Parabolic reflector antenna using 2D FDTD code with PML ABC

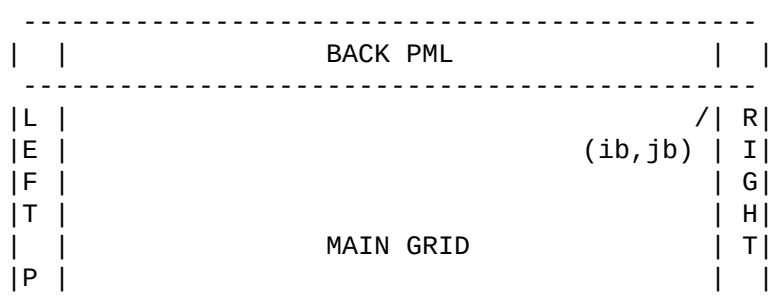
Table of Contents

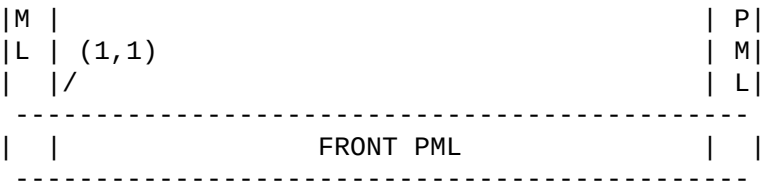
This program is a modified version of 'fdtd2D.m' by Susan Hagness, dated Feb 2000. This modified version was made in Sep 2020. Authored by K. Foutzopoulos kfoutzop@auth.gr.

Theory

This MATLAB M-file implements the finite-difference time-domain solution of Maxwell's curl equations over a two-dimensional Cartesian space lattice comprised of uniform square grid cells.

The computational domain is truncated using the perfectly matched layer (PML) absorbing boundary conditions. The formulation used in this code is based on the original split-field Berenger PML. The PML regions are labeled as shown in the following diagram:





To execute this M-file, type "fdtd2D" at the MATLAB prompt. This M-file displays the FDTD-computed Ex, Ey, and Hz fields at prespecified time intervals (differs from simulation time step).

Code

The annotated source code of the program follows bellow.

```
clear
```

Fundamental constants

The units are in SI. For source excitation frequency, we choose the 1400 MHz band. When the antenna is utilized as a receiver, this band is used for hydrogen line measurements.

```
cc=2.99792458e8;       %speed of light in free space
muz=4.0*pi*1.0e-7;     %permeability of free space
epsz=1.0/(cc*cc*muz);  %permittivity of free space

freq=1.4e+9;           %center frequency of source excitation
lambda=cc/freq;        %center wavelength of source excitation
omega=2.0*pi*freq;
```

Grid parameters

Number of grid cells in x-direction and y-direction.

```
ie=400;
je=400;
```

Auxiliary numbers for grid cells.

```
ib=ie+1;
jb=je+1;
```

Location of z-directed hard source.

```
is=round(ie/4);
js=round(je/2);
```

Space increment of square lattice and time step.

```
dx=lambda/10;
dt=dx/(2.0*cc);
```

Total number of time steps.

```
nmax=800;
```

Thickness of PML regions.

```
iebc=8;                 %left and right
jebc=8;                 %front and back
rmax=0.00001;
orderbc=2;
```

Auxiliary numbers for PML regions.

```
ibbc=iebc+1;
jbbc=jebc+1;
iefbc=ie+2*iebc;
jefbc=je+2*jebc;
ibfbc=iefbc+1;
jbfbc=jefbc+1;
```

Visualization time step set through total frames.

```
frames=10;
visdt=round(nmax/frames);
```

Material parameters

Those define the behavior of the inserted object. Those have been chosen arbitarily to match the behavior of an antenna.

```
eps=[1.0 1.0e+9];
sig=[0.0 0.0];
mur=[1.0 1.0e+9];
sim=[0.0 0.0];
```

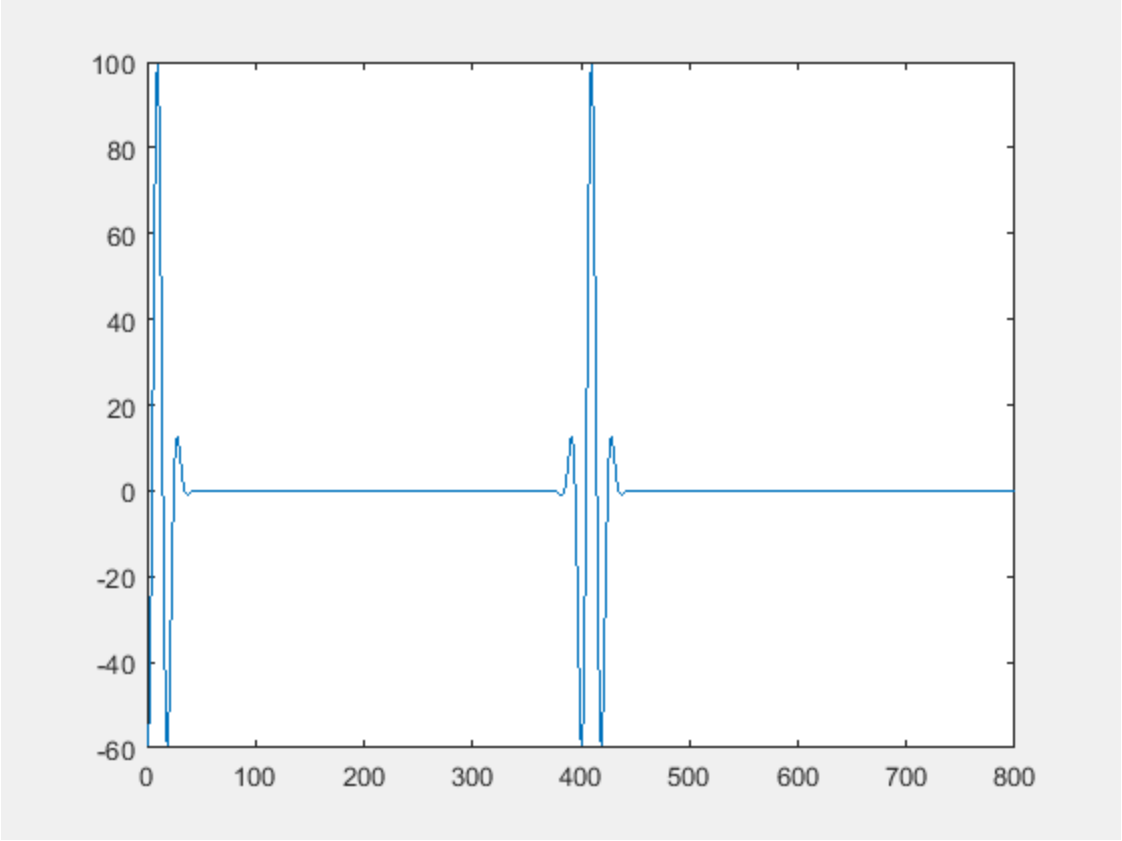
Handle errors if one forgets or adds extra parameters.

```
media=length(eps);
if media~=length(sig) || media~=length(mur) || media~=length(sim)
    disp("Missing or extra material parameters.")
    return
end
```

Wave excitation

Here we consider the source of the excitation. The source (in the program an array) is chosen to be two Gaussian pulses.

```
tc=gauspuls('cutoff',freq,0.8,[],-60);
t=(1:nmax).*dt-dt;
source=1e2*gauspuls(t-tc/4,freq,0.8);

move=nmax*dt/2;
source=source+1e2*gauspuls(t-tc/4-move,freq,0.8);

figure
plot(source)
```



Field arrays

In this section, we preallocate the fields for the grid and PML region. The PML regions are boundaries of the computing grid.

Fields in main grid.

```
ex=zeros(ie,jb);
ey=zeros(ib,je);
hz=zeros(ie,je);
```

Fields in front PML region.

```
exbcf=zeros(iefbc,jebc);
eybcf=zeros(ibfbc,jebc);
hzxbcf=zeros(iefbc,jebc);
hzybcf=zeros(iefbc,jebc);
```

Fields in back PML region.

```
exbcb=zeros(iefbc,jbbc);
eybcb=zeros(ibfbc,jebc);
hzxbcb=zeros(iefbc,jebc);
hzybcb=zeros(iefbc,jebc);
```

Fields in left PML region.

```
exbcl=zeros(iebc,jb);
eybcl=zeros(iebc,je);
hzxbcl=zeros(iebc,je);
hzybcl=zeros(iebc,je);
```

Fields in right PML region.

```
exbcr=zeros(iebc,jb);
eybcr=zeros(ibbc,je);
hzxbcr=zeros(iebc,je);
hzybcr=zeros(iebc,je);
```

Updating coefficients

For performance, similarly as before, we preallocate the coefficients. Not a terrific increase in case of one homogeneous object in free-space.

```
ca=zeros(1,media);
cb=zeros(1,media);
da=zeros(1,media);
db=zeros(1,media);
```

Then we compute their values.

```
for i=1:media
    eaf=dt*sig(i)/(2.0*epsz*eps(i));
    ca(i)=(1.0-eaf)/(1.0+eaf);
    cb(i)=dt/epsz/eps(i)/dx/(1.0+eaf);
    haf=dt*sim(i)/(2.0*muz*mur(i));
    da(i)=(1.0-haf)/(1.0+haf);
    db(i)=dt/muz/mur(i)/dx/(1.0+haf);
end
```

Geometry specification (main grid)

Initialize entire main grid to free space.

```
caex(1:ie,1:jb)=ca(1);
cbex(1:ie,1:jb)=cb(1);
caey(1:ib,1:je)=ca(1);
cbey(1:ib,1:je)=cb(1);
dahz(1:ie,1:je)=da(1);
dbhz(1:ie,1:je)=db(1);
```

Add interference object (a parabolic antenna)

The equation of an horizontal parabola, its vertex and its focus are:

```
      .
     .          (C): x = a (y-k)^2 + h
V .   F
  x x           where:
```

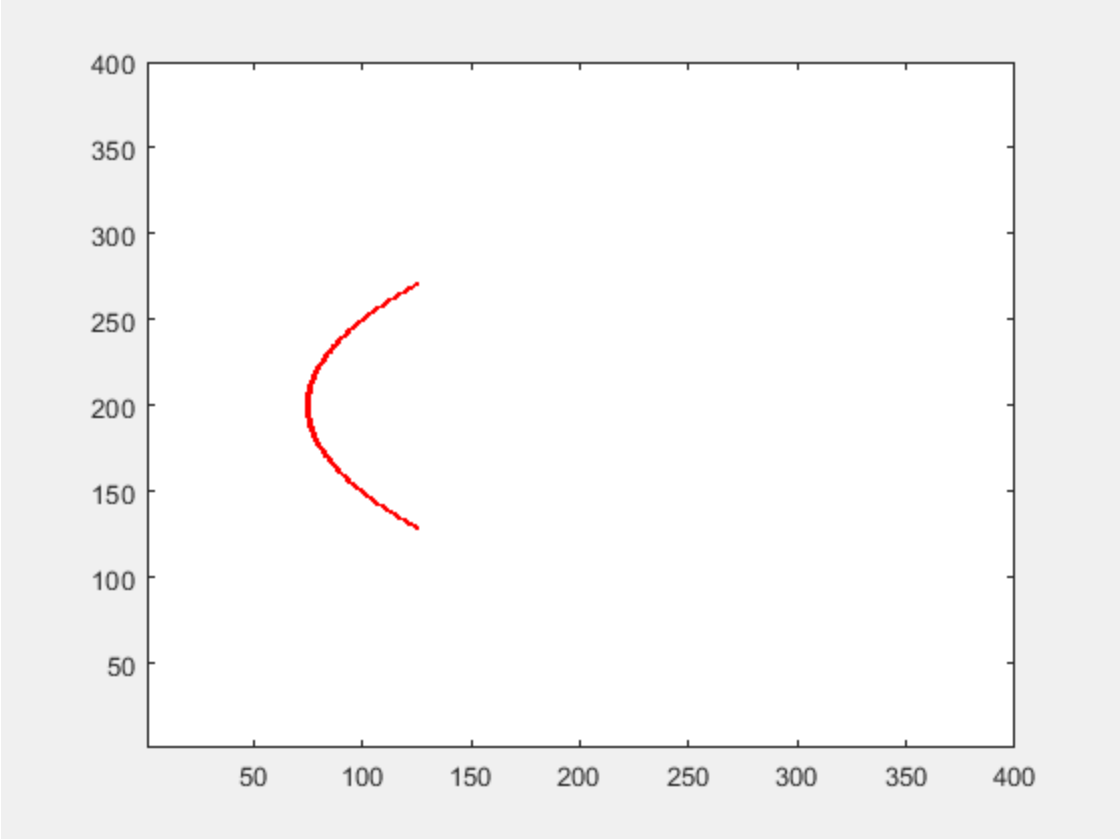
```
              .            V(h, k)
               .           F(h + 1/(4a), k)
                .
         Fig: Parabola with (h,k,a)=(5,4,0.16).
```

The number $|a|$ defines the opening. The larger (smaller) $|a|$ is, the more narrower (wider) is the parabola, or else more stretched (shrinked).

For our problem we need the source to be at the focus, and we take the vertex to be at $(is-dv, js)$ where dv the distance vertex-focus. Thus $is = is - dv + 1/(4a)$ and $js = k$. The parabola whence is defined by the equation $x = 1/(4dv) \cdot (y-js)^2 + is - dv$

We consider the cells satisfying that equation as the antenna. We also create a plot of the object to superimpose on our animation. The commented section is a second flipped antenna discussed at the end.

```
dv=25;
alpha=1/(4*dv);

obj=zeros(ie,je);

for i=1:(is+dv)
    for j=1:je
        x = alpha*(j-js)^2+is-dv;
        if round(x)==i
            caex(i,j)=ca(2);
            caey(i,j)=ca(2);
            cbex(i,j)=cb(2);
            cbey(i,j)=cb(2);
            dahz(i,j)=da(2);
            dbhz(i,j)=db(2);
            obj(i,j)=1.0;
        end
    end
end

% for i=(ie-is-dv):ie
%     for j=1:je
%         x = -alpha*(j-je+js)^2+ie-is+dv;
%         if round(x)==i
%             caex(i,j)=ca(2);
%             caey(i,j)=ca(2);
%             cbex(i,j)=cb(2);
%             cbey(i,j)=cb(2);
%             dahz(i,j)=da(2);
%             dbhz(i,j)=db(2);
%             obj(i,j)=1.0;
%         end
%     end
% end

obj=obj';
figure
contour(obj,'r-','LineWidth',1.0)
```



Fill the PML regions

The algorithms in this section are from Susan Hagness' program. Some preallocations have been added to improve performance.

```
delbc=iebc*dx;
sigmam=-log(rmax/100.0)*epsz*cc*(orderbc+1)/(2*delbc);
bcfactor=eps(1)*sigmam/(dx*(delbc^orderbc)*(orderbc+1));
```

PML front region.

```
caexbcf(1:iefbc,1)=1.0;
cbexbcf(1:iefbc,1)=0.0;
for j=2:jebc
    y1=(jebc-j+1.5)*dx;
    y2=(jebc-j+0.5)*dx;
    sigmay=bcfactor*(y1^(orderbc+1)-y2^(orderbc+1));
    ca1=exp(-sigmay*dt/(epsz*eps(1)));
    cb1=(1.0-ca1)/(sigmay*dx);
    caexbcf(1:iefbc,j)=ca1;
    cbexbcf(1:iefbc,j)=cb1;
end
sigmay = bcfactor*(0.5*dx)^(orderbc+1);
ca1=exp(-sigmay*dt/(epsz*eps(1)));
cb1=(1-ca1)/(sigmay*dx);
caex(1:ie,1)=ca1;
```

```
cbex(1:ie,1)=cb1;
caexbcl(1:iebc,1)=ca1;
cbexbcl(1:iebc,1)=cb1;
caexbcr(1:iebc,1)=ca1;
cbexbcr(1:iebc,1)=cb1;

% preallocate
dahzybcf=zeros(iefbc,jebc);
dbhzybcf=zeros(iefbc,jebc);
caeybcf=zeros(ibfbc,jebc);
cbeybcf=zeros(ibfbc,jebc);
dahzxbcf=zeros(iefbc,jebc);
dbhzxbcf=zeros(iefbc,jebc);

for j=1:jebc
    y1=(jebc-j+1)*dx;
    y2=(jebc-j)*dx;
    sigmay=bcfactor*(y1^(orderbc+1)-y2^(orderbc+1));
    sigmays=sigmay*(muz/(epsz*eps(1)));
    da1=exp(-sigmays*dt/muz);
    db1=(1-da1)/(sigmays*dx);
    dahzybcf(1:iefbc,j)=da1;
    dbhzybcf(1:iefbc,j)=db1;
    caeybcf(1:ibfbc,j)=ca(1);
    cbeybcf(1:ibfbc,j)=cb(1);
    dahzxbcf(1:iefbc,j)=da(1);
    dbhzxbcf(1:iefbc,j)=db(1);
end
```

PML back region.

```
caexbcb(1:iefbc,jbbc)=1.0;
cbexbcb(1:iefbc,jbbc)=0.0;
for j=2:jebc
    y1=(j-0.5)*dx;
    y2=(j-1.5)*dx;
    sigmay=bcfactor*(y1^(orderbc+1)-y2^(orderbc+1));
    ca1=exp(-sigmay*dt/(epsz*eps(1)));
    cb1=(1-ca1)/(sigmay*dx);
    caexbcb(1:iefbc,j)=ca1;
    cbexbcb(1:iefbc,j)=cb1;
end
sigmay = bcfactor*(0.5*dx)^(orderbc+1);
ca1=exp(-sigmay*dt/(epsz*eps(1)));
cb1=(1-ca1)/(sigmay*dx);
caex(1:ie,jb)=ca1;
cbex(1:ie,jb)=cb1;
caexbcl(1:iebc,jb)=ca1;
cbexbcl(1:iebc,jb)=cb1;
caexbcr(1:iebc,jb)=ca1;
cbexbcr(1:iebc,jb)=cb1;

% preallocate
dahzybcb=zeros(iefbc,jebc);
```

```
dbhzybcb=zeros(iefbc,jebc);
caeybcb=zeros(ibfbc,jebc);
cbeybcb=zeros(ibfbc,jebc);
dahzxbcb=zeros(iefbc,jebc);
dbhzxbcb=zeros(iefbc,jebc);

for j=1:jebc
    y1=j*dx;
    y2=(j-1)*dx;
    sigmay=bcfactor*(y1^(orderbc+1)-y2^(orderbc+1));
    sigmays=sigmay*(muz/(epsz*eps(1)));
    da1=exp(-sigmays*dt/muz);
    db1=(1-da1)/(sigmays*dx);
    dahzybcb(1:iefbc,j)=da1;
    dbhzybcb(1:iefbc,j)=db1;
    caeybcb(1:ibfbc,j)=ca(1);
    cbeybcb(1:ibfbc,j)=cb(1);
    dahzxbcb(1:iefbc,j)=da(1);
    dbhzxbcb(1:iefbc,j)=db(1);
end
```

PML left region.

```
caeybcl(1,1:je)=1.0;
cbeybcl(1,1:je)=0.0;
for i=2:iebc
    x1=(iebc-i+1.5)*dx;
    x2=(iebc-i+0.5)*dx;
    sigmax=bcfactor*(x1^(orderbc+1)-x2^(orderbc+1));
    ca1=exp(-sigmax*dt/(epsz*eps(1)));
    cb1=(1-ca1)/(sigmax*dx);
    caeybcl(i,1:je)=ca1;
    cbeybcl(i,1:je)=cb1;
    caeybcf(i,1:jebc)=ca1;
    cbeybcf(i,1:jebc)=cb1;
    caeybcb(i,1:jebc)=ca1;
    cbeybcb(i,1:jebc)=cb1;
end
sigmax=bcfactor*(0.5*dx)^(orderbc+1);
ca1=exp(-sigmax*dt/(epsz*eps(1)));
cb1=(1-ca1)/(sigmax*dx);
caey(1,1:je)=ca1;
cbey(1,1:je)=cb1;
caeybcf(iebc+1,1:jebc)=ca1;
cbeybcf(iebc+1,1:jebc)=cb1;
caeybcb(iebc+1,1:jebc)=ca1;
cbeybcb(iebc+1,1:jebc)=cb1;

% preallocate
dahzxbcl=zeros(iebc,je);
dbhzxbcl=zeros(iebc,je);
dahzybcl=zeros(iebc,je);
dbhzybcl=zeros(iebc,je);
```

```
for i=1:iebc
    x1=(iebc-i+1)*dx;
    x2=(iebc-i)*dx;
    sigmax=bcfactor*(x1^(orderbc+1)-x2^(orderbc+1));
    sigmaxs=sigmax*(muz/(epsz*eps(1)));
    da1=exp(-sigmaxs*dt/muz);
    db1=(1-da1)/(sigmaxs*dx);
    dahzxbcl(i,1:je)=da1;
    dbhzxbcl(i,1:je)=db1;
    dahzxbcf(i,1:jebc)=da1;
    dbhzxbcf(i,1:jebc)=db1;
    dahzxbcb(i,1:jebc)=da1;
    dbhzxbcb(i,1:jebc)=db1;
    caexbcl(i,2:je)=ca(1);
    cbexbcl(i,2:je)=cb(1);
    dahzybcl(i,1:je)=da(1);
    dbhzybcl(i,1:je)=db(1);
end
```

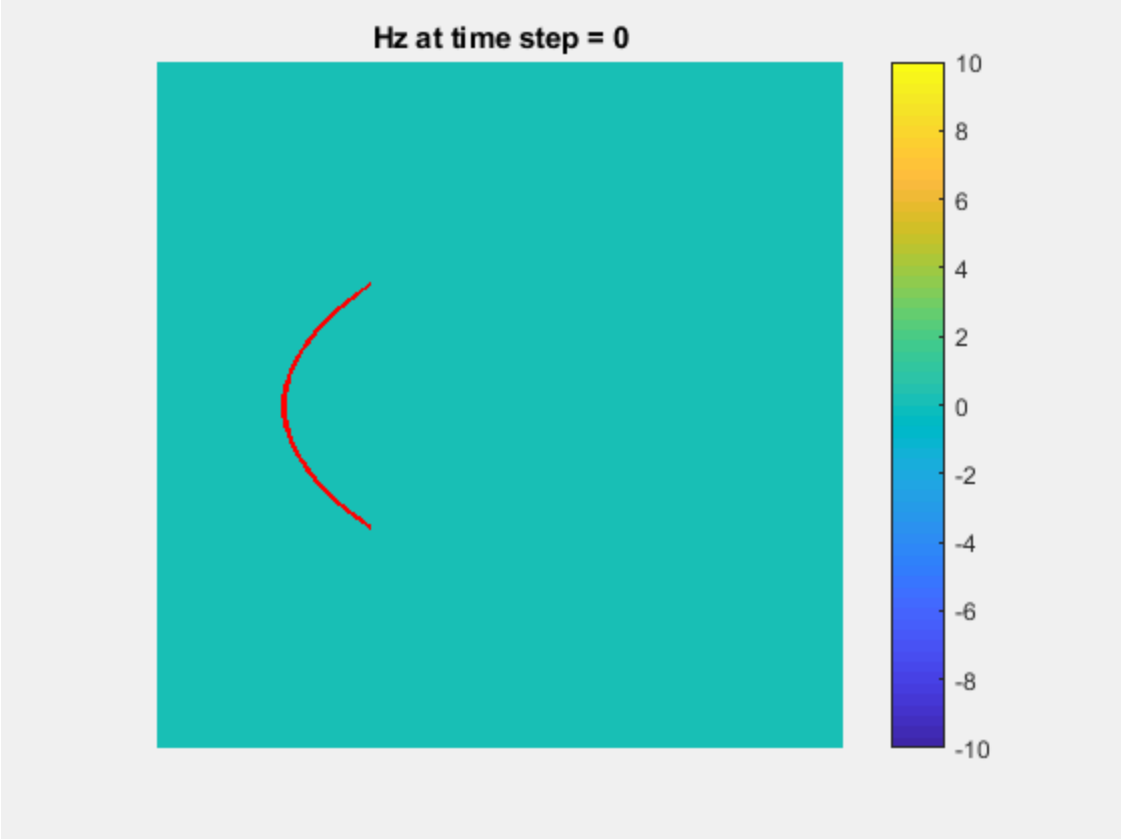
PML right region.

```
caeybcr(ibbc,1:je)=1.0;
cbeybcr(ibbc,1:je)=0.0;
for i=2:iebc
    x1=(i-0.5)*dx;
    x2=(i-1.5)*dx;
    sigmax=bcfactor*(x1^(orderbc+1)-x2^(orderbc+1));
    ca1=exp(-sigmax*dt/(epsz*eps(1)));
    cb1=(1-ca1)/(sigmax*dx);
    caeybcr(i,1:je)=ca1;
    cbeybcr(i,1:je)=cb1;
    caeybcf(i+iebc+ie,1:jebc)=ca1;
    cbeybcf(i+iebc+ie,1:jebc)=cb1;
    caeybcb(i+iebc+ie,1:jebc)=ca1;
    cbeybcb(i+iebc+ie,1:jebc)=cb1;
end
sigmax=bcfactor*(0.5*dx)^(orderbc+1);
ca1=exp(-sigmax*dt/(epsz*eps(1)));
cb1=(1-ca1)/(sigmax*dx);
caey(ib,1:je)=ca1;
cbey(ib,1:je)=cb1;
caeybcf(iebc+ib,1:jebc)=ca1;
cbeybcf(iebc+ib,1:jebc)=cb1;
caeybcb(iebc+ib,1:jebc)=ca1;
cbeybcb(iebc+ib,1:jebc)=cb1;

% preallocate
dahzxbcr=zeros(iebc,je);
dbhzxbcr=zeros(iebc,je);
dahzybcr=zeros(iebc,je);
dbhzybcr=zeros(iebc,je);

for i=1:iebc
    x1=i*dx;
```

```
    x2=(i-1)*dx;
    sigmax=bcfactor*(x1^(orderbc+1)-x2^(orderbc+1));
    sigmaxs=sigmax*(muz/(epsz*eps(1)));
    da1=exp(-sigmaxs*dt/muz);
    db1=(1-da1)/(sigmaxs*dx);
    dahzxbcr(i,1:je) = da1;
    dbhzxbcr(i,1:je) = db1;
    dahzxbcf(i+ie+iebc,1:jebc)=da1;
    dbhzxbcf(i+ie+iebc,1:jebc)=db1;
    dahzxbcb(i+ie+iebc,1:jebc)=da1;
    dbhzxbcb(i+ie+iebc,1:jebc)=db1;
    caexbcr(i,2:je)=ca(1);
    cbexbcr(i,2:je)=cb(1);
    dahzybcr(i,1:je)=da(1);
    dbhzybcr(i,1:je)=db(1);
end
```

Visualization initialization

```
caxis_limits=[-10 10];

figure
pcolor(hz')
shading flat
caxis(caxis_limits)
axis([1 ie 1 je])
colorbar
axis image
axis off
title('Hz at time step = 0')
hold on
contour(obj,'r-','LineWidth',1.0)
hold off

rect=get(gcf,'Position');
rect(1:2)=[0 0];
```



Time-stepping loop

This is the main loop of the program; the time advancement. The algorithms in this section are verbatim from Susan Hagness' program. As they're used in vectorized form caution should be taken, for indices to match the PML algorithm.

```
j=0;

for n=1:nmax
```

Update electric fields (EX and EY) in main grid

```
ex(:,2:je)=caex(:,2:je).*ex(:,2:je)+...
    cbex(:,2:je).*(hz(:,2:je)-hz(:,1:je-1));
ey(2:ie,:)=caey(2:ie,:).*ey(2:ie,:)+...
    cbey(2:ie,:).*(hz(1:ie-1,:)-hz(2:ie,:));
```

Update EX in PML regions

```
% FRONT
exbcf(:,2:jebc)=caexbcf(:,2:jebc).*exbcf(:,2:jebc)-...
    cbexbcf(:,2:jebc).*(hzxbcf(:,1:jebc-1)+hzybcf(:,1:jebc-1)-...
    hzxbcf(:,2:jebc)-hzybcf(:,2:jebc));
ex(1:ie,1)=caex(1:ie,1).*ex(1:ie,1)-...
```

```
    cbex(1:ie,1).*(hzxbcf(ibbc:iebc+ie,jebc)+...
    hzybcf(ibbc:iebc+ie,jebc)-hz(1:ie,1));
% BACK
exbcb(:,2:jebc)=caexbcb(:,2:jebc).*exbcb(:,2:jebc)-...
    cbexbcb(:,2:jebc).*(hzxbcb(:,1:jebc-1)+hzybcb(:,1:jebc-1)-...
    hzxbcb(:,2:jebc)-hzybcb(:,2:jebc));
ex(1:ie,jb)=caex(1:ie,jb).*ex(1:ie,jb)-...
    cbex(1:ie,jb).*(hz(1:ie,jb-1)-hzxbcb(ibbc:iebc+ie,1)-...
    hzybcb(ibbc:iebc+ie,1));
% LEFT
exbcl(:,2:je)=caexbcl(:,2:je).*exbcl(:,2:je)-...
    cbexbcl(:,2:je).*(hzxbcl(:,1:je-1)+hzybcl(:,1:je-1)-...
    hzxbcl(:,2:je)-hzybcl(:,2:je));
exbcl(:,1)=caexbcl(:,1).*exbcl(:,1)-...
    cbexbcl(:,1).*(hzxbcf(1:iebc,jebc)+hzybcf(1:iebc,jebc)-...
    hzxbcl(:,1)-hzybcl(:,1));
exbcl(:,jb)=caexbcl(:,jb).*exbcl(:,jb)-...
    cbexbcl(:,jb).*(hzxbcl(:,je)+hzybcl(:,je)-...
    hzxbcb(1:iebc,1)-hzybcb(1:iebc,1));
% RIGHT
exbcr(:,2:je)=caexbcr(:,2:je).*exbcr(:,2:je)-...
    cbexbcr(:,2:je).*(hzxbcr(:,1:je-1)+hzybcr(:,1:je-1)-...
    hzxbcr(:,2:je)-hzybcr(:,2:je));
exbcr(:,1)=caexbcr(:,1).*exbcr(:,1)-...
    cbexbcr(:,1).*(hzxbcf(1+iebc+ie:iefbc,jebc)+...
    hzybcf(1+iebc+ie:iefbc,jebc)-...
    hzxbcr(:,1)-hzybcr(:,1));
exbcr(:,jb)=caexbcr(:,jb).*exbcr(:,jb)-...
    cbexbcr(:,jb).*(hzxbcr(:,je)+hzybcr(:,je)-...
    hzxbcb(1+iebc+ie:iefbc,1)-...
    hzybcb(1+iebc+ie:iefbc,1));
```

Update EY in PML regions

```
% FRONT
eybcf(2:iefbc,:)=caeybcf(2:iefbc,:).*eybcf(2:iefbc,:)-...
    cbeybcf(2:iefbc,:).*(hzxbcf(2:iefbc,:)+hzybcf(2:iefbc,:)-...
    hzxbcf(1:iefbc-1,:)-hzybcf(1:iefbc-1,:));
% BACK
eybcb(2:iefbc,:)=caeybcb(2:iefbc,:).*eybcb(2:iefbc,:)-...
    cbeybcb(2:iefbc,:).*(hzxbcb(2:iefbc,:)+hzybcb(2:iefbc,:)-...
    hzxbcb(1:iefbc-1,:)-hzybcb(1:iefbc-1,:));
% LEFT
eybcl(2:iebc,:)=caeybcl(2:iebc,:).*eybcl(2:iebc,:)-...
    cbeybcl(2:iebc,:).*(hzxbcl(2:iebc,:)+hzybcl(2:iebc,:)-...
    hzxbcl(1:iebc-1,:)-hzybcl(1:iebc-1,:));
ey(1,:)=caey(1,:).*ey(1,:)-...
    cbey(1,:).*(hz(1,:)-hzxbcl(iebc,:)-hzybcl(iebc,:));
% RIGHT
eybcr(2:iebc,:)=caeybcr(2:iebc,:).*eybcr(2:iebc,:)-...
    cbeybcr(2:iebc,:).*(hzxbcr(2:iebc,:)+hzybcr(2:iebc,:)-...
    hzxbcr(1:iebc-1,:)-hzybcr(1:iebc-1,:));
ey(ib,:)=caey(ib,:).*ey(ib,:)-...
```

```
    cbey(ib,:).*(hzxbcr(1,:)+hzybcr(1,:)- hz(ie,:));
```

Update magnetic fields (HZ) in main grid

```
hz(1:ie,1:je)=dahz(1:ie,1:je).*hz(1:ie,1:je)+...
    dbhz(1:ie,1:je).*(ex(1:ie,2:jb)-ex(1:ie,1:je)+...
    ey(1:ie,1:je)-ey(2:ib,1:je));
hz(is,js)=source(n);
```

Update HZX in PML regions

```
% FRONT
hzxbcf(1:iefbc,:)=dahzxbcf(1:iefbc,:).*hzxbcf(1:iefbc,:)-...
    dbhzxbcf(1:iefbc,:).*(eybcf(2:ibfbc,:)-eybcf(1:iefbc,:));
% BACK
hzxbcb(1:iefbc,:)=dahzxbcb(1:iefbc,:).*hzxbcb(1:iefbc,:)-...
    dbhzxbcb(1:iefbc,:).*(eybcb(2:ibfbc,:)-eybcb(1:iefbc,:));
% LEFT
hzxbcl(1:iebc-1,:)=dahzxbcl(1:iebc-1,:).*hzxbcl(1:iebc-1,:)-...
    dbhzxbcl(1:iebc-1,:).*(eybcl(2:iebc,:)-eybcl(1:iebc-1,:));
hzxbcl(iebc,:)=dahzxbcl(iebc,:).*hzxbcl(iebc,:)-...
    dbhzxbcl(iebc,:).*(ey(1,:)-eybcl(iebc,:));
% RIGHT
hzxbcr(2:iebc,:)=dahzxbcr(2:iebc,:).*hzxbcr(2:iebc,:)-...
    dbhzxbcr(2:iebc,:).*(eybcr(3:ibbc,:)-eybcr(2:iebc,:));
hzxbcr(1,:)=dahzxbcr(1,:).*hzxbcr(1,:)-...
    dbhzxbcr(1,:).*(eybcr(2,:)-ey(ib,:));
```

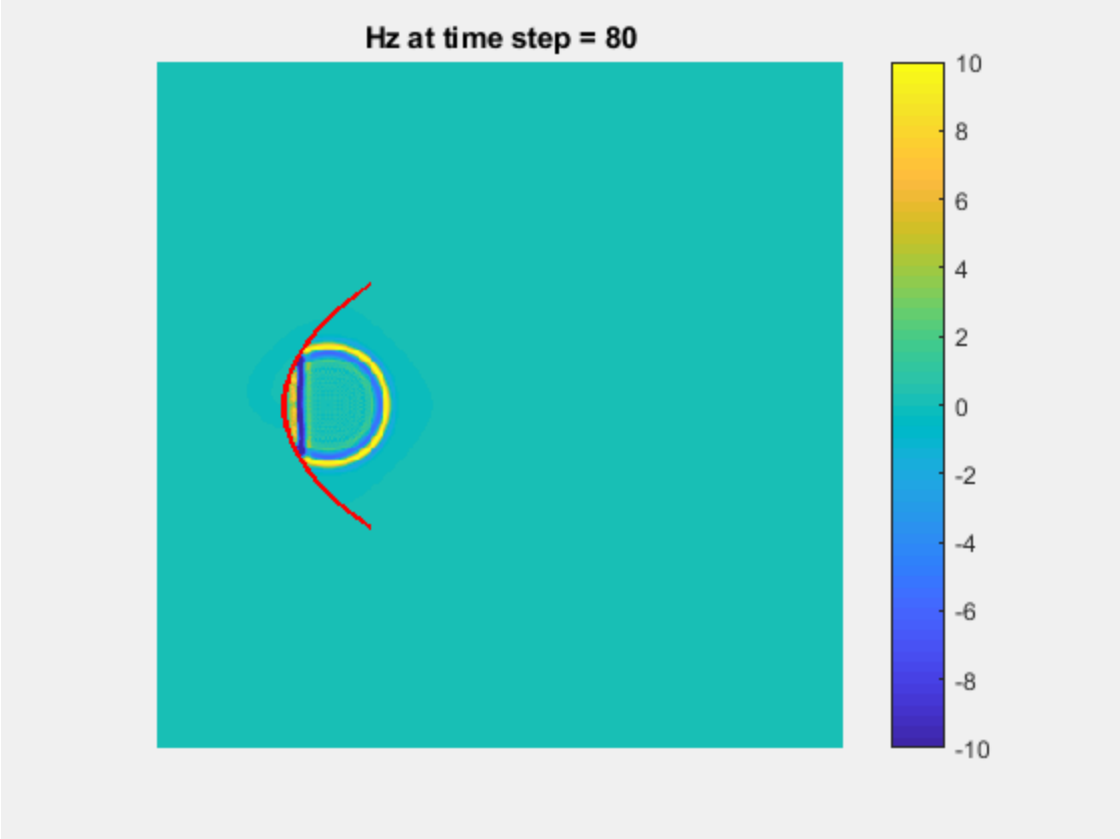
Update HZY in PML regions

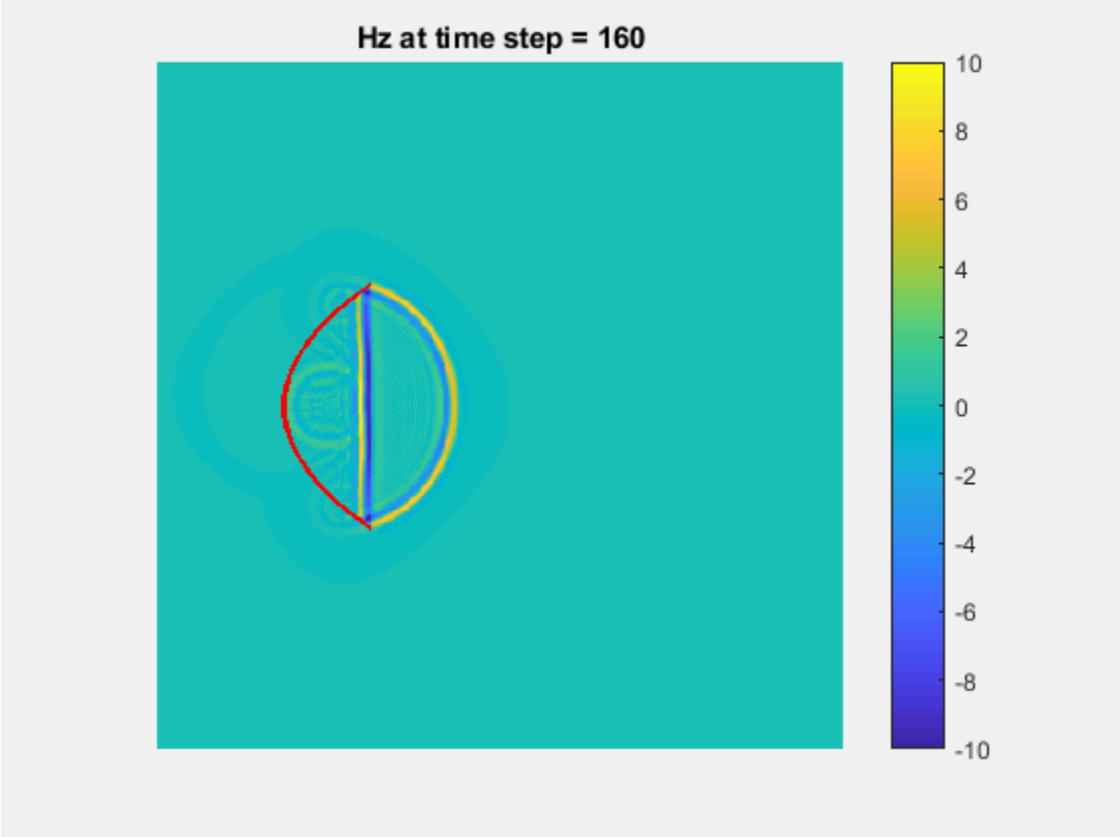
```
% FRONT
hzybcf(:,1:jebc-1)=dahzybcf(:,1:jebc-1).*hzybcf(:,1:jebc-1)-...
    dbhzybcf(:,1:jebc-1).*(exbcf(:,1:jebc-1)-exbcf(:,2:jebc));
hzybcf(1:iebc,jebc)=dahzybcf(1:iebc,jebc).*hzybcf(1:iebc,jebc)-...
    dbhzybcf(1:iebc,jebc).*(exbcf(1:iebc,jebc)-exbcl(1:iebc,1));
hzybcf(iebc+1:iebc+ie,jebc)=...
    dahzybcf(iebc+1:iebc+ie,jebc).*hzybcf(iebc+1:iebc+ie,jebc)-...
    dbhzybcf(iebc+1:iebc+ie,jebc).*(exbcf(iebc+1:iebc+ie,jebc)-...
    ex(1:ie,1));
hzybcf(iebc+ie+1:iefbc,jebc)=...
    dahzybcf(iebc+ie+1:iefbc,jebc).*hzybcf(iebc+ie
+1:iefbc,jebc)-...
    dbhzybcf(iebc+ie+1:iefbc,jebc).*(exbcf(iebc+ie
+1:iefbc,jebc)-...
    exbcr(1:iebc,1));
% BACK

 hzybcb(1:iefbc,2:jebc)=dahzybcb(1:iefbc,2:jebc).*hzybcb(1:iefbc,2:jebc)-...
    dbhzybcb(1:iefbc,2:jebc).*(exbcb(1:iefbc,2:jebc)-
exbcb(1:iefbc,3:jbbc));
hzybcb(1:iebc,1)=dahzybcb(1:iebc,1).*hzybcb(1:iebc,1)-...
```

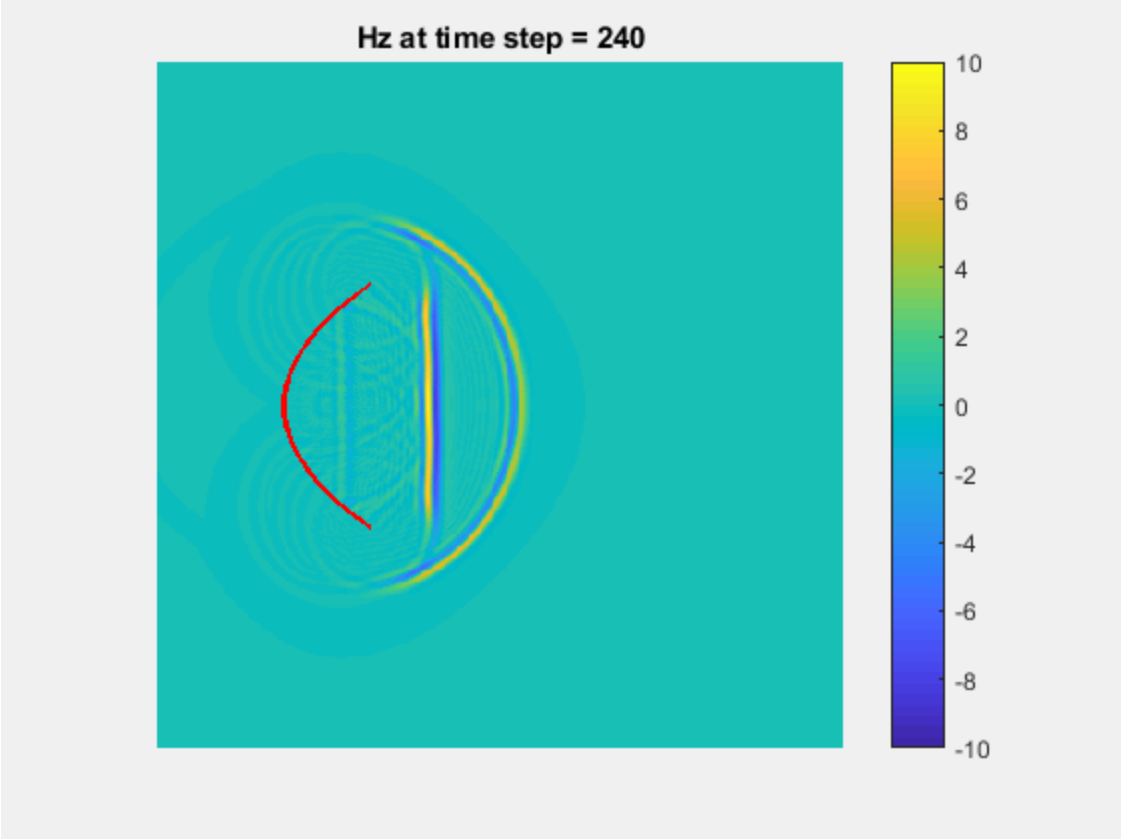
```
        dbhzybcb(1:iebc,1).*(exbcl(1:iebc,jb)-exbcb(1:iebc,2));
    hzybcb(iebc+1:iebc+ie,1)=...
        dahzybcb(iebc+1:iebc+ie,1).*hzybcb(iebc+1:iebc+ie,1)-...
        dbhzybcb(iebc+1:iebc+ie,1).*(ex(1:ie,jb)-exbcb(iebc+1:iebc
+ie,2));
    hzybcb(iebc+ie+1:iefbc,1)=...
        dahzybcb(iebc+ie+1:iefbc,1).*hzybcb(iebc+ie+1:iefbc,1)-...
        dbhzybcb(iebc+ie+1:iefbc,1).*(exbcr(1:iebc,jb)-...
        exbcb(iebc+ie+1:iefbc,2));
    % LEFT
    hzybcl(:,1:je)=dahzybcl(:,1:je).*hzybcl(:,1:je)-...
        dbhzybcl(:,1:je).*(exbcl(:,1:je)-exbcl(:,2:jb));
    % RIGHT
    hzybcr(:,1:je)=dahzybcr(:,1:je).*hzybcr(:,1:je)-...
        dbhzybcr(:,1:je).*(exbcr(:,1:je)-exbcr(:,2:jb));
```

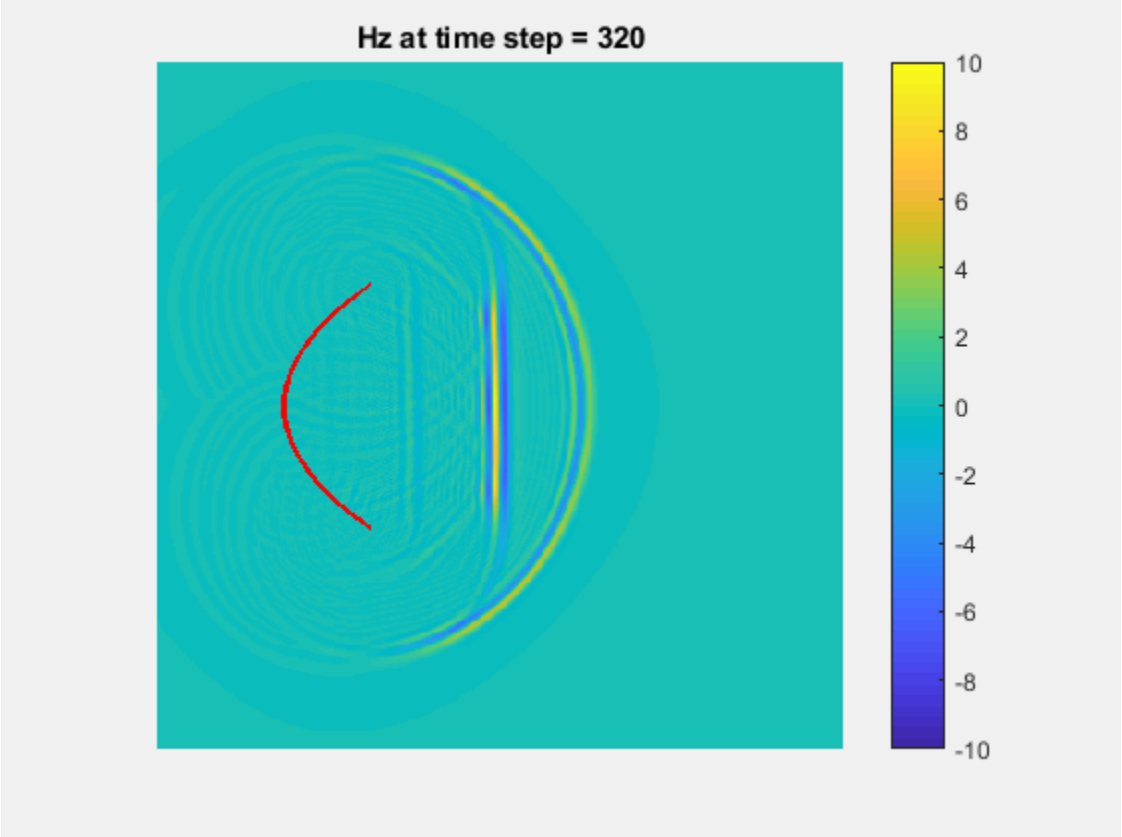
Visualize fields

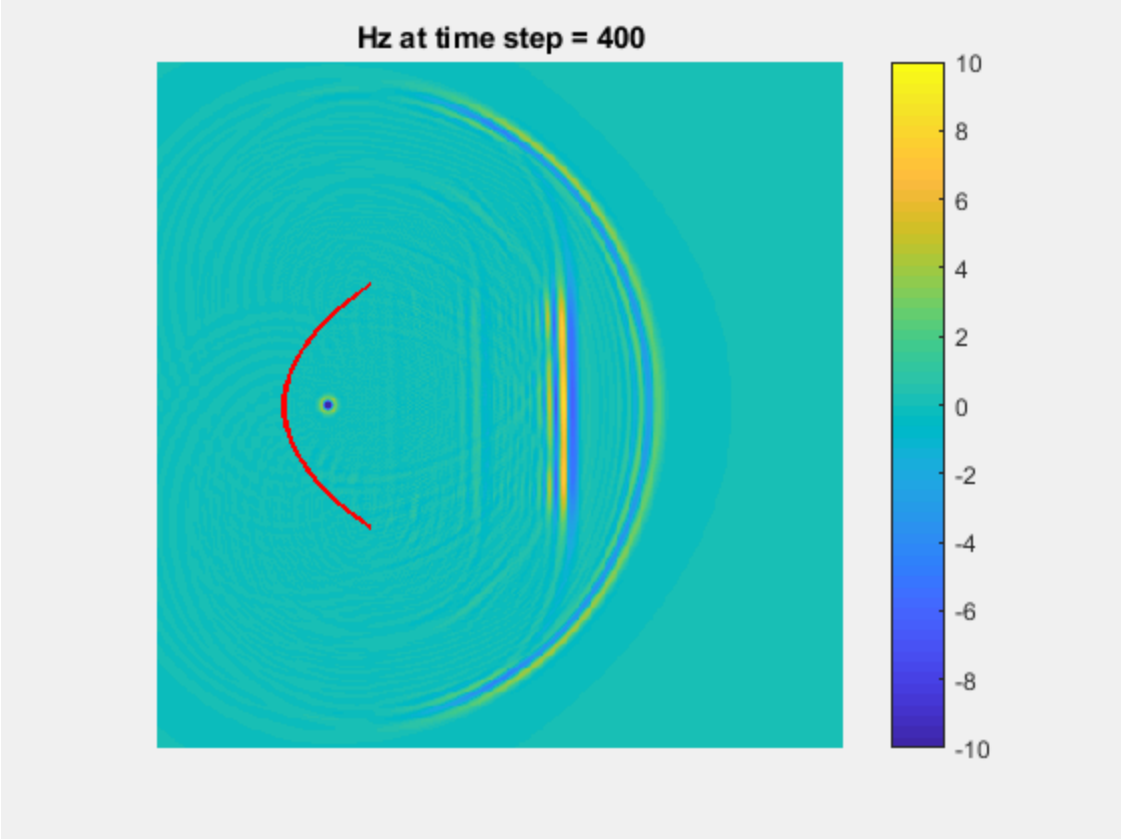
```
    if mod(n,visdt)==0
        timestep=int2str(n);

        pcolor(hz')
        shading flat
        caxis(caxis_limits)
        axis([1 ie 1 je])
        colorbar
        axis image
        axis off
        title(['Hz at time step = ',timestep])
        hold on
        contour(obj,'r-','LineWidth',1.0)
        hold off

        % MATLAB specifies that a frames vector shouldn't be
 preallocated.
        j=j+1;
        M(j)=getframe(gcf,rect); %#ok<SAGROW>
    end
```

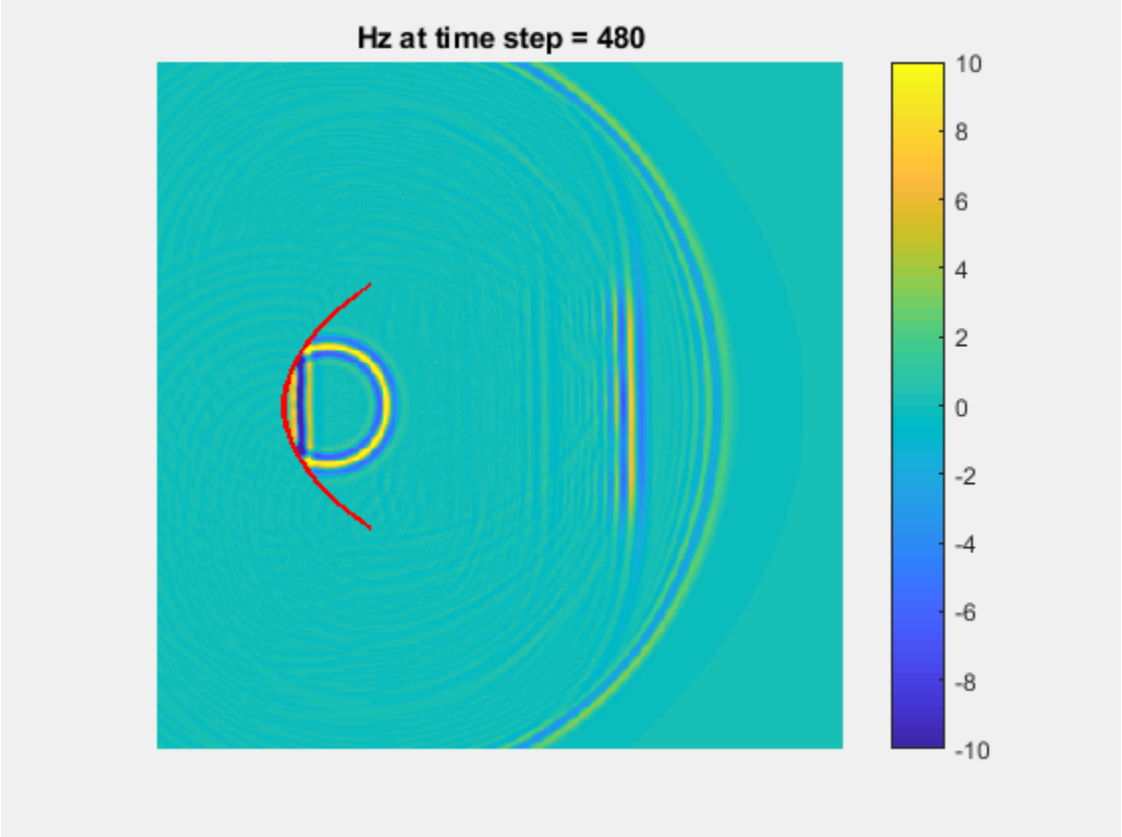

Hz at time step = 80
10
8
6
4
2
0
-2
-4
-6
-8
-10

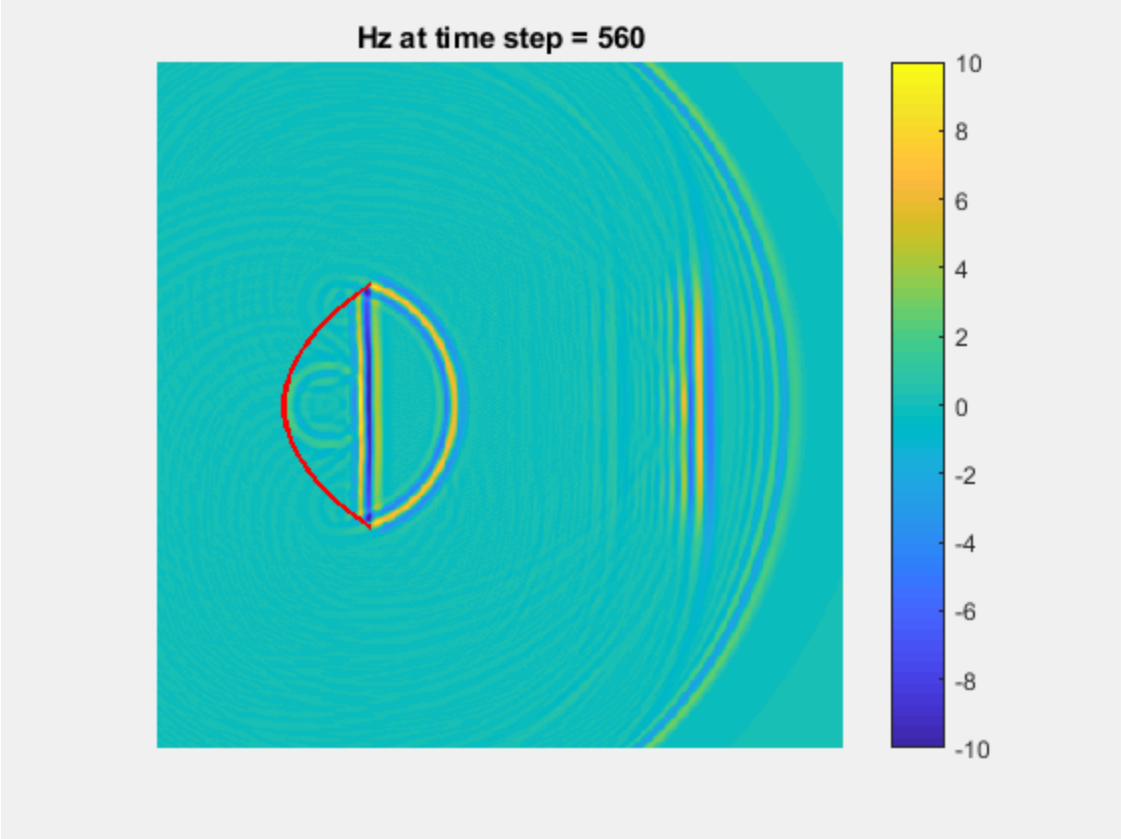

Hz at time step = 160
10
8
6
4
2
0
-2
-4
-6
-8
-10

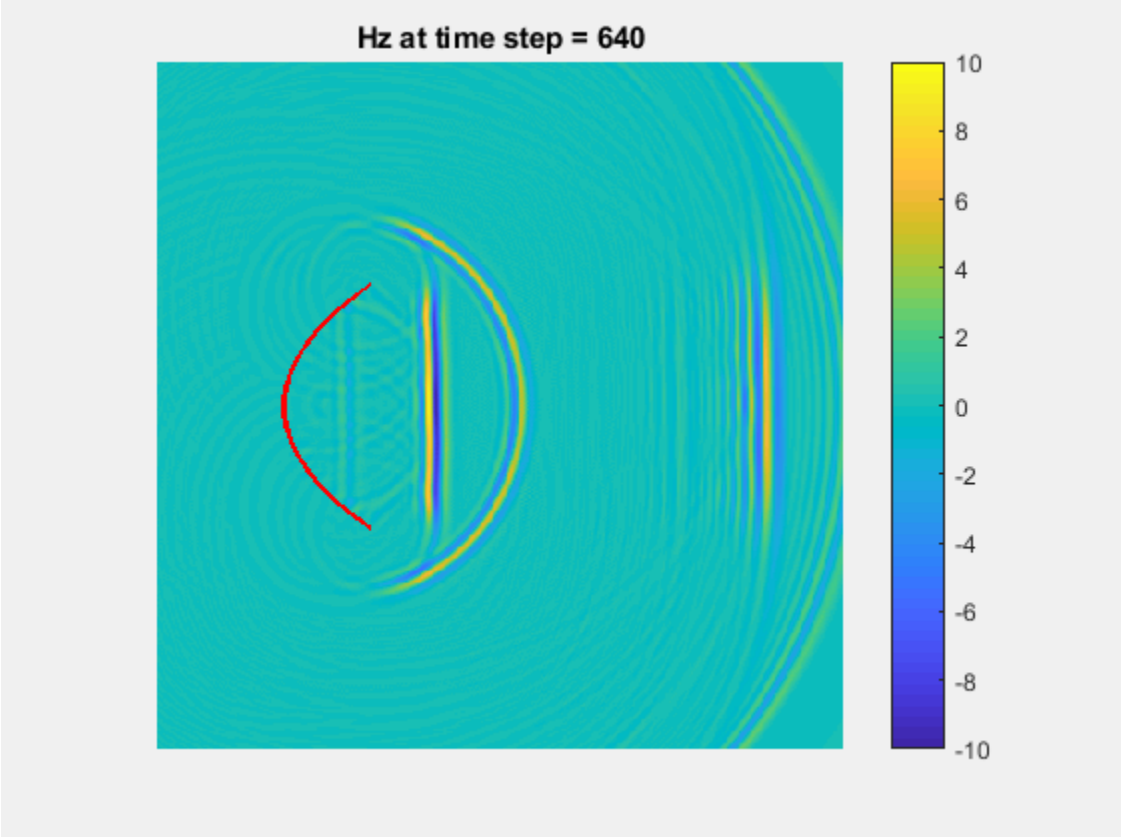

Hz at time step = 240
10
8
6
4
2
0
-2
-4
-6
-8
-10

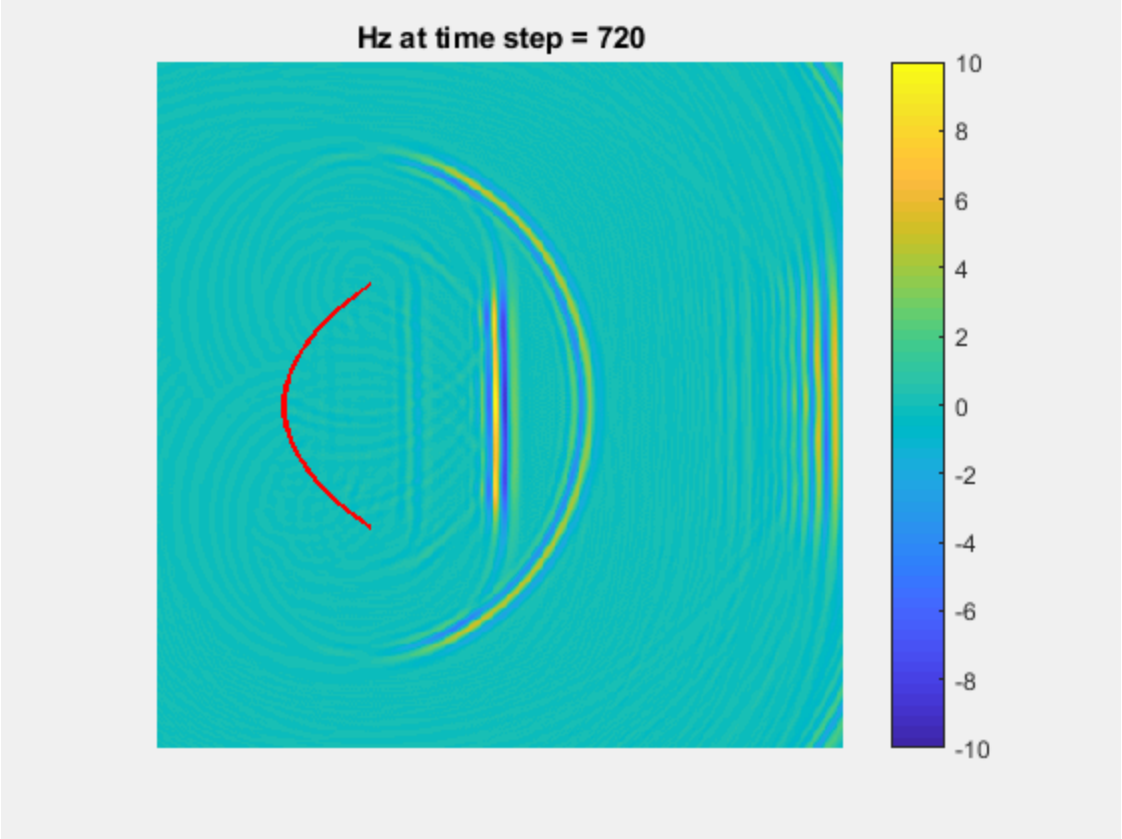

Hz at time step = 320
10
8
6
4
2
0
-2
-4
-6
-8
-10

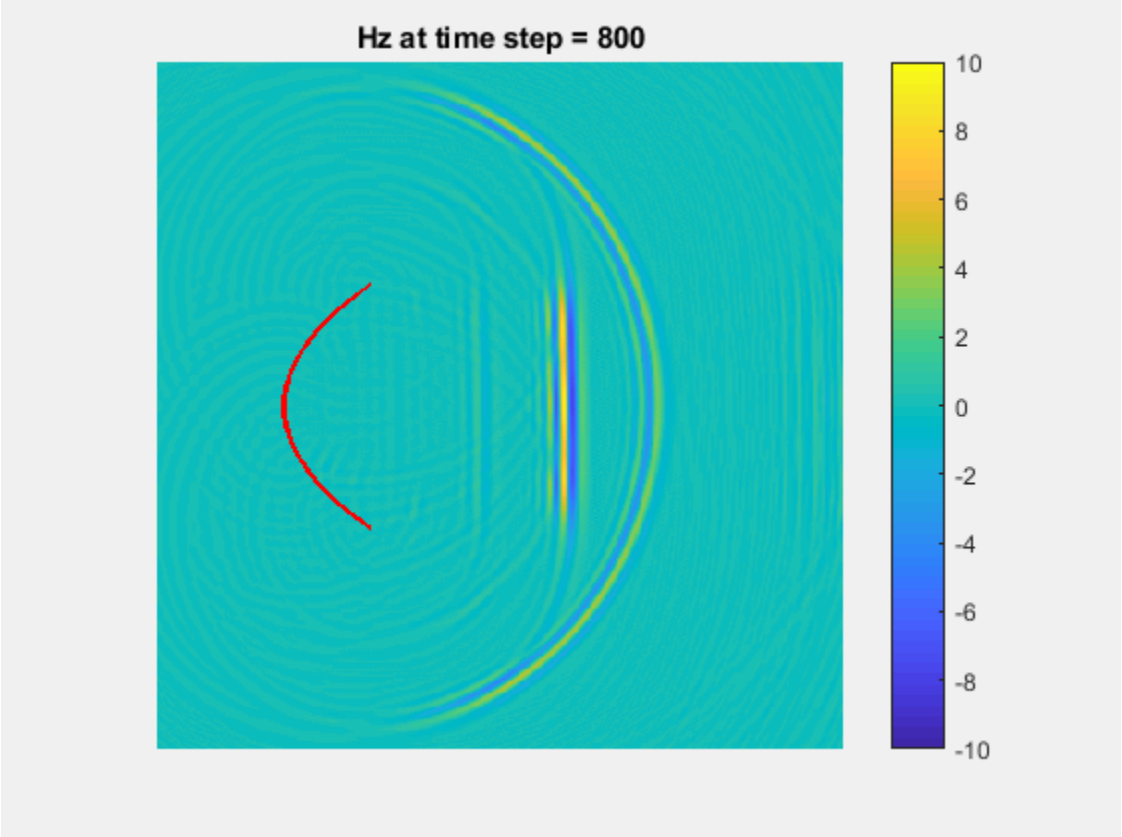

Hz at time step = 400
10
8
6
4
2
0
-2
-4
-6
-8
-10


Hz at time step = 480
10
8
6
4
2
0
-2
-4
-6
-8
-10


Hz at time step = 560
10
8
6
4
2
0
-2
-4
-6
-8
-10


Hz at time step = 640
10
8
6
4
2
0
-2
-4
-6
-8
-10

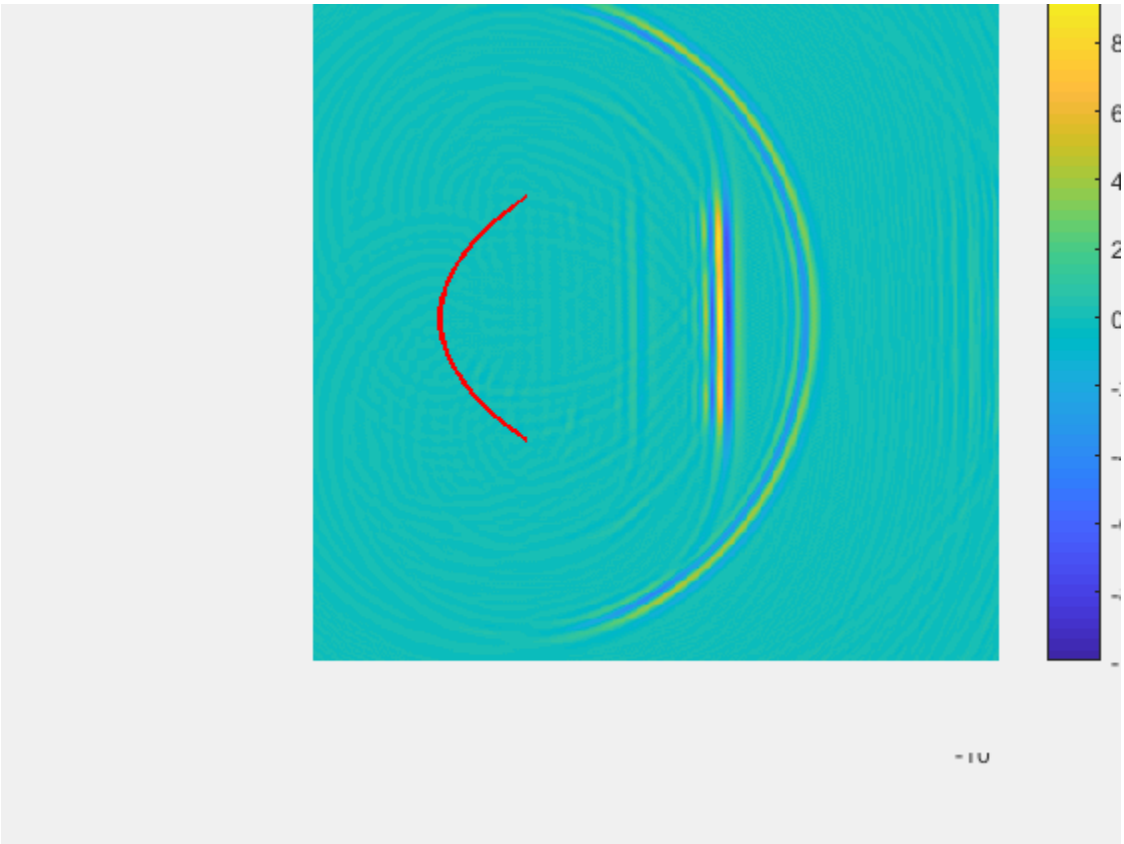

Hz at time step = 720
10
8
6
4
2
0
-2
-4
-6
-8
-10


Hz at time step = 800
10
8
6
4
2
0
-2
-4
-6
-8
-10

```
end
```

The following command is used when running file to playback a fluid animation. When publishing it does nothing; the image shown is merely the final state.

```
movie(M)
```



Conclusion

Reflector antennas are highly directive. For them to work as intended, the source (called antenna feed) should be located at the focal point. The wavefronts emitted at the focus are spherical but when they get reflected by the paraboloid surface they become planar.

In order to simulate a receiving antenna we need to have a plane wave being sent to the antenna. Though the TF/SF technique can be used to efficiently create plane waves, a simple way in this program will be adding a flipped antenna at the right of the computation grid. Then the incident plane wave will concentrate to the focus point. Finally we require to set the focus point as a black hole, that is it should behave like an absorber.

Bibliography

- [Taflove2000] A. Taflove and S.C. Hagness, Computational Electrodynamics: The Finite-Difference Time-Domain Method, 2nd edition. 2000.
- [GedneyEE624] Gedney Stephen, University of Kentucky, EE624 Notes, Fall 2005, "Absorbing Boundary Conditions".

- [YTVideo] meyavuz, https://www.youtube.com/watch?v=785kRIZ7aeI

Published with MATLAB® R2017b