-1][k])/dy;
                            dHy = (Hy[i][j][k] - Hy[i][j][ZCELLS-1])/dz;
                            Dx[i][j][k] = C1Dx[i][j][k]*Dx[i][j][k] + C2Dx[i][j][k]*
                            *(dHz - dHy);
                        }
                    }
                    else {
                        Dx[i][j][k] = 0;
                    }
                    //PBC on zmax
                    else if(k == ZCELLS) {
                        if(!SBC) {
                            Dx[i][j][k] = Dx[i][j][0];
                        }
                    }
                    else {
                        Dx[i][j][k] = 0;
                    }
                }
            }
        }
    }
}
dHz = (Hz[i][j][k] - Hz[i][j-1][k])/dy;
dHy = (Hy[i][j][k] - Hy[i][j][k-1])/dz;

Dx[i][j][k] = C1Dx[i][j][k]*Dx[i][j][k] + C2Dx[i][j][k]*(dHz - dHy);

//Ex update at n
for(j = 0;j < PL + 1;j++) {
  for(i = 0;i < XCELLS;i++) {
    for(k = 0;k < ZCELLS + 1;k++) {
      Ex[i][j][k] = Ex[i][j][k] + (1/epso)*(Dx[i][j][k] - Dxo[i][j][k]);
    }
  }
}

//Dy update at n
for(i = 0;i < XCELLS + 1;i++) {
  for(j = 0;j < PL;j++) {
    for(k = 0;k < ZCELLS + 1;k++) {
      Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
    }
  }
}

// BC on corners
if(i == 0 & & k == 0) {
  if(!SBC) {
    dHz = (Hz[i][j][k] - Hz[XCELLS-1][j][k])/dx;
    dHx = (Hx[i][j][k] - Hx[i][j][ZCELLS-1])/dz;
    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
  }
  else {
    Dy[i][j][k] = 0;
  }
}

// PBC on corners
else if((i == 0 & & k == ZCELLS) || (i == XCELLS & & k == ZCELLS) || (i == XCELLS & & k == 0)) {
  if(!SBC) {
    Dy[i][j][k] = Dy[0][j][0];
  }
  else {
    Dy[i][j][k] = 0;
  }
}

// BC on xmin
else if(i == 0) {
  if(!SBC) {
    dHx = -2*(Hz[XCELLS-1][j][k])/dx;
    dHz = (Hz[i][j][k] - Hz[i][j][k-1])/dz;
    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
  }
  else {
    dHz = (Hz[i][j][k] - Hz[i][j][k-1])/dz;
    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
  }
}

// BC on xmax
else if(i == XCELLS) {
  if(!SBC) {
    Dy[i][j][k] = Dy[0][j][k];
  }

  else {
    dHz = -2*(Hz[XCELLS-1][j][k])/dx;
  }
}
\[
dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
\]
\[
Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
\]

// BC on zmin
else if(k == 0) {
    if(!SBC) {
        dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx;
        dHx = (Hx[i][j][k] - Hx[i][j][ZCELLS-1])/dz;
        Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
    }
    else {
        Dy[i][j][k] = 0;
    }
}
}
// BC on zmax
else if(k == ZCELLS) {
    if(!SBC) {
        Dy[i][j][k] = Dy[i][j][0];
    }
    else {
        Dy[i][j][k] = 0;
    }
}
else {
    dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
    dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx;
    Dy[i][j][k] = Dy[i][j][k] + dt*(dHx - dHz);
}
}
}

// Ey update at n
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < PL; j++) {
        for(k = 0; k < ZCELLS + 1; k++) {
            Ey[i][j][k] = Ey[i][j][k] + C2Ey[i][j][k]*Dy[i][j][k] - C3Ey[i][j][k]*Dyo[i][j][k];
        }
    }
}

// Dz update
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < (PL + 1); j++) {
        for(k = 0; k < ZCELLS; k++) {
            // Dzo = Dz at n - 1
            Dzo[i][j][k] = Dz[i][j][k];
            // PEC
            if(j == PL) {
                Dz[i][j][k] = 0;
            }
            // BC on xmin
            else if(i == 0) {
                if(!SBC) {
                    dHy = (Hy[i][j][k] - Hy[XCELLS-1][j][k])/dx;
                    if(j == 0) {
                        dHx = (Hx[i][j][k] - Hxb[i][k])/dy;
                    }
                    else {
                        dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;
                    }
                    else {
                        dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;
                    }
                }
            }
        }
    }
}

// Dz update
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < (PL + 1); j++) {
        for(k = 0; k < ZCELLS; k++) {
            // Dzo = Dz at n - 1
            Dzo[i][j][k] = Dz[i][j][k];
            // PEC
            if(j == PL) {
                Dz[i][j][k] = 0;
            }
            // BC on xmin
            else if(i == 0) {
                if(!SBC) {
                    dHy = (Hy[i][j][k] - Hy[XCELLS-1][j][k])/dx;
                    if(j == 0) {
                        dHx = (Hx[i][j][k] - Hxb[i][k])/dy;
                    }
                    else {
                        dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;
                    }
                }
            }
        }
    }
}

// Ey update at n
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < PL; j++) {
        for(k = 0; k < ZCELLS + 1; k++) {
            Ey[i][j][k] = Ey[i][j][k] + C2Ey[i][j][k]*Dy[i][j][k] - C3Ey[i][j][k]*Dyo[i][j][k];
        }
    }
}

// Dz update
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < (PL + 1); j++) {
        for(k = 0; k < ZCELLS; k++) {
            // Dzo = Dz at n - 1
            Dzo[i][j][k] = Dz[i][j][k];
            // PEC
            if(j == PL) {
                Dz[i][j][k] = 0;
            }
            // BC on xmin
            else if(i == 0) {
                if(!SBC) {
                    dHy = (Hy[i][j][k] - Hy[XCELLS-1][j][k])/dx;
                    if(j == 0) {
                        dHx = (Hx[i][j][k] - Hxb[i][k])/dy;
                    }
                    else {
                        dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;
                    }
                }
            }
        }
    }
}
else {
    dHy = 2*(Hy[i][j][k])/dx;
    if(j == 0)
        dHx = (Hx[i][j][k] - Hxb[i][k])/dy;
    else
        dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;
    Dz[i][j][k] = Dz[i][j][k] + dt*(dHy - dHx);
}
// BC on xmax
else if(i == XCELLS) {
    if(!SBC) {
        Dz[i][j][k] = Dz[0][j][k];
    } else {
        dHy = -2*(Hy[XCELLS-1][j][k])/dx;
        if(j == 0)
            dHx = (Hx[i][j][k] - Hxb[i][k])/dy;
        else
            dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;
        Dz[i][j][k] = Dz[i][j][k] + dt*(dHy - dHx);
    }
} else {
    dHy = (Hy[i][j][k] - Hy[i-1][j][k])/dx;
    if(j == 0)
        dHx = (Hx[i][j][k] - Hxb[i][k])/dy;
    else
        dHx = (Hx[i][j][k] - Hx[i][j-1][k])/dy;
    Dz[i][j][k] = Dz[i][j][k] + dt*(dHy - dHx);
}

// Ez update
for(i = 0;i < XCELLS + 1;i++) {
    for(j = 0;j < PL+1;j++) {
        for(k = 0;k < ZCELLS;k++) {
            Ez[i][j][k] = C1Ez[i][j][k]*Ez[i][j][k] + C2Ez[i][j][k]*
                        (Dz[i][j][k] - Dzo[i][j][k]);
        }
    }
}

// Send Ex,Ez to rank 'NProcs - 1'
for(mpidx = 0;mpidx < XCELLS;mpidx++) {
    MPI_Send(&Ex[mpidx][0][0],ZCELLS+1,MPI_DOUBLE,NProcs - 1,2*10+(NProcs-1)*1+0,
             MPI_COMM_WORLD);
}
for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
    MPI_Send(&Ez[mpidx][0][0],ZCELLS,MPI_DOUBLE,NProcs - 1,2*10+(NProcs-1)*1+1,
             MPI_COMM_WORLD);
}
time = time + 0.5*dt;

// Bx update
for(i = 0;i < XCELLS + 1;i++) {
    for(j = 0;j < PL+1;j++) {
        for(k = 0;k < ZCELLS;k++) {
            // Bxo = Bx at n - 0.5
            Bxo[i][j][k] = Bx[i][j][k];
        }
    }
}
\[\text{dE}_z = \frac{\text{E}_z[i][j+1][k] - \text{E}_z[i][j][k]}{dy};\]
\[\text{dE}_y = \frac{\text{E}_y[i][j][k+1] - \text{E}_y[i][j][k]}{dz};\]
\[\text{B}_x[i][j][k] = C_1 \text{B}_x[i][j][k] \times \text{B}_x[i][j][k] - C_2 \text{B}_x[i][j][k] \times (\text{dE}_z - \text{dE}_y);\]

//Hx update
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < PL; j++) {
        for(k = 0; k < ZCELLS; k++) {
            if(i == XCELLS && !SBC) \{ \text{H}_x[i][j][k] = \text{H}_x[0][j][k]; \}
            else \{
                \text{H}_x[i][j][k] = \text{H}_x[i][j][k] + \frac{1}{\mu_o} (\text{B}_x[i][j][k] - \text{B}_xo[i][j][k]);
            \}
        }
    }
}

//By update (By at n + 0.5)
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < PL + 1; j++) {
        for(k = 0; k < ZCELLS; k++) {
            \text{B}_y[i][j][k] = \text{B}_y[i][j][k] - dt \times (\text{dE}_x - \text{dE}_z);
        }
    }
}

//Hy update (Hy at n + 0.5)
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < PL + 1; j++) {
        for(k = 0; k < ZCELLS; k++) {
            \text{H}_y[i][j][k] = \text{H}_y[i][j][k] + C_2 \text{H}_y[i][j][k] \times \text{B}_y[i][j][k] -
            C_3 \text{H}_y[i][j][k] \times \text{B}_yo[i][j][k];
        }
    }
}

//Bz update (Bz at n + 0.5)
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < PL; j++) {
        for(k = 0; k < ZCELLS + 1; k++) {
            \text{B}_z[i][j][k] = \text{B}_z[i][j][k] - dt \times (\text{dE}_y - \text{dE}_x);
        }
    }
}

//Hz update (Hz at n + 0.5)
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < PL; j++) {
        for(k = 0; k < ZCELLS + 1; k++) {
            \text{H}_z[i][j][k] = \text{H}_z[i][j][k] + C_1 \text{H}_z[i][j][k] \times \text{H}_z[i][j][k] + C_2 \text{H}_z[i][j][k] 
            \times (\text{B}_z[i][j][k] - \text{B}_zo[i][j][k]);
        }
    }
}
for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
    MPI_Recv(&Hxb[mpidx][0],ZCELLS,MPI_DOUBLE,NProcs - 1,
             (NProcs-1)*10+(rank*1)+0,MPI_COMM_WORLD,&status);
}
for(mpidx = 0;mpidx < XCELLS;mpidx++) {
    MPI_Recv(&Hzb[mpidx][0],ZCELLS + 1,MPI_DOUBLE,NProcs - 1,
             (NProcs-1)*10+(rank*1)+1,MPI_COMM_WORLD,&status);
}
time = time + 0.5*dt;
}

if(rank >=3 && rank <= NProcs - 2) {
    for(n = 0;n < T;n++) {
        //Ex main grid update equations
        for(i = 0;i < XCELLS;i++) {
            for(j = 0;j < yeePerProc;j++) {
                for(k = 0;k < ZCELLS + 1;k++) {
                    Exo = Ex[i][j][k];
                    RAExtot = 0;
                    dispcond = true;
                    for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
                        RAExtot = RAExtot + RAEx[i][j][k][pidx-1];
                    }
                    if(k == 0) {
                        if(!SBC) {
                            if(j == yeePerProc - 1)
                                dHz = (Hzb[i][k] - Hz[i][j][k])/dy;
                            else
                                dHz = (Hz[i][j+1][k] - Hz[i][j][k])/dy;
                            dHy = (Hy[i][j][k] - Hy[i][j][ZCELLS-1])/dz;
                            Ex[i][j][k] = (dt/(epso*K1Ex[i][j][k]))*(dHz - dHy)
                                        + (K2Ex[i][j][k]/K1Ex[i][j][k])*Ex[i][j][k] +
                                        RAExtot/K1Ex[i][j][k];
                        } else {
                            Ex[i][j][k] = 0;
                        }
                    }
                    //PBC on zmax
                    else if(k == ZCELLS) {
                        if(!SBC) {
                            Ex[i][j][k] = Ex[i][j][0];
                        } else {
                            Ex[i][j][k] = 0;
                        }
                        dispcond = false;
                    }
                    else {
                        if(j == yeePerProc - 1)
                            dHz = (Hzb[i][k] - Hz[i][j][k])/dy;
                        else
                            dHz = (Hz[i][j+1][k] - Hz[i][j][k])/dy;
                        dHy = (Hy[i][j][k] - Hy[i][j][k-1])/dz;
                        Ex[i][j][k] = (dt/(epso*K1Ex[i][j][k]))*(dHz - dHy)
                                        + (K2Ex[i][j][k]/K1Ex[i][j][k])*Ex[i][j][k] +
                                        RAExtot/K1Ex[i][j][k];
                    }
                }
            }
        }
    }
}

int rankoff = (rank - 3)*yeePerProc + 1;
// Update RAEx for the next update
if (k > ZCELLS/2 && !SBC) { kk = k - 1; }
else { kk = k; }
// dispcond will be false for k = ZCELLS
dispcond = dispcond && IsDispersive(MediaFullDomain[i][j+PL+rankoff][kk]);
if (dispcond) {
  for (pidx = 1;pidx <= MAXPOLESEPS;pidx++)
    (DXeEx0 = GetXeDebye(i,j+PL+rankoff,kk,0,dt,pidx) - GetXeDebye(i,j+PL+rankoff,kk,1,dt,pidx);
    DXieEx0 = GetXieDebye(i,j+PL+rankoff,kk,0,dt,pidx) - GetXieDebye(i,j+PL+rankoff,kk,1,dt,pidx);
    CEx = exp(-dt/ GetTaueps(MediaFullDomain[i][j+PL+rankoff][kk],pidx));
    RAEx[i][j][k][pidx-1] = DXeEx0*Ex[i][j][k] + DXieEx0*(Exo - Ex[i][j][k]) +
    CEx*RAEx[i][j][k][pidx-1];
}
}
}
}

// Ey main grid update equations
for (i = 0;i < XCELLS + 1;i++) {
  for (j = 0;j < yeePerProc;j++) {
    for (k = 0;k < ZCELLS + 1;k++) {
      Eyo = Ey[i][j][k];
      RAEytot = 0;
      dispcond = true;
      for (pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
        RAEytot = RAEytot + RAEy[i][j][k][pidx-1];
      }
      if (i == 0 && k == 0) {
        if (!SBC) {
          dHx = (Hx[i][j][k] - Hx[i][j][ZCELLS-1])/dz;
          dHz = (Hz[i][j][k] - Hz[XCELLS-1][j][k])/dx;
          Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*dHx - dHz +
          (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
          RAEytot/K1Ey[i][j][k];
        }
        else {
          Ey[i][j][k] = 0;
        }
      }
      else if ((i == 0 && k == ZCELLS) || (i == XCELLS && k == 0) ||
        (i == XCELLS && k == ZCELLS)) {
        if (!SBC) {
          Ey[i][j][k] = Ey[0][j][0];
        }
        else {
          Ey[i][j][k] = 0;
        }
        dispcond = false;
      }
      else if (i == 0) {
        if (!SBC) {
          dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
          dHz = (Hz[i][j][k] - Hz[XCELLS-1][j][k])/dx;
          Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*dHx - dHz +
          (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
          RAEytot/K1Ey[i][j][k];
        }
        else {
          dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
          dHz = 2*(Hx[i][j][k])/dx;
        }
      }
    }
  }
}

\[ Ey[i][j][k] = \left( \frac{dt}{\varepsilon_o K_1 Ey[i][j][k]} \right) (dHx - dHz) + \left( \frac{K_2 Ey[i][j][k]}{K_1 Ey[i][j][k]} \right) Ey[i][j][k] + \frac{RAEytot}{K_1 Ey[i][j][k]}; \]

else if(i == XCELLS) {
if(!SBC) {
Ey[i][j][k] = Ey[0][j][k];
}
else {
\[ dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz; \]
\[ dHz = -2*(Hz[XCELLS-1][j][k])/dx; \]
\[ Ey[i][j][k] = \left( \frac{dt}{\varepsilon_o K_1 Ey[i][j][k]} \right) (dHx - dHz) + \left( \frac{K_2 Ey[i][j][k]}{K_1 Ey[i][j][k]} \right) Ey[i][j][k] + \frac{RAEytot}{K_1 Ey[i][j][k]}; \]
}
}

else if(k == 0) {
if(!SBC) {
\[ dHx = (Hx[i][j][k] - Hx[i][j][ZCELLS-1])/dz; \]
\[ dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx; \]
\[ Ey[i][j][k] = \left( \frac{dt}{\varepsilon_o K_1 Ey[i][j][k]} \right) (dHx - dHz) + \left( \frac{K_2 Ey[i][j][k]}{K_1 Ey[i][j][k]} \right) Ey[i][j][k] + \frac{RAEytot}{K_1 Ey[i][j][k]}; \]
}
else {
Ey[i][j][k] = 0;
}
}

else if(k == ZCELLS) {
if(!SBC) {
Ey[i][j][k] = Ey[i][j][0];
}
else {
Ey[i][j][k] = 0;
}

\[ dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz; \]
\[ dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx; \]
\[ Ey[i][j][k] = \left( \frac{dt}{\varepsilon_o K_1 Ey[i][j][k]} \right) (dHx - dHz) + \left( \frac{K_2 Ey[i][j][k]}{K_1 Ey[i][j][k]} \right) Ey[i][j][k] + \frac{RAEytot}{K_1 Ey[i][j][k]}; \]
}

} // Update RAEy for the next update

if(i > XCELLS/2 && !SBC) { ii = i - 1; }
else { ii = i; }
if(k > ZCELLS/2 && !SBC) { kk = k - 1; }
else { kk = k; }
int rankoff = (rank - 3)*yeePerProc;
dispcond = dispcond && IsDispersive(MediaFullDomain[ii][j+PL+rankoff][kk]);
if(dispcond) {
    for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
        DXeEy0 = GetXeDebye(ii,j+PL+rankoff,kk,0,dt,pidx)-GetXeDebye(ii,j+PL+rankoff,kk,1,dt,pidx);
        DXieEy0 = GetXieDebye(ii,j+PL+rankoff,kk,0,dt,pidx)-GetXieDebye(ii,j+PL+rankoff,kk,1,dt,pidx);
        CEy = exp(-dt/GetTaueps(MediaFullDomain[ii][j+PL+rankoff][kk],pidx));
        RAEy[i][j][k][pidx-1] = DXeEy0*Ey[i][j][k] + DXieEy0*(Ey0 - Ey[i][j][k]) + CEy*RAEy[i][j][k][pidx-1];
    }
}
}
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < yeePerProc; j++) {
        for(k = 0; k < ZCELLS; k++) {
            Ez[i][j][k] = RAEztot = 0;
            dispcond = true;
            for(pidx = 1; pidx <= MAXPOLESEPS; pidx++) {
                RAEztot = RAEztot + RAEz[i][j][k][pidx-1];
            }
            if(i == 0) {
                if(!SBC) {
                    dHy = (Hy[i][j][k] - Hy[XCELLS-1][j][k])/dx;
                    dHx = (Hxb[i][k] - Hx[i][j][k])/dy;
                    Ez[i][j][k] = (dt/(epso*K1Ez[i][j][k]))*(dHy - dHx) +
                                    (K2Ez[i][j][k]/K1Ez[i][j][k])*Ez[i][j][k] +
                                    RAEztot/K1Ez[i][j][k];
                }
                else {
                    dHy = 2*(Hy[i][j][k])/dx;
                    dHx = (Hxb[i][k] - Hx[i][j][k])/dy;
                    Ez[i][j][k] = (dt/(epso*K1Ez[i][j][k]))*(dHy - dHx) +
                                    (K2Ez[i][j][k]/K1Ez[i][j][k])*Ez[i][j][k] +
                                    RAEztot/K1Ez[i][j][k];
                }
            }
            else if(i == XCELLS) {
                if(!SBC) {
                    Ez[i][j][k] = Ez[0][j][k];
                    dispcond = false;
                }
                else {
                    dHy = -2*(Hy[XCELLS-1][j][k])/dx;
                    dHx = (Hxb[i][k] - Hx[i][j][k])/dy;
                    Ez[i][j][k] = (dt/(epso*K1Ez[i][j][k]))*(dHy - dHx) +
                                    (K2Ez[i][j][k]/K1Ez[i][j][k])*Ez[i][j][k] +
                                    RAEztot/K1Ez[i][j][k];
                }
            }
            else {
                dHy = (Hy[i][j][k] - Hy[i-1][j][k])/dx;
                dHx = (Hxb[i][k] - Hx[i][j][k])/dy;
                Ez[i][j][k] = (dt/(epso*K1Ez[i][j][k]))*(dHy - dHx) +
                                    (K2Ez[i][j][k]/K1Ez[i][j][k])*Ez[i][j][k] +
                                    RAEztot/K1Ez[i][j][k];
            }
        }
    }
}

//Update RAEz for next update
if(i > XCELLS/2 && !SBC) { ii = i - 1; }
else { ii = i; }
int rankoff = (rank - 3)*yeePerProc + 1;
dispcond = dispcond && IsDispersive(MediaFullDomain[ii][j+PL+rankoff][k]);
if(dispcond) {
    for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
        DXeEz0 = GetXeDebye(ii,j+PL+rankoff,k,0,dt,pidx)
          - GetXeDebye(ii,j+PL+rankoff,k,1,dt,pidx);
        DXieEz0 = GetXieDebye(ii,j+PL+rankoff,k,0,dt,pidx)
          - GetXieDebye(ii,j+PL+rankoff,k,1,dt,pidx);
        CEz = exp(-dt / GetTaueps(MediaFullDomain[ii][j+PL+rankoff][k],pidx));
        RAEz[i][j][k][pidx-1] = DXeEz0*Ez[i][j][k] + DXieEz0*(Ezo - Ez[i][j][k]) +
          CEz*RAEz[i][j][k][pidx-1];
    }
}
if(rank == 3) {
    //Recv Ez,Ex
    for(mpidx = 0;mpidx < XCELLS;mpidx++) {
        MPI_Recv(&Exb[mpidx][0],ZCELLS + 1,MPI_DOUBLE,rank - 2,1*10+3*1+0,
          MPI_COMM_WORLD,&status);
    }
    for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
        MPI_Recv(&Ezb[mpidx][0],ZCELLS,MPI_DOUBLE,rank - 2,1*10+3*1+1,
          MPI_COMM_WORLD,&status);
    }
}
else {
    //Recv Ez,Ex
    for(mpidx = 0;mpidx < XCELLS;mpidx++) {
        MPI_Recv(&Exb[mpidx][0],ZCELLS + 1,MPI_DOUBLE,rank - 1,(rank - 1)*10 +
          (rank*1)+0,MPI_COMM_WORLD,&status);
    }
    for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
        MPI_Recv(&Ezb[mpidx][0],ZCELLS,MPI_DOUBLE,rank - 1,(rank - 1)*10 +
          (rank*1)+1,MPI_COMM_WORLD,&status);
    }
}
//Send Ex,Ez to rank+1 proc
for(mpidx = 0;mpidx < XCELLS;mpidx++) {
    MPI_Send(&Ex[mpidx][yeePerProc-1][0],ZCELLS + 1,MPI_DOUBLE,rank + 1,
      rank*10+(rank + 1)*1+0,MPI_COMM_WORLD);
}
for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
    MPI_Send(&Ez[mpidx][yeePerProc-1][0],ZCELLS,MPI_DOUBLE,rank + 1,
      rank*10+(rank + 1)+1,MPI_COMM_WORLD);
}
//Source excitation
if(rank == SPrank) {
    for(int is = 0 ; is < XCELLS + 1;is++) {
        for(int ks = 0;ks < ZCELLS;ks++) {
            if(source == EXPRAMPSINE) {
                if(n <= 960)
                    Ez[is][SPidx][ks] = Ez[is][SPidx][ks] + exp(-1*pow((time-960*dt)/(200*dt),2)) *
                      sin(2*pi*time*fs);
                else
                    Ez[is][SPidx][ks] = Ez[is][SPidx][ks] + sin(2*pi*time*fs);
            }
            else if(source == GAUSSIAN) {
                Ez[is][SPidx][ks] = Ez[is][SPidx][ks] + exp(-1*pow((time-to)/(tau),2));
            }
        }
    }
}
else if(source == MODGAUSSIAN) {
    Ez[is][SPidx][ks] = Ez[is][SPidx][ks] + exp(-1*pow((time-to)/(tau),2))*cos(2*pi*time*fc);
}
else if(source == DIFFGAUSSIAN) {
    Ez[is][SPidx][ks] = Ez[is][SPidx][ks] + sqrt(2*e)*((time - 2*to)/Tb)*exp(-1*pow((time - 2*to)/Tb,2));
}

if(INCSIM) {
    if(rank == OPrank) {
        outEz<<Ez[Round(XCELLS/2)][OPidx][Round(ZCELLS/2)]<<endl;
        outHx<<(Hx[Round(XCELLS/2)][OPidx][Round(ZCELLS/2)] + Hx[Round(XCELLS/2)][OPidx+1][Round(ZCELLS/2)])/2<<endl;
    }
}

//n1 - First index, n2 - Last index for writing data to .dat file
int n1 = 0;
int n2 = T;
//Ex - total field
if(n >= n1 && n <= n2 && rank == OPrank && !INCSIM){
    for(int xidx = 0;xidx < XCELLS + 1;xidx++){
        for(int zidx = 0;zidx < ZCELLS + 1;zidx++){
            double val;
            if(zidx == 0){
                val = Ez[xidx][OPidx][zidx];
            } else if(zidx == ZCELLS){
                val = Ez[xidx][OPidx][zidx -1];
            } else{
                val = (Ez[xidx][OPidx][zidx] + Ez[xidx][OPidx][zidx-1])/2;
            }
            outEz<<val<<endl;
        }
    }
    //Ey - total field
    for(int xidx = 0;xidx < XCELLS + 1;xidx++){
        for(int zidx = 0;zidx < ZCELLS + 1;zidx++){
            double val;
            val = (Ey[xidx][OPidx][zidx] + Ey[xidx][OPidx-1][zidx])/2;
            outEy<<val<<endl;
        }
    }
    //Hx - total field
    for(int xidx = 0;xidx < XCELLS + 1;xidx++){
        for(int zidx = 0;zidx < ZCELLS + 1;zidx++){
            double val;
            if(zidx == 0){
                val = (Hx[xidx][OPidx][zidx] + Hx[xidx][OPidx+1][zidx])/2;
            } else{
                val = (Hx[xidx][OPidx][zidx-1] + Hx[xidx][OPidx+1][zidx-1])/2;
            }
        }
    }
}
else{
    val = (Hz[xidx][OPidx][zidx] + Hz[xidx-1][OPidx][zidx]
        + Hz[xidx][OPidx+1][zidx] + Hz[xidx-1][OPidx+1][zidx])/4;
    outHz<<val<<endl;
}
}
}

//Hz - total field
if(n >= n1 && n <= n2 && rank == OPrank && !INCSIM){
    for(int xidx = 0;xidx < XCELLS + 1;xidx++){
        for(int zidx = 0;zidx < ZCELLS + 1;zidx++){
            double val;
            if(xidx == 0 || xidx == XCELLS || zidx == 0 || zidx == ZCELLS){
                val = 0;
            } else{
                val = (Hz[xidx][OPidx][zidx] + Hz[xidx-1][OPidx][zidx]
                    + Hz[xidx][OPidx+1][zidx] + Hz[xidx-1][OPidx+1][zidx])/4;
            }
            outHz<<val<<endl;
        }
    }
}

//Hy
/*if(n >= n1 && n <= n2 && rank == OPrank && !INCSIM){
    for(int xidx = 0;xidx < XCELLS + 1;xidx++){
        for(int zidx = 0;zidx < ZCELLS + 1;zidx++){
            double val;
            if(xidx == 0 || xidx == XCELLS) {
                val = 0;
            } else if(zidx == 0) {
                val = (Hy[xidx-1][OPidx][zidx] + Hy[xidx][OPidx][zidx])/2;
            } else if(zidx == ZCELLS) {
                val = (Hy[xidx-1][OPidx][zidx-1] + Hy[xidx][OPidx][zidx-1])/2;
            } else {
                val = (Hy[xidx-1][OPidx][zidx-1] + Hy[xidx][OPidx][zidx-1] +
                    Hy[xidx][OPidx][zidx] + Hy[xidx-1][OPidx][zidx])/4;
            }
            outHy<<val<<endl;
        }
    }
}*/

//Ex
if(n >= n1 && n <= n2 && rank == OPrank && !INCSIM){
    for(int xidx = 0;xidx < XCELLS + 1;xidx++){
        for(int zidx = 0;zidx < ZCELLS + 1;zidx++){
            double val;
            if(xidx == 0 || zidx == 0 || xidx == XCELLS || zidx == ZCELLS){
                val = 0;
            } else{
                val = (Ex[xidx][OPidx][zidx] + Ex[xidx-1][OPidx][zidx])/2;
            }
            outEx<<val<<endl;
        }
    }
}
time = time + 0.5*dt;

// Hx update
for(i = 0; i < XCELLS + 1; i++) {
    for(j = 0; j < yeePerProc; j++) {
        for(k = 0; k < ZCELLS; k++) {
            // Hx update in main grid
            Hxo = Hx[i][j][k];
            dEy = (Ey[i][j][k+1] - Ey[i][j][k])/dz;
            if(j == 0)
                dEz = (Ez[i][j][k] - Ezb[i][k])/dy;
            else
                dEz = (Ez[i][j][k] - Ez[i][j-1][k])/dy;
            RAHxtot = 0;
            for(pidx = 1; pidx <= MAXPOLESMU; pidx++) {
                RAHxtot = RAHxtot + RAHx[i][j][k][pidx-1];
            }
            if(i == XCELLS && !SBC) {
                Hx[i][j][k] = Hx[0][j][k];
            } else {
                Hx[i][j][k] = (dt/(muo*K1Hx[i][j][k]))*(dEy - dEz) +
                    (K2Hx[i][j][k]/K1Hx[i][j][k])*Hx[i][j][k] +
                    RAHxtot/K1Hx[i][j][k];
            }
            // Update RAHx for the next update of Hx
            if((i > XCELLS/2 && !SBC)) {
                ii = i - 1;
            } else {
                ii = i;
            }
            dispcond = IsDispersive(MediaFullDomain[ii][j+rankoff+PL][k]);
            if(dispcond) {
                int rankoff = (rank - 3) * yeePerProc;
                for(pidx = 1; pidx <= MAXPOLESMU; pidx++) {
                    DXmHx0 = GetXmDebye(ii,j+rankoff+PL,k,0,dt,pidx) -
                        GetXmDebye(ii,j+rankoff+PL,k,1,dt,pidx);
                    DXimHx0 = GetXimDebye(ii,j+rankoff+PL,k,0,dt,pidx) -
                        GetXimDebye(ii,j+rankoff+PL,k,1,dt,pidx);
                    CHx = exp(-dt/GetTaumu(MediaFullDomain[ii][j+rankoff+PL][k],pidx));
                    RAHx[i][j][k][pidx-1] = DXmHx0*Hx[i][j][k] + DXimHx0*(Hxo - Hx[i][j][k]) +
                        CHx*RAHx[i][j][k][pidx-1];
                }
            }
        }
    }
}

// Hy update (n + 0.5)
for(i = 0; i < XCELLS; i++) {
    for(j = 0; j < yeePerProc; j++) {
        for(k = 0; k < ZCELLS; k++) {
            // Hy update in main grid
            Hyo = Hy[i][j][k];
            dEz = (Ez[i+1][j][k] - Ez[i][j][k])/dx;
            dEx = (Ex[i][j+1][k] - Ex[i][j][k])/dz;
            RAHytot = 0;
            for(pidx = 1; pidx <= MAXPOLESMU; pidx++) {
RAHytot = RAHytot + RAHy[i][j][k][pidx-1];
}
Hy[i][j][k] = (dt/(muo*KH1Hy[i][j][k]))*(dEx - dEy)
+ (K2Hy[i][j][k]/KH1Hy[i][j][k])*Hy[i][j][k]
+ RAHytot/KH1Hy[i][j][k];

// Update RAHy for the next update of Hy

dispcond = false;
double DXmHy0,DXimHy0,CHy;
for(rankoff = (rank - 3)*yeePerProc + 1; rankoff < MAXPOLESMU; rankoff++) {
    dispcond = IsDispersive(MediaFullDomain[i][j+PL+rankoff][k]);
    if(dispcond) {
        for(pidx = 1;pidx <= MAXPOLESMU; pidx++) {
            DXmHy0 = GetXmDebye(i,j+PL+rankoff,k,0,dt,pidx) - GetXmDebye(i,j+PL+rankoff,k,1,dt,pidx);
            DXimHy0 = GetXimDebye(i,j+PL+rankoff,k,0,dt,pidx) - GetXimDebye(i,j+PL+rankoff,k,1,dt,pidx);
            CHy = exp(-dt/GetTaumu(MediaFullDomain[i][j+PL+rankoff][k],pidx));
            RAHy[i][j][k][pidx-1] = DXmHy0*Hy[i][j][k] + DXimHy0*(Hzo - Hy[i][j][k])
                            + CHy*RAHy[i][j][k][pidx-1];
        }
    }
}

//Hz update (at n + 0.5)
for(j = 0;j < yeePerProc;j++) {
    for(i = 0;i < XCELLS;i++) {
        for(k = 0;k < ZCELLS + 1;k++) {
            //Hz update in main grid
            Hz0 = Hz[i][j][k];
            if(j == 0)
                dEx = (Ex[i][j][k] - Exb[i][k])/dy;
            else
                dEx = (Ex[i][j][k] - Ex[i][j-1][k])/dy;
            dEy = (Ey[i+1][j][k] - Ey[i][j][k])/dx;
            RAHztot = 0;
            for(pidx = 1;pidx <= MAXPOLESMU; pidx++) {
                RAHztot = RAHztot + RAHz[i][j][k][pidx-1];
            }
            Hz[i][j][k] = (dt/(muo*KH1Hz[i][j][k]))*(dEx - dEy)
                            + (K2Hz[i][j][k]/KH1Hz[i][j][k])*Hz[i][j][k]
                            + RAHztot/KH1Hz[i][j][k];
            //Update RAHz for the next update of Hz
            double DXmHz0,DXimHz0,CHz;
            dispcond = false;
            if(k > ZCELLS/2 && !SBC) { kk = k - 1;}
            else { kk = k;}
            int rankoff = (rank - 3)*yeePerProc;
            dispcond = IsDispersive(MediaFullDomain[i][j+PL+rankoff][kk]);
            if(dispcond) {
                for(pidx = 1;pidx <= MAXPOLESMU; pidx++) {
                    DXmHz0 = GetXmDebye(i,j+PL+rankoff,kk,0,dt,pidx) - GetXmDebye(i,j+PL+rankoff,kk,1,dt,pidx);
                    DXimHz0 = GetXimDebye(i,j+PL+rankoff,kk,0,dt,pidx) - GetXimDebye(i,j+PL+rankoff,kk,1,dt,pidx);
                    CHz = exp(-dt/GetTaumu(MediaFullDomain[i][j+PL+rankoff][kk],pidx));
                    RAHz[i][j][k][pidx-1] = DXmHz0*Hz[i][j][k] + DXimHz0*(Hz0 - Hz[i][j][k])
                                    + CHz*RAHz[i][j][k][pidx-1];
                }
            }
        }
    }
}
if(rank == 3) {
    for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
        MPI_Send(&Hx[mpidx][0][0], ZCELLS, MPI_DOUBLE, 1, 
                  rank*10+(1*1)+0, MPI_COMM_WORLD);
    }
    for(mpidx = 0;mpidx < XCELLS;mpidx++) {
        MPI_Send(&Hz[mpidx][0][0], ZCELLS + 1, MPI_DOUBLE, 1, 
                  rank*10+(1*1)+1, MPI_COMM_WORLD);
    }
}
else {
    for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
        MPI_Send(&Hx[mpidx][0][0], ZCELLS, MPI_DOUBLE, rank - 1, 
                  rank*10+((rank-1)*1)+0, MPI_COMM_WORLD);
    }
    for(mpidx = 0;mpidx < XCELLS;mpidx++) {
        MPI_Send(&Hz[mpidx][0][0], ZCELLS + 1, MPI_DOUBLE, rank - 1, 
                  rank*10+((rank-1)*1)+1, MPI_COMM_WORLD);
    }
}
//Recv Hxb,Hzb
for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
    MPI_Recv(&Hxb[mpidx][0], ZCELLS, MPI_DOUBLE, rank + 1, 
              (rank+1)*10+(rank*1)+0, MPI_COMM_WORLD, &status);
}
for(mpidx = 0;mpidx < XCELLS;mpidx++) {
    MPI_Recv(&Hzb[mpidx][0], ZCELLS + 1, MPI_DOUBLE, rank + 1, 
              (rank+1)*10+(rank*1)+1, MPI_COMM_WORLD, &status);
}
time = time + 0.5*dt;
} //rank >=3 && rank <= NProcs - 2
if(rank == NProcs - 1) {
    for(n = 0;n < T;n++) {
        // Ex main grid update equations
        for(i = 0;i < XCELLS;i++) {
            for(j = 0;j < yeePerProc - 1;j++) {
                for(k = 0;k < ZCELLS + 1;k++) {
                    Exo = Ex[i][j][k];
                    RAExtot = 0;
                    dispcond = true;
                    for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
                        RAExtot += RAEx[i][j][k][pidx-1];
                    }
                    if(k == 0) {
                        if(ISBC) {
                            dHz = (Hz[i][j+1][k] - Hz[i][j][k])/dy;
                            dHy = (Hy[i][j][k] - Hy[i][j][ZCELLS+1])/dz;
                            Ex[i][j][k] = (dt/(epso*K1Ex[i][j][k]))*(dHz - dHy) + 
                                          (K2Ex[i][j][k]*K1Ex[i][j][k])*Ex[i][j][k] + 
                                          RAExtot/K1Ex[i][j][k];
                        } else {
                            Ex[i][j][k] = 0;
                        }
                    }
                    //PBC on zmax
                    else if(k == ZCELLS) {
                        if(ISBC) {
                            Ex[i][j][k] = Ex[i][j][0];
                        }
                    }
                }
            }
        }
    }
}
else {
    Ex[i][j][k] = 0;
}
    dispcond = false;
}
else {
    dHz = (Hz[i][j+1][k] - Hz[i][j][k])/dy;
    dHy = (Hy[i][j][k] - Hy[i][j][k-1])/dz;
    Ex[i][j][k] = (dt/(epso*K1Ex[i][j][k]))*(dHz - dHy) + (K2Ex[i][j][k]/K1Ex[i][j][k])*Ex[i][j][k] + RAExtot/K1Ex[i][j][k];
}

    int rankoff = (rank - 3)*yeePerProc + 1;
    //Update RAEx for the next update
    if(k > ZCELLS/2 && !SBC) { kk = k - 1; }
    else { kk = k; }
    //dispcond will be false for k = ZCELLS
    dispcond = dispcond && IsDispersive(MediaFullDomain[i][j+PL+rankoff][kk]);
    if(dispcond) {
        for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
            DXeEx0 = GetXeDebye(i,j+PL+rankoff,kk,0,dt,pidx) - GetXeDebye(i,j+PL+rankoff,kk,1,dt,pidx);
            DXieEx0 = GetXieDebye(i,j+PL+rankoff,kk,0,dt,pidx) - GetXieDebye(i,j+PL+rankoff,kk,1,dt,pidx);
            CEx = exp(-dt/GetTaueps(MediaFullDomain[i][j+PL+rankoff][kk],pidx));
            RAEx[i][j][k][pidx-1] = DXeEx0*Ex[i][j][k] + DXieEx0*(Exo - Ex[i][j][k]) + CEx*RAEx[i][j][k][pidx-1];
        }
    }
    else {

    }

    //Ey main grid update equations
    for(i = 0;i < XCELLS + 1;i++) {
        for(j = 0;j < yeePerProc;j++) {
            for(k = 0;k < ZCELLS + 1;k++) {
    Eyo = Ey[i][j][k];
    RAExtot = 0;
    dispcond = true;
    for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
        RAExtot = RAExtot + RAEx[i][j][k][pidx-1];
    }
    if(i == 0 && k == 0) {
        if(!SBC) {
            dHx = (Hx[i][j][k] - Hx[i][j][ZCELLS-1])/dz;
            dHz = (Hz[i][j][k] - Hz[XCELLS-1][j][k])/dx;
            Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*(dHx - dHz) + (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] + RAExtot/K1Ey[i][j][k];
        }
    }
    else {
        Ey[i][j][k] = 0;
    }
    }
    }
    }
    else if((i == 0 && k == ZCELLS) || (i == XCELLS && k == 0) || (i == XCELLS && k == ZCELLS)) {
        if(!SBC) {
            Ey[i][j][k] = Ey[0][j][0];
        }
    else {
        Ey[i][j][k] = 0;
    }
    dispcond = false;
else if(i == 0) {
    if(!SBC) {
        dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
        dHz = (Hz[i][j][k] - Hz[XCELLS-1][j][k])/dx;
        Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*(dHx - dHz) +
                      (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
                      RAEytot/K1Ey[i][j][k];
    } else {
        dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
        dHz = 2*(Hz[i][j][k])/dx;
        Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*(dHx - dHz) +
                      (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
                      RAEytot/K1Ey[i][j][k];
    }
}
else if(i == XCELLS) {
    if(!SBC) {
        dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
        dHz = -2*(Hz[XCELLS-1][j][k])/dx;
        Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*(dHx - dHz) +
                      (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
                      RAEytot/K1Ey[i][j][k];
    } else {
        dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
        dHz = 2*(Hz[i][j][k])/dx;
        Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*(dHx - dHz) +
                      (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
                      RAEytot/K1Ey[i][j][k];
    }
}
else if(k == 0) {
    if(!SBC) {
        dHx = (Hx[i][j][k] - Hx[i][j][ZCELLS-1])/dz;
        dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx;
        Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*(dHx - dHz) +
                      (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
                      RAEytot/K1Ey[i][j][k];
    } else {
        dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
        dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx;
        Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*(dHx - dHz) +
                      (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
                      RAEytot/K1Ey[i][j][k];
    }
}
else if(k == ZCELLS) {
    if(!SBC) {
        dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
        dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx;
        Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*(dHx - dHz) +
                      (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
                      RAEytot/K1Ey[i][j][k];
    } else {
        dHx = (Hx[i][j][k] - Hx[i][j][k-1])/dz;
        dHz = (Hz[i][j][k] - Hz[i-1][j][k])/dx;
        Ey[i][j][k] = (dt/(epso*K1Ey[i][j][k]))*(dHx - dHz) +
                      (K2Ey[i][j][k]/K1Ey[i][j][k])*Ey[i][j][k] +
                      RAEytot/K1Ey[i][j][k];
    }
}
else { //Update RAEy for the next update
    if(i > XCELLS/2 && !SBC) { ii = i - 1; }
    else { ii = i; }
    if(k > ZCELLS/2 && !SBC) { kk = k - 1; }
    else { kk = k; }
    int rankoff = (rank - 3)*yeePerProc;
    dispcond = dispcond && IsDispersive(MediaFullDomain[ii][j+PL+rankoff][kk]);
    if(dispcond) {
for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
  DXeEy0 = GetXeDebye(ii,j+PL+rankoff,kk,0,dt,pidx)
  -GetXeDebye(ii,j+PL+rankoff,kk,1,dt,pidx);
  DXieEy0 = GetXieDebye(ii,j+PL+rankoff,kk,0,dt,pidx)
  -GetXieDebye(ii,j+PL+rankoff,kk,1,dt,pidx);
  CEy = exp(-dt/GetTaueps(MediaFullDomain[ii][j+PL+rankoff][kk],pidx));
  RAEy[i][j][k][pidx-1] = DXeEy0*Ey[i][j][k] + DXieEy0*(Eyo - Ey[i][j][k]) +
    CEy*RAEy[i][j][k][pidx-1];
}
}
}
}
}
}

//Ez update
for(i = 0;i < XCELLS + 1;i++) {
  for(j = 0;j < yeePerProc - 1;j++) {
    for(k = 0;k < ZCELLS;k++) {
      Ez[i][j][k];
      RAEztot = 0;
      dispcond = true;
      for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
        RAEztot = RAEztot + RAEz[i][j][k][pidx-1];
      }
      if(i == 0) {
        if(ISBC) {
          dHy = (Hy[i][j][k] - Hy[XCELLS-1][j][k])/dx;
          dHx = (Hx[i][j+1][k] - Hx[i][j][k])/dy;
          Ez[i][j][k] = (dt/(epso*K1Ez[i][j][k]))*(dHy - dHx) +
            (K2Ez[i][j][k]/K1Ez[i][j][k])*Ez[i][j][k] +
            RAEztot/K1Ez[i][j][k];
        }
        else {
          dHy = 2*(Hy[i][j][k])/dx;
          dHx = (Hx[i][j+1][k] - Hx[i][j][k])/dy;
          Ez[i][j][k] = (dt/(epso*K1Ez[i][j][k]))*(dHy - dHx) +
            (K2Ez[i][j][k]/K1Ez[i][j][k])*Ez[i][j][k] +
            RAEztot/K1Ez[i][j][k];
        }
      } else if(i == XCELLS) {
        if(ISBC) {
          Ez[i][j][k] = Ez[0][j][k];
          dispcond = false;
        }
        else {
          dHy = 2*(Hy[XCELLS-1][j][k])/dx;
          dHx = (Hx[i][j+1][k] - Hx[i][j][k])/dy;
          Ez[i][j][k] = (dt/(epso*K1Ez[i][j][k]))*(dHy - dHx) +
            (K2Ez[i][j][k]/K1Ez[i][j][k])*Ez[i][j][k] +
            RAEztot/K1Ez[i][j][k];
        }
      } else {
        dHy = (Hy[i][j][k] - Hy[i-1][j][k])/dx;
        dHx = (Hx[i][j+1][k] - Hx[i][j][k])/dy;
        Ez[i][j][k] = (dt/(epso*K1Ez[i][j][k]))*(dHy - dHx) +
          (K2Ez[i][j][k]/K1Ez[i][j][k])*Ez[i][j][k] +
          RAEztot/K1Ez[i][j][k];
      }
    }
  }
}

//Update RAEz for next update
if(i > XCELLS/2 && !SBC) { ii = i - 1; }
else {
    ii = i;
}
int rankoff = (rank - 3)*yeePerProc + 1;
dispcond = dispcond && IsDispersive(MediaFullDomain[ii][j+PL+rankoff][k]);
if(dispcond) {
    for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
        DXeDebye = GetXeDebye(ii,j+PL+rankoff,k,0,dt,pidx);
        DXeDebye = GetXeDebye(ii,j+PL+rankoff,k,0,dt,pidx);
        CEx = exp(-dt / GetTaueps(MediaFullDomain[ii][j+PL+rankoff][k],pidx));
        RAEz[i][j][k][pidx-1] = DXeEz0*Ez[i][j][k] + DXieEz0*(Ezo - Ez[i][j][k]) +
            CEx*RAEz[i][j][k][pidx-1];
    }
}

//Recv Ez,Ex
for(mpidx = 0;mpidx < XCELLS;mpidx++) {
    MPI_Recv(&Exb[mpidx][0],ZCELLS + 1,MPI_DOUBLE,rank - 1,
            (rank-1)*10+(rank*1)+0,MPI_COMM_WORLD,&status);
}
for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
    MPI_Recv(&Ezb[mpidx][0],ZCELLS,MPI_DOUBLE,rank - 1,
            (rank-1)*10+(rank*1)+1,MPI_COMM_WORLD,&status);
}
//Recv Ezb1,Exb1
for(mpidx = 0;mpidx < XCELLS;mpidx++) {
    MPI_Recv(&Exb1[mpidx][0],ZCELLS + 1,MPI_DOUBLE,2,
            2*10+(NProcs-1)*1+0,MPI_COMM_WORLD,&status);
}
for(mpidx = 0;mpidx < XCELLS + 1;mpidx++) {
    MPI_Recv(&Ezb1[mpidx][0],ZCELLS,MPI_DOUBLE,2,
            2*10+(NProcs-1)*1+1,MPI_COMM_WORLD,&status);
}
time = time + 0.5*dt;

//Hx update
for(i = 0;i < XCELLS + 1;i++) {
    for(j = 0;j < yeePerProc;j++) {
        for(k = 0;k < ZCELLS;k++) {
        // Hx update in main grid
            Hxo = Hx[i][j][k];
            dEy = (Ey[i][j][k+1] - Ey[i][j][k])/dz;
            if(j == 0) dEz = (Ez[i][j][k] - Ezb[i][k])/dy;
            else if(j == yeePerProc - 1) dEz = (Ezb1[i][k] - Ez[i][j-1][k])/dy;
            else dEz = (Ez[i][j][k] - Ez[i][j-1][k])/dy;
            RAHxtot = 0;
            for(pidx = 1;pidx <= MAXPOLESEPS;pidx++) {
                RAHxtot = RAHxtot + RAHx[i][j][k][pidx-1];
            }
            if(i == XCELLS && !SBC) {
                Hx[i][j][k] = Hx[0][j][k];
            } else {
                Hx[i][j][k] = (dt/(muo*K1Hx[i][j][k]))*(dEy - dEz) + (K2Hx[i][j][k]/K1Hx[i][j][k])*Hx[i][j][k]
            }
        }
    }
}
double DXmHx = 0, DXimHx = 0, CHx;
bool dispcond = false;
if((i > XCELLS/2 && !SBC)) {
    ii = i - 1;
} else {
    ii = i;
}
if(dispcond) {
    int rankoff = (rank - 3)*yeePerProc;
    dispcond = IsDispersive(MediaFullDomain[ii][j+PL+rankoff][k]);
    if(dispcond) {
        for(pidx = 1;pidx <= MAXPOLESMU;pidx++) {
            DXmHx = GetXmDebye(ii,j+PL+rankoff,k,0,dt,pidx) - GetXmDebye(ii,j+PL+rankoff,k,1,dt,pidx);
            DXimHx = GetXimDebye(ii,j+PL+rankoff,k,0,dt,pidx) - GetXimDebye(ii,j+PL+rankoff,k,1,dt,pidx);
            CHx = exp(-dt/GetTaumu(MediaFullDomain[ii][j+PL+rankoff][k],pidx));
            RAHx[i][j][k][pidx-1] = DXmHx*Hx[i][j][k] + DXimHx*(Hxo - Hx[i][j][k]) + CHx*RAHx[i][j][k][pidx-1];
        }
    }
}

for(i = 0;i < XCELLS;i++) {
    for(j = 0;j < yeePerProc - 1;j++) {
        for(k = 0;k < ZCELLS;k++) {
            // Hy update in main grid
            Hyo = Hy[i][j][k];
            dEz = (Ez[i+1][j][k] - Ez[i][j][k])/dx;
            dEx = (Ex[i][j][k+1] - Ex[i][j][k])/dz;
            RAHytot = 0;
            for(pidx = 1;pidx <= MAXPOLESMU;pidx++) {
                RAHytot = RAHytot + RAHy[i][j][k][pidx-1];
            }
            Hy[i][j][k] = (dt/(muo*K1Hy[i][j][k]))*(dEz - dEx) + (K2Hy[i][j][k]/K1Hy[i][j][k])*Hy[i][j][k] + RAHytot/K1Hy[i][j][k];
            // Update RAHy for the next update of Hy
            dispcond = false;
            double DXmHy, DXimHy, CHy;
            int rankoff = (rank - 3)*yeePerProc + 1;
            dispcond = IsDispersive(MediaFullDomain[i][j+PL+rankoff][k]);
            if(dispcond) {
                for(pidx = 1;pidx <= MAXPOLESMU;pidx++) {
                    DXmHy = GetXmDebye(i,j+PL+rankoff,k,0,dt,pidx) - GetXmDebye(i,j+PL+rankoff,k,1,dt,pidx);
                    DXimHy = GetXimDebye(i,j+PL+rankoff,k,0,dt,pidx) - GetXimDebye(i,j+PL+rankoff,k,1,dt,pidx);
                    CHy = exp(-dt/GetTaumu(MediaFullDomain[i][j+PL+rankoff][k],pidx));
                    RAHy[i][j][k][pidx-1] = DXmHy*Hy[i][j][k] + DXimHy*(Hy0 - Hy[i][j][k]) + CHy*RAHy[i][j][k][pidx-1];
                }
            }
        }
    }
}
//Hz update
for(i = 0; i < XCELLS; i++) {
    for(j = 0; j < yeePerProc; j++) {
        for(k = 0; k < ZCELLS + 1; k++) {
            // Hz update in main grid
            Hzo = Hz[i][j][k];

            if(j == 0)
                dEx = (Ex[i][j][k] - Exb[i][k])/dy;
            else if(j == yeePerProc - 1)
                dEx = (Exb1[i][k] - Ex[i][j-1][k])/dy;
            else
                dEx = (Ex[i][j][k] - Ex[i][j-1][k])/dy;

            dEy = (Ey[i+1][j][k] - Ey[i][j][k])/dx;

            RAHztot = 0;
            for(pidx = 1; pidx <= MAXPOLESMU; pidx++) {
                RAHztot = RAHztot + RAHz[i][j][k][pidx-1];
            }

            Hz[i][j][k] = (dt/(muo*K1Hz[i][j][k]))*(dEx - dEY) +
                        (K2Hz[i][j][k]/K1Hz[i][j][k])*Hz[i][j][k]
                        + RAHztot/K1Hz[i][j][k];

            //Update RAHz for the next update of Hz
            double DXmHz0, DXmHz0, CHz;
            dispcond = false;
            if(k > ZCELLS/2 && !SBC) { kk = k - 1;}
            else { kk = k;}
            int rankoff = (rank - 3)*yeePerProc;
            dispcond = !isDispersive(MediaFullDomain[i][j+PL+rankoff][kk]);
            if(dispcond) {
                for(pidx = 1; pidx <= MAXPOLESMU; pidx++) {
                    DXmHz0 = GetXmDebye(i,j+PL+rankoff,kk,0,dt,pidx)
                            - GetXmDebye(i,j+PL+rankoff,kk,1,dt,pidx);
                    DXmHz0 = GetXmDebye(i,j+PL+rankoff,kk,0,dt,pidx)
                            - GetXmDebye(i,j+PL+rankoff,kk,1,dt,pidx);
                    CHz = exp(-dt/GetTaumu(MediaFullDomain[i][j+PL+rankoff][kk],pidx));
                    RAHz[i][j][k][pidx-1] = DXmHz0*Hz[i][j][k] +
                                            DXmHz0*(Hzo - Hz[i][j][k])
                                            + CHz*RAHz[i][j][k][pidx-1];
                }
            }
        }
    }
}

//Send Hx,Hz to rank 2
for(mpidx = 0; mpidx < XCELLS + 1; mpidx++) {
    MPI_Send(&Hx[mpidx][yeePerProc-1][0],ZCELLS,MPI_DOUBLE,2,
             rank*10+(2*1)+0,MPI_COMM_WORLD);
}

for(mpidx = 0; mpidx < XCELLS; mpidx++) {
    MPI_Send(&Hz[mpidx][yeePerProc-1][0],ZCELLS + 1,MPI_DOUBLE,2,
             rank*10+(2*1)+1,MPI_COMM_WORLD);
}

//Send Hx,Hz to rank 'NProcs - 2'
for(mpidx = 0; mpidx < XCELLS + 1; mpidx++) {
    MPI_Send(&Hx[mpidx][0][0],ZCELLS,MPI_DOUBLE,NProcs - 2,
             rank*10+(NProcs - 2)+0,MPI_COMM_WORLD);
}
for(mpidx = 0; mpidx < XCELLS; mpidx++) {
    MPI_Send(&Hz[mpidx][0][0],ZCELLS + 1,MPI_DOUBLE,NProcs - 2,
             rank*10+(NProcs - 2)+1,MPI_COMM_WORLD);
}
time = time + 0.5*dt;
}

} // n
} // rank = NProcs - 1

// Delete the boundary field matrices
if(rank == 1) {
 DeleteMatrix(Hzb,XCELLS,ZCELLS + 1);
 DeleteMatrix(Hxb,XCELLS + 1,ZCELLS);
}
if(rank == 2) {
 DeleteMatrix(Hzb,XCELLS,ZCELLS + 1);
 DeleteMatrix(Hxb,XCELLS + 1,ZCELLS);
 DeleteMatrix(Ezb,XCELLS + 1,ZCELLS);
 DeleteMatrix(Exb,XCELLS,ZCELLS + 1);
}
if(rank == NProcs - 1) {
 DeleteMatrix(Ezb,XCELLS + 1,ZCELLS);
 DeleteMatrix(Ezb1,XCELLS + 1,ZCELLS);
 DeleteMatrix(Exb,XCELLS,ZCELLS + 1);
 DeleteMatrix(Exb1,XCELLS,ZCELLS + 1);
}
DeleteAllMatrices(rank);
if(rank == OPrank) {
 if(!INCSIM) {
 outEz.close();
 outEy.close();
 outEx.close();
 outHx.close();
 outHz.close();
 }
 else {
 outEz.close();
 outHx.close();
 }
}
pproc:
MPI_Finalize();
//------------------------------------------------------------------------
// Postprocessing
if(!INCSIM && !rank) {
 cout<<"Postprocessing..."<<endl;
 ifstream Ezf,Eyf,Exf,Hxf,Hzf,Hyf,Ezincf,Hxincf;
 int idx;
 char str[200];
 CmplxPtr3 Ezsfr,Eysfr,Exsfr,Hzsfr,Hysfr,Exsfr,Hysfr,S;
 double f1 = 6e9;
 double f2 = 200e9;
 cout<<"Postprocessing : f1 = "<<f1<<endl;
 cout<<"Postprocessing : f2 = "<<f2<<endl;
 // Get the incident field data
 Ezincf.open("/panl1/projects/awmra/Ezinc.dat",ios::in);
 Hxincf.open("/panl1/projects/awmra/Hxinc.dat",ios::in);
 double* Ezinc;
 double* Hxinc;
 Ezinc = new double[T];
 Hxinc = new double[T];
 // Read Ezinc,Hxinc data files
 idx = 0;

while(!Ezincf.eof()) {
    Ezincf.getline(str,200);
    Ezinc[idx] = strtod(str,'\0');
    idx++;
}
idx = 0;
while(!Hxincf.eof()) {
    Hxincf.getline(str,200);
    Hxinc[idx] = strtod(str,'\0');
    idx++;
}cout<<"Read Ezinc,Hxinc"<<endl;

//Frequency points
uint Nres = pow(2,17) - T;
double df = (1/dt)/(T+Nres);
uint N1 = (uint)Round(f1/df);
uint N2 = (uint)Round(f2/df);

cout<<"Creating freq domain field matrices"<<endl;
//Frequency domain complex fields
CreateMatrix(Ezsfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
CreateMatrix(Exsfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
//CreateMatrix(Eysfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
CreateMatrix(Hxsfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
//CreateMatrix(Hysfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);

//Ez
complex* Eztpxl = new complex[T+Nres];
complex* Ezfrpxl = new complex[T+Nres];
double* Ezdata = new double[(XCELLS + 1)*(ZCELLS + 1)*T];
Ezf.open(\"/pan1/projects/awmra/Ez.dat\",ios::in);
idx = 0;
while(!Ezf.eof()) {
    Ezf.getline(str,200);
    Ezdata[idx++] = strtod(str,'\0');
}
for(int i = 0;i < XCELLS + 1;i++) {
    for(int k = 0;k < ZCELLS + 1;k++) {
        int offset = i*(ZCELLS + 1) + k;
        for(int q = 0;q < T;q++) {
            Eztpxl[q] = complex(Ezdata[offset + q*(XCELLS + 1)*(ZCELLS + 1)] - Ezinc[q],0);
        }
        CFFT::Forward(&Eztpxl[0],&Ezfrpxl[0],T+Nres);
        for(int q = 0;q < N2 - N1 + 1;q++) {
            Ezsfr[k][i][q] = Ezfrpxl[q + N1];
        }
    }
}
delete [] Eztpxl; delete [] Ezfrpxl; delete [] Ezdata;
Ezf.close();
cout<<"Filled Ezsfr matrix"<<endl;

//Ey
/*complex* Eytpxl = new complex[T+Nres];
complex* Eyfrpxl = new complex[T+Nres];
double* Eydata = new double[(XCELLS + 1)*(ZCELLS + 1)*T];
Eyf.open("/pan1/projects/awmra/Ey.dat\",ios::in);
idx = 0;
while(!Eyf.eof()) {
    Eyf.getline(str,200);
    Eydata[idx++] = strtod(str,'\0');
}
for(int i = 0;i < XCELLS + 1;i++) {
    for(int k = 0;k < ZCELLS + 1;k++) {
        int offset = i*(ZCELLS + 1) + k;
        for(int q = 0;q < T;q++) {
            Eytpxl[q] = complex(Eydata[offset + q*(XCELLS + 1)*(ZCELLS + 1)] - Hxinc[q],0);
        }
        CFFT::Forward(&Eytpxl[0],&Eyfrpxl[0],T+Nres);
        for(int q = 0;q < N2 - N1 + 1;q++) {
            Eysfr[k][i][q] = Eyfrpxl[q + N1];
        }
    }
}
delete [] Eytpxl; delete [] Eyfrpxl; delete [] Eydata;
Eyf.close();
cout<<"Filled Eysfr matrix"<<endl;
for(int q = 0; q < T; q++) {
    Eytpxl[q] = complex(Eydata[offset + q*(XCELLS + 1)*(ZCELLS + 1)],0);
}
CFFT::Forward(&Eytpxl[0], &Eyfrpxl[0], T+Nres);
for(int q = 0; q < N2 - N1 + 1; q++) {
    Eysfr[k][i][q] = Eyfrpxl[q + N1];
}
}
}
delete [] Eytpxl; delete [] Eyfrpxl; delete [] Eydata;
Eyf.close();

//Ex
complex* Extpxl = new complex[T+Nres];
complex* Exfrpxl = new complex[T+Nres];
double* Exdata = new double[(XCELLS + 1)*(ZCELLS + 1)*T];
Exf.open("/pan1/projects/awmra/Ex.dat", ios::in);
idx = 0;
while(!Exf.eof()) {
    Exf.getline(str, 200);
    Exdata[idx++] = strtod(str, NULL);
}
for(int i = 0; i < XCELLS + 1; i++) {
    for(int k = 0; k < ZCELLS + 1; k++) {
        int offset = i*(ZCELLS + 1) + k;
        for(int q = 0; q < T; q++) {
            Extpxl[q] = complex(Exdata[offset + q*(XCELLS + 1)*(ZCELLS + 1)],0);
        }
    }
    CFFT::Forward(&Extpxl[0], &Exfrpxl[0], T+Nres);
    for(int q = 0; q < N2 - N1 + 1; q++) {
        Exsfr[k][i][q] = Exfrpxl[q + N1];
    }
}
}
delete [] Extpxl; delete [] Exfrpxl; delete [] Exdata;
Exf.close();

//Hz
complex* Hztpxl = new complex[T+Nres];
complex* Hzfrpxl = new complex[T+Nres];
double* Hzdata = new double[(XCELLS + 1)*(ZCELLS + 1)*T];
Hzf.open("/pan1/projects/awmra/Hz.dat", ios::in);
idx = 0;
while(!Hzf.eof()) {
    Hzf.getline(str, 200);
    Hzdata[idx++] = strtod(str, NULL);
}
for(int i = 0; i < XCELLS + 1; i++) {
    for(int k = 0; k < ZCELLS + 1; k++) {
        int offset = i*(ZCELLS + 1) + k;
        for(int q = 0; q < T; q++) {
            Hztpxl[q] = complex(Hzdata[offset + q*(XCELLS + 1)*(ZCELLS + 1)],0);
        }
    }
    TimeAlignHfield(&Hztpxl[0], T + Nres);
    CFFT::Forward(&Hztpxl[0], &Hzfrpxl[0], T+Nres);
    for(int q = 0; q < N2 - N1 + 1; q++) {
        Hzsfr[k][i][q] = Hzfrpxl[q + N1];
    }
}
}
delete [] Hztpxl; delete [] Hzfrpxl; delete [] Hzdata;
Hzf.close();
cout<<"Filled Hzsfr matrix"<<endl;

//Hy
/* complex* Hytpxl = new complex[T+Nres];
complex* Hyfrpxl = new complex[T+Nres];
double* Hydata = new double[(XCELLS + 1)*(ZCELLS + 1)*T];
Hyf.open("/pan1/projects/awmra/Hy.dat",ios::in);
idx = 0;
while(!Hyf.eof()) {
    Hyf.getline(str,200);
    Hydata[idx++] = strtod(str,\"\0");
}
for(int i = 0;i < XCELLS + 1;i++) {
    for(int k = 0;k < ZCELLS + 1;k++) {
        int offset = i*(ZCELLS + 1) + k;
        for(int q = 0;q < T;q++) {
            Hytpxl[q] = complex(Hydata[offset + q*(XCELLS + 1)*(ZCELLS + 1)],0);
        }
        TimeAlignHfield(&Hytpxl[0],T + Nres);
        CFFT::Forward(&Hytpxl[0],&Hyfrpxl[0],T+Nres);
        for(int q = 0;q < N2 - N1 + 1;q++) {
            Hysfr[k][i][q] = Hyfrpxl[q + N1];
        }
    }
}
delete [] Hytpxl; delete [] Hyfrpxl; delete [] Hydata;
Hyf.close();*/

//Hx
complex* Hxtpxl = new complex[T+Nres];
complex* Hxfrpxl = new complex[T+Nres];
double* Hxdata = new double[(XCELLS + 1)*(ZCELLS + 1)*T];
Hxf.open("/pan1/projects/awmra/Hx.dat",ios::in);
idx = 0;
while(!Hxf.eof()) {
    Hxf.getline(str,200);
    Hxdata[idx++] = strtod(str,\"\0");
}
for(int i = 0;i < XCELLS + 1;i++) {
    for(int k = 0;k < ZCELLS + 1;k++) {
        int offset = i*(ZCELLS + 1) + k;
        for(int q = 0;q < T;q++) {
            Hxtpxl[q] = complex(Hxdata[offset + q*(XCELLS + 1)*(ZCELLS + 1)] - Hxinc[q],0);
        }
        TimeAlignHfield(&Hxtpxl[0],T + Nres);
        CFFT::Forward(&Hxtpxl[0],&Hxfrpxl[0],T+Nres);
        for(int q = 0;q < N2 - N1 + 1;q++) {
            Hxsfr[k][i][q] = Hxfrpxl[q + N1];
        }
    }
}
delete [] Hxtpxl; delete [] Hxfrpxl; delete [] Hxdata;
Hxf.close();
cout<<"Filled Hxsfr matrix"<<endl;

//Write time average power spectral density to output file
ofstream Sout;
Sout.open("Sout.dat",ios::out);
Sout<<XCELLS + 1<<endl;
Sout<<ZCELLS + 1<<endl;
Sout<<N2 - N1 + 1<<endl;
Sout<<dx<<endl;
Sout<<dt<<endl;
Sout<<df<<endl;
CreateMatrix(S,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
for(int i = 0;i < XCELLS + 1;i++) {
    for(int k = 0;k < ZCELLS + 1;k++) {
        for(int q = 0;q < N2 - N1 + 1;q++) {
            S[k][i][q] = complex(((Exsfr[k][i][q]*(Hzsfr[k][i][q].conjugate())).re()) -
((Ezsfr[k][i][q]*Hxsfr[k][i][q].conjugate()).re()),0);
Sout<<2*(S[k][i][q].re())<<endl;
}
}
}
Sout.close();
cout<<"Created Sout.dat"<<endl;

//Write the Ezinc magnitude (freq domain) to file
ofstream Ezincmagf;
Ezincmagf.open("Ezincmagf.dat",ios::out);
//Incident field
complex* Ezinct = new complex[T+Nres];
complex* Ezincfr = new complex[T+Nres];
for(int q = 0;q < T;q++) {
    Ezinct[q] = complex(Ezinc[q],0);
}
CFFT::Forward(&Ezinct[0],&Ezincfr[0],T+Nres);
for(int q = 0;q < N2 - N1 + 1;q++) {
    Ezincmagf<<sqrt(Ezincfr[q+N1].norm())<<endl;
}
delete [] Ezinct; delete [] Ezincfr;
Ezincmagf.close();
cout<<"Created Ezincmagf.dat"<<endl;

//Write the list of frequencies
ofstream Flist;
Flist.open("Flist.dat",ios::out);
for(int q = 0;q < N2 - N1 + 1;q++) {
    Flist<<(q + N1)/dt/(T+Nres)<<endl;
}
Flist.close();
cout<<"Created Flist.dat"<<endl;

/*
cout<<"Floquet mode analysis"<<endl;
ofstream ExFloquet,ExFloquet,EyFloquet;
ofstream HxFloquet,HxFloquet,HyFloquet;
ExFloquet.open("Exfloquet.dat",ios::out);
EyFloquet.open("Eyfloquet.dat",ios::out);
HxFloquet.open("Hxfloquet.dat",ios::out);
HyFloquet.open("Hyfloquet.dat",ios::out);
for(int j = 0;j < NumofOutFreq;j++) {
    int qf;
    double radiometFreq = OutFreqs[j];
    for(int q = 0;q < N2 - N1 + 1;q++) {
        if(((q + N1)/dt/(T+Nres)) > radiometFreq) {
            double diff1 = fabs(((double)(q + N1 - 1))/dt/((double)(T+Nres)) - radiometFreq);
            double diff2 = fabs(((double)(q + N1))/dt/((double)(T+Nres)) - radiometFreq);
            if(diff1 < diff2) {
                qf = q - 1;
                break;
            } else {
                qf = q;
                break;
            }
        }
    }
    for(int i = 0;i < XCELLS + 1;i++) {
        for(int k = 0;k < ZCELLS + 1;k++) {
            ExFloquet<<Exsfr[k][i][qf].re()<<endl;
            EyFloquet<<Eysfr[k][i][qf].re()<<endl;
            EzFloquet<<Ezsfr[k][i][qf].re()<<endl;
        }
    }
}*/
ExFloquet<<Exsfr[k][i][qf].im()<<endl;
EyFloquet<<Eysfr[k][i][qf].im()<<endl;
ExFloquet<<Exsfr[k][i][qf].im()<<endl;

HxFloquet<<Hxsfr[k][i][qf].re()<<endl;
HyFloquet<<Hysfr[k][i][qf].re()<<endl;
HzFloquet<<Hzsfr[k][i][qf].re()<<endl;
HxFloquet<<Hxsfr[k][i][qf].im()<<endl;
HyFloquet<<Hysfr[k][i][qf].im()<<endl;
HzFloquet<<Hzsfr[k][i][qf].im()<<endl;

ExFloquet.close();EyFloquet.close();
ExFloquet.close();
HzFloquet.close();HyFloquet.close();
HzFloquet.close();
*/
cout<<"Finished"<<endl;
delete [] Ezinc;
delete [] Hxinc;
Ezincf.close();
Hxincf.close();
DeleteMatrix(Ezsfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
DeleteMatrix(Exsfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
DeleteMatrix(Hzsfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
DeleteMatrix(Hxsfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
//DeleteMatrix(Hysfr,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
DeleteMatrix(S,ZCELLS + 1,XCELLS + 1,N2 - N1 + 1);
}

return 0;
}

void TimeAlignHfield(double* field,uint sz) {
for(int idx = 0;idx < sz;idx++) {
if(idx != 0 && idx != sz - 1) {
field[idx] = 0.5*(field[idx] + field[idx + 1]);
}
}
}
void TimeAlignHfield(complex* field,uint sz) {
for(int idx = 0;idx < sz;idx++) {
if(idx != 0 && idx != sz - 1) {
field[idx] = 0.5*(field[idx] + field[idx + 1]);
}
}

//Range of xidx:[0,PSX],zidx:[0,PSZ],yidx:[0,PSY]
double GetFieldVal(uint xidx,uint yidx,uint zidx,uint fieldComp) {
do
}
else if(fieldComp == Eyc) {
    if(yidx == 0 || yidx == PSY)
        val = Ey[xidx][PL+yidx][zidx];
    else
        val = (Ey[xidx][PL+yidx][zidx] + Ey[xidx][PL+yidx-1][zidx])/2;
} else if(fieldComp == Ezc) {
    if(zidx == 0)
        val = Ez[xidx][PL+yidx][zidx];
    else if(zidx == PSZ)
        val = Ez[xidx][PL+yidx][zidx -1];
    else
        val = (Ez[xidx][PL+yidx][zidx] + Ez[xidx][PL+yidx][zidx-1])/2;
} else if(fieldComp == Hxc) {
    if(zidx == 0)
        val = (Hx[xidx][PL+yidx][zidx] + Hx[xidx][PL+yidx-1][zidx])/2;
    else if(zidx == ZCELLS)
        val = (Hx[xidx][PL+yidx][zidx-1] + Hx[xidx][PL+yidx-1][zidx-1])/2;
    else
        val = (Hx[xidx][PL+yidx][zidx] + Hx[xidx][PL+yidx-1][zidx] + Hx[xidx][PL+yidx][zidx-1] + Hx[xidx][PL+yidx-1][zidx-1])/4;
} else if(fieldComp == Hyc) {
}
return val;
}

void InitMaterialMatrix()
{
    for(int i = 0;i < XCELLS;i++) {
        for(int j = 0;j < YCELLS;j++) {
            for(int k = 0;k < ZCELLS;k++) {
                if(j < PL || j > PL + PSY - 1) {
                    //MediaFullDomain[i][j][k] = PML;
                } else if(j >= 50 && j <= 75) {
                    //MediaFullDomain[i][j][k] = MF112;
                } else if(j >= 93 && j <= 95 && i >= 10 && i <= 20 && k >= 10 && k <= 20) {
                    //MediaFullDomain[i][j][k] = MF112;
                } else {
                    MediaFullDomain[i][j][k] = AIR;
                }
            }
        }
    }
}

uint AddMetalPyr(double dx,uint material)
{
    //PL + PSY - 1 : AIR
    //Base : PL + PSY - 2,...,PL + PSY - 2 = dbase + 1
    for(int i = 0;i < XCELLS;i++) {
        for(int j = PL + PSY - aircells - dbase;j <= PL + PSY - 1 - aircells;j++) {
            for(int k = 0;k < ZCELLS;k++) {
                MediaFullDomain[i][j][k] = material;
            }
        }
    }
}

double z,d,u;
int j, hw, yidx;
d = (basewidth/mp)*1e-2;
\[ j = PL + PSY - aircells - dbase - 1; \]
\[ z = (PL + PSY - aircells - dbase - 1 - j) \times dx; \]
\[ w = \text{basewidth} \times 1e-2 \times (1 - z/d); \]
\[ hw = \text{Round}(w/2/dx); \]
\[ yidx = PL + PSY - aircells - dbase - 1; \]

\[
\text{do } \{
\text{for(int xidx = PSX/2 - hw; xidx <= PSX / 2 + hw - 1; xidx++) } \{
\text{for(int zidx = PSX/2 - hw; zidx <= PSX / 2 + hw - 1; zidx++) } \{
\text{MediaFullDomain}[xidx][yidx][zidx] = material; \}
\}
\}
\]

\[
yidx = yidx - 1; \]
\[
j--; \]
\[
z = (PL + PSY - aircells - dbase - 1 - j) \times dx; \]
\[
w = \text{basewidth} \times 1e-2 \times (1 - z/d); \]
\[
hw = \text{Round}(w/2/dx); \]
\[
\text{while (hw} \geq 1); \]
\]
\[
\text{return yidx + 1}; \]
\]

//Add absorbent coating
uint AddCoating(double dx, uint material)
{
uint pyrtopyidxtemp = PL + PSY;

for(int i = 0; i < XCELLS; i++) {
    for(int k = 0; k < ZCELLS; k++) {
        for(int j = PL; j < PL + PSY - 1; j++) {
            if(MediaFullDomain[i][j][k] == AL || MediaFullDomain[i][j][k] == PEC) {
                //Add the coating
                for(uint c = j - 1; c >= j - coating; c--) {
                    MediaFullDomain[i][c][k] = material;
                    if(c < pyrtopyidxtemp) {
                        pyrtopyidxtemp = c;
                    }
                    break;
                }
            }
        }
    }
}

return pyrtopyidxtemp;
}

void FillUpdateEqnConstantMatrices(double dx, double sigmao, double d, double m, double dt, int rank)
{
    double Dx_sigmay, Bx_sigmay;
    double Ey_sigmay, Ez_sigmay;
    double Hy_sigmay, Hz_sigmay;
    int i, ii, kk;

    //UPML coefficients
    if(rank == 1 || rank == 2) {
        //C1Dx, C2Dx
        for(int i = 0; i < XCELLS; i++) {
            for(int j = 0; j < PL + 1; j++) {
                for(int k = 0; k < ZCELLS + 1; k++) {
                    //Dx_sigmay
                    if(rank == 1)
Dx_sigmay = sigmao * pow((PL-j)*dx/d,m);
else
Dx_sigmay = sigmao * pow(j*dx/d,m);

//C1Dx,C2Dx
C1Dx[i][j][k] = (2 * epso - Dx_sigmay * dt)
/(2 * epso + Dx_sigmay * dt);
C2Dx[i][j][k] = (2 * epso * dt)
/(2 * epso + Dx_sigmay * dt);
}
}
}

//C1Ey,C2Ey
for(i = 0;i < XCELLS + 1; i++) {
for(int j = 0;j < PL; j++) {
for(int k = 0;k < ZCELLS + 1; k++) {
//Ey_sigmay
if(rank == 1)
Ey_sigmay = sigmao * pow((PL-j-0.5)*dx/d,m);
else
Ey_sigmay = sigmao * pow((j+0.5)*dx/d,m);

//C2Ey,C3Ey
C2Ey[i][j][k] = (2 * epso + Ey_sigmay * dt)
/(2 * epso);
C3Ey[i][j][k] = (2 * epso - Ey_sigmay * dt)
/(2 * epso);
}
}
}

//C1Ez,C2Ez
for(i = 0;i < XCELLS + 1; i++) {
for(int j = 0;j < PL + 1; j++) {
for(int k = 0;k < ZCELLS; k++) {
//Ez_sigmay
if(rank == 1)
Ez_sigmay = sigmao * pow((PL-j)*dx/d,m);
else
Ez_sigmay = sigmao * pow((j+0.5)*dx/d,m);

//C1Ez,C2Ez
C1Ez[i][j][k] = (2 * epso - Ez_sigmay * dt)
/(2 * epso + Ez_sigmay * dt);
C2Ez[i][j][k] = (2 * epso)
/(2 * epso + Ez_sigmay * dt);
}
}
}

//C1Bx,C2Bx
for(i = 0;i < XCELLS + 1; i++) {
for(int j = 0;j < PL;j++) {
for(int k = 0;k < ZCELLS;k++) {
//Bx_sigmay
if(rank == 1)
Bx_sigmay = sigmao * pow((PL-j-0.5)*dx/d,m);
else
Bx_sigmay = sigmao * pow((j+0.5)*dx/d,m);

//C1Bx,C2Bx
C1Bx[i][j][k] = (2 * epso - Bx_sigmay * dt)
C2Bx[i][j][k] = \( \frac{2 \cdot \varepsilon_{\text{o}} \cdot dt}{2 \cdot \varepsilon_{\text{o}} + B_{\text{x sigmay}} \cdot dt} \)

\{ 
  \{ 
    for(i = 0; i < XCELLS; i++) {
      for(int j = 0; j < PL + 1; j++) {
        for(int k = 0; k < ZCELLS; k++) {
          if(rank == 1)
            Hy_sigmay = \sigma_{\text{o}} \cdot \text{pow}((PL-j) \cdot dx/d, m);
          else
            Hy_sigmay = \sigma_{\text{o}} \cdot \text{pow}(j \cdot dx/d, m);
          C2Hy[i][j][k] = \frac{2 \cdot \varepsilon_{\text{o}} + Hy_{\text{sigmay}} \cdot dt}{(2 \cdot \varepsilon_{\text{o}}) \cdot \mu_{\text{o}}};
          C3Hy[i][j][k] = \frac{2 \cdot \varepsilon_{\text{o}} - Hy_{\text{sigmay}} \cdot dt}{(2 \cdot \varepsilon_{\text{o}}) \cdot \mu_{\text{o}}};
        }
      }
    }
  }
\} 

//C2Hz,C1Hz
for(i = 0; i < XCELLS; i++) {
  for(int j = 0; j < PL; j++) {
    for(int k = 0; k < ZCELLS + 1; k++) {
      Hz_sigmay = \sigma_{\text{o}} \cdot \text{pow}((k - j - 0.5) \cdot dx/d, m);
      C2Hz[i][j][k] = \frac{2 \cdot \varepsilon_{\text{o}}}{(2 \cdot \varepsilon_{\text{o}} + Hz_{\text{sigmay}} \cdot dt) \cdot \mu_{\text{o}}};
      C1Hz[i][j][k] = \frac{2 \cdot \varepsilon_{\text{o}} - Hz_{\text{sigmay}} \cdot dt}{2 \cdot \varepsilon_{\text{o}} + Hz_{\text{sigmay}} \cdot dt};
    }
  }
}

if(rank == 0 || rank == 1 || rank == 2)
  return;

//rank 3 .. rank NProcs-1 : Processors for problem space domain
//rank 3 adjacent to PML0
int jmax, rankoff;

//K1Ex,K2Ex
if(rank == NProcs - 1)
  jmax = yeePerProc - 1;
else
  jmax = yeePerProc;
  rankoff = (rank - 3) \cdot yeePerProc;
double \varepsilon_{\text{infEx}}, \sigma_{\text{Ex}}, X_{\text{Ex0}}, X_{\text{ieEx0}};
for(i = 0; i < XCELLS; i++) {
for(int j = 0; j < jmax; j++) {
    for(int k = 0; k < ZCELLS + 1; k++) {
        if(k <= ZCELLS/2) {
            kk = k;
        } else {
            kk = k - 1;
        }

        sigmaEx = GetSigma(i, j+PL+rankoff, kk);
        epsinfEx = GetEpsinf(i, j+PL+rankoff, kk);
        XeEx0 = GetXeDebye(i, j+PL+rankoff, kk, 0, dt, 0);
        XieEx0 = GetXieDebye(i, j+PL+rankoff, kk, 0, dt, 0);

        K1Ex[i][j][k] = epsinfEx + XeEx0 - XieEx0 + (sigmaEx*dt/2/epso);
        K2Ex[i][j][k] = epsinfEx - XieEx0 - (sigmaEx*dt/2/epso);
    }
}

//K1Ey,K2Ey
jmax = yeePerProc;
rankoff = (rank - 3)*yeePerProc;
double epsinfEy, sigmaEy, XeEy0, XieEy0;
for(i = 0; i < XCELLS + 1; i++) {
    for(int j = 0; j < yeePerProc; j++) {
        for(int k = 0; k < ZCELLS + 1; k++) {
            if(k > ZCELLS/2) {
                kk = k - 1;
            } else {
                kk = k;
            }

            if(i > XCELLS/2) {
                ii = i - 1;
            } else {
                ii = i;
            }

            epsinfEy = GetEpsinf(ii, j+PL+rankoff, kk);
            sigmaEy = GetSigma(ii, j+PL+rankoff, kk);
            XeEy0 = GetXeDebye(ii, j+PL+rankoff, kk, 0, dt, 0);
            XieEy0 = GetXieDebye(ii, j+PL+rankoff, kk, 0, dt, 0);

            K1Ey[i][j][k] = epsinfEy + XeEy0 - XieEy0 + (sigmaEy*dt/2/epso);
            K2Ey[i][j][k] = epsinfEy - XieEy0 - (sigmaEy*dt/2/epso);
        }
    }
}

//K1Ez,K2Ez
if(rank == NProcs - 1)
    jmax = yeePerProc - 1;
else
    jmax = yeePerProc;
rankoff = (rank - 3)*yeePerProc;
double epsinfEz, sigmaEz, XeEz0, XieEz0;
for(i = 0; i < XCELLS + 1; i++) {
    for(int j = 0; j < jmax; j++) {
        for(int k = 0; k < ZCELLS; k++) {
            if(i > XCELLS/2) {
                ii = i - 1;
            } else {
                ii = i;
            }

            epsinfEz = GetEpsinf(ii, j+PL+rankoff, k);
            sigmaEz = GetSigma(ii, j+PL+rankoff, k);
            XeEz0 = GetXeDebye(ii, j+PL+rankoff, k, 0, dt, 0);
XieEz0 = GetXieDebye(ii,j+PL+rankoff,k,0,dt,0);
K1Ez[i][j][k] = epsinfEz + XeEz0 - XieEz0 + (sigmaEz*dt/2/epso);
K2Ez[i][j][k] = epsinfEz - XieEz0 - (sigmaEz*dt/2/epso);
} }
}
}
}
//K1Hx,K2Hx
jmax = yeePerProc;
rankoff = (rank - 3)*yeePerProc;
for(i = 0;i < XCELLS + 1;i++) {
  for(int j = 0;j < jmax;j++) {
    for(int k = 0;k < ZCELLS;k++) {
      double MuinfHx,XmHx0,XimHx0;
      if(i == XCELLS) {
        ii = 0;
      } else if(i > XCELLS/2) {
        ii = i - 1;
      } else {
        ii = i;
      }
      MuinfHx = GetMuinf(ii,j+PL+rankoff,k);
      XmHx0 = GetXmDebye(ii,j+PL+rankoff,k,0,dt,0);
      XimHx0 = GetXimDebye(ii,j+PL+rankoff,k,0,dt,0);
      K1Hx[i][j][k] = MuinfHx + XmHx0 - XimHx0;
      K2Hx[i][j][k] = MuinfHx - XimHx0;
    }
  }
}
//K1Hy,K2Hy
//Note : Only PSY - 2 Hy's inside the problem space
if(rank == NProcs - 1)
  jmax = yeePerProc - 1;
else
  jmax = yeePerProc;
rankoff = (rank - 3)*yeePerProc;
for(i = 0;i < XCELLS;i++) {
  for(int j = 0;j < jmax;j++) {
    for(int k = 0;k < ZCELLS;k++) {
      double MuinfHy,XmHy0,XimHy0;
      MuinfHy = GetMuinf(i,j+PL+1+rankoff,k);
      XmHy0 = GetXmDebye(i,j+PL+1+rankoff,k,0,dt,0);
      XimHy0 = GetXimDebye(i,j+PL+1+rankoff,k,0,dt,0);
      K1Hy[i][j][k] = MuinfHy + XmHy0 - XimHy0;
      K2Hy[i][j][k] = MuinfHy - XimHy0;
    }
  }
}
//K1Hz,K2Hz
jmax = yeePerProc;
rankoff = (rank - 3)*yeePerProc;
for(i = 0;i < XCELLS;i++) {
  for(int j = 0;j < jmax;j++) {
    for(int k = 0;k < ZCELLS + 1;k++) {
      double MuinfHz,XmHz0,XimHz0;
      if(k == ZCELLS) {
        kk = 0;
      } else if(k > ZCELLS/2) {
        kk = k - 1;
      } else {
        kk = k;
      }
    }
  }
}
double GetMuinf(int xidx, int yidx, int zidx)
{
    int xmax, ymax, zmax;
    if(!SBC) {
        xmax = 2*XCELLS - 1;
        ymax = 2*YCELLS - 1;
        zmax = 2*ZCELLS - 1;
    } else {
        xmax = XCELLS;
        ymax = YCELLS;
        zmax = ZCELLS;
    }
    if((xidx >= 0 && xidx <= xmax) &&
       (yidx >= 0 && yidx <= ymax) &&
       (zidx >= 0 && zidx <= zmax)) {
        switch(MediaFullDomain[xidx][yidx][zidx]) {
            case AIR: case PML: case PEC: case AL: case PYREX:
            return 1.0;
            break;
            case MF110:
            return MF110Debyemur[1];
            break;
            case MF112:
            return MF112Debyemur[1];
            break;
            case MF114:
            return MF114Debyemur[1];
            break;
            default:
            return 1.0;
            break;
        }
    } else {
        cout<<"GetMuinf() : Error in indices"<<endl;
        return 1.0;
    }
}

double GetEpsinf(int xidx, int yidx, int zidx)
{
    int xmax, ymax, zmax;
    if(!SBC) {
        xmax = 2*XCELLS - 1;
        ymax = 2*YCELLS - 1;
        zmax = 2*ZCELLS - 1;
    } else {
        xmax = XCELLS;
        ymax = YCELLS;
        zmax = ZCELLS;
    }
    if((xidx >= 0 && xidx <= xmax) &&
       (yidx >= 0 && yidx <= ymax) &&
       (zidx >= 0 && zidx <= zmax)) {
        switch(MediaFullDomain[xidx][yidx][zidx]) {
            case AIR: case PML: case PEC: case AL: case PYREX:
            return 1.0;
            break;
            case MF110:
            return MF110Debyemur[1];
            break;
            case MF112:
            return MF112Debyemur[1];
            break;
            case MF114:
            return MF114Debyemur[1];
            break;
            default:
            return 1.0;
            break;
        }
    } else {
        cout<<"GetEpsinf() : Error in indices"<<endl;
        return 1.0;
    }
}
if((xidx >= 0 && xidx <= xmax) &&
   (yidx >= 0 && yidx <= ymax) &&
   (zidx >= 0 && zidx <= zmax)) {
    switch(MediaFullDomain[xidx][yidx][zidx]) {
      case AIR: case PML: case PEC: case AL:
        return 1.0;
        break;
      case PYREX:
        return 4.2;
        break;
      case MF110:
        return MF110Debyemur[1];
        break;
      case MF112:
        return MF112Debyepsr[1];
        break;
      case MF114:
        return MF114Debyepsr[1];
        break;
      default:
        return 1.0;
        break;
    }
  }
  else {
    cout<<xidx<<endl;
    cout<<zidx<<endl;
    cout<<yidx<<endl;
    cout<<"GetEpsinf() : Error in indices"<<endl;
    return 1.0;
  }
}

// Get the value of sigma (electrical conductivity)
// in the main grid cell
double GetSigma(int xidx,int yidx,int zidx) {
  int xmax,ymax,zmax;
  if(!SBC) {
    xmax = 2*XCELLS - 1;
    ymax = 2*YCELLS - 1;
    zmax = 2*ZCELLS - 1;
  } else {
    xmax = XCELLS;
    ymax = YCELLS;
    zmax = ZCELLS;
  }
  if((xidx >= 0 && xidx <= xmax) &&
     (yidx >= 0 && yidx <= ymax) &&
     (zidx >= 0 && zidx <= zmax)) {
    switch(MediaFullDomain[xidx][yidx][zidx]) {
      case AIR: case PYREX:
        return 0.0;
        break;
      case PML:
        cout<<"GetSigma() : PML conductivity error"<<endl;
        return 0.0;
        break;
    }
case PEC:
    return PECsigma;
break;
case AL:
    return ALsigma;
break;
default:
    return 0.0;
break;
}
}
else {
    cout<"GetSigma() : Error in indices"<<endl;
return 1.0;
}
}

//pole = [1,MAXPOLESEPS]
//pole = 0 -- total
double GetXeDebye(int xidx,int yidx,int zidx,int p,double dt,uint pole)
{
    int xmax,ymax,zmax;
if(!SBC) {
    xmax = 2*XCELLS - 1;
    ymax = 2*YCELLS - 1;
    zmax = 2*ZCELLS - 1;
} else {
    xmax = XCELLS;
    ymax = YCELLS;
    zmax = ZCELLS;
}
if((xidx >= 0 && xidx <= xmax) &&
   (yidx >= 0 && yidx <= ymax) &&
   (zidx >= 0 && zidx <= zmax)) {
    double Xe = 0,deltaeps,taueps;
    uint pidx;
    switch(MediaFullDomain[xidx][yidx][zidx]) {
    case AIR: case PML: case PEC: case AL:
        return 0.0;
break;
case MF110:
    Xe = 0;
    if(pole > MF110Debyeepsr[0]) {
        cout<"GetXeDebye :: Pole index error."<<endl;
        return 0.0;
    } else if(pole == 0) {
        for(pidx = 1;pidx <= MF110Debyeepsr[0];pidx++) {
            deltaeps = MF110Debyeepsr[2*pidx];
            taueps = MF110Debyeepsr[2*pidx + 1];
            Xe = Xe + deltaeps*(exp(-p*dt/taueps)-
                             exp(-(p+1)*dt/taueps));
        }
    } else {
        deltaeps = MF110Debyeepsr[2*pole];
        taueps = MF110Debyeepsr[2*pole + 1];
        Xe = deltaeps*(exp(-p*dt/taueps)- exp(-(p+1)*dt/taueps));
    }
return Xe;
break;
case MF112:
    Xe = 0;
if(pole > MF112Debyeepsr[0]) {
    cout<<"GetXeDebye :: Pole index error."<<endl;
    return 0.0;
} 
else if(pole == 0) {
    for(pidx = 1;pidx <= MF112Debyeepsr[0];pidx++) {
        deltaeps = MF112Debyeepsr[2*pidx];
        taueps = MF112Debyeepsr[2*pidx + 1];
        Xe = Xe + deltaeps*(exp(-p*dt/taueps) -
                         exp(-(p+1)*dt/taueps));
    }
}
else {
    deltaeps = MF112Debyeepsr[2*pole];
    taueps = MF112Debyeepsr[2*pole + 1];
    Xe = deltaeps*(exp(-p*dt/taueps) - exp(-(p+1)*dt/taueps));
}
return Xe;
break;

case MF114:
    Xe = 0;
    if(pole > MF114Debyeepsr[0]) {
        cout<<"GetXeDebye :: Pole index error."<<endl;
        return 0.0;
    } 
else if(pole == 0) {
    for(pidx = 1;pidx <= MF114Debyeepsr[0];pidx++) {
        deltaeps = MF114Debyeepsr[2*pidx];
        taueps = MF114Debyeepsr[2*pidx + 1];
        Xe = Xe + deltaeps*(exp(-p*dt/taueps) -
                         exp(-(p+1)*dt/taueps));
    }
}
else {
    deltaeps = MF114Debyeepsr[2*pole];
    taueps = MF114Debyeepsr[2*pole + 1];
    Xe = deltaeps*(exp(-p*dt/taueps) - exp(-(p+1)*dt/taueps));
}
return Xe;
break;
default:
    return 0.0;
break;
}
}
else {
    cout<<"GetXeDebye() : Error in indices"<<endl;
    return 0.0;
}
}

//pole - [1,MAXPOLESMU]
//pole = 0 -- total
double GetXmDebye(int xidx,int yidx,int zidx,int p,double dt,uint pole)
{
    int xmax,ymax,zmax;
    int xmax,ymax,zmax;
    if(!SBC) {
        xmax = 2*XCELLS - 1;
        ymax = 2*YCELLS - 1;
        zmax = 2*ZCELLS - 1;
    } 
else {
        xmax = XCELLS;
        ymax = YCELLS;
        zmax = ZCELLS;
    }
    if((xidx >= 0 && xidx <= xmax) &&
\begin{verbatim}
(yidx >= 0 && yidx <= ymax) &&
(zidx >= 0 && zidx <= zmax)) {

double Xm = 0, deltamu, tauamu;
uint pidx;

switch(MediaFullDomain[xidx][yidx][zidx]) {
    case AIR: case PML: case PEC: case AL:
        return 0.0;
        break;
    case PYREX:
        return 0.0;
        break;
    case MF110:
        Xm = 0;
        if(pole > MF110Debyemur[0]) {
            cout<<"GetXmDebye :: Pole index error."<<endl;
            return 0.0;
        } else if(pole == 0) {
            for(pidx = 1;pidx <= MF110Debyemur[0];pidx++) {
                deltamu = MF110Debyemur[2*pidx];
                tauamu = MF110Debyemur[2*pidx + 1];
                Xm = Xm + deltamu*(exp(-p*dt/tauamu) -
                                exp(-(p+1)*dt/tauamu));
            }
        } else {
            deltamu = MF110Debyemur[2*pole];
            tauamu = MF110Debyemur[2*pole + 1];
            Xm = deltamu*(exp(-p*dt/tauamu) -
                             exp(-(p+1)*dt/tauamu));
        }
        return Xm;
        break;
    case MF112:
        Xm = 0;
        if(pole > MF112Debyemur[0]) {
            cout<<"GetXmDebye :: Pole index error."<<endl;
            return 0.0;
        } else if(pole == 0) {
            for(pidx = 1;pidx <= MF112Debyemur[0];pidx++) {
                deltamu = MF112Debyemur[2*pidx];
                tauamu = MF112Debyemur[2*pidx + 1];
                Xm = Xm + deltamu*(exp(-p*dt/tauamu) -
                                exp(-(p+1)*dt/tauamu));
            }
        } else {
            deltamu = MF112Debyemur[2*pole];
            tauamu = MF112Debyemur[2*pole + 1];
            Xm = deltamu*(exp(-p*dt/tauamu) -
                             exp(-(p+1)*dt/tauamu));
        }
        return Xm;
        break;
    case MF114:
        Xm = 0;
        if(pole > MF114Debyemur[0]) {
            cout<<"GetXmDebye :: Pole index error."<<endl;
            return 0.0;
        } else if(pole == 0) {
            for(pidx = 1;pidx <= MF114Debyemur[0];pidx++) {
                deltamu = MF114Debyemur[2*pidx];
                tauamu = MF114Debyemur[2*pidx + 1];
                Xm = Xm + deltamu*(exp(-p*dt/tauamu) -
                                exp(-(p+1)*dt/tauamu));
            }
        }
        return Xm;
    }
\end{verbatim}
else {
    deltamu = MF114Debyemur[2*pole];
    taumu = MF114Debyemur[2*pole + 1];
    Xm = deltamu*(exp(-p*dt/taumu)- exp(-(p+1)*dt/taumu));
}  
return Xm;
break;
default:
    return 0.0;
    break;
}
}
else {
    cout<"GetXmDebye() : Error in indices"<<endl;
    return 0.0;
}
}
//Returns the pth Xim for the given spatial indices.
//pole = 0. Returns the sum of the contribution due to each pole.
//Otherwise returns the pole contribution
//pth Xim for the dth pole = deltamu*exp(-p*dt/taumu)*
// (taumu/dt - (taumu/dt + 1)*exp(-dt/taumu))
//pole - [1,MAXPOLES]
double GetXieDebye(int xidx,int yidx,int zidx,int p,double dt,uint pole)
{
    int xmax,ymax,zmax;
    if(!SBC) {
        xmax = 2*XCELLS - 1;
        ymax = 2*YCELLS - 1;
        zmax = 2*ZCELLS - 1;
    } else {
        xmax = XCELLS;
        ymax = YCELLS;
        zmax = ZCELLS;
    }
    if((xidx >= 0 && xidx <= xmax) &&
      (yidx >= 0 && yidx <= ymax) &&
      (zidx >= 0 && zidx <= zmax)) {
        double Xie = 0,deltaeps,taueps;
        uint pidx;

        switch(MediaFullDomain[xidx][yidx][zidx]) {
            case AIR: case PML: case PEC: case AL:
            case PYREX:
                return 0.0;
                break;
            case MF110:
                Xie = 0;
                if(pole > MF110Debyeepsr[0]) {
                    cout<"GetXieDebye : Pole index error."<<endl;
                    return 0.0;
                } else if(pole == 0) {
                    for(pidx = 1;pidx <= MF110Debyeepsr[0];pidx++) {
                        deltaeps = MF110Debyeepsr[2*pidx];
                        taueps = MF110Debyeepsr[2*pidx + 1];
                        Xie = Xie + deltaeps*exp(-p*dt/taueps)*(taueps/dt -
                            (taueps/dt + 1)*exp(-dt/taueps));
                    }
                } else {
                    deltaeps = MF110Debyeepsr[2*pole];
                    taueps = MF110Debyeepsr[2*pole + 1];
                    Xie = deltaeps*exp(-p*dt/taueps)*(taueps/dt -
                        (taueps/dt + 1)*exp(-dt/taueps));
                }
        }
    } else {
        Xie = 0;
    }
    return Xie;
}
(taueps/dt + 1)*exp(-dt/taueps));
} return Xie;
break;
case MF112:
    Xie = 0;
if(pole > MF112Debyeepsr[0]) {
    cout("GetXieDebye :: Pole index error."<<endl;
    return 0.0;
} else if(pole == 0) {
    for(pidx = 1;pidx <= MF112Debyeepsr[0];pidx++) {
        deltaeps = MF112Debyeepsr[2*pidx];
        taueps = MF112Debyeepsr[2*pidx + 1];
        Xie = Xie + deltaeps*exp(-p*dt/taueps)*
             (taueps/dt - (taueps/dt + 1)*exp(-dt/taueps));
    }
} else {
    deltaeps = MF112Debyeepsr[2*pole];
    taueps = MF112Debyeepsr[2*pole + 1];
    Xie = deltaeps*exp(-p*dt/taueps)*
         (taueps/dt - (taueps/dt + 1)*exp(-dt/taueps));
} return Xie;
break;
case MF114:
    Xie = 0;
if(pole > MF114Debyeepsr[0]) {
    cout("GetXieDebye :: Pole index error."<<endl;
    return 0.0;
} else if(pole == 0) {
    for(pidx = 1;pidx <= MF114Debyeepsr[0];pidx++) {
        deltaeps = MF114Debyeepsr[2*pidx];
        taueps = MF114Debyeepsr[2*pidx + 1];
        Xie = Xie + deltaeps*exp(-p*dt/taueps)*
             (taueps/dt - (taueps/dt + 1)*exp(-dt/taueps));
    }
} else {
    deltaeps = MF114Debyeepsr[2*pole];
    taueps = MF114Debyeepsr[2*pole + 1];
    Xie = deltaeps*exp(-p*dt/taueps)*
         (taueps/dt - (taueps/dt + 1)*exp(-dt/taueps));
} return Xie;
break;
default:
    return 0.0;
break;
}
else {
    cout("GetXieDebye() : Error in indices"<<endl;
    return 0.0;
}
}

// Returns the pth Xim for the given spatial indices.
// pole = 0. Returns the sum of the contribution due to each pole.
// Otherwise returns the pole contribution
// pth Xim for the dth pole = deltamu*exp(-p*dt/taumu)
// (taumu/dt - (taumu/dt + 1)*exp(-dt/taumu))
// pole - [1,MAXPOLESMU]
double GetXimDebye(int xidx,int yidx,int zidx,int p,double dt,uint pole) {

int xmax, ymax, zmax;
if(!SBC) {
    xmax = 2*XCELLS - 1;
    ymax = 2*YCELLS - 1;
    zmax = 2*ZCELLS - 1;
} else {
    xmax = XCELLS;
    ymax = YCELLS;
    zmax = ZCELLS;
}
if((xidx >= 0 && xidx <= xmax) &&
    (yidx >= 0 && yidx <= ymax) &&
    (zidx >= 0 && zidx <= zmax)) {

double Xim = 0, deltamu, taumu;
uint pidx;

switch(MediaFullDomain[xidx][yidx][zidx]) {
    case AIR: case PML: case PEC: case AL: case PYREX:
        return 0.0;
        break;
    case MF110:
        Xim = 0;
        if(pole > MF110Debyemur[0]) {
            cout<<"GetXimDebye :: Pole index error.\n"<<endl;
            return 0.0;
        } else if(pole == 0) {
            for(pidx = 1;pidx <= MF110Debyemur[0];pidx++) {
                deltamu = MF110Debyemur[2*pidx];
                taumu = MF110Debyemur[2*pidx + 1];
                Xim = Xim + deltamu*exp(-(p*dt/taumu)*(taumu/dt -
                    (taumu/dt + 1)*exp(-dt/taumu)));
            }
        } else {
            deltamu = MF110Debyemur[2*pole];
            taumu = MF110Debyemur[2*pole + 1];
            Xim = deltamu*exp(-(p*dt/taumu)*
                (taumu/dt + 1)*exp(-dt/taumu));
        }
        return Xim;
        break;
    case MF112:
        Xim = 0;
        if(pole > MF112Debyemur[0]) {
            cout<<"GetXimDebye :: Pole index error.\n"<<endl;
            return 0.0;
        } else if(pole == 0) {
            for(pidx = 1;pidx <= MF112Debyemur[0];pidx++) {
                deltamu = MF112Debyemur[2*pidx];
                taumu = MF112Debyemur[2*pidx + 1];
                Xim = Xim + deltamu*exp(-(p*dt/taumu)*
                    (taumu/dt + 1)*exp(-dt/taumu));
            }
        } else {
            deltamu = MF112Debyemur[2*pole];
            taumu = MF112Debyemur[2*pole + 1];
            Xim = deltamu*exp(-(p*dt/taumu)*
                (taumu/dt - (taumu/dt + 1)*exp(-dt/taumu)));
        }
        return Xim;
        break;
}
}
case MF114:
    Xim = 0;
    if(pole > MF114Debyemur[0]) {
        cout<<"GetXimDebye :: Pole index error."<<endl;
        return 0.0;
    } else if(pole == 0) {
        for(pidx = 1;pidx <= MF114Debyemur[0];pidx++) {
            deltamu = MF114Debyemur[2*pidx];
            taumu = MF114Debyemur[2*pidx + 1];
            Xim = Xim + deltamu*exp(-p*dt/taumu)*(taumu/dt
                - (taumu/dt + 1)*exp(-dt/taumu));
        }
    } else {
        deltamu = MF114Debyemur[2*pole];
        taumu = MF114Debyemur[2*pole + 1];
        Xim = deltamu*exp(-p*dt/taumu)*(taumu/dt
                - (taumu/dt + 1)*exp(-dt/taumu));
    }
    return Xim;
    break;
default:
    return 0.0;
    break;
}
else {
    cout<<"GetXimDebye() : Error in indices"<<endl;
    return 0.0;
}
}

double GetTaumu(uint media,uint pole)
{
    if(!(pole >=1 && pole <= MAXPOLESMU)) {
        cout<<"GetTaumu :: Pole index error"<<endl;
        return 0;
    }
    switch(media) {
    case MF112:
        return MF112Debyemur[2*pole+1];
        break;
    case MF110:
        return MF110Debyemur[2*pole+1];
        break;
    case MF114:
        return MF114Debyemur[2*pole+1];
        break;
    default:
        return 0;
    }
}

double GetTaueps(uint media,uint pole)
{
    if(!(pole >=1 && pole <= MAXPOLESEPS)) {
        cout<<"GetTaueps :: Pole index error"<<endl;
        return 0;
    }
    switch(media) {
    case MF112:
        return MF112Debyeepsr[2*pole+1];
        break;
    case MF110:
        return MF110Debyeepsr[2*pole+1];
        break;
    case MF114:
        return MF114Debyeepsr[2*pole+1];
        break;
    default:
        return 0;
    }
case MF114:
    return MF114Debyeepsr[2*pole+1];
    break;

default:
    return 0;
}

// Create all the matrices used in the program
void CreateAllMatrices(int rank) {
    int XCELLSP, ZCELLSP;
    XCELLSP = XCELLS;
    ZCELLSP = ZCELLS;

    // Field matrices
    if(SBC) {
        XCELLS = Round(XCELLS/2);
        ZCELLS = Round(ZCELLS/2);
    }

    // Rank 1 = PML0
    // Rank 2 = PML1
    if(rank == 1 || rank == 2) {
        CreateMatrix(Ex, XCELLS, PL + 1, ZCELLS + 1);
        CreateMatrix(Ey, XCELLS + 1, PL, ZCELLS + 1);
        CreateMatrix(Ez, XCELLS + 1, PL + 1, ZCELLS + 1);
        CreateMatrix(Dx, XCELLS + 1, PL + 1, ZCELLS); 
        CreateMatrix(Dy, XCELLS + 1, PL, ZCELLS + 1);
        CreateMatrix(Dz, XCELLS + 1, PL, ZCELLS + 1);
        CreateMatrix(Dxo, XCELLS, PL + 1, ZCELLS + 1);
        CreateMatrix(Dyo, XCELLS + 1, PL, ZCELLS + 1);
        CreateMatrix(Dzo, XCELLS + 1, PL + 1, ZCELLS);
    }

    else if(rank != 0) {
        // Non-PML domains
        // Rank 3 .. Rank(NProcs-1)
        // Rank 3 adjacent to PML0 (ymin PML)
        // Rank(NProcs-1) adjacent to PML1
        if(rank != NProcs - 1) {
            CreateMatrix(Ex, XCELLS, yeePerProc, ZCELLS + 1);
            CreateMatrix(Ez, XCELLS + 1, yeePerProc - 1, ZCELLS);
            CreateMatrix(Hy, XCELLS, yeePerProc, ZCELLS);
        }
        else {
            CreateMatrix(Ex, XCELLS, yeePerProc - 1, ZCELLS + 1);
            CreateMatrix(Ez, XCELLS + 1, yeePerProc - 1, ZCELLS);
            CreateMatrix(Hx, XCELLS, yeePerProc, ZCELLS);
        }
    }

    CreateMatrix(Ey, XCELLS, yeePerProc, ZCELLS + 1);
    CreateMatrix(Hx, XCELLS + 1, yeePerProc, ZCELLS + 1);
    CreateMatrix(Hz, XCELLS, yeePerProc, ZCELLS + 1);
}

// Coefficient matrices are first filled in PBC condition
// and then split for the SBC case
if(SBC) {
    XCELLS = XCELLSP;
    ZCELLS = ZCELLSP;
}

// UML coefficients
if(rank == 1 || rank == 2) {
    CreateMatrix(C1Dx, XCELLS, PL + 1, ZCELLS + 1);
    CreateMatrix(C2Dx, XCELLS, PL + 1, ZCELLS + 1);
    CreateMatrix(C1Bx, XCELLS + 1, PL, ZCELLS);
    CreateMatrix(C2Bx, XCELLS + 1, PL, ZCELLS);
    CreateMatrix(C1Ey, XCELLS + 1, PL, ZCELLS + 1);
    CreateMatrix(C2Ey, XCELLS + 1, PL, ZCELLS + 1);
    CreateMatrix(C1Ez, XCELLS + 1, PL + 1, ZCELLS);
    CreateMatrix(C2Ez, XCELLS + 1, PL + 1, ZCELLS);
    CreateMatrix(C2Hy, XCELLS, PL + 1, ZCELLS);
    CreateMatrix(C3Hy, XCELLS, PL + 1, ZCELLS);
    CreateMatrix(C1Hz, XCELLS + 1, ZCELLS + 1);
    CreateMatrix(C2Hz, XCELLS + 1, ZCELLS + 1);
}

// PLRC coefficients
if(rank != 0 && rank != 1 && rank != 2) {
    // Non PML domains
    // Rank3 .. Rank(NProcs-1)
    // Rank3 adjacent to PML0 (ymin PML)
    // Rank(NProcs-1) adjacent to PML1
    if(rank != NProcs - 1) {
        CreateMatrix(K1Ex, XCELLS, yeePerProc, ZCELLS + 1);
        CreateMatrix(K2Ex, XCELLS, yeePerProc, ZCELLS + 1);
        CreateMatrix(K1Ez, XCELLS + 1, yeePerProc, ZCELLS);
        CreateMatrix(K2Ez, XCELLS + 1, yeePerProc, ZCELLS);
        CreateMatrix(K1Hy, XCELLS, yeePerProc, ZCELLS);
        CreateMatrix(K2Hy, XCELLS, yeePerProc, ZCELLS);
        CreateMatrix(RAEx, XCELLS, yeePerProc, ZCELLS + 1, MAXPOLESEPS);
        CreateMatrix(RAEz, XCELLS + 1, yeePerProc, ZCELLS, MAXPOLESEPS);
        CreateMatrix(RAHy, XCELLS, yeePerProc, ZCELLS, MAXPOLEMU);
    } else {
        CreateMatrix(K1Ex, XCELLS, yeePerProc - 1, ZCELLS + 1);
        CreateMatrix(K2Ex, XCELLS, yeePerProc - 1, ZCELLS + 1);
        CreateMatrix(K1Ez, XCELLS + 1, yeePerProc - 1, ZCELLS);
        CreateMatrix(K2Ez, XCELLS + 1, yeePerProc - 1, ZCELLS);
        CreateMatrix(K1Hy, XCELLS, yeePerProc - 1, ZCELLS);
        CreateMatrix(K2Hy, XCELLS, yeePerProc - 1, ZCELLS);
        CreateMatrix(RAEx, XCELLS, yeePerProc - 1, ZCELLS + 1, MAXPOLESEPS);
        CreateMatrix(RAEz, XCELLS + 1, yeePerProc - 1, ZCELLS, MAXPOLESEPS);
        CreateMatrix(RAHy, XCELLS, yeePerProc - 1, ZCELLS, MAXPOLEMU);
    }
    CreateMatrix(K1Ey, XCELLS + 1, yeePerProc, ZCELLS + 1);
    CreateMatrix(K2Ey, XCELLS + 1, yeePerProc, ZCELLS + 1);
    CreateMatrix(K1Hz, XCELLS + 1, yeePerProc, ZCELLS);
    CreateMatrix(K2Hz, XCELLS + 1, yeePerProc, ZCELLS);
    CreateMatrix(K1Ey, XCELLS + 1, yeePerProc, ZCELLS + 1);
    CreateMatrix(K2Ey, XCELLS + 1, yeePerProc, ZCELLS + 1);
    CreateMatrix(K1Hz, XCELLS, yeePerProc, ZCELLS + 1);
    CreateMatrix(K2Hz, XCELLS, yeePerProc, ZCELLS + 1);
    CreateMatrix(RAEy, XCELLS + 1, yeePerProc, ZCELLS + 1, MAXPOLESEPS);
    CreateMatrix(RAHx, XCELLS + 1, yeePerProc, ZCELLS, MAXPOLEMU);
    CreateMatrix(RAHz, XCELLS, yeePerProc, ZCELLS + 1, MAXPOLEMU);
}

 tileSize = XCELLS * ZCELLS;

// Media matrix
if(rank == 1 || rank == 2) {
    CreateMatrix(Media, XCELLS, PL, ZCELLS);
} else if(rank != 0) {
CreateMatrix(Media,XCELLS,yeePerProc,ZCELLS);
*/

//MediaFullDomain is for all the processors
CreateMatrix(MediaFullDomain,XCELLS,YCELLS,ZCELLS);
}

// Delete all the matrices used in the program
void DeleteAllMatrices(int rank)
{
    //Rank1 = PML0
    //Rank2 = PML1
    if(rank == 1 || rank == 2) {
        DeleteMatrix(Ex,XCELLS,PL + 1,ZCELLS + 1);
        DeleteMatrix(Ey,XCELLS + 1,PL,ZCELLS + 1);
        DeleteMatrix(Ez,XCELLS + 1,PL + 1,ZCELLS + 1);
        DeleteMatrix(Dx,XCELLS,PL + 1,ZCELLS + 1);
        DeleteMatrix(Dy,XCELLS + 1,PL,ZCELLS + 1);
        DeleteMatrix(Dz,XCELLS + 1,PL + 1,ZCELLS);
        DeleteMatrix(Dxo,XCELLS,PL + 1,ZCELLS + 1);
        DeleteMatrix(Dyo,XCELLS + 1,PL,ZCELLS + 1);
        DeleteMatrix(Dzo,XCELLS + 1,PL + 1,ZCELLS);
        DeleteMatrix(Hx,XCELLS + 1,PL,ZCELLS);
        DeleteMatrix(Hy,XCELLS,PL + 1,ZCELLS + 1);
        DeleteMatrix(Hz,XCELLS,PL,ZCELLS + 1);
        DeleteMatrix(Bx,XCELLS + 1,PL,ZCELLS);
        DeleteMatrix(By,XCELLS,PL + 1,ZCELLS);
        DeleteMatrix(Bz,XCELLS,PL,ZCELLS + 1);
        DeleteMatrix(Bxo,XCELLS + 1,PL,ZCELLS);
        DeleteMatrix(Byo,XCELLS,PL + 1,ZCELLS);
        DeleteMatrix(Bzo,XCELLS,PL,ZCELLS + 1);
    }
    else if(rank != 0) {
        //Non-PML domains
        //Rank3 .. Rank(NProcs-1)
        //Rank3 adjacent to PML0 (ymin PML)
        //Rank(NProcs-1) adjacent to PML1
        if(rank != NProcs - 1) {
            DeleteMatrix(Ex,XCELLS,yeePerProc,ZCELLS + 1);
            DeleteMatrix(Ez,XCELLS + 1,yeePerProc,ZCELLS);
        }
        else {
            DeleteMatrix(Ex,XCELLS,yeePerProc - 1,ZCELLS + 1);
            DeleteMatrix(Ey,XCELLS + 1,yeePerProc - 1,ZCELLS);
        }
        DeleteMatrix(Ey,XCELLS + 1,yeePerProc,ZCELLS + 1);
        DeleteMatrix(Hx,XCELLS + 1,yeePerProc,ZCELLS);
        DeleteMatrix(Hz,XCELLS,yeePerProc,ZCELLS + 1);
    }
    //UPML coefficients
    if(rank == 1 || rank == 2) {
        DeleteMatrix(C1Bx,XCELLS,PL + 1,ZCELLS + 1);
        DeleteMatrix(C2Bx,XCELLS + 1,PL,ZCELLS + 1);
        DeleteMatrix(C2Bx,XCELLS + 1,PL,ZCELLS);
        DeleteMatrix(C2Bx,XCELLS,PL + 1,ZCELLS);
        DeleteMatrix(C3Bx,XCELLS + 1,PL,ZCELLS + 1);
        DeleteMatrix(C3Bx,XCELLS + 1,PL,ZCELLS);
        DeleteMatrix(C3Bx,XCELLS,PL + 1,ZCELLS);
        DeleteMatrix(C2Hy,XCELLS,PL + 1,ZCELLS);
        DeleteMatrix(C3Hy,XCELLS,PL + 1,ZCELLS);
        DeleteMatrix(C1Hz,XCELLS,PL,ZCELLS + 1);
        DeleteMatrix(C2Hz,XCELLS,PL,ZCELLS + 1);
    }
if(rank != 0 && rank != 1 && rank != 2) {
    //Non PML domains
    //Rank3 .. Rank(NProcs-1)
    //Rank3 adjacent to PML0 (ymin PML)
    //Rank(NProcs-1) adjacent to PML1
    if(rank != NProcs - 1) {
        DeleteMatrix(K1Ex, XCELLS, yeePerProc, ZCELLS + 1);
        DeleteMatrix(K2Ex, XCELLS, yeePerProc, ZCELLS + 1);
        DeleteMatrix(K1Ez, XCELLS + 1, yeePerProc, ZCELLS);
        DeleteMatrix(K2Ez, XCELLS + 1, yeePerProc, ZCELLS);
        DeleteMatrix(K1Hy, XCELLS, yeePerProc, ZCELLS);
        DeleteMatrix(K2Hy, XCELLS, yeePerProc, ZCELLS);
        DeleteMatrix(RAEx, XCELLS, yeePerProc, ZCELLS + 1, MAXPOLESEPS);
        DeleteMatrix(RAEz, XCELLS + 1, yeePerProc, ZCELLS, MAXPOLESEPS);
        DeleteMatrix(RAHy, XCELLS, yeePerProc, ZCELLS, MAXPOLESMU);
    } else {
        DeleteMatrix(K1Ex, XCELLS, yeePerProc - 1, ZCELLS + 1);
        DeleteMatrix(K2Ex, XCELLS, yeePerProc - 1, ZCELLS + 1);
        DeleteMatrix(K1Ez, XCELLS + 1, yeePerProc - 1, ZCELLS);
        DeleteMatrix(K2Ez, XCELLS + 1, yeePerProc - 1, ZCELLS);
        DeleteMatrix(K1Hy, XCELLS, yeePerProc - 1, ZCELLS);
        DeleteMatrix(K2Hy, XCELLS, yeePerProc - 1, ZCELLS);
        DeleteMatrix(RAEx, XCELLS, yeePerProc - 1, ZCELLS + 1, MAXPOLESEPS);
        DeleteMatrix(RAEz, XCELLS + 1, yeePerProc - 1, ZCELLS, MAXPOLESEPS);
        DeleteMatrix(RAHy, XCELLS, yeePerProc - 1, ZCELLS, MAXPOLESMU);
    }
    DeleteMatrix(K1Ey, XCELLS + 1, yeePerProc, ZCELLS + 1);
    DeleteMatrix(K2Ey, XCELLS + 1, yeePerProc, ZCELLS + 1);
    DeleteMatrix(K1Hy, XCELLS + 1, yeePerProc, ZCELLS);
    DeleteMatrix(K2Hy, XCELLS + 1, yeePerProc, ZCELLS);
    DeleteMatrix(RAEy, XCELLS + 1, yeePerProc, ZCELLS + 1, MAXPOLESEPS);
    DeleteMatrix(RAHx, XCELLS + 1, yeePerProc, ZCELLS, MAXPOLESMU);
    DeleteMatrix(RAHx, XCELLS, yeePerProc, ZCELLS + 1, MAXPOLESMU);
}

void ConvToSBC(int rank) {
    int XCELLSP, ZCELLSP;
    XCELLSP = XCELLS;
    ZCELLSP = ZCELLS;
    XCELLS = Round(XCELLS/2);
    ZCELLS = Round(ZCELLS/2);

    //Copy the UPML coefficients
    if(rank == 1 || rank == 2) {
        CopyMatrix(C1Dx, XCELLS, PL + 1, ZCELLS + 1, XCELLSP, PL + 1, ZCELLSP + 1);
        CopyMatrix(C2Dx, XCELLS, PL + 1, ZCELLS + 1, XCELLSP, PL + 1, ZCELLSP + 1);
        CopyMatrix(C1Bx, XCELLS + 1, PL, ZCELLS, XCELLSP + 1, PL, ZCELLSP);
        CopyMatrix(C2Bx, XCELLS + 1, PL, ZCELLS, XCELLSP + 1, PL, ZCELLSP);
        CopyMatrix(C1Ey, XCELLS + 1, PL, ZCELLS + 1, XCELLSP + 1, PL, ZCELLSP + 1);
        CopyMatrix(C2Ey, XCELLS + 1, PL, ZCELLS + 1, XCELLSP + 1, PL, ZCELLSP + 1);
        CopyMatrix(C3Ey, XCELLS + 1, PL, ZCELLS + 1, XCELLSP + 1, PL, ZCELLSP + 1);
        CopyMatrix(C1Ez, XCELLS + 1, PL + 1, ZCELLS, XCELLSP + 1, PL + 1, ZCELLSP);
        CopyMatrix(C2Ez, XCELLS + 1, PL + 1, ZCELLS, XCELLSP + 1, PL + 1, ZCELLSP);
        CopyMatrix(C3Ez, XCELLS + 1, PL + 1, ZCELLS, XCELLSP + 1, PL + 1, ZCELLSP);
        CopyMatrix(C1Hy, XCELLS, PL + 1, ZCELLS, XCELLSP + 1, PL + 1, ZCELLSP);
        CopyMatrix(C2Hy, XCELLS, PL + 1, ZCELLS, XCELLSP + 1, PL + 1, ZCELLSP);
        CopyMatrix(C3Hy, XCELLS, PL + 1, ZCELLS, XCELLSP + 1, PL + 1, ZCELLSP);
        CopyMatrix(C1Hz, XCELLS, PL, ZCELLS + 1, XCELLSP, PL, ZCELLSP + 1);
        CopyMatrix(C2Hz, XCELLS, PL, ZCELLS + 1, XCELLSP, PL, ZCELLSP + 1);
        CopyMatrix(C3Hz, XCELLS, PL, ZCELLS + 1, XCELLSP, PL, ZCELLSP + 1);
    }
}
if(rank != 0 && rank != 1 && rank != 2) {
if(rank != NProcs-1) {
    CopyMatrix(K1Ex,XCELLS,yeePerProc,ZCELLS + 1,XCELLSP,yeePerProc,ZCELLSP + 1);
    CopyMatrix(K2Ex,XCELLS,yeePerProc,ZCELLS + 1,XCELLSP,yeePerProc,ZCELLSP + 1);
    CopyMatrix(K1Ez,XCELLS + 1,yeePerProc,ZCELLS,XCELLSP + 1,yeePerProc,ZCELLSP);
    CopyMatrix(K2Ez,XCELLS + 1,yeePerProc,ZCELLS,XCELLSP + 1,yeePerProc,ZCELLSP);
    CopyMatrix(K1Hy,XCELLS,yeePerProc,ZCELLS,XCELLSP,yeePerProc,ZCELLSP);
    CopyMatrix(K2Hy,XCELLS,yeePerProc,ZCELLS,XCELLSP,yeePerProc,ZCELLSP);
    CopyMatrix(RAEx,XCELLS,yeePerProc,ZCELLS + 1,MAXPOLESEPS,
                XCELLSP,yeePerProc,ZCELLSP + 1,MAXPOLESEPS);
    CopyMatrix(RAEz,XCELLS + 1,yeePerProc,ZCELLS,MAXPOLESEPS,
                XCELLSP + 1,yeePerProc,ZCELLSP,MAXPOLESEPS);
    CopyMatrix(RAHy,XCELLS,yeePerProc,ZCELLS,MAXPOLESMU,
                XCELLSP,yeePerProc,ZCELLSP,MAXPOLESMU);
} else {
    CopyMatrix(K1Ex,XCELLS,yeePerProc - 1,ZCELLS + 1,XCELLSP,yeePerProc - 1,ZCELLSP + 1);
    CopyMatrix(K2Ex,XCELLS,yeePerProc - 1,ZCELLS + 1,XCELLSP,yeePerProc - 1,ZCELLSP + 1);
    CopyMatrix(K1Ez,XCELLS + 1,yeePerProc - 1,ZCELLS,XCELLSP + 1,yeePerProc - 1,XCELLSP);
    CopyMatrix(K2Ez,XCELLS + 1,yeePerProc - 1,ZCELLS,XCELLSP + 1,yeePerProc - 1,XCELLSP);
    CopyMatrix(K1Hy,XCELLS,yeePerProc - 1,ZCELLS,XCELLSP,yeePerProc - 1,ZCELLSP);
    CopyMatrix(K2Hy,XCELLS,yeePerProc - 1,ZCELLS,XCELLSP,yeePerProc - 1,ZCELLSP);
    CopyMatrix(RAEx,XCELLS,yeePerProc - 1,ZCELLS + 1,MAXPOLESEPS,
                XCELLSP,yeePerProc - 1,ZCELLSP + 1,MAXPOLESEPS);
    CopyMatrix(RAEz,XCELLS + 1,yeePerProc - 1,ZCELLS,MAXPOLESEPS,
                XCELLSP + 1,yeePerProc - 1,ZCELLSP,MAXPOLESEPS);
    CopyMatrix(RAHy,XCELLS,yeePerProc - 1,ZCELLS,MAXPOLESMU,
                XCELLSP,yeePerProc - 1,ZCELLSP,MAXPOLESMU);
}

    CopyMatrix(K1Ey,XCELLS + 1,yeePerProc,ZCELLS + 1,XCELLSP + 1,yeePerProc,ZCELLSP + 1);
    CopyMatrix(K2Ey,XCELLS + 1,yeePerProc,ZCELLS + 1,XCELLSP + 1,yeePerProc,ZCELLSP + 1);
    CopyMatrix(K1Hx,XCELLS + 1,yeePerProc,ZCELLS,XCELLSP + 1,yeePerProc,ZCELLSP);
    CopyMatrix(K2Hx,XCELLS + 1,yeePerProc,ZCELLS,XCELLSP + 1,yeePerProc,ZCELLSP);
    CopyMatrix(K1Hz,XCELLS,yeePerProc,ZCELLS + 1,XCELLSP,yeePerProc,ZCELLSP + 1);
    CopyMatrix(K2Hz,XCELLS,yeePerProc,ZCELLS + 1,XCELLSP,yeePerProc,ZCELLSP + 1);
    CopyMatrix(RAEy,XCELLS + 1,yeePerProc,ZCELLS + 1,MAXPOLESEPS,
                XCELLSP + 1,yeePerProc,ZCELLSP + 1,MAXPOLESEPS);
    CopyMatrix(RAhx,XCELLS + 1,yeePerProc,ZCELLS,MAXPOLESMU,
                XCELLSP + 1,yeePerProc,ZCELLSP,MAXPOLESMU);
    CopyMatrix(RAhz,XCELLS,yeePerProc,ZCELLS + 1,MAXPOLESMU,
                XCELLSP,yeePerProc,ZCELLSP + 1,MAXPOLESMU);
}
XCELLS = XCELLSP;
ZCELLS = ZCELLSP;
Appendix B

Genetic algorithm Debye series fitting code

% Genetic algorithm based Debye-series fitting (4pole) for mu/eps
% Random initial condition (GA is not dependent on this)
xo = [1.001023455843576e+001,...
2.77094027512e+001,...
2.71277867365e-009,...
2.99984604823153e+001,...
8.597467505694346e-008,...
2.993848406191331e+001,...
2.813650593172032e-008,...
5.716714775731167e-008];

gaopts = gaoptimset(@ga);
% Four iterations of GA
for i = 1 : 4
    gaopts = gaoptimset(gaopts,'PopInitRange',[-50 ; 100],...
        'Generations',4e4,'StallGenLimit',3e4,'MutationFcn',...
        @mutationadaptfeasible,'InitialPopulation',xo,...
        'CrossoverFcn',@crossoverheuristic,'PlotFcns',@gaplotbestf,'TolFun',1e-8);
    % The first set is the minimum and the second set maximum
    [x fval] = ga(@gafitfunc,9,[],[],[],...
        [0 0 3e-11 0 3e-11 0 3e-11 ],...
        [200 100 1e-5 100 1e-5 100 1e-5 ],[],gaopts);
x'
    xo = x;
end

function [fitval] = gafitfunc(x)
A = x(1);de1 = x(2);t1 = x(3);
de2 = x(4);t2 = x(5);
de3 = x(6);t3 = x(7);
de4 = x(8);t4 = x(9);
de5 = x(10);t5 = x(11);
f = linspace(8.25e9,12.35e9,3e3);
w = 2*pi*f;
epsr = GetEpsr(f/1e9);
epsrm = A + de1./((1+sqrt(-1)*w*t1) + de2./((1+sqrt(-1)*w*t2) + de3./((1+sqrt(-1)*w*t3) + ...)
    ./(1+sqrt(-1)*w*t4) + de5./((1+sqrt(-1)*w*t5));
fitval = sum(abs(epsr - epsrm).^2);

function epsr = GetEpsr(f/1e9);
function [epsr] = GetEpsr(freq)
% Get the measured data at frequency 'freq'

% Measured data and frequency points
global fr;
global mf14eps;

epsr = [];
epsr = zeros(1,length(freq));
epsr = interp1(fr,mf14eps,freq,'cubic');
Appendix C

Periodic surface reflectivity using Geometric Optics (GO)

function \[Pi,Pr\] = GO(fr,t,eps1,mu1,tr)

% fr - Frequency (Hz)
% t  - Coating thickness (m)
% eps1, mu1 - Relative, imaginary part > 0
% base - Base dimension
% asp_rat - Aspect ratio
% h  - Height
% ds - Ray incremental
% dx - Source plane discretization interval
% SP - Source plane
% Pi - Incident power

% 0.5 inch base
base = 0.5*2.54e-2;
asp_rat = 4;
h = asp_rat * base;
ds = base / 300;
dx = base / 200;
SP = (asp_rat+1)*base;
ko = 2*pi*fr*sqrt(pi*4e7*8.85e-12);
etao = sqrt(pi*4e7/8.85e-12);
Pi = 0;
m = t*sqrt(1+(2*h/base)^2);

% inipos - Initial position of the ray,
% inipos(row,col,1) = xini of the ray (row,col)
% inipos(row,col,2) = yini of the ray (row,col)
% If the ray is hitting the edge the values will be 'nan'
% ray_mode(row,col) --> 1 (TE) , ray_mode(row,col) --> 0 (TM)
% ray_mode(row,col) --> nan (ray hitting edge)
cplx_amp = ones(numel([dx : dx : base]),numel([dx : dx : base]));
ray_mode = zeros(numel([dx : dx : base]),numel([dx : dx : base]));
inipos = zeros(numel([dx : dx : base]),numel([dx : dx : base]),2);
finpos = zeros(numel([dx : dx : base]),numel([dx : dx : base]),3);
S_ave = zeros(numel([dx : dx : base]),numel([dx : dx : base]));

row = 1;
col = 1;

for xini = [dx : dx : base]
    row = 1;
    for yini = [dx : dx : base]
        Pi = Pi + (1/2/etao)*dx*dx;
        % Initial position of the ray
        inipos(row,col,1) = xini;
        inipos(row,col,2) = yini;
    end
end
\[ z, r \] = SurfaceHeightPyr(xini, yini, base, asp_rat, t, tr);
% ray_mode and cplx_amp
if r == 1 || r == 2
    \% TE
    ray_mode(row, col) = 1;
    cplx_amp(row, col) = 1;
elseif r == 3 || r == 4 || r == 5
    \% TM
    ray_mode(row, col) = 0;
    cplx_amp(row, col) = 1/etao;
end

\% ray_pts - Points for plotting
\% ray_pos - Current position of ray
\% s - Unit vector along propagation direction
ray_pts = [xini; yini; SP];
ray_pos = [xini; yini; SP];
s = [0; 0; -1];
cnt = 1;
while 1
    ray_pos_p = ray_pos;
    ray_pos = ray_pos + ds*s;
cplx_amp(row, col) = cplx_amp(row, col) * exp(-1i*ko*ds);
if ray_pos(3) > SP
    \% Ray has reached SP
    ray_pos = 0.5*(ray_pos_p + ray_pos);
    ray_pts = [ray_pts ray_pos];
cplx_amp(row, col) = cplx_amp(row, col) * exp(1i*0.5*ko*ds);
finpos(row, col, 1) = ray_pos(1);
finpos(row, col, 2) = ray_pos(2);
finpos(row, col, 3) = ray_pos(3);
if ray_mode(row, col) == 1
    \% TE
    S_ave(row, col) = (0.5/etao)*(abs(cplx_amp(row, col))^2)*dot(s, [0; 0; 1]);
elseif ray_mode(row, col) == 0
    \% TM
    S_ave(row, col) = (0.5*etao)*(abs(cplx_amp(row, col))^2)*dot(s, [0; 0; 1]);
end
    break;
else
    [zs, r] = SurfaceHeightPyr(ray_pos(1), ray_pos(2), base, asp_rat, t, tr);
if ray_pos(3) < zs
    ray_pos = 0.5*(ray_pos_p + ray_pos);
cplx_amp(row, col) = cplx_amp(row, col) * exp(1i*0.5*ko*ds);
    ray_pts = [ray_pts ray_pos];
[n, r] = SurfaceNormalPyr(ray_pos(1), ray_pos(2), base, asp_rat, t, tr);
if ray_mode(row, col) == 1
    \% TE
    if r == 5
    \[R\] = TE_singlelayer_PECback(fr, eps1, mu1, t*10, acos(dot(-s, n)));
    else
    \[R\] = TE_singlelayer_PECback(fr, eps1, mu1, t, acos(dot(-s, n)));
    end
    cplx_amp(row, col) = cplx_amp(row, col) * R;
else
    ray_mode(row, col) == 0
    \% TM
    if r == 5
    \[R\] = TM_singlelayer_PECback(fr, eps1, mu1, t*10, acos(dot(-s, n)));
    else
    \[R\] = TM_singlelayer_PECback(fr, eps1, mu1, t, acos(dot(-s, n)));
    end
    cplx_amp(row, col) = cplx_amp(row, col) * R;
end
re = 2*n*dot(-s, n) + s;
re = re/norm(re);
s = re;
if ray_pos(3) < 0
    disp('Something is wrong');
    ray_pos(3);
end

cnt = cnt + 1;
if cnt > 2e4
    disp('Ray taking too long to come back to SP');
    break;
end

plot3(ray_pts(1,2:size(ray_pts,2)),ray_pts(2,2:size(ray_pts,2)),ray_pts(3,2:size(ray_pts,2)),'r');
hold on;
plot3(ray_pts(1,1:2),ray_pts(2,1:2),ray_pts(3,1:2),'g');
grid on;
row = row + 1;
col = col + 1;
end
row = 1;col = 1;
S_ave_tot = 0;
for xini = [dx : dx : base]
    row = 1;
    for yini = [dx : dx : base]
        if(~isnan(S_ave(row,col)))
            S_ave_tot = S_ave_tot + S_ave(row,col);
        end
        row = row + 1;
    end
    col = col + 1;
end
Pr = S_ave_tot * dx * dx;

function [z,r] = SurfaceHeightPyr(x,y,base,asp_rat,t,tr)
% Surface height z = f(x,y)

% Transform x,y to the unit cell
if x > base || x < 0
    if x > 0
        x = x - floor(x/base)*base;
    else
        x = base - (abs(x) - floor(abs(x)/base)*base);
    end
end
if y > base || y < 0
    if y > 0
        y = y - floor(y/base)*base;
    else
        y = base - (abs(y) - floor(abs(y)/base)*base);
    end
end

h = asp_rat * base;
m = t*sqrt((2*asp_rat)^2);
p = (tr/100)*(h*m);
b = p/2/asp_rat;

if (x > (0.5*base - b)) && (x < (0.5*base + b)) && ... 
(y > (0.5*base - b)) && (y < (0.5*base + b))
    % region 5 - Top face
    z = h + m - p;
    r = 5;
elseif (x < 0.5*base) && (y >= x) && (y <= (-x + base))
% Region 1
z = 2*h*x/base + m;
r = 1;
elseif (x > 0.5*base) && (y <= x) && (y >= (-x + base))
% Region 2
z = 2*h*(base - x)/base + m;
r = 2;
elseif (y < 0.5*base) && (x >= y) && (x <= base - y)
% Region 3
z = 2*h*y/base + m;
r = 3;
elseif (y > 0.5*base) && (x >= base - y) && (x <= y)
% Region 4
z = 2*h*(base - y)/base + m;
r = 4;
end

******************************************************************************
function [n,r] = SurfaceNormalPyr(x,y,base,asp_rat,t,tr)
% Surface normal : Depends on the facet
if x > base || x < 0
  if x > 0
    x = x - floor(x/base)*base;
  else
    x = base - (abs(x) - floor(abs(x)/base)*base);
  end
end
if y > base || y < 0
  if y > 0
    y = y - floor(y/base)*base;
  else
    y = base - (abs(y) - floor(abs(y)/base)*base);
  end
end
h = asp_rat * base;
mag_n = sqrt(1 + (base^2)/4/(h^2));
m = t*sqrt(1+(2*asp_rat)^2);
p = (tr/100)*(h+m);
b = p/2/asp_rat;

if (x > (0.5*base - b)) && (x < (0.5*base + b)) && ...
  (y > (0.5*base - b)) && (y < (0.5*base + b))
% Region 5 - Top face
  n = [0;0;1];
r = 5;
elseif (x < 0.5*base) && (y >= x) && (y <= (-x + base))
% Region 1
  n = [-1;0;base/2/h]./mag_n;
r = 1;
elseif (x > 0.5*base) && (y <= x) && (y >= (-x + base))
% Region 2
  n = [1;0;base/2/h]./mag_n;
r = 2;
elseif (y < 0.5*base) && (x >= y) && (x <= base - y)
% Region 3
  n = [0;-1;base/2/h]./mag_n;
r = 3;
elseif (y > 0.5*base) && (x >= base - y) && (x <= y)
% Region 4
  n = [0;1;base/2/h]./mag_n;
r = 4;
end

******************************************************************************
function [R] = TE_singlelayer_PECback(f,eps1,mu1,d,th)
muo = 4*pi*1e-7;
epso = 8.85e-12;
ko = 2*pi*f*sqrt(muo*epso);
k1 = 2*pi*f*sqrt(muo*epso*eps1*mu1);
kx = ko*sin(th);
kz = sqrt(ko^2 - kx.^2);
kz1 = sqrt(kx^2 - kz.^2);
m = -1i*tan(kz1*d).*((kz*mu1*muo)./(kz1*muo));
R = (m - 1)./(m + 1);

function [R] = TM_singlelayer_PECback(f,eps1,mu1,d,th)

muo = 4*pi*1e-7;
epso = 8.85e-12;
ko = 2*pi*f*sqrt(muo*epso);
k1 = 2*pi*f*sqrt(muo*epso*eps1*mu1);

% Complex angle of refraction in thin layer
th1 = asin(ko * sin(th) ./ k1);
R1 = exp(-2*1i*d*k1*cos(th1));

m = ((R1 + 1)./(R1 - 1)).*(ko*epso*eps1*cos(th))./(k1*epso*cos(th1));
R = (m - 1)./(m + 1);
Appendix D

Truncated pyramid geometry

Figure D.1: Truncated pyramid geometry

$$m = t \sqrt{1 + \left(\frac{2H}{\Lambda}\right)^2}$$

$$x = (H + m) \frac{\Lambda}{2H} - \frac{\Lambda}{2}$$

$$p = \frac{tr}{100} (H + m) \ ; \ tr \text{ is the truncation percentage}$$

$$b = \frac{p\Lambda}{2H}$$
In order to fit a spherical cap on a truncated pyramid, the slope of the tangent of the circle has to be equal to the slope of the face of the pyramid (see figure below). $\theta_c$ is fixed by the aspect ratio of the pyramid. The second degree of freedom is either the truncation percentage $tr$ or radius of curvature of the spherical cap. If $tr$ is given, then $r_c$ can be calculated. On the other hand if $r_c$ is given, $tr$ can be calculated.

\[ \theta_c = \cot^{-1} \left( \frac{2H}{\Lambda} \right) \]

\[ r_c = b \cdot \sec(\theta_c) \]  \hspace{1cm} (D.2)

For the case of 1:4 pyramid and $r_c = t$, $\theta_c = 7.125^\circ$. $b = 0.9923t$, $p = 7.9382t$.

\[ tr = \frac{793.82t}{H + \sqrt{65t}} \]  \hspace{1cm} (D.3)

For $t = 2$ mm, $tr = 23.7228\%$.

For spherical cap truncation, the following two conclusions can be reached.

1) If you want a “good spherical top” (not a one which is nearly flat), the truncation percentage has to be significant, resulting in high reflectivity. For example, as derived above if the radius of

\[ b = 0.9923t, p = 7.9382t. \]

\[ tr = \frac{793.82t}{H + \sqrt{65t}} \]  \hspace{1cm} (D.3)

For $t = 2$ mm, $tr = 23.7228\%$.

For spherical cap truncation, the following two conclusions can be reached.

1) If you want a “good spherical top” (not a one which is nearly flat), the truncation percentage has to be significant, resulting in high reflectivity. For example, as derived above if the radius of
curvature of the spherical cap is $r_c = t = 2$ mm, the truncation percentage for 1:4 pyramid is given by $tr = 23.7228 \%$. Truncation percentage of $tr = 23.7228 \%$ means most of the tip coating is removed.

2) If the truncation percentage is not large enough, the radius of curvature $r_c$ will be large and it will be nearly a flat truncated top. The reflectivity spectrum of flat truncated square pyramids with various truncation ratios are given in chapter 3.
Appendix E

Laguerre MoD code

//C++ code for Laguerre coefficient generation and post processing
//Run before and after the MATLAB main code

#ifndef __AUXFUNCS_H__
#define __AUXFUNCS_H__

struct GaussianPulse {
    double to;
    double tau;
};

double GaussianPulseVal(GaussianPulse* pulse,double t);
double DebyeRelaxation(double* data,int len,double t);

#endif

#include "AuxFuncs.h"
#include <cmath>

double GaussianPulseVal(GaussianPulse* pulse,double t) {
    //return exp(-pow((t-pulse->to)/pulse->tau,2));
    return exp(-pow((t-pulse->to)/pulse->tau,2))*(-1/pulse->tau/pulse->tau)*2*(t-pulse->to);
}

double DebyeRelaxation(double data[],int len,double t) {
    int p = (int)((len - 1)/2);
double epsinf = data[0];
double sum = 0;
for(int pole = 0;pole < p;pole++) {
    sum += ((data[2*pole+1] - epsinf)/data[2*pole+2])*exp(-t/data[2*pole+2]);
}
    return sum;
}

#include <stdio.h>
#include <cmath>
#include <iostream>
#include <fstream>
#include <gsl/gsl_sf_laguerre.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_const.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_permutation.h>
#include <gsl/gsl_linalg.h>
#include "AuxFuncs.h"

using namespace std;

double LaguerreBasis(int n,double st);

//Fundamental constants
double pi = 3.14159;
double c = 2.99792458e8;
double muo = 4*pi*1e-7;
double epso = 8.85e-12;
double etao = sqrt(muo/epso);

int main() {
    //BW : sqrt(2.3)/pi/tau
    //HWTM (Half Width Tenth Maximum) : 1.517*tau
    //to : Time shift of the Gaussian pulse
    //Tf : Simulation stop time
    //M : Total number of Laguerre basis functions
    double fc = 1000e9;
double fBW = 100e9;
double tau = sqrt(2.3)/pi/fBW;
double to = 4*tau;
double Tf = to + 180*1.517*tau;
double dx = c/(fBW)/120;
double t = 0;
    //s x Tf = 9 x M
    double s = 1*M/Tf;
    //double debdata[] = {2,3,1e-10,3.8,1e-11};
    cout.precision(20);
    cout<<"M = "<<M<<endl;
    cout<<"Tf = "<<Tf<<endl;
    cout<<"s = "<<s<<endl;
    cout<<"dx = "<<dx<<endl;

    int IL = 20;
    double phi = 60*pi/180;
    //Source
    //TF/SF boundary is located at 'iL'th column Pz
    //x = 0 is at the PEC behind the UPML
    //So TF/SF boundary is at (iL - 0.5)*dx from x = 0
    //We need Pz_{iL-1,j}
    GaussianPulse PzPulse,QyPulse,PzPulse_1;
PzPulse.to = to + (IL - 1.5)*dx*cos(phi)/c;
PzPulse.tau = tau;
PzPulse_1.to = to + (IL - 0.5)*dx*cos(phi)/c;
PzPulse_1.tau = tau;
    QyPulse.to = to + (IL - 1)*dx*cos(phi)/c;
    QyPulse.tau = tau;
gsl_matrix* Pz_inc_p = gsl_matrix_calloc(M,1);
gsl_matrix* Pz_inc_p1 = gsl_matrix_calloc(M,1);
gsl_matrix* Qy_inc_p = gsl_matrix_calloc(M,1);
gsl_matrix* debCoeffs = gsl_matrix_calloc(M,1);
    int panels = 1e4;
double dst = s*Tf/panels;
    double st1,st2,st3;
double csum,csum1,csum2,csum3;
    //Calculate Pz_inc_p, Qy_inc_p
    for(int n = 0;n < M; n++) {
        csum = 0;csum1 = 0;csum2 = 0;csum3 = 0;
        for(int p = 0;p < panels/2;p++) {
            st1 = 2*p*dst;
    
}
st2 = (2*p+1)*dst;  
st3 = (2*p+2)*dst;  
  //Simpson's quadrature  
csum += dst*(GaussianPulseVal(&PzPulse,st1/s)*LaguerreBasis(n,st1) +  
  4*GaussianPulseVal(&PzPulse,st2/s)*LaguerreBasis(n,st2) +  
  GaussianPulseVal(&PzPulse,st3/s)*LaguerreBasis(n,st3))/3;   
csum1 += -cos(phi)*dst*(GaussianPulseVal(&QyPulse,st1/s)*LaguerreBasis(n,st1) +  
  4*GaussianPulseVal(&QyPulse,st2/s)*LaguerreBasis(n,st2) +  
  GaussianPulseVal(&QyPulse,st3/s)*LaguerreBasis(n,st3))/3;  
csum2 += dst*(GaussianPulseVal(&PzPulse_1,st1/s)*LaguerreBasis(n,st1) +  
  4*GaussianPulseVal(&PzPulse_1,st2/s)*LaguerreBasis(n,st2) +  
  GaussianPulseVal(&PzPulse_1,st3/s)*LaguerreBasis(n,st3))/3;  
  /*csum3 += dst*(DebyeRelaxation(debdata,5,st1/s)*LaguerreBasis(n,st1) +  
  4*DebyeRelaxation(debdata,5,st2/s)*LaguerreBasis(n,st2) +  
  DebyeRelaxation(debdata,5,st3/s)*LaguerreBasis(n,st3))/3;*/  
}

gsl_matrix_set(Pz_inc_p,n,0,csum);  
gsl_matrix_set(Pz_inc_p1,n,0,csum2);  
gsl_matrix_set(Qy_inc_p,n,0,csum1);  
  //gsl_matrix_set(debCoeffs,n,0,csum3);  
}

ofstream Pzq;  
ofstream Qyq;  
ofstream Pzl;  
ofstream debq;  
Pzq.open("Pzq.dat",ios::out);  
Pzl.open("Pzq.dat",ios::out);  
Qyq.open("Qyq.dat",ios::out);  
Pzq.precision(25);  
Pzl.precision(25);  
Qyq.precision(25);  
//debq.open("debq.dat",ios::out);  
for(int n = 0;n < M;n++)  
  {
    Pzq<<gsl_matrix_get(Pz_inc_p,n,0)<<endl;  
Pzl<<gsl_matrix_get(Pz_inc_p1,n,0)<<endl;  
    Qyq<<gsl_matrix_get(Qy_inc_p,n,0)<<endl;  
    //debq<<gsl_matrix_get(debCoeffs,n,0)<<endl;  
  }

Pzq.close();  
Pzl.close();  
Qyq.close();  
//debq.close();  

double* coeffs = new double[(int)M];  
FILE* f;  
f= fopen("Pzmatlabout.dat","r");  
char a[100];  
for(int n = 0;n < M;n++)  
  {  
    fscanf(f,"%s \n",a);  
    coeffs[n] = atof(a);  
  }
fclose(f);  

ofstream Pzcppout;  
Pzcppout.open("Pzcppout.dat",ios::out);  
Pzcppout.precision(20);  
for(double time = 0;time < Tf;time += Tf/3e4)  
  {  
    double val = 0;  
    for(int n = 0;n < M;n++)  
      {  
        val += coeffs[n]*LaguerreBasis(n,s*time);  
      }  
    //Pz<<GaussianPulseVal(&PzPulse,time)<<endl;  
Pzcppout<<val<<endl;  
  }  
Pzcppout.close();
MATLAB Laguerre MoD main code

\[ \phi_p(st) = \exp(-0.5*st)*L_p(st) \]

\[
\text{double LaguerreBasis(int n, double st) }
\text{ }
\{
\text{ return gsl_sf_laguerre_n(n,0,st) * exp(-0.5*st); }
\}
\]

---

\[ \text{pi} = 3.14159; \]
\[ \text{c} = 2.99792458e8; \]
\[ \text{muo} = 4*pi*1e-7; \]
\[ \text{epso} = 8.85e-12; \]
\[ \text{etao} = \sqrt{\text{muo/epso}}; \]

\[ \%BW : \text{sqrt}(2.3)/\text{pi}/\text{tau} \]
\[ \%HWTM (Half Width Tenth Maximum) : 1.517*\text{tau} \]
\[ \%to : \text{Time shift of the Gaussian pulse} \]
\[ \%Tf : \text{Simulation stop time} \]
\[ \%M : \text{Total number of Laguerre basis functions} \]
\[ \% fc = 500e9; \]
\[ \text{fBW} = 100e9; \]
\[ \text{tau} = \text{sqrt}(2.3)/\text{pi}/\text{fBW}; \]
\[ \text{to} = 4*\text{tau}; \]
\[ \text{Tf} = \text{to} + 180*1.517*\text{tau}; \]
\[ \text{M} = \text{ceil}(2*(\text{fBW})*\text{Tf} + 1) + 80; \]
\[ \text{dx} = \text{c}/(\text{fBW})/120; \]
\[ \text{s} = 1*\text{M}/\text{Tf}; \]

\[ \% \text{Materials} \]
\[ \text{VACUUM} = 0; \]
\[ \text{UPML} = 1; \]
\[ \text{MED1} = 3; \]
\[ \text{PEC} = 2; \]
\[ \text{VacuumEpsDebye} = [1]; \]
\[ \text{Med1EpsDebye} = [3,4.7e-8,4.5,2e-10]; \]
\[ \% \text{Med1EpsDebye} = [3]; \]
\[ \text{PL} = 47; \]
\[ \text{XCELLS_PS} = 200; \]
\[ \text{XCELLS} = \text{XCELLS_PS} + 2*\text{PL}; \]
\[ \text{YCELLS} = 20; \]
\[ \text{IL} = 30; \]

\text{load Pzq.dat; load Qyq.dat; load Pzq1.dat;}
\text{Pzinc = Pzq;}
\text{Qyinc = Qyq;}
\text{Pzinc1 = Pzq1; clear Pzq; clear Qyq; clear Pzq1;}
\text{phi = 60*pi/180;}
\text{A = zeros(XCELLS*YCELLS,XCELLS*YCELLS); Media = zeros(YCELLS,XCELLS); Sigma = zeros(YCELLS,XCELLS); Pz = zeros(YCELLS,XCELLS,M); Qy = zeros(YCELLS,XCELLS+1,M); Qx = zeros(YCELLS,XCELLS,M); Media(:,1:PL) = UPML; Media(:,XCELLS_PS+PL+1:YCELLS) = UPML; \% Media(:,35:135) = MED1;
\% Media(:,66:70) = PEC;

for row = 1 : YCELLS
    for col = 1 : XCELLS
        if Media(row,col) == PEC
            Sigma(row,col) = 1e8;
        else
            Sigma(row,col) = 0;
        end
    end
end

m = 3;
sigmo = -\log(\exp(-16))*(m+1)/(2*120*pi*PL*dx);
d = PL*dx;
SigCon = sigmo/(d^m);

for row = 1 : YCELLS
    for col = 1 : XCELLS
        if col <= PL + 1
            A6 = 2*c/s/dx;
            A5 = 2*c/s/dx;
        else
            A6 = (2*c*epso)/(s*epso*dx + 2*dx*SigCon*(((PL - col + 0.5)*dx)^m));
            A5 = (2*c*epso)/(s*epso*dx + 2*dx*SigCon*(((PL - col)*dx)^m));
        end
        A((col-1)*YCELLS + row,(col)*YCELLS + row) = A6*A5;
        % A6|i,j
        % A5|i+1,j
        if col == PL + 1
            A6 = 2*c/s/dx;
        end
        if col == PL + 1
            A1 = -2*c/s/dx + 0.5*sin(phi);
            A2 = 2*c/s/dx + 0.5*sin(phi);
        else
            A1 = -2*c/s/dx - 4*c*SigCon*(((PL - col + 0.5)*dx)^m)/s/s/epso/dx + 0.5*sin(phi) + SigCon*(((PL - col + 0.5)*dx)^m)*sin(phi)/s/epso;
            A2 = 2*c/s/dx + 4*c*SigCon*(((PL - col + 0.5)*dx)^m)/s/s/epso/dx + 0.5*sin(phi) + SigCon*(((PL - col + 0.5)*dx)^m)*sin(phi)/s/epso;
        end
        A((col-1)*YCELLS + row,(col-1)*YCELLS + row) = -A6*A5 - A6*A5_1 + A7*A2 + A8*A1 - 1;
    end
end

if col == 1
    A((col-1)*YCELLS + row,(col-2)*YCELLS + row) = A6*A5_1;
end

if row == YCELLS
    A((col-1)*YCELLS + row,(col-1)*YCELLS + row + 1) = A7*A1;
else
    A((col-1)*YCELLS + row,(col-1)*YCELLS + 1) = A7*A1;
end

if row == 1
    A((col-1)*YCELLS + row,(col-1)*YCELLS + row - 1) = A8*A2;
else
    A((col-1)*YCELLS + row,(col-1)*YCELLS + YCELLS) = A8*A2;
end

elseif col == XCELS_PS + PL
    \% A6|i,j
    \% A5|i+1,j
    if col == XCELS_PS + PL
        A6 = 2*c/s/dx;
A5 = 2*c/s/dx;
else
    A6 = (2*c*epsilon)/(s*epsilon*dx + 2*dx*Sigma(((col - XCELLS_PS - PL)*dx)^m));
A5 = (2*c*epsilon)/(s*epsilon*dx + 2*dx*Sigma(((col - XCELLS_PS - PL)*dx)^m));
end

if col ~= XCELLS
    A((col-1)*YCELLS + row,(col)*YCELLS + row) = A6*A5;
end

% A(i,j)
if col ~= XCELLS_PS + PL
    A5_1 = 2*c/s/dx;
else
    A5_1 = (2*c*epsilon)/(s*epsilon*dx + 2*dx*Sigma(((col - XCELLS_PS - PL - 0.5)*dx)^m));
end
A7 = (s*sin(phi)*dx - 4*c)*dx*A6/4/c/dx;
A8 = (s*sin(phi)*dx + 4*c)*dx*A6/4/c/dx;
if col ~= XCELLS_PS + PL
    A1 = -2*c/s/dx + 0.5*sin(phi);
    A2 = 2*c/s/dx + 0.5*sin(phi);
else
    A1 = -2*c/s/dx - 4*c*Sigma(((col - XCELLS_PS - PL - 0.5)*dx)^m)/(s/s/epsilon/dx) + 0.5*sin(phi) + Sigma(((col - XCELLS_PS - PL - 0.5)*dx)^m)*sin(phi)/(s/epsilon);
    A2 = 2*c/s/dx + 4*c*Sigma(((col - XCELLS_PS - PL - 0.5)*dx)^m)/(s/s/epsilon/dx) + 0.5*sin(phi) + Sigma(((col - XCELLS_PS - PL - 0.5)*dx)^m)*sin(phi)/(s/epsilon);
end
A((col-1)*YCELLS + row,(col-1)*YCELLS + row) = -A6*A5 - A6*A5_1 + A7*A2 + A8*A1 - 1;
A((col-1)*YCELLS + row,(col-2)*YCELLS + row) = A6*A5_1;

if row ~= YCELLS
    A((col-1)*YCELLS + row,(col-1)*YCELLS + row + 1) = A7*A1;
else
    A((col-1)*YCELLS + row,(col-1)*YCELLS + row + 1) = A7*A1;
end

if row ~= 1
    A((col-1)*YCELLS + row,(col-1)*YCELLS + row - 1) = A8*A2;
else
    A((col-1)*YCELLS + row,(col-1)*YCELLS + YCELLS) = A8*A2;
end
else
if Media(row,col) == VACUUM
    K = 2*c/(DebyeMediaLaguerre(VacuumEpsDebye,s,0) + 2*c*etao*Sigma(row,col));
elseif Media(row,col) == MED1
    K = 2*c/(DebyeMediaLaguerre(Med1EpsDebye,s,0) + 2*c*etao*Sigma(row,col));
else
    K = 2*c/(s + 2*c*etao*Sigma(row,col));
end
A((col-1)*YCELLS + row,(col-1)*YCELLS + row) = -8*c*K/s/dx/dx - 1 + s*sin(phi)*sin(phi)*K/2/c;
A((col-1)*YCELLS + row,(col-2)*YCELLS + row) = 2*c*K/s/dx/dx;
A((col-1)*YCELLS + row,(col)*YCELLS + row) = 2*c*K/s/dx/dx;
if row ~= YCELLS
    A((col-1)*YCELLS + row,(col-1)*YCELLS + row + 1) = 2*c*K/s/dx/dx - sin(phi)*K/dx;
else
    A((col-1)*YCELLS + row,(col-1)*YCELLS + row + 1) = 2*c*K/s/dx/dx - sin(phi)*K/dx;
end
if row ~= 1
    A((col-1)*YCELLS + row,(col-1)*YCELLS + row - 1) = 2*c*K/s/dx/dx + sin(phi)*K/dx;
else
\begin{align*}
A((col-1)\cdot YCELLS + \text{row},(col-1)\cdot YCELLS + YCELLS) &= 2cK/s/dx/dx + \sin(\phi)K/dx; \\
\text{end} & \text{ end} \\
\text{end} & \text{ end} \\
Ainv &= \text{inv}(A); \\
b &= \text{zeros}(XCELLS\cdot YCELLS,1); \\
\text{for } q = 0 : 1 : M-1 \\
\text{q} & \text{ for } \text{row} = 1 : YCELLS \\
& \text{ for } \text{col} = 1 : XCELLS \\
brow &= ((\text{col}-1)\cdot YCELLS + \text{row}); \\
bsum &= 0; \\
\text{if } \text{col} <= \text{PL} + 1 \\
\text{A6} &= 2c/s/dx; \\
A5 &= 2c/s/dx; \\
B1 &= 0; \\
\text{else} \\
A6 &= 2c/\text{epsol}dx + 2dx\text{SigCon}((\text{PL} - \text{col} + 0.5)dx)^m); \\
A5 &= 2c/\text{epsol}dx + 2dx\text{SigCon}((\text{PL} - \text{col})dx)^m); \\
B1 &= 2\sin(\phi)\text{SigCon}((\text{PL} - \text{col} + 0.5)dx)^m)/s/\text{epsol}; \\
\text{end} \\
\text{A6,i,j} & \text{ A5,i+1,j} \\
\text{A5_1} &= 2c/\text{epsol}dx + 2dx\text{SigCon}((\text{PL} - \text{col} + 1)dx)^m); \\
A7 &= \text{epsol}\phi/dx + 4c/dx/\text{A6}/4/c/dx; \\
A8 &= \text{epsol}\phi/dx + 4c/dx/\text{A6}/4/c/dx; \\
\text{for } p = 0 : 1 : q - 1 \\
A3 &= B1 + 2\sin(\phi)(q-p) - 4c/s/dx; \\
A4 &= B1 + 2\sin(\phi)(q-p) + 4c/s/dx; \\
\text{if } \text{row} = \text{YCELLS} \\
bsum &= bsum + (sdx/c)\text{A6}\text{A5}\text{Qy(\text{row},\text{col}+1,p+1)} - (sdx/c)\text{A6}\text{A5_1}\text{Qy(\text{row},\text{col},p+1)} - (sdx/c)\text{A6}\text{A5}\text{A7}\text{Pz(\text{row},\text{col}+1,p+1)} + (sdx/c)\text{A6}\text{A5_1}\text{A7}\text{Pz(\text{row},\text{col},p+1)} + (sdx/c)\text{A6}\text{A5}\text{A8}\text{Qx(\text{row},\text{col},p+1)} - (sdx/c)\text{A6}\text{A5_1}\text{A8}\text{Qx(\text{row},\text{col},p+1)}; \\
\text{elseif } \text{row} = 1 \\
bsum &= bsum + (sdx/c)\text{A6}\text{A5}\text{Qy(\text{row},\text{col},p+1)} - (sdx/c)\text{A6}\text{A5_1}\text{Qy(\text{row},\text{col},p+1)} - (sdx/c)\text{A6}\text{A5}\text{A7}\text{Pz(\text{row},\text{col},p+1)} + (sdx/c)\text{A6}\text{A5_1}\text{A7}\text{Pz(\text{row},\text{col},p+1)} + (sdx/c)\text{A6}\text{A5}\text{A8}\text{Qx(\text{row},\text{col},p+1)} - (sdx/c)\text{A6}\text{A5_1}\text{A8}\text{Qx(\text{row},\text{col},p+1)}; \\
\text{else} \\
bsum &= bsum + (sdx/c)\text{A6}\text{A5}\text{Qy(\text{row},\text{col},p+1)} - (sdx/c)\text{A6}\text{A5_1}\text{Qy(\text{row},\text{col},p+1)} - (sdx/c)\text{A6}\text{A5}\text{A7}\text{Pz(\text{row},\text{col},p+1)} + (sdx/c)\text{A6}\text{A5_1}\text{A7}\text{Pz(\text{row},\text{col},p+1)} + (sdx/c)\text{A6}\text{A5}\text{A8}\text{Qx(\text{row},\text{col},p+1)} - (sdx/c)\text{A6}\text{A5_1}\text{A8}\text{Qx(\text{row},\text{col},p+1}); \\
\text{end} \\
b(brow,1) &= bsum; \\
\text{elseif } \text{col} = \text{XCELLS_PS} + \text{PL} \\
\text{A6} &= 2c/s/dx; \\
A5 &= 2c/s/dx; \\
\text{else} \\
\text{end} \\
\text{end} \\
\text{end}
A6 = \(\frac{(2c*\varepsilon_0)}{(s*\varepsilon_0*dx + 2*dx*\Sigma_{\text{Con}}*((col - XCELLS_{\text{PS}} - PL - 0.5)*dx)^m)}\);
A5 = \(\frac{(2c*\varepsilon_0)}{(s*\varepsilon_0*dx + 2*dx*\Sigma_{\text{Con}}*((col - XCELLS_{\text{PS}} - PL)*dx)^m)}\);

end

\% A5|i,j
if col == XCELLS_{\text{PS}} + PL
A5_1 = \frac{2c}{s/dx};
B1 = 0;
else
A5_1 = \(\frac{(2c*\varepsilon_0)}{(s*\varepsilon_0*dx + 2*dx*\Sigma_{\text{Con}}*((col - XCELLS_{\text{PS}} - PL - 1)*dx)^m)}\);
B1 = \(\frac{(2*\sin(phi)*\Sigma_{\text{Con}}*((col - XCELLS_{\text{PS}} - PL)*dx)^m)}{s/\varepsilon_0}\);
end
A7 = \(\frac{(s*\sin(phi)*dx - 4*c)*dx*A6/4/c/dx}\);
A8 = \(\frac{(s*\sin(phi)*dx + 4*c)*dx*A6/4/c/dx}\);

for p = 0 : 1 : q - 1
A3 = B1 + 2*\sin(phi)*(q-p) - 4*c/s/dx;
A4 = B1 + 2*\sin(phi)*(q-p) + 4*c/s/dx;
if row == YCELLS
bsum = bsum + (\(s*\frac{dx}{c}\)*A6*A5*Qy(row,col+1,p+1) - (\(s*\frac{dx}{c}\)*A6*A5_1*Qy(row,col,p+1) - A7*A3*Pz(1,col,p+1) - A7*A4 + A8*A3) + s*dx*A6/c) + (4*A7*(q - p) - \(s*\sin(phi)*dx*A6/2/c\)*Qz(row,1,col,p+1) + (4*A8*(q - p) - \(s*\sin(phi)*dx*A6/2/c\)*Qz(row,col,p+1));
else
bsum = bsum + (\(s*\frac{dx}{c}\)*A6*A5*Qy(row,col+1,p+1) - (\(s*\frac{dx}{c}\)*A6*A5_1*Qy(row,col,p+1) - A7*A3*Pz(row+1,col,p+1) - A7*A4 + A8*A3 - s*dx*A6/c) + (4*A7*(q - p) - \(s*\sin(phi)*dx*A6/2/c\)*Qz(row+1,col,p+1) + (4*A8*(q - p) - \(s*\sin(phi)*dx*A6/2/c\)*Qz(row,col,p+1));
end
end
b(brow,1) = bsum;
else
if Media(row,col) == VACUUM
K = \frac{2c}{s + 2c*\eta_0*\Sigma_{\text{row,col}}};
\varepsilon_{\text{r}_1} = s;
\varepsilon_{\text{r}_2} = s;
elseif Media(row,col) == MED1
K = \frac{2c}{\text{DebyeMediaLaguerre(MediEpsDebye,s,0) + 2c*\eta_0*\Sigma_{\text{row,col}})};
\varepsilon_{\text{r}_1} = \text{DebyeMediaLaguerre(MediEpsDebye,s,q-p)};
\varepsilon_{\text{r}_2} = \text{DebyeMediaLaguerre(MediEpsDebye,s,q-p-1)};
else
K = \frac{2c}{s + 2c*\eta_0*\Sigma_{\text{row,col}}};
\varepsilon_{\text{r}_1} = s;
\varepsilon_{\text{r}_2} = s;
end
for p = 0 : 1 : q - 1
if row == 1
bsum = bsum + (\(2*\frac{K}{dx}\)*(Qy(row,col+1,p+1)-Qy(row,col,p+1)) + (\(\sin(phi)*\frac{K}{dx}\)+Pz(row+1,col,p+1)-Pz(YCELLS_{\text{col}},col,p+1)) + (-2*\frac{K}{dx}) + (Qz(row+1,col,p+1)-Qz(row,col,p+1)) + (K/c)*\(0.5*\varepsilon_{\text{r}_1} + 0.5*\varepsilon_{\text{r}_2}\) - \(s*\sin(phi)*\sin(phi)\)*Pz(row,col,p+1));
else
bsum = bsum + (\(2*\frac{K}{dx}\)*(Qy(row,col+1,p+1)-Qy(row,col,p+1)) + (\(\sin(phi)*\frac{K}{dx}\)+Pz(1,col,p+1)-Pz(row-1,col,p+1)) + (-2*\frac{K}{dx}) + (Qz(1,col,p+1)-Qz(row,col,p+1)) + (K/c)*\(0.5*\varepsilon_{\text{r}_1} + 0.5*\varepsilon_{\text{r}_2}\) - \(s*\sin(phi)*\sin(phi)\)*Pz(row,col,p+1));
end
end
\[ bsum = bsum + (2K/dx)(Qy(row,col+1,p+1) - Qy(row,col,p+1)) + \\
(sin(phi)K/dx)(Pz(row+1,col+1,p+1) - Pz(row,col,p+1)) + \\
(Qx(row+1,col+1,p+1) - Qx(row,col,p+1)) + (K/c)(0.5*\epsilon_s_1 + 0.5*\epsilon_s_2 \\
- s*sin(phi)*sin(phi))Pz(row,col,p+1); \]

\%TF/SF

if col == IL
\[ bsum = bsum + (-2cK/s/dx/dx)*Pzinc(q+1,1); \]
for p = 0 : 1 : q - 1
\[ bsum = bsum + (2K/dx)*(-Qyinc(p+1,1)); \]
end
end
\[ b(brow,1) = bsum; \]
end
end
end
end

\[ x = Ainv*b; \]

for row = 1 : YCELLS
for col = 1 : XCELLS
\[ Pz(row,col,q+1) = x((col-1)*YCELLS + row,1); \]
if col == IL
\%Pz(row,col,q+1) = Pz(row,col,q+1) + Pzinc(q+1,1);
end
end
end

A1 = (s*dx*sin(phi) - 4c)/2/s/dx;
B = (s*dx*sin(phi) + 4c)/2/s/dx;
C = 2c/s/dx;
for row = 1 : YCELLS
for col = 1 : XCELLS
\[ bsum = 0; \]
if col <= PL + 1
if col == PL + 1
\[ A1u = -2c/s/dx + 0.5*sin(phi); \]
\[ A2 = 2c/s/dx + 0.5*sin(phi); \]
\[ B1 = 0; \]
else
\[ A1u = -2c/s/dx - 4c*SigCon*(((PL - col + 0.5)*dx)^m)/s/s/epso/dx \\
+ 0.5*sin(phi) + SigCon*(((PL - col + 0.5)*dx)^m)*sin(phi)/s/epso; \]
\[ A2 = 2c/s/dx + 4c*SigCon*(((PL - col + 0.5)*dx)^m)/s/s/epso/dx \\
+ 0.5*sin(phi) + SigCon*(((PL - col + 0.5)*dx)^m)*sin(phi)/s/epso; \]
\[ B1 = (2*sin(phi)*SigCon*(((PL - col + 0.5)*dx)^m))/s/epso; \]
end
if row == 1
\[ bsum = A1u*Pz(row,col,q+1) + A2*Pz(YCELLS,col,q+1); \]
else
\[ bsum = A1u*Pz(row,col,q+1) + A2*Pz(row-1,col,q+1); \]
end
for p = 0 : 1 : q - 1
\[ A3 = B1 + 2*sin(phi)*(q-p) - 4c/s/dx; \]
\[ A4 = B1 + 2*sin(phi)*(q-p) + 4c/s/dx; \]
if row == 1
\[ bsum = bsum + A3*Pz(row,col,p+1) + A4*Pz(YCELLS,col,p+1) \\
- 4*(q-p)*Qx(row,col,p+1); \]
else
\[ bsum = bsum + A3*Pz(row,col,p+1) + A4*Pz(row-1,col,p+1) \\
- 4*(q-p)*Qx(row,col,p+1); \]
end
end
\[ Qx(row,col,q+1) = bsum; \]
elseif col >= XCELLS_PS + PL
if col == XCELLS_PS + PL
    A1u = -2*c/s/dx + 0.5*sin(phi);
    A2 = 2*c/s/dx + 0.5*sin(phi);
    B1 = 0;
else
    A1u = -2*c/s/dx - 4*c*SigCon*(((col - XCELLS_PS - PL - 0.5)*dx)^m)/
         s/s/epso/dx + 0.5*sin(phi) + SigCon*(((col - XCELLS_PS - PL - 0.5)*dx)^m)
         *sin(phi)/s/epso;
    A2 = 2*c/s/dx + 4*c*SigCon*(((col - XCELLS_PS - PL - 0.5)*dx)^m)/
         s/s/epso/dx + 0.5*sin(phi) + SigCon*(((col - XCELLS_PS - PL - 0.5)*dx)^m)
         *sin(phi)/s/epso;
    B1 = (2*sin(phi)*SigCon*(((col - XCELLS_PS - PL - 0.5)*dx)^m))/s/epso;
end
if row == 1
    bsum = A1u*Pz(row,col,q+1) + A2*Pz(YCELLS,col,q+1);
else
    bsum = A1u*Pz(row,col,q+1) + A2*Pz(row-1,col,q+1);
end
for p = 0 : 1 : q - 1
    A3 = B1 + 2*sin(phi)*(q-p) - 4*c/s/dx;
    A4 = B1 + 2*sin(phi)*(q-p) + 4*c/s/dx;
    if row == 1
        bsum = bsum + A3*Pz(row,col,p+1) + A4*Pz(YCELLS,col,p+1)
        - 4*(q-p)*Qx(row,col,p+1);
    else
        bsum = bsum + A3*Pz(row,col,p+1) + A4*Pz(row-1,col,p+1)
        - 4*(q-p)*Qx(row,col,p+1);
    end
end
Qx(row,col,q+1) = bsum;
else
    bsum = (-2*c/s/dx + 0.5*sin(phi))*Pz(row,col,q+1);
    if row == 1
        bsum = bsum + (2*c/s/dx + 0.5*sin(phi))*Pz(row-1,col,q+1);
    else
        bsum = bsum + (2*c/s/dx + 0.5*sin(phi))*Pz(YCELLS,col,q+1);
    end
for p = 0 : 1 : q - 1
    if row == 1
        bsum = bsum + sin(phi)*Pz(row,col,p+1) + ...
        sin(phi)*Pz(row-1,col,p+1) - 2*Qx(row,col,p+1);
    else
        bsum = bsum + sin(phi)*Pz(row,col,p+1) + ...
        sin(phi)*Pz(YCELLS,col,p+1) - 2*Qx(row,col,p+1);
    end
end
Qx(row,col,q+1) = bsum;
end
end
end
for row = 1 : YCELLS
    for col = 1 : XCELLS + 1
        bsum = 0;
        if col == PL + 1
            A5 = (2*c*epso)/(s*epso*dx + 2*dx*SigCon*(((PL - col + 1)*dx)^m));
            if col == 1
                bsum = A5*Pz(row,col,q+1);
            else
                bsum = A5*Pz(row,col,q+1) - A5*Pz(row,col-1,q+1);
            end
        end
        for p = 0 : 1 : q - 1
            bsum = bsum + (s*dx/c)*A5*Qy(row,col,p+1);
end
Qy(row,col,q+1) = bsum;
elseif col >= XCELLS_PS + PL + 1
  \% A5|i,j
  A5 = \frac{(2*c*epso)/(s*epso*dx + 2*dx*SigCon*)((col - XCELLS_PS - PL - 1)*dx)^m)}{dx};
  if col == XCELLS + 1
    bsum = -A5*Pz(row,col-1,q+1);
  else
    bsum = A5*Pz(row,col,q+1) - A5*Pz(row,col-1,q+1);
  end
  for p = 0 : 1 : q - 1
    bsum = bsum - (s*dx/c)*A5*Qy(row,col,p+1);
  end
  Qy(row,col,q+1) = bsum;
else
  bsum = \frac{(2*c/s/dx)*Pz(row,col,q+1) - (2*c/s/dx)*Pz(row,col-1,q+1)}{dx};
  if col == IL
    bsum = bsum + (-2*c/s/dx)*Pzinc1(q+1,1);
  end
  for p = 0 : 1 : q - 1
    bsum = bsum - 2*Qy(row,col,p+1);
  end
  Qy(row,col,q+1) = bsum;
end
end
end
fid = fopen('Pzmatlabout.dat','w');
for idx = 0 : 1 : M - 1
  fprintf(fid,'%15.12f
',Pz(10,200,idx+1));
end
fclose(fid);
Appendix F

ADE (Auxiliary Differential Equation) formulation for $TM_z$ case (Debye media)

\[
E_{z|ij}^{n+1} = \left[ \frac{2\varepsilon_0 \varepsilon_{\infty} + \sum_{p=1}^{N} \beta_p - \sigma \Delta t}{2\varepsilon_0 \varepsilon_{\infty} + \sum_{p=1}^{N} \beta_p + \sigma \Delta t} \right] E_{z|ij}^{n} + \frac{2\Delta t}{2\varepsilon_0 \varepsilon_{\infty} + \sum_{p=1}^{N} \beta_p + \sigma \Delta t} \sum_{p=1}^{N} (1 + k_p) J_{z|ij}^{p,n}
\]

\[
\left[ \frac{H_{y|i+0.5,j} - H_{y|i,j}}{\Delta x} - \frac{H_{x|i,j+0.5} - H_{x|i+1,j}}{\Delta y} \right] - \frac{1}{2} \sum_{p=1}^{N} (1 + k_p) J_{z|ij}^{p,n}
\]

\[
J_{z|ij}^{p,n+1} = k_p J_{z|ij}^{p,n} + \beta_p \left[ \frac{E_{z|i,j+1}^{n+1} - E_{z|i,j}^{n}}{\Delta t} \right]
\]

\[
k_p = \frac{1 - \frac{\Delta t}{\tau_p}}{1 + \frac{\Delta t}{\tau_p}} \quad \beta_p = \frac{\varepsilon_0 \Delta \varepsilon_p \Delta t}{\tau_p (1 + \frac{\Delta t}{\tau_p})}
\]

\[
H_{x|i,j+1}^{n+0.5} = H_{x|i,j+1}^{n-0.5} - \frac{\Delta t}{\mu_0 \Delta y} \left[ E_{z|i,j+1}^{n} - E_{z|i,j}^{n} \right]
\]

\[
H_{y|i+1,j}^{n+0.5} = H_{y|i+1,j}^{n-0.5} + \frac{\Delta t}{\mu_0 \Delta x} \left[ E_{z|i+1,j}^{n} - E_{z|i,j}^{n} \right]
\]