Notes on Optimization using Cgmx

William D. Henshaw,
Extreme team ...

Department of Mathematical Sciences,
Rensselaer Polytechnic Institute (RPI),
Troy, NY, USA, 12180.

November 23, 2017

Abstract:
This document describes the using the Overture CgMx time-domain Maxwell solver for optimization problems. The initial version uses some simple optimization routines from Matlab.

Contents

1 Introduction

We consider using the time-domain solver for Maxwell's equations, CgMx [1] to solve some optimization problems.

The optimizer is built with a few Matlab functions:

- `optimizer.m` : simple optimizer using Matlab functions.
- `runMaxwell.m` : run CgMx for a given case, and given parameters, and return appropriate results.

2 Minimizing the reflection coefficient of a slab

Consider a dielectric block with (ϵ_1, μ_1) occupying the region $x \in [-W/2, W/2]$ embedded in a material with (ϵ, μ).

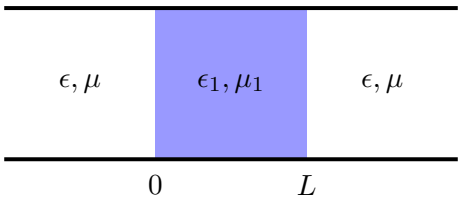

Figure 1: Dielectric block.

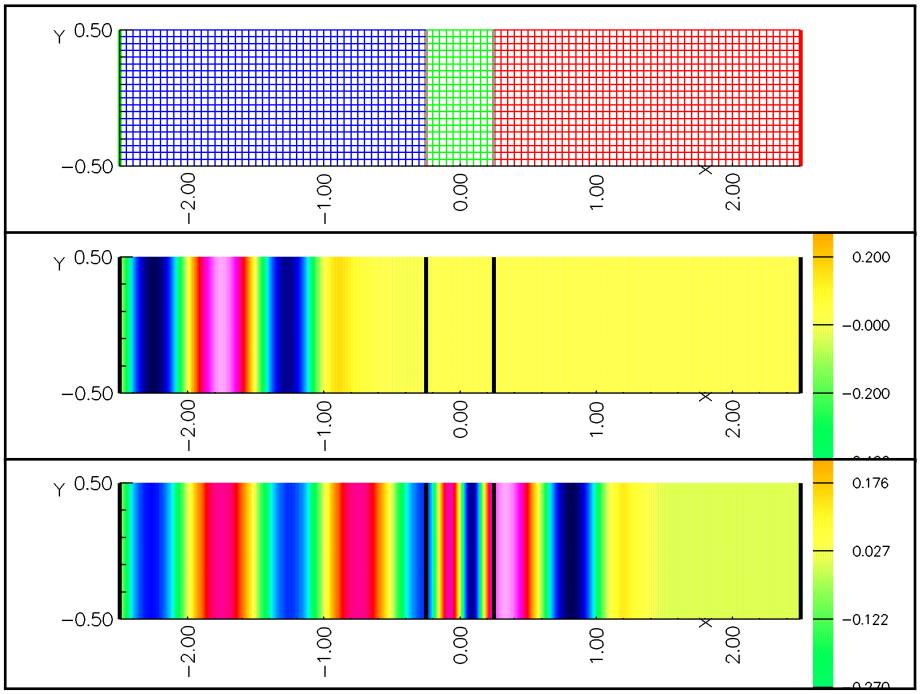

Figure 2: Grid and sample solutions for the dielectric block, $k_x = 1$, $\epsilon = 8$.

2.1 Minimizing the reflection coefficient of a slab by varying ϵ_1

For this example we attempt to find the value of ϵ_1 that minimizes the reflection coefficient. The analytic solution has one minimum at $\epsilon_1 = 4$ for this case (there are others, see Fig.3).

Notes:

1. the incident plane wave has a wave number of $k_x = 1$.
2. the width of the slab is $W = .5$ (the grid actually occupies $x \in [-.25, .25]$).
3. the grid can be made with the Makefile (`make dielectricBlock2d`).
4. a special *user defined probe* is used in this case that estimates the reflection and transmission coefficients.
5. Typical output from a CgMx run is shown in Fig. 4 where the reflection and transmission coefficients are shown over time. Once the solution has reached a near time-periodic state the coefficients become constant in time.

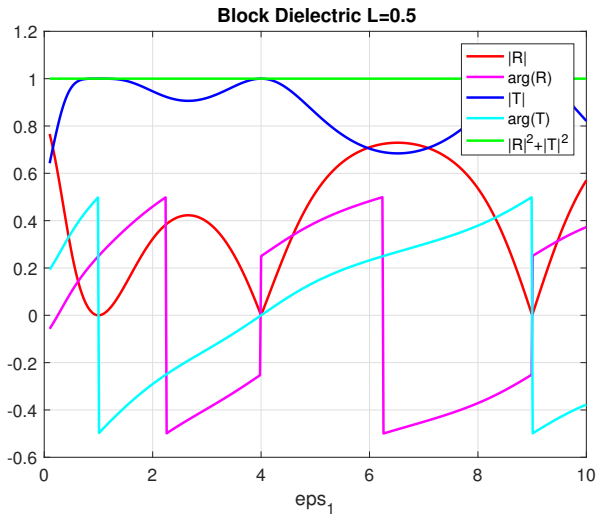

Figure 3: Dielectric block: reflection and transmission coefficients versus ϵ_1 for $W = .5$ from the analytic solution.

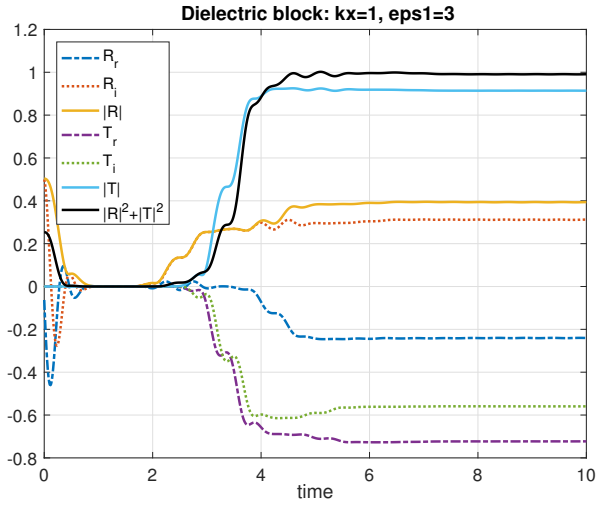

Figure 4: Dielectric block: reflection and transmission coefficients versus time as computed by CgMx. Shown are the real and imaginary parts of R and T, along with their magnitudes. The magnitude of the reflection coefficient at the final time is optimized.

Here is the output from `optimizer.m` with messages from the Matlab function `fminsearch`. A value of $\epsilon_1 = 3.975$ is obtained satisfying the tolerances.

```
>>> optimizer: caseName=block, method=fminsearch, infoLevel=0, plotOption=1, tFinal=1.000e+01, probeType=transmission,
call fminsearch...
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=3, kx=1

 Iteration   Func-count     min f(x)         Procedure
     0           1         0.394126
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=3.15, kx=1
     1           2         0.364856         initial simplex
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=3.3, kx=1
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=3.45, kx=1
     2           4           0.2733         expand
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=3.75, kx=1
```

```
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=4.05, kx=1
     3            6        0.0297798         expand
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=4.65, kx=1
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=3.75, kx=1
     4            8        0.0297798         contract inside
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=4.35, kx=1
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=3.9, kx=1
     5           10        0.0297798         contract inside
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=4.2, kx=1
runMaxwell: caseName=block, tFinal=1.000e+01 probeType=transmission, eps1=3.975, kx=1
     6           12         0.015101         contract inside

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-01
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-01

...DONE fminsearch: x=3.975, fval=0.015101
```

2.2 Minimizing the reflection coefficient of a slab by varying the width

For this example we attempt to find the value of the block width W that minimizes the reflection coefficient. The analytic solution has one minimum at $W = .5$ for this case (there are others).

Here is the output of a run. The optimizer `fminsearch` finds the value of $W = .495$ satisfying the tolerances.

Listing 1: blockWidth

```
>> optimizer -caseName=blockWidth -probeType=transmission -tf=10 -infoLevel=0 -method=fminsearch -blockWidth=.45
>>> optimizer: caseName=blockWidth, method=fminsearch, infoLevel=0, plotOption=1, gridFactor=4, tFinal=1.000e+01,
    probeType=transmission,
             : kx=1, blockWidth=0.45, eps1=4
call fminsearch...
runMaxwell: caseName=blockWidth, RENGERATE THE GRID blockWidth=0.45
runMaxwell: caseName=blockWidth, tFinal=1.000e+01 probeType=transmission, eps1=4, kx=1 blockWidth=0.45

 Iteration   Func-count     min f(x)         Procedure
     0           1         0.401063
runMaxwell: caseName=blockWidth, RENGERATE THE GRID blockWidth=0.4725
runMaxwell: caseName=blockWidth, tFinal=1.000e+01 probeType=transmission, eps1=4, kx=1 blockWidth=0.4725
     1           2          0.24439         initial simplex
runMaxwell: caseName=blockWidth, RENGERATE THE GRID blockWidth=0.495
runMaxwell: caseName=blockWidth, tFinal=1.000e+01 probeType=transmission, eps1=4, kx=1 blockWidth=0.495
runMaxwell: caseName=blockWidth, RENGERATE THE GRID blockWidth=0.5175
runMaxwell: caseName=blockWidth, tFinal=1.000e+01 probeType=transmission, eps1=4, kx=1 blockWidth=0.5175
     2           4        0.0466456         reflect
runMaxwell: caseName=blockWidth, RENGERATE THE GRID blockWidth=0.5175
runMaxwell: caseName=blockWidth, tFinal=1.000e+01 probeType=transmission, eps1=4, kx=1 blockWidth=0.5175
runMaxwell: caseName=blockWidth, RENGERATE THE GRID blockWidth=0.50625
runMaxwell: caseName=blockWidth, tFinal=1.000e+01 probeType=transmission, eps1=4, kx=1 blockWidth=0.50625
     3           6        0.0466456         contract outside

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-01
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-01

...DONE fminsearch: x=0.495, fval=0.0466456
```

3 Adjusting the shape of a lens

In this example the shape of a *lens* is adjusted to achieve some objective. The curves defining the left and right edges of the lens are defined with NURBS curves. The shape can be controlled by adjusting the control points of the NURBS.

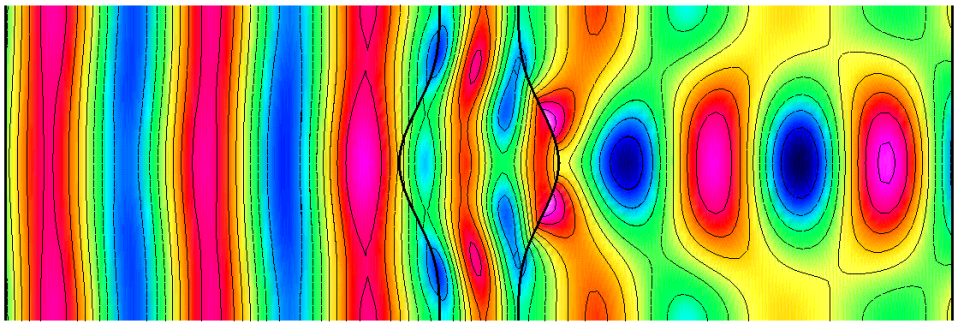

Figure 5: Target lens shape and solution (E_y).

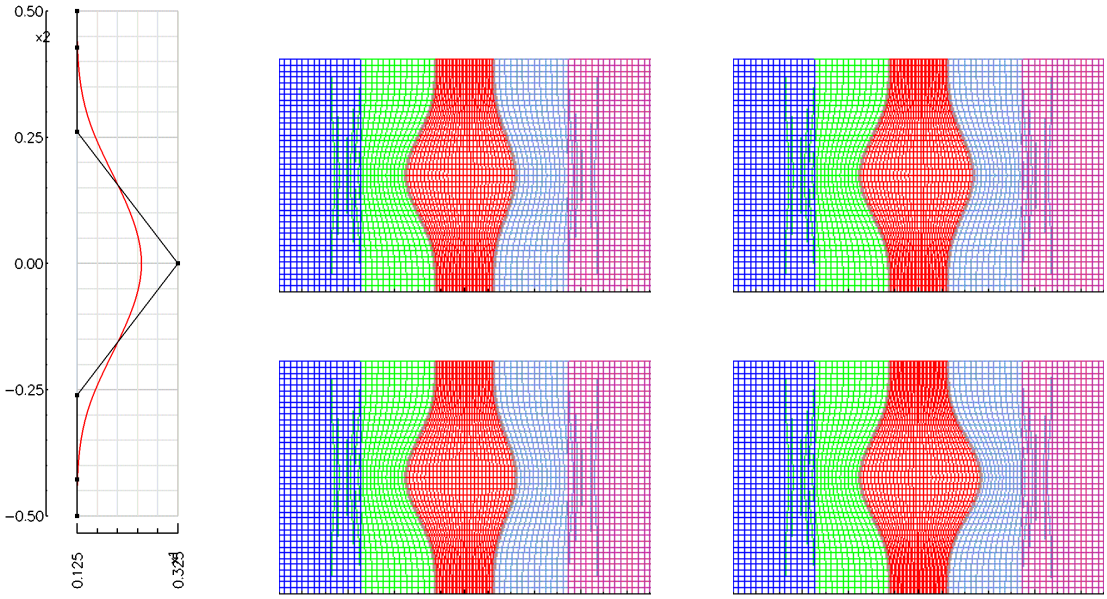

Figure 6: Left: Nurbs curve for lens shape with control points. Right: sequence of grids for optimization of the shape of a lens (adjusting the right face of the lens).

Figure 6 shows the results of a simple test. The code was first run with a given lens shape and the transmission coefficent (averaged over a box to the right of the lens) was saved as the *target transmission coefficient* (see Figure 5). The objective of the optimization was then to start from a different lens shape and adjust the central control point on the right face of the lens to match the target transmission coefficient.

Here are the results from the optimizer using `fminsearch`

A Matlab codes

A.1 optimizer.m

Listing 2: optimizer.m

```
%
%  Simple Optimizer that connects to CgMx
%
% Usage:
%    optimizer -caseName=[cyl|block|blockWidth|lens] -plotOption=[0|1] -tf=<f> -probeType=[point|transmission] ...
%              -method=[fake,fminsearch] -gridFactor=<i> -plotGrid=[0|1] -plotSolution=[0|1] ...
%              -objective=[minimizeReflection|targetTransmission] -targetFile=<>
%
% caseName:
%     block        : scattering from a dielectric block, change epsilon
%     blockWidth   : scattering from a dielectric block, change the width
% -plotGrid = 1 : plot the grid afet each optimizer step
% -plotSolution = 1 : plot the solution after each optimizer step
%
% Examples
%
%
function optimizer(varargin)

 fontSize=14;  lineWidth=2;  markerSize=5;   % for plots

 % Define some global variables to avoid pass so many args to runMaxwell
 globalDeclarations

 method = 'fake';

 caseName = 'cyl';   % case to run
 plotOption=1;
 plotGrid=1;
 plotSolution=1;
 infoLevel=1;

 objective='minimizeReflection';
 targetFile='none'; % data file for target

 pointProbe=0; transmissionProbe=1; % probe types
 probeType='point';
 tFinal=1;         % final time
 gridFactor=4;     % defines grid resolution 1,2,4,8
 kx=1;             % wave number
 blockWidth=.5;    % default width of dielectric block
 eps1=4.;          % default block epsilon
 tolFun=.1; tolX=.1; % tolerences for fminserach

  % --- read command line args ---
  for i = 1 : nargin
    line = varargin{i};
    caseName      = getString( line,'-caseName',caseName );
    probeType     = getString( line,'-probeType',probeType );
    method        = getString( line,'-method',method );
    objective     = getString( line,'-objective',objective );
    targetFile    = getString( line,'-targetFile',targetFile );
    tFinal        =   getReal( line,'-tf',tFinal );
    kx            =   getReal( line,'-kx',kx );
    eps1          =   getReal( line,'-eps1',eps1 );
    tolFun        =   getReal( line,'-tolFun',tolFun );
    tolX          =   getReal( line,'-tolX',tolX );
    blockWidth    =   getReal( line,'-blockWidth',blockWidth );
    infoLevel     =    getInt( line,'-infoLevel',infoLevel );
    plotOption    =    getInt( line,'-plotOption',plotOption );
    plotGrid      =    getInt( line,'-plotGrid',plotGrid );
    plotSolution  =    getInt( line,'-plotSolution',plotSolution );
    gridFactor    =    getInt( line,'-gridFactor',gridFactor );
  end

  fprintf('>>> optimizer: caseName=%s, method=%s, objective=%s (targetFile=%s, tolFun=%g, tolX=%g), infoLevel=%d, plotOption=%d, plotGrid=%d, plotSolution=%d, gridFactor=%d, tFinal=%9.3e, probeType=%s, \n',...
                caseName,method,objective,targetFile,tolFun,tolX,infoLevel,plotOption,plotGrid,plotSolution,gridFactor,tFinal,probeType);
  fprintf('                :kx=%g, blockWidth=%g, eps1=%g\n',kx,blockWidth,eps1);
```

```
69
70
71  iteration=0;      % keeps track of how many times runMaxwell is called.
72  maxIterations=10;
73
74  % ------------------------------------------------------
75  %   Define an objective function for minimization
76  % ------------------------------------------------------
77  function ff = cgmxFunction( xx )
78    % fprintf(' objectiveFunction: x=%g\n',xx(1));
79
80    if( strcmp(objective,'targetTransmission') )
81
82      % -- LENS: for now just adjust the right control point:
83      par(1)=kx;
84      par(2)=eps1;
85      dxLeft =-.2; dxRight=.2 + xx(1);
86      par(3)=dxLeft;    % shift left control point
87      par(4)=dxRight;   % shift right control point
88
89    elseif( strcmp(objective,'minimizeReflection') )
90
91      if( strcmp(caseName,'blockWidth') )
92        blockWidth=xx(1);
93      else
94        eps1=xx(1);
95      end
96      par(1)=kx;
97      par(2)=eps1;
98      par(3)=blockWidth;
99
100   else
101
102     fprintf('optimizer: ERROR: unknown objective =[%s]\n',objective);
103     pause; pause;
104   end;
105
106
107
108   [ values ]  = runMaxwell( caseName,tFinal,probeType,gridFactor,infoLevel,plotOption, par );
109   ff=values(1); % reflection coefficient
110
111 end
112
113 % ff = @(x) x(1)^2;
114 ff = @(x) cgmxFunction( x ); % for some reason we need to call cgMxFunction this way
115
116 if( strncmp(method,'fminsearch',length('fminsearch')) )
117
118   % ---- Find minium using fminsearch: ----
119
120   options = optimset('Display','iter','TolFun',tolFun, 'TolX',tolX);
121
122   if( strcmp(objective,'targetTransmission') )
123
124     % Lens:
125     x0 = [ -.05 ]; % initial guess for dxRight
126
127   elseif( strcmp(objective,'minimizeReflection') )
128
129     if( strcmp(caseName,'blockWidth') )
130       x0=[ blockWidth ]; %  blockWidth
131     else
132       x0=[ 3.]; %  3. ,  8. initial guess
133     end;
134   else
135
136     fprintf('optimizer: ERROR: unknown objective =[%s]\n',objective);
137     pause; pause;
138   end;
139
140   % fval = myObjectiveFunction( x0 );
```

```
    % fprintf('myObjectiveFunction: x0=%g, f=%g\n',x0(1),fval);

    fprintf('call␣fminsearch...\n');

    % [x,fval] = fminsearch(myObjectiveFunction,x0,options);
    [x,fval] = fminsearch(ff,x0,options);

    fprintf('...DONE␣fminsearch:␣x=%g,␣fval=%g\n',x(1),fval);


  else

    % -------------------- FAKE OPTIMIZATION LOOP  ------------------------
    for iter=1:maxIterations

      if( strcmp(caseName,'blockWidth') )
        kx=1;
        eps1=4;
        blockWidthNew = blockWidth+.1*iter;
        par(1)=kx;
        par(2)=eps1;
        par(3)=blockWidthNew

      elseif( strcmp(caseName,'block') )
        kx=1;
        eps1= 2+ .5*iter;
        par(1)=kx;
        par(2)=eps1;
        par(3)=blockWidth

      elseif( strcmp(caseName,'lens') )

        % NOTE: for now we just shift one control point at the center of the left and right sides.

        par(1)=kx;
        par(2)=eps1;
        dxLeft = -.2 + .025*(iter-1); dxRight=.2 - .025*(iter-1);
        par(3)=dxLeft;    % shift left control point
        par(4)=dxRight;   % shift right control point

      else

        eps1=4;
        kx = 2 + iter; % incident wave number
        par(1)=kx;
        par(2)=eps1;
        par(3)=blockWidth
      end;

      [ values ]  = runMaxwell( caseName,tFinal,probeType,gridFactor,infoLevel,plotOption, par );

      fprintf('optimizer:␣iter=%d:␣return-values=[%g,%g]\n',iter,values(1),values(2));

      if( 1==1 || ( plotGrid==0 && plotSolution==0) )
         pause
      end
    end; % end for iter
  end;

fprintf('done\n');

end

% --- Utility functions ---




% Function getReal: read a command line argument for a real variable
function [ val ] = getReal( line,name,val)
 % fprintf('getReal: val=%g line=[%s] name=[%s]\n',val,line,name);
```

```
 if( strncmp(line,strcat(name,'='),length(name)+1) )
   val = sscanf(line,sprintf('%s=%%e',name));
   % fprintf('getReal: scan for val=%g\n',val);
 end
end

% Function getInt: read a command line argument for an integer variable
function [ val ] = getInt( line,name,val)
 if( strncmp(line,strcat(name,'='),length(name)+1) )
   val = sscanf(line,sprintf('%s=%%d',name));
 end
end

% Function getString: read a command line argument for a string variable
function [ val ] = getString( line,name,val)
 if( strncmp(line,strcat(name,'='),length(name)+1) )
   val = sscanf(line,sprintf('%s=%%s',name));
 end
end
```

A.2 runMaxwell.m

Listing 3: runMaxwell.m

```
%
%  Run the Maxwell Solver CgMx and return the requested results
%
% Usage:
%
% Parameters:
%   caseName    (input) : ['cyl'|'block'|'blockWidth'|'lens'],
%   tFinal      (input) : final time
%   probeType   (input) : ['point','transmission']
%   gridFactor  (input) : =1,2,4,8 - grid is this much finer than coarsest grid available. Usually dx=1/(10*gridFactor)
%   infolevel   (input) :
%   plotOption  (input) :
%   par(1:)     (input) : input parameters
%
%   values      (output) : array of output values
%
% Examples
%
%
function [ values ]  = runMaxwell( caseName,tFinal,probeType,gridFactor,infoLevel,plotOption, par )

 % Define some global variables to avoid pass so many args to runMaxwell
 globalDeclarations

 iteration = iteration+1; % counts the number of times runMaxwell is called

% Overture = getenv('Overture')
 % Here is cgmx:  *fix me*
 cgmx = '/Users/henshaw/cg.g/mx/bin/cgmx';
 % Here is Ogen
 ogen = '/Users/henshaw/Overture.g/bin/ogen';

 % Here is plotStuff
 plotStuff = '/Users/henshaw/Overture.g/bin/plotStuff';

 fontSize=14;  lineWidth=2;  markerSize=5;   % for plots
 gridName='none';
 showFileName='optimizer.show';

 % OLD pointProbe=0; transmissionProbe=1; % probe types

  values(1)=0; values(2)=0;

  if( strcmp(caseName,'block') )

```

```
46      % -------------------------------------------------
47      % ---- BLOCK: scattering from a dielectric block -----
48      % -------------------------------------------------
49
50      kx=par(1);
51      eps1=par(2);
52
53      fprintf('runMaxwell: it=%d, caseName=%s, tFinal=%9.3e probeType=%s, eps1=%g, kx=%g plotOption=%d\n',iteration,
         caseName,tFinal,probeType,eps1,kx,plotOption);
54
55
56      titleLabel=sprintf('Block: eps1=%g, kx=%i',eps1,kx);
57
58      cgmxCommand = sprintf('%s -noplot dielectricBodies -g=dielectricBlockGrid2de%d.order4 -backGround=leftBackGround -
         rbc=rbcNonLocal -kx=%i -eps1=%g -eps2=1. -diss=2 -tf=%g -tp=.1 -probeFileName=OptProbe -go=go >! cgmxOptimzer.out
         ',cgmx,gridFactor,kx,eps1,tFinal);
59
60      titleLabel=sprintf('Dielectric block: kx=%i, eps1=%g',kx,eps1);
61
62      if( strncmp(probeType,'transmission',length('transmission')) )
63        % reflection/transmission probes:
64        probeDataFile = 'OptProbe.dat';
65      else
66        % point probes:
67        probeDataFile = 'leftOptProbe.dat';
68      end;
69
70    elseif( strcmp(caseName,'blockWidth') )
71
72      % ------------------------------------------------------------------------
73      % ---- BLOCKWIDTH: scattering from a dielectric block that changes width -----
74      % ------------------------------------------------------------------------
75
76      kx=par(1);
77      eps1=par(2);
78      blockWidth=par(3);
79
80      fprintf('runMaxwell: caseName=%s, RENGERATE THE GRID blockWidth=%g\n',caseName,blockWidth);
81      % Note: new name chosen for grid:
82      ogenCommand = sprintf('%s -noplot dielectricBlockGrid2d -prefix=dieBlockOpt -interp=e -order=4 -width=%g -factor=%d
         >&! ogenOpt.out',...
83                            ogen,blockWidth,gridFactor);
84      if( infoLevel>0 )  fprintf('Run ogen: %s\n',ogenCommand); end;
85      system(ogenCommand);
86      if( rt ~= 0 )
87        fprintf('runMaxwell:ERROR return from ogen: rt=[%d]\n',rt);
88        pause; pause; pause;
89      end
90      if( infoLevel>0 ) fprintf('..done ogen\n'); end;
91
92      fprintf('runMaxwell: caseName=%s, tFinal=%9.3e probeType=%s, eps1=%g, kx=%g blockWidth=%g \n',...
93              caseName,tFinal,probeType,eps1,kx,blockWidth);
94
95
96      titleLabel=sprintf('Block: eps1=%g, kx=%i, width=%g',eps1,kx,blockWidth);
97
98      cgmxCommand = sprintf('%s -noplot dielectricBodies -g=dieBlockOpte%d.order4 -backGround=leftBackGround -rbc=
         rbcNonLocal -kx=%i -eps1=%g -eps2=1. -diss=2 -tf=%g -tp=.1 -probeFileName=OptProbe -go=go >! cgmxOptimzer.out ',
         cgmx,gridFactor,kx,eps1,tFinal);
99
100     titleLabel=sprintf('Dielectric block: kx=%i, eps1=%g, width=%g',kx,eps1,blockWidth);
101
102     if( strncmp(probeType,'transmission',length('transmission')) )
103       % reflection/transmission probes:
104       probeDataFile = 'OptProbe.dat';
105     else
106       % point probes:
107       probeDataFile = 'leftOptProbe.dat';
108     end;
109
110   elseif( strcmp(caseName,'cyl') )
111
```

```
    % ------------------------------------------------
    % CYL: ---- scattering from a PEC cylinder ---
    % ------------------------------------------------

    kx=par(1);
    eps1=par(2);
    titleLabel=sprintf('cylScat: kx=%i',kx);

    fprintf('runMaxwell: caseName=%s, tFinal=%9.3e probeType=%s, eps1=%g, kx=%g \n',caseName,tFinal,probeType,eps1,kx);

    cgmxCommand = sprintf('%s -noplot cylOpt -g=cice%d.order4.hdf -probeFileName=OptProbe -tf=%g -tp=.1 -kx=%g -go=go >! cgmxOptimzer.out ',...
        cgmx,gridFactor,tFinal,kx);

    probeDataFile = 'rightOptProbe.dat';


  elseif( strcmp(caseName,'lens') )

    % ------------------------------------------------
    % LENS: ---- adjust the shape of a lens  ------------
    % ------------------------------------------------

    kx=par(1);
    eps1=par(2);
    dxLeft=par(3);   % shift left control point
    dxRight=par(4);  % shift right control point
    titleLabel=sprintf('Lens: kx=%i, eps1=%g, dxLeft=%g, dxRight=%g',kx,eps1,dxLeft,dxRight);

    fprintf('runMaxwell: caseName=%s, RENGERATE THE GRID dxLeft=%g, dxRight=%g\n',caseName,dxLeft,dxRight);
    % NOTE: for now we just shift one control point at the center of the left and right sides.
    % Note: new name chosen for grid:
    ogenCommand = sprintf('%s -noplot curvedBlockGrid2d -prefix=lensOptGrid -order=4 -interp=e -width=.25 -
     interfaceGridWidth=.4 -dxLeft=0 0 0 0 %g 0 0 0 0 -dxRight=0 0 0 0 %g 0 0 0 0 -factor=%d >&! ogenOpt.out',...
                          ogen,dxLeft,dxRight,gridFactor);
    if( infoLevel>0 )  fprintf('Run ogen: %s\n',ogenCommand); end;
    rt = system(ogenCommand);
    if( rt ~= 0 )
      fprintf('runMaxwell:ERROR return from ogen: rt=[%d]\n',rt);
      pause; pause; pause;
    end
    if( infoLevel>0 ) fprintf('..done ogen\n'); end;


    fprintf('runMaxwell: caseName=%s, tFinal=%9.3e probeType=%s, eps1=%g, kx=%g dxLeft=%g dxRight=%g\n',caseName,tFinal
     ,probeType,eps1,kx,dxLeft,dxRight);

    gridName = sprintf('lensOptGride%d.order4.hdf',gridFactor);
    cgmxCommand = sprintf('%s -noplot dielectricBodies -g=%s -probeFileName=OptProbe -tf=%g -tp=.5 -kx=%g -backGround=
     leftBackGround -rbc=rbcNonLocal -eps1=%g -eps2=1. -diss=2 -xb=-1. -show=%s -go=go >! cgmxOptimzer.out ',...
        cgmx,gridName,tFinal,kx,eps1,showFileName);

    probeDataFile = 'OptProbe.dat';

  else
    fprintf('Unknown caseName=[%s]\n',caseName)
    pause;
  end;



  % ----------------------------------------
  % ----------- RUN CGMX -------------------
  % ----------------------------------------
  if( infoLevel>0 )
    fprintf('Run cgmx...\n');
    fprintf('>> %s\n',cgmxCommand);
  end;
  rt = system(cgmxCommand);
  if( rt ~= 0 )
    fprintf('runMaxwell:ERROR return from cgmx: rt=[%d]\n',rt);
    pause; pause; pause;
```

```
  end

%     system(sprintf('/Users/henshaw/cg.g/mx/bin/cgmx -noplot cylOpt -g=cice4.order4.hdf -probeFileName=OptProbe -tf=1
    -tp=.1 -kx=%g -go=go >! cgmxOptimizer.out ',kx));

  if( infoLevel>0 )
    fprintf('...done\n');
  end;

  % ------------------------------------------------
  % ---------- Optionally plot the grid ------------
  % ------------------------------------------------

  if( plotGrid==1 && strcmp(caseName,'lens') )
    fprintf('Plot␣the␣current␣grid=[%s]...\n',gridName);
    plotName=sprintf('lensGridIteration%d.ps',iteration);
    system(sprintf('%s␣plotCurrentGrid.cmd␣-show=%s␣-plotName=%s>!␣plotGrid.out',plotStuff,gridName,plotName));
  end;

  if( plotSolution==1 && strcmp(caseName,'lens') )
    fprintf('Plot␣the␣solution,␣showFileName=[%s]...\n',showFileName);
    system(sprintf('%s␣plotSolution.cmd␣-show=%s␣>!␣plotSolution.out',plotStuff,showFileName));
  end;


  % fileName='/Users/henshaw/runs/mx/optimizer/leftOptProbe.dat';
  % fileName='rightOptProbe.dat';
  % fileName='leftOptProbe.dat';


  if( strncmp(probeType,'point',length('point')) )

    % --- PLOT POINT PROBE RESULTS ----

    if( infoLevel>0 )
      fprintf('POINT-PROBE:␣Read␣probe␣file␣=␣[%s]\n',probeDataFile)
    end;
    referenceFile=0;
    [ t, Ex, Ey, Hz ] = getCgMxProbeData( probeDataFile, referenceFile,infoLevel  );

    fprintf('Plot␣probe␣data...\n')
    plot(t,Ex,'r-', t,Ey,'g-', t,Hz,'b-','LineWidth',lineWidth );
    title(titleLabel);
    legend('E_x','E_y','H_z' ); set(gca,'FontSize',fontSize);
    xlabel('t');
    grid on;

  elseif( strncmp(probeType,'transmission',length('transmission')) )

    % --- PLOT REFLECTION/TRANSMISSION PROBE RESULTS ----

    if( infoLevel>0 )
      fprintf('REFLECTION/TRANSMISSION-PROBE:␣Read␣probe␣file␣=␣[%s]\n',probeDataFile)
    end;
    [ t, Rr, Ri, Tr, Ti ] = getCgMxProbeReflectionTransmissionData( probeDataFile, infoLevel  );

    Rnorm = sqrt( Rr.^2 + Ri.^2 );
    Tnorm = sqrt( Tr.^2 + Ti.^2 );
    rtNorm = Rnorm.^2 + Tnorm.^2;

    % google 'matlab colrs rgb' --> CSS3 color names
    % myColours = [rgb('Crimson'); rgb('Red'); rgb('Orange'); rgb('Blue'); rgb('DodgerBlue'); rgb('Turquoise')];
    % myColours = [rgb('Red'); rgb('OrangeRed'); rgb('Orange'); rgb('Blue'); rgb('DodgerBlue'); rgb('Turquoise')];
    % set(gcf,'DefaultAxesColorOrder',myColours);

    plot(t,Rr,'-.',t,Ri,':',t,Rnorm,'-', ...
         t,Tr,'-.',t,Ti,':',t,Tnorm,'-', ...
         t,rtNorm,'k-', ...
         'LineWidth',lineWidth,'MarkerSize',markerSize);

    legend('R_r','R_i','|R|', 'T_r','T_i','|T|','|R|^2+|T|^2','Location','NorthWest');

```

```
    set(gca,'FontSize',fontSize);
    xlab =xlabel('time');                      % add axis labels and plot title
    set(xlab, 'Units', 'Normalized', 'Position', [.5, -0.05, 0]); % shifty x label upward

    title(titleLabel);
    grid on;
    drawnow;
    if( plotOption>0 )
      plotFileName=sprintf('%sReflectionTransmission.eps',caseName);
      fprintf('runMaxwell: save plot: %s\n',plotFileName);
      print('-depsc2',plotFileName); % save as an eps file
    end;
    if( infoLevel>0 )
      fprintf('t=%9.3e: R=(%12.5e,%12.5e) |R|=%12.5e, T=(%12.5e,%12.5e) |T|=%12.5e, \n',...
             t(end), ...
            Rr(end),Ri(end),Rnorm(end), ...
            Tr(end),Ti(end),Tnorm(end) );
    end;

    % ------- DEFINE THE OBJECTIVE -----------
    if( strcmp(objective,'minimizeReflection') || strcmp(objective,'none') )

      % -- Objective: minimize the reflection
      values(1)=Rnorm(end); % reflection coefficient
      values(2)=Tnorm(end); % transmission coefficient

   elseif( strcmp(objective,'targetTransmission') )

      % -- Objective: minimize the error between the transmission coeff and the target transmission
      [ tTarget, RrTarget, RiTarget, TrTarget, TiTarget ] = getCgMxProbeReflectionTransmissionData( targetFile,
     infoLevel  );

      fprintf('runMaxwell: T=[%g,%g] : target: T=[%g,%g]\n',Tr(end),Ti(end),TrTarget(end),TiTarget(end));

      Rdiff = sqrt( (Rr(end)-RrTarget(end))^2 + (Ri(end)-RiTarget(end))^2 );
      Tdiff = sqrt( (Tr(end)-TrTarget(end))^2 + (Ti(end)-TiTarget(end))^2 );

      values(1)=Rdiff;
      values(2)=Tdiff;


   else
      fprintf('runMaxwell: ERROR: unknown objective=[%s]\n',objective);
      pause; pause;
   end;

  else
    fprintf('ERROR: unknown probeType=[%s]\n',probeType )
  end;
  if( infoLevel>0 )
    fprintf('...done runMaxwell\n');
  end;

end

% --- Utility functions ---

% ---------------  Read and Extract REFLECTION/TRANSMISSON Probe data from a CgMx probe file -----
% Parameters:
%   fileName (input) : name of the reflection/transmisson probe file
%   referenceFile (input) : (optional) name of the "reference file data" to be subtracted
%                           to get reflected field from total field
%  infoLevel (input) : > 0 : output extra info
%
%  t (output) : array of timne values
%  Rr,Ri (output) : eral and imaginary parts of the reflection coefficient
%  Tr,Ti (output) : eral and imaginary parts of the transmission coefficient
% ------------------------------------------------------------------------------------------
function [ t, Rr, Ri, Tr, Ti ] = getCgMxProbeReflectionTransmissionData( fileName, infoLevel  )


%  Read data:
```

```
322
323 % reflecton data:
324 reflectionFileName='␣';
325 reflectionFileName=sprintf('reflection%s',fileName);
326 if( infoLevel>0 )
327   fprintf('ReflectionTransmissionProbe:␣Read␣file=[%s]\n',reflectionFileName);
328 end;
329 [headers,labels,t,qr] = readProbeFile(reflectionFileName,infoLevel);
330
331 % transmission data:
332 transmissionFileName=sprintf('transmission%s',fileName);
333 if( infoLevel>0 )
334   fprintf('ReflectionTransmissionProbe:␣Read␣file=[%s]\n',transmissionFileName);
335 end;
336 [headers,labels,t,qt] = readProbeFile(transmissionFileName,infoLevel);
337
338 [numHeaders,headerLen] = size(headers);
339 if( infoLevel>0 )
340   fprintf('Header␣comments:\n');
341   for i=1:numHeaders
342     fprintf('%s\n',headers(i,:));
343   end;
344 end;
345
346 % labels
347
348 [numColumns,columnLen] = size(labels);
349 numVars=numColumns-1;   % t is not counted
350
351 if( infoLevel>0 )
352   fprintf(1,'ReflectionTransmissionProbe:␣There␣are␣%d␣solution␣variables␣in␣the␣data.\n',numVars);
353 end;
354
355 cStart=4; % first component
356
357 numToPlot=numVars-j+1;
358
359 cr=cStart;
360 ci=cStart+1;
361
362 Rr = qr(:,cr);
363 Ri = qr(:,ci);
364
365 Tr = qt(:,cr);
366 Ti = qt(:,ci);
367
368 % Rnorm = sqrt( qr(:,cr).^2 + qr(:,ci).^2 );
369 % Tnorm = sqrt( qt(:,cr).^2 + qt(:,ci).^2 );
370 % rtNorm = Rnorm.^2 + Tnorm.^2;
371
372 end
373
374
375
376
377
378 % ---------------  Read and Extract Probe data from a CgMx probe file -----
379 % Parameters:
380 %   fileName (input) : name of the probe file
381 %   referenceFile (input) : (optional) name of the "reference file data" to be subtracted
382 %                           to get reflected field from total field
383 %  infoLevel (input) : > 0 : output extra info
384 %
385 %  t (output) : array of timne values
386 %  Ex, Ey, Hz (output) : time sequence probe data
387 % ------------------------------------------------------------------------------
388 function [ t, Ex, Ey, Hz ] = getCgMxProbeData( fileName, referenceFile, infoLevel  )
389
390
391   %  Read data:
392   [headers,labels,t,q] = readProbeFile(fileName,infoLevel);
393
```

```
  [numHeaders,headerLen] = size(headers);
  fprintf('Header comments:\n');
  if( infoLevel >0 )
    for i=1:numHeaders
      fprintf('%s\n',headers(i,:));
    end;
  end;

  % labels

  if( referenceFile ~= 0 )
    fprintf('Reading the reference file=[%s]\n',referenceFile);
    [headersRef,labelsRef,tRef,qRef] = readProbeFile(referenceFile);

    tDiff = max(abs(t-tRef));
    fprintf('Checking consistency of reference: |t-tRef| = %9.2e\n',tDiff);

    % Subtract off the reference solution (but not from x,y,z)
    q(:,4:end) = q(:,4:end)- qRef(:,4:end);

  end



  [numColumns,columnLen] = size(labels);
  numVars=numColumns-1;   % t is not counted
  fprintf(1,'There are %d solution variables in the data.\n',numVars);

  x = q(:,1);
  y = q(:,2);
  z = q(:,3);

  xMin = min(x);  xMax = max(x);
  yMin = min(y);  yMax = max(y);
  zMin = min(z);  zMax = max(z);

  fixedPosition=0;
  if xMin==xMax && yMin==yMax && zMin==zMax
    fixedPosition=1;
    fprintf(1,' Probe is located at the fixed position (x,y,z)=(%9.3e,%9.3e,%9.3e)\n',xMin,yMin,zMin);
  end;

  j=1;
  if fixedPosition==1
    j=j+3;  % do not plot (x,y,z) in this case
  end;
  cStart=j; % first component

  numToPlot=numVars-j+1;

  exc=cStart+0;
  eyc=cStart+1;
  hzc=cStart+2;

  Ex = q(:,exc);
  Ey = q(:,eyc);
  Hz = q(:,hzc);


end
```

References

[1] W. D. Henshaw, *Cgmx user guide: An Overture solver for Maxwell's equations on composite grids*, Software Manual LLNL-SM-523971, Lawrence Livermore National Laboratory, 2012.