FMPS solver

Version 1.0

FORTRAN

1 Introduction

This is the first release of the FMPS library, which solves the multiple electromagnetic scattering problem for a collection of arbitrary oriented dielectric inclusions.

Currently, the library contains two solvers.

Step 1. The first solver evaluates the electromagnetic multipole reflection coefficients from an arbitrary dielectric inclusion. This effectively reduces the number of degree of freedoms to represent the inclusion, especially if a complicated geometry is used in sub-wavelength regime. We use the Muller integral equation to evaluate the above coefficients for arbitrary incoming fields, which we expand in terms of local vector spherical harmonics up to a user-defined number of modes p on a sphere enclosing the inclusion. (We will refer to this expansion as an EM-multipole expansion.) The resulting reflection coefficients are stored in a large $(2\cdot(2p+1)\cdot(p+1))\times(2\cdot(2p+1)\cdot(p+1))$ complex valued matrix.

Note: This *scattering matrix* is frequency dependent, so that if a large number of frequencies are required, the precomputation step can be quite expensive. The Muller integral equation solver can use either flat or quadratic triangulations.

Step 2. The second solver uses the precomputed reflection matrices to evaluate multiple scattering from a collection of arbitrary oriented dielectric inclusions. The equation to be solved is

$$(a,b)_i^{outgoing} = R_i \left[(a,b)_i^{incident} + \sum_{j\neq i} T_{ij}(a,b)_j^{outgoing} \right],$$

where $(a,b)_i^{outgoing}$ is the outgoing EM-multipole expansion for the ith inclusion, $(a,b)_i^{incident}$ is the incoming EM-multipole expansion for the incident field, R_i is the reflection coefficient matrix, and T_{ij} is the translation operator converting the outgoing EM-multipole expansion due the jth inclusion into the incoming EM-multipole expansion on the sphere enclosing the ith inclusion.

For general geometries, the matrix R_i is computed in Step 1 using the Muller solver. If the inclusions are dielectric spheres, the coefficients have a much simple analytic form, and the reflection matrices can be formed much faster, which could be useful in some model situations. (The spheres do not have to be the same size.)

Once the coefficients of the outgoing EM-multipole expansions are found, we can evaluate the scattered and total E and H fields at user-defined targets. This allows us to evaluate other parameters of interest such as scattering, absorption and extinction cross sections, etc.

2 Detailed description of the codes

In both solvers, the external media parameters ε_0 and μ_0 are assumed to be normalized to 1, and dielectric inclusion parameters are always relative to the exterior.

2.1 Muller solver

The user has to provide the inclusion geometry in either "Cart3D" format (flat triangulations) or a Cart3D-like quadratic triangulation (see appendix for description of these formats); the wavelength of the incoming field; the real and imaginary parts of the refractive index; the radius of enclosing sphere and the number of terms for reflection matrix. Matlab scripts write a Fortran readable configuration file, after that, all processing is done by an external executable.

```
gold_jc_data;

%     Johnson-Christy gold parameters
```

```
gold_jc = [
       1.0200,1215.6863,0.3500,8.1450,
       1.0400,1192.3077,0.3325,7.9655,
       1.0600,1169.8113,0.3170,7.7914,
       ....
       4.4400, 279.2793,1.4531,1.8585,
       4.4600, 278.0269,1.4454,1.8545,
       4.4800, 276.7857,1.4357,1.8497];

%% do frequency sweep
for freq=12:12

wavelength=gold_jc(freq,2);
re_n=gold_jc(freq,3);   % real part of refractive index
im_n=gold_jc(freq,4);   % imaginary part of refractive index

geom_type = 3;   % for flat triangulation, set geom_type = 2
                 % for quadratic triangulation, set geom_type = 3
filename_geo = '2ellipsoids-25x25x75-sep40.q.tri';
scale_geo=[1, 1, 1];
shift_geo=[0, 0, 0];
radius=50;    % radius of enclosing sphere
nterms=6;     % maximum number of terms in reflection coefficient matrix

solver_type=2;       % use GMRES iterative algorithm
eps=1d-8;            % stopping criterion for GMRES
numit=40;            % maximum number of iterations

filename_out = ['scat.freq.' num2str(freq)];    % name of the output file

%[ write the configuration file and do all processing ]

config = ['config.freq.' num2str(freq)];

fid = fopen(config,'w');

fprintf(fid,'%d\n',geom_type);
fprintf(fid,'%s\n',filename_geo);
fprintf(fid,'%g %g %g %g %g %g\n', scale_geo(1:3)', shift_geo(1:3)');
fprintf(fid,'%g\n', radius);
fprintf(fid,'%g %g %g\n', wavelength, re_n, im_n);
fprintf(fid,'%d\n', nterms);
fprintf(fid,'%d %g %d\n', solver_type,eps,numit);
fprintf(fid,'%s\n',filename_out);

fclose(fid);

system(['int2_w32.exe ' config]);     % windows 32 bit executable
%%%system(['int2_w64.exe ' config]);     % windows 64 bit executable
%%%system(['int2_lnx64_openmp ' config]);   % linux 64 bit executable
```

```
end
```

2.2 FMPS solver

The solver mainly uses Matlab to setup geometry, material parameters, incoming fields, and post-process the solution. The GMRES solver for the main scattering equation heavily uses external Fortran MEX files in order to accelerate all computations. All length units are in nanometers.

2.2.1 Solver setup

```
nterms=3;   % In this section, this is the only user-defined
            % parameter, used to control accuracy of simulation

itype=1;
nquad=nterms;
nphi=2*nquad+1;
ntheta=nquad+1;

A.nterms = nterms;
A.nquad = nquad;
A.nphi = nphi;
A.ntheta = ntheta;

[A.rnodes,A.weights,A.nnodes]=e3fgrid(itype,nquad,nphi,ntheta);
```

2.2.2 Geometry setup

```
nspheres=4;

center = zeros(3,nspheres);
center(1:3,1) = [0,0,0];
center(1:3,2) = [0,200,0];
center(1:3,3) = [200,0,0];
center(1:3,4) = [200,200,0];

radius = zeros(1,nspheres);
radius(1) = 50;
radius(2) = 50;
radius(3) = 50;
radius(4) = 50;

sphere_type = zeros(1,nspheres);
for i=1:nspheres
  sphere_type(1,i) = 3;
end

sphere_eps = zeros(1,nspheres)+1i*zeros(1,nspheres);
sphere_cmu = zeros(1,nspheres)+1i*zeros(1,nspheres);

for i=1:nspheres
  sphere_eps(1,i) = 1;
  sphere_cmu(1,i) = 1;
```

```
end

sphere_rot = zeros(3,nspheres);
for i=1:nspheres
    sphere_rot(1:3,i)=[0,0,0];
end
```

2.2.3 Inclusion rotation parameters

Currently, inclusion orientation is specified via Euler angles, see test3.m for examples.

2.2.4 External media parameters

```
gold_jc_data;

%     Johnson-Christy gold parameters
gold_jc = [
       1.0200,1215.6863,0.3500,8.1450,
       1.0400,1192.3077,0.3325,7.9655,
       1.0600,1169.8113,0.3170,7.7914,
       ....
       4.4400, 279.2793,1.4531,1.8585,
       4.4600, 278.0269,1.4454,1.8545,
       4.4800, 276.7857,1.4357,1.8497];

ifreq = 61;

wavelength=gold_jc(ifreq,2)
omega=2*pi/wavelength
eps0=1;
cmu0=1;
```

2.2.5 Incoming field

Incoming plane wave can be specified by arbitrary k propagation and E polarization vectors.

```
zk=omega*sqrt(eps0*cmu0);

ima = 1i;
kvec = zk*[0 0 -1];    % k vector
epol = [1 0 0];        % E polarization direction


%
%  Initialize the incoming field
%

%
%  Grab E and H fields on all spheres
%
nnodes = A.nnodes;
nphi = A.nphi;
```

```
ntheta = A.ntheta;

evecs = zeros(3,nnodes,nspheres) + 1i*zeros(3,nnodes,nspheres);
hvecs = zeros(3,nnodes,nspheres) + 1i*zeros(3,nnodes,nspheres);

ima = 1i;
kvec = zk*[0 0 -1];
epol = [1 0 0];


for j=1:nspheres

ntargets = nphi*ntheta;
targets = repmat(center(:,j),1,nnodes)+A.rnodes*radius(j);

[evecs(:,:,j),hvecs(:,:,j)] = ...
    em3d_planearb_targeval(kvec,epol,ntargets,targets);

end
```

2.2.6 Reflection matrix

The reflection matrix is retrieved from a file and converted into a Matlab matrix. Currently, the code can handle just one type on inclusion.

```
%%%ifreq = 61;
scatfile = ['data' filesep 'scat.2ellipsoids-25x25x75-sep40-gold-draft' filesep ...
            'scat.freq.' num2str(ifreq)]
copyfile(scatfile,'fort.19');

%
%  Retrieve reflection matrix from 'fort.19'
%
[A.raa,A.rab,A.rba,A.rbb,A.nterms_r,ier]=rmatr_file('fort.19');
```

2.2.7 The FMPS solver

Now, we are ready to solve the main equation. We use the GMRES iterative algorithm to find the coefficients of outgoing EM-multipole expansions due to each enclosing sphere. For non-overlaping spheres, only few iterations are needed.

```
%
%  Construct the right hand side
%
[arhs,brhs]=em3d_multa_r(center,radius,aimpole,bimpole,nterms,A);

%
%  Call the solver
%
'FMPS solver for the Maxwell equation in R^3, Matlab + Fortran 90'
tic
```

```
rhs = reshape([arhs brhs],A.ncoefs*A.nspheres*2, 1);
sol = gmres_simple(@(x) em3d_multa_fmps(A,x), rhs, 1e-3, 20);

sol0 = reshape(sol,A.ncoefs,A.nspheres,2);
aompole = sol0(:,:,1);
bompole = sol0(:,:,2);

time_gmres=toc

[asmpole,bsmpole] = em3d_multa_mptaf90(A.nspheres,A.nterms,A.ncoefs,...
    A.omega,A.eps0,A.cmu0,A.center,A.radius,...
    aompole,bompole,A.rnodes,A.weights,A.nphi,A.ntheta);
```

2.2.8 Postprocessor

- Evaluate E and H fields on all spheres.
- Evaluate absorption, scattering, and extinction cross sections.
- Evaluate electric and magnetic dipole moments average over the spheres.
- Evaluate E and H fields at a target grid.

3 Notes

We highly recommend using the most resent versions of Matlab for Windows, (R2010b as of time of this writing). The older version of Matlab may execute the same precompiled code 2-3 times slower, user beware.

Neither the Muller solver, nor the FMPS module contain fast multipole accelerations, but the codes should be fast enough to handle triangulations for up to 400 triangles, and multiple scattering for up to 1000 spheres or so. All codes are multithreaded with OpenMP and should run at a reasonable speed on a modern 4-core desktop workstation.

The inclusions are assumed to be enclosed by non-intersecting spheres. This is required to ensure convergence of scattered fields.

4 Appendix

Cart3d configuration file format:

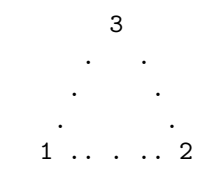
(adapted from http://people.nas.nasa.gov/aftosmis/cart3d/cart3dTriangulations.html)

```
nverts, ntri              <- total # of unique verts/tris, (int, int)
x_1, y_1, z_1             <- Geometry of Vertex 1 (float, float, float)
x_2, y_2, z_2             <- Geometry of Vertex 2 (float, float, float)
x_3, y_3, z_3             <- ...
.
.
.
x_nverts, y_nverts, z_nverts <- geom of last unique vertex (nverts)
v1_t1, v2_t1, v3_t1             <- Vertices of triangle 1 (int, int, int)
v1_t2, v2_t2, v3_t2             <- Vertices of triangle 2 (int, int, int)
v1_t3, v2_t3, v3_t3             <- Vertices of triangle 3 (int, int, int)
```

```
.
.
.
v1_ntri, v2_ntri, v3_ntri <- Vertices of last triangle (int,int,int)
1 1 1 1 1 . . . 1 1 1 2 2 2 . . . 2 2 2 <-component numbers of all
                                          triangles in the configuraiton
                                          sequentially starting at "1"

-------------
Example of F77 code for reading a *.a.tri file
read(iUnit,*) nverts, ntri
read(iUnit,*) ( verts(1,j), verts(2,j), verts(3,j), j=1,nverts)
read(iUnit,*) ( itrivert(1,j), itrivert(2,j), itrivert(3,j), j=1,ntri)
read(iUnit,*) ( comp(j), j=1,ntri)
-------------
```

Ordering of vertices and mid points:



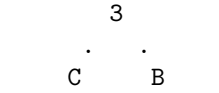
4.1 Quadratic triangulation configuration file format

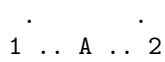
```
nverts, ntri              <- total # of unique verts/tris, (int, int)
x_1, y_1, z_1             <- Geometry of Vertex 1 (float, float, float)
x_2, y_2, z_2             <- Geometry of Vertex 2 (float, float, float)
x_3, y_3, z_3             <- ...
.
.
.
x_nverts, y_nverts, z_nverts <- geom of last unique vertex (nverts)

v1_t1, v2_t1, v3_t1, va_t1, vb_t1, vc_t1  <- Vertices and mid points of
                            triangles 1 .. ntri (int,int,int,int,int,int)
.
.
v1_ntri, v2_ntri, v3_ntri, va_ntri, vb_ntri, vc_ntri


-------------
Example of F77 code for reading a *.q.tri file
read(iUnit,*) nverts, ntri
read(iUnit,*) ( verts(1,j), verts(2,j), verts(3,j), j=1,nverts)
read(iUnit,*) ( (itrivert(i,j), i=1,6), j=1,ntri)
-------------
```

Ordering of vertices and mid points:




1 .. A .. 2