Notes on Optimization using Cgmx

William D. Henshaw,
Extreme team ...

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

June 21, 2018

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 [**?**] 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).

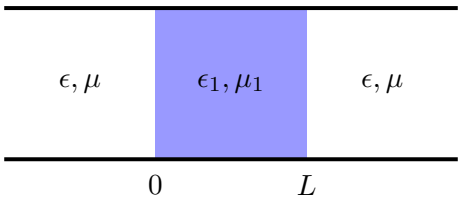

Figure 1: Dielectric block.

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.**??**).
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. **??** 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.

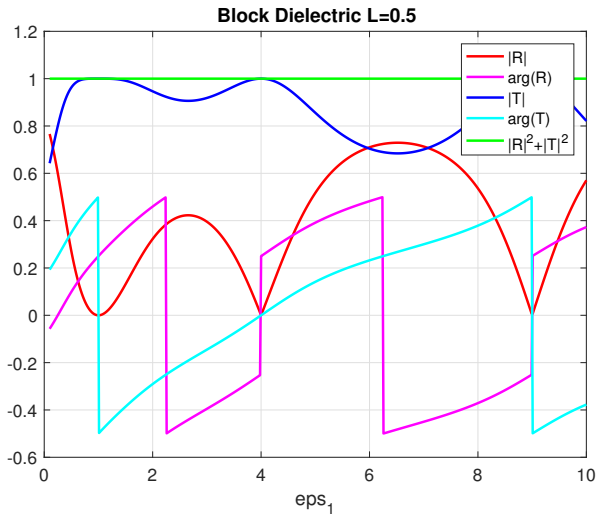

Figure 3: Dielectric block: reflection and transmission coefficients versus ϵ_1 for $W = .5$ from the analytic solution.

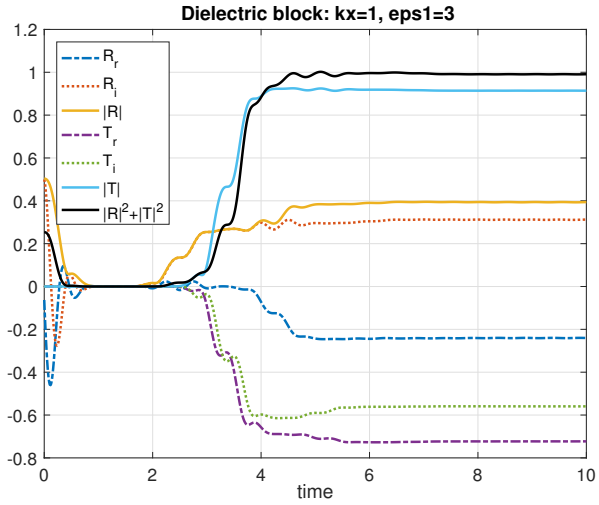

Figure 4: Dielectric block: reflection and transmission coefficients versus time as computed by CgMx. Shown are the real and imaginary parts of R and T, along with their magnitudes. The magnitude of the reflection coefficient at the final time is optimized.

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 **??** shows a Nurbs curve with control points. The curve was defined by a set of points that were interpolated to form the Nurbs.

Note: It is possible to change the control points explicitly, but there is currently a problem in that the parameterization of the curve (which is usually based on arclength) is not updated – this can result in a poor quality grid. This needs to be fixed before more general changes to the control points will be viable.

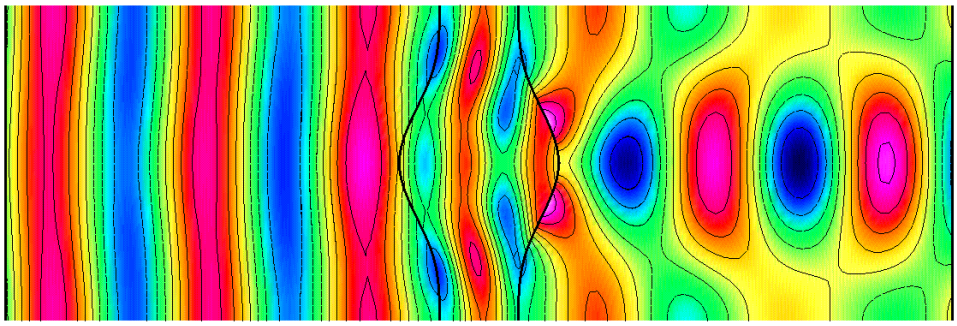

Figure 5: Target lens shape and solution (E_y).

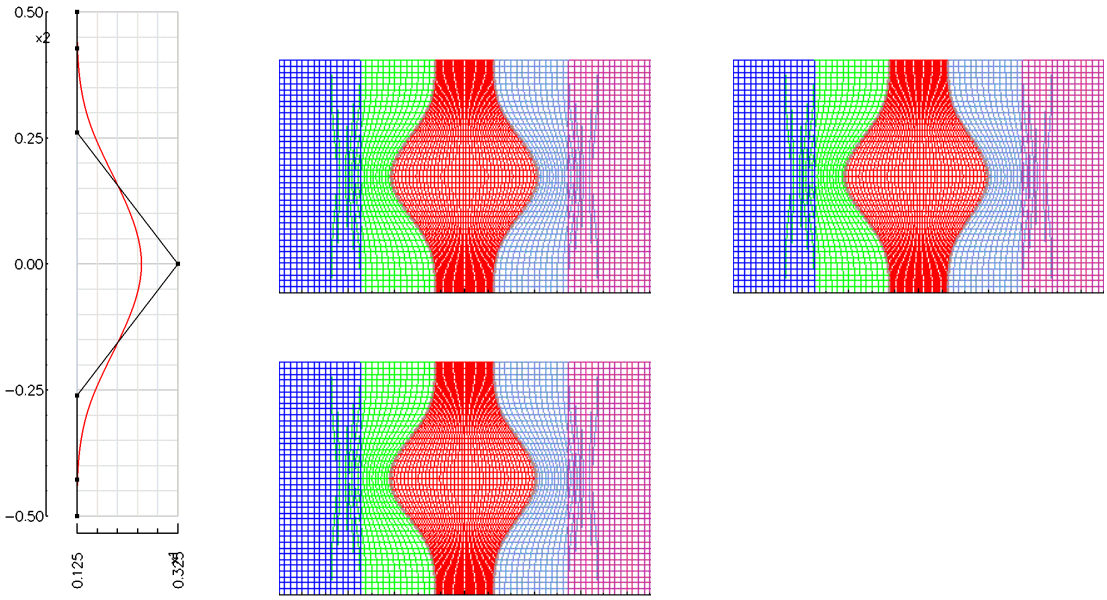

Figure 6: Left: Nurbs curve for lens shape with control points. Right: sequence of grids for optimization of the shape of a lens (adjusting the left and right faces of the lens).

Figure **??** 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 **??**). The objective of the optimization was then to start from a different lens shape and adjust the central control points on the left and right faces of the lens to match the target transmission coefficient.

Here are shome of the output from the optimizer using `fminsearch`

Figure 7: Convergence history of fminsearch.

```
>> optimizer -caseName=lens -probeType=transmission -kx=2 -eps1=4 -tf=5 -infoLevel=0 -plotGrid=0 -plotSolution=0
         -method=fminsearch -objective=targetTransmission -targetFile=TargetLens.dat -tolFun=.001 -tolX=.05 -x0=-.1
>>> optimizer: caseName=lens, method=fminsearch,
 objective=targetTransmission (targetFile=TargetLens.dat, tolFun=0.001, tolX=0.05), infoLevel=0,
 plotOption=1, plotGrid=0, plotSolution=0, gridFactor=4, tFinal=5.000e+00, probeType=transmission,
             : kx=2, blockWidth=0.5, eps1=4
call fminsearch...
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.3, dxRight=0.3
runMaxwell: T=[-0.0791327,0.0239978] : target: T=[0.164138,0.461024]

 Iteration   Func-count     min f(x)         Procedure
     0           1         0.129852
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.305, dxRight=0.3
runMaxwell: T=[-0.0784375,0.00793146] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.3, dxRight=0.305
runMaxwell: T=[-0.0780648,0.00801869] : target: T=[0.164138,0.461024]
     1           3         0.129852         initial simplex
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.305, dxRight=0.295
runMaxwell: T=[-0.0793239,0.0238] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.3075, dxRight=0.29
runMaxwell: T=[-0.0794118,0.0313547] : target: T=[0.164138,0.461024]
     2           5         0.120426         expand
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.3025, dxRight=0.29
runMaxwell: T=[-0.079017,0.0472384] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.30125, dxRight=0.285
runMaxwell: T=[-0.0780445,0.0658418] : target: T=[0.164138,0.461024]
     3           7         0.115185         expand
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.30875, dxRight=0.275
runMaxwell: T=[-0.0768177,0.0718712] : target: T=[0.164138,0.461024]
     4           8         0.115185         reflect
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.3025, dxRight=0.27
runMaxwell: T=[-0.0727761,0.103641] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.30375, dxRight=0.275
runMaxwell: T=[-0.0756175,0.086624] : target: T=[0.164138,0.461024]
     5          10         0.115185         contract outside
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.29625, dxRight=0.285
runMaxwell: T=[-0.0767729,0.0805756] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.29, dxRight=0.29
runMaxwell: T=[-0.076188,0.0843362] : target: T=[0.164138,0.461024]
     6          12         0.112223         expand
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.2875, dxRight=0.3
runMaxwell: T=[-0.0774894,0.0623941] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.299687, dxRight=0.28125
runMaxwell: T=[-0.0766232,0.081142] : target: T=[0.164138,0.461024]
     7          14         0.112223         contract inside
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.288437, dxRight=0.28625
runMaxwell: T=[-0.0743235,0.0991321] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.282031, dxRight=0.286875
runMaxwell: T=[-0.0712715,0.114746] : target: T=[0.164138,0.461024]
```

```
     8           16         0.105611         expand
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.272344, dxRight=0.295625
runMaxwell: T=[-0.0697294,0.116487] : target: T=[0.164138,0.461024]
     9           17         0.105611         reflect
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.264375, dxRight=0.2925
runMaxwell: T=[-0.0638157,0.144137] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.251562, dxRight=0.29375
runMaxwell: T=[-0.0559129,0.170297] : target: T=[0.164138,0.461024]
    10           19        0.0938313         expand
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.26125, dxRight=0.285
runMaxwell: T=[-0.0578973,0.169759] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.255703, dxRight=0.279687
runMaxwell: T=[-0.0514997,0.194407] : target: T=[0.164138,0.461024]
    11           21        0.0839358         expand
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.225234, dxRight=0.286562
runMaxwell: T=[-0.0333342,0.242685] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.196836, dxRight=0.286406
runMaxwell: T=[-0.00564524,0.301778] : target: T=[0.164138,0.461024]
    12           23         0.081498         reflect
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.229375, dxRight=0.2725
runMaxwell: T=[-0.0266702,0.267519] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.218281, dxRight=0.261875
runMaxwell: T=[-0.00424015,0.317287] : target: T=[0.164138,0.461024]
    13           25        0.0576806         expand
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.187812, dxRight=0.26875
runMaxwell: T=[0.0317859,0.364031] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.23873, dxRight=0.276953
runMaxwell: T=[-0.0377415,0.237048] : target: T=[0.164138,0.461024]
    14           27        0.0576806         contract inside
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.231777, dxRight=0.252266
runMaxwell: T=[-0.00967372,0.310309] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.235049, dxRight=0.235117
runMaxwell: T=[0.00808226,0.342775] : target: T=[0.164138,0.461024]
    15           29        0.0536635         expand
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.2146, dxRight=0.220039
runMaxwell: T=[0.0772318,0.41604] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.202534, dxRight=0.191582
runMaxwell: T=[0.17862,0.465646] : target: T=[0.164138,0.461024]
    16           31        0.0143141         expand
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.219302, dxRight=0.164824
runMaxwell: T=[0.199735,0.470431] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.218536, dxRight=0.237612
runMaxwell: T=[0.0322374,0.373396] : target: T=[0.164138,0.461024]
    17           33        0.0143141         contract inside
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.186022, dxRight=0.194077
runMaxwell: T=[0.213742,0.473329] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.161508, dxRight=0.173557
runMaxwell: T=[0.307342,0.48502] : target: T=[0.164138,0.461024]
    18           35       0.00485421         reflect
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.17002, dxRight=0.148047
runMaxwell: T=[0.338131,0.490642] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.206407, dxRight=0.215221
runMaxwell: T=[0.10896,0.436509] : target: T=[0.164138,0.461024]
    19           37       0.00485421         contract inside
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.182149, dxRight=0.170438
runMaxwell: T=[0.273346,0.48141] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.200343, dxRight=0.204025
runMaxwell: T=[0.153053,0.457001] : target: T=[0.164138,0.461024]
    20           39       0.00485421         contract inside
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.18383, dxRight=0.20652
runMaxwell: T=[0.188464,0.467793] : target: T=[0.164138,0.461024]
runMaxwell: caseName=lens, RENGERATE THE GRID dxLeft=-0.197858, dxRight=0.195317
runMaxwell: T=[0.181322,0.466346] : target: T=[0.164138,0.461024]
    21           41       0.00485421         contract inside

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 5.000000e-02
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-03

...DONE fminsearch: x=[0.0139783,-0.00592285], fval=0.00485421
optimizer: save plot file=[lensFminsearchConvergence.eps]
done
```

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
 x0Default=-1234567.;
 x0 = x0Default;     % initial guess  set to a real number to change the default initial guess

  % --- 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 );
    x0            =   getReal( line,'-x0',x0 );
    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, \n objective=%s (targetFile=%s, tolFun=%g, tolX=%g), infoLevel=%d, \n
   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);


iteration=0;      % keeps track of how many times runMaxwell is called.
maxIterations=10;

% ------------------------------------------------
%   Define an objective function for minimization
% ------------------------------------------------
function ff = cgmxFunction( xx )
  % fprintf(' objectiveFunction: x=%g\n',xx(1));
  % fprintf(' objectiveFunction: size(xx,2)=%d\n',size(xx,2));

  if( strcmp(objective,'targetTransmission') )

    % -- LENS: for now just adjust the right control point:
    par(1)=kx;
    par(2)=eps1;

    if( size(xx,2)==1 )
      % shift right control point
      dxLeft =-.2; dxRight=.2 + xx(1);
    else
      % shift left and right control points
      dxLeft =-.2 + xx(1); dxRight=.2 + xx(2);
    end

    par(3)=dxLeft;    % shift left control point
    par(4)=dxRight;   % shift right control point

  elseif( strcmp(objective,'minimizeReflection') )

    if( strcmp(caseName,'blockWidth') )
      blockWidth=xx(1);
    else
      eps1=xx(1);
    end
    par(1)=kx;
    par(2)=eps1;
    par(3)=blockWidth;

  else

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



  [ values ]  = runMaxwell( caseName,tFinal,probeType,gridFactor,infoLevel,plotOption, par );
  ff=values(1); % reflection coefficient

end

% ff = @(x) x(1)^2;
ff = @(x) cgmxFunction( x ); % for some reason we need to call cgMxFunction this way

if( strncmp(method,'fminsearch',length('fminsearch')) )

  % ---- Find minium using fminsearch: ----

  options = optimset('Display','iter','TolFun',tolFun, 'TolX',tolX,'PlotFcns',@optimplotfval);

  % ------ INITIAL GUESS FOR FMINSEARCH -----
  if( strcmp(objective,'targetTransmission') )
```

```
136
137       % --- Adjust the shape of the lens:
138
139       if( x0 == x0Default  )
140         x0 = [ -.05, .05  ]; % initial guess for dxRight (offset from 'exact')
141       else
142         x0 = [ x0, -x0 ];  % user specified initial guess
143       end;
144
145     elseif( strcmp(objective,'minimizeReflection') )
146
147       if( strcmp(caseName,'blockWidth') )
148         x0=[ blockWidth ]; %  blockWidth
149       else
150         x0=[ 3.]; %  3. ,  8. initial guess
151       end;
152     else
153
154       fprintf('optimizer: ERROR: unknown objective =[%s]\n',objective);
155       pause; pause;
156     end;
157
158     % fval = myObjectiveFunction( x0 );
159     % fprintf('myObjectiveFunction: x0=%g, f=%g\n',x0(1),fval);
160
161     fprintf('call fminsearch...\n');
162     figure(2);
163
164     % [x,fval] = fminsearch(myObjectiveFunction,x0,options);
165     [x,fval] = fminsearch(ff,x0,options);
166
167     if( size(x,2)==1 )
168       fprintf('...DONE fminsearch: x=%g, fval=%g\n',x(1),fval);
169     else
170       fprintf('...DONE fminsearch: x=[%g,%g], fval=%g\n',x(1),x(2),fval);
171     end;
172     if( plotOption > 0  )
173       grid on; set(gca,'FontSize',fontSize);
174       plotFileName=sprintf('%sFminsearchConvergence.eps',caseName);
175       fprintf('optimizer: save plot file=[%s]\n',plotFileName);
176       print('-depsc2',plotFileName); % save as an eps file
177     end
178
179   else
180
181     % -------------------- FAKE OPTIMIZATION LOOP  ------------------------
182     for iter=1:maxIterations
183
184       if( strcmp(caseName,'blockWidth') )
185         kx=1;
186         eps1=4;
187         blockWidthNew = blockWidth+.1*iter;
188         par(1)=kx;
189         par(2)=eps1;
190         par(3)=blockWidthNew
191
192       elseif( strcmp(caseName,'block') )
193         kx=1;
194         eps1= 2+ .5*iter;
195         par(1)=kx;
196         par(2)=eps1;
197         par(3)=blockWidth
198
199       elseif( strcmp(caseName,'lens') )
200
201         % NOTE: for now we just shift one control point at the center of the left and right sides.
202
203         par(1)=kx;
204         par(2)=eps1;
205         dxLeft = -.2 + .025*(iter-1); dxRight=.2 - .025*(iter-1);
206         par(3)=dxLeft;    % shift left control point
207         par(4)=dxRight;   % shift right control point
```

```
208
209       else
210
211         eps1=4;
212         kx = 2 + iter; % incident wave number
213         par(1)=kx;
214         par(2)=eps1;
215         par(3)=blockWidth
216       end;
217
218       [ values ]  = runMaxwell( caseName,tFinal,probeType,gridFactor,infoLevel,plotOption, par );
219
220       fprintf('optimizer: iter=%d: return-values=[%g,%g]\n',iter,values(1),values(2));
221
222       if( 1==1 || ( plotGrid==0 && plotSolution==0) )
223          pause
224       end
225     end; % end for iter
226   end;
227
228 fprintf('done\n');
229
230 end
231
232 % --- Utility functions ---
233
234
235
236
237
238 % Function getReal: read a command line argument for a real variable
239 function [ val ] = getReal( line,name,val)
240  % fprintf('getReal: val=%g line=[%s] name=[%s]\n',val,line,name);
241  if( strncmp(line,strcat(name,'='),length(name)+1) )
242    val = sscanf(line,sprintf('%s=%%e',name));
243    % fprintf('getReal: scan for val=%g\n',val);
244  end
245 end
246
247 % Function getInt: read a command line argument for an integer variable
248 function [ val ] = getInt( line,name,val)
249  if( strncmp(line,strcat(name,'='),length(name)+1) )
250    val = sscanf(line,sprintf('%s=%%d',name));
251  end
252 end
253
254 % Function getString: read a command line argument for a string variable
255 function [ val ] = getString( line,name,val)
256  if( strncmp(line,strcat(name,'='),length(name)+1) )
257    val = sscanf(line,sprintf('%s=%%s',name));
258  end
259 end
```

A.2 runMaxwell.m

Listing 3: runMaxwell.m

```
1  %
2  %  Run the Maxwell Solver CgMx and return the requested results
3  %
4  % Usage:
5  %
6  % Parameters:
7  %   caseName    (input) : ['cyl'|'block'|'blockWidth'|'lens'],
8  %   tFinal      (input) : final time
9  %   probeType   (input) : ['point','transmission']
10 %   gridFactor  (input) : =1,2,4,8 - grid is this much finer than coarsest grid available. Usually dx=1/(10*gridFactor)
11 %   infolevel   (input) :
12 %   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') )

    % -----------------------------------------------------
    % ---- BLOCK: scattering from a dielectric block -----
    % -----------------------------------------------------

    kx=par(1);
    eps1=par(2);

    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);


    titleLabel=sprintf('Block: eps1=%g, kx=%i',eps1,kx);

    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);

    titleLabel=sprintf('Dielectric block: kx=%i, eps1=%g',kx,eps1);

    if( strncmp(probeType,'transmission',length('transmission')) )
      % reflection/transmission probes:
      probeDataFile = 'OptProbe.dat';
    else
      % point probes:
      probeDataFile = 'leftOptProbe.dat';
    end;

  elseif( strcmp(caseName,'blockWidth') )

    % -----------------------------------------------------------------------------
    % ---- BLOCKWIDTH: scattering from a dielectric block that changes width -----
    % -----------------------------------------------------------------------------

    kx=par(1);
    eps1=par(2);
    blockWidth=par(3);

    fprintf('runMaxwell: caseName=%s, RENGERATE THE GRID blockWidth=%g\n',caseName,blockWidth);
    % Note: new name chosen for grid:
```

```
    ogenCommand = sprintf('%s -noplot dielectricBlockGrid2d -prefix=dieBlockOpt -interp=e -order=4 -width=%g -factor=%d >&! ogenOpt.out',...
                          ogen,blockWidth,gridFactor);
    if( infoLevel>0 )  fprintf('Run ogen: %s\n',ogenCommand); end;
    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 blockWidth=%g \n',...
            caseName,tFinal,probeType,eps1,kx,blockWidth);


    titleLabel=sprintf('Block: eps1=%g, kx=%i, width=%g',eps1,kx,blockWidth);

    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);

    titleLabel=sprintf('Dielectric block: kx=%i, eps1=%g, width=%g',kx,eps1,blockWidth);

    if( strncmp(probeType,'transmission',length('transmission')) )
      % reflection/transmission probes:
      probeDataFile = 'OptProbe.dat';
    else
      % point probes:
      probeDataFile = 'leftOptProbe.dat';
    end;

  elseif( strcmp(caseName,'cyl') )

    % ------------------------------------------------
    % 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, RENERATE 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 %g 0 0 0 0 -dxRight=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;


    if( infoLevel>0 ) 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); end;

    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';


  figure(1);
  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);
      if( infoLevel>0 ) fprintf('runMaxwell: save plot: %s\n',plotFileName); end;
      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:

% reflecton data:
reflectionFileName=' ';
reflectionFileName=sprintf('reflection%s',fileName);
if( infoLevel>0 )
  fprintf('ReflectionTransmissionProbe: Read file=[%s]\n',reflectionFileName);
end;
[headers,labels,t,qr] = readProbeFile(reflectionFileName,infoLevel);

% transmission data:
transmissionFileName=sprintf('transmission%s',fileName);
if( infoLevel>0 )
  fprintf('ReflectionTransmissionProbe: Read file=[%s]\n',transmissionFileName);
end;
[headers,labels,t,qt] = readProbeFile(transmissionFileName,infoLevel);

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

% labels

[numColumns,columnLen] = size(labels);
numVars=numColumns-1;   % t is not counted

if( infoLevel>0 )
  fprintf(1,'ReflectionTransmissionProbe: There are %d solution variables in the data.\n',numVars);
end;

cStart=4; % first component

numToPlot=numVars-j+1;

cr=cStart;
```

```
ci=cStart+1;

Rr = qr(:,cr);
Ri = qr(:,ci);

Tr = qt(:,cr);
Ti = qt(:,ci);

% Rnorm = sqrt( qr(:,cr).^2 + qr(:,ci).^2 );
% Tnorm = sqrt( qt(:,cr).^2 + qt(:,ci).^2 );
% rtNorm = Rnorm.^2 + Tnorm.^2;

end




% ---------------  Read and Extract Probe data from a CgMx probe file -----
% Parameters:
%   fileName (input) : name of the 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
%  Ex, Ey, Hz (output) : time sequence probe data
% ------------------------------------------------------------------------------------
function [ t, Ex, Ey, Hz ] = getCgMxProbeData( fileName, referenceFile, infoLevel  )


  %  Read data:
  [headers,labels,t,q] = readProbeFile(fileName,infoLevel);

  [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
```