APPENDIX A: DELAY-AND-SUM BEAMFORMING SCRIPTS

A.1 Beamformer.m Script

```
% beamformer.m
%
% Jonathan Mamou
% January 2002
%
% Form a B-mode image by delay and sum beamforming
%
% Can be used for both real data and simulated data
%
% Version 1:
%
% Simple Delay and Sum, no Time Gain Compensation,
% no dynamic focusing.
% All signals are hence focused at a fixed depth.

function [z1, z2] = beamformer(name, paramfile, paramflag, focus,
                                                          realsim)

%name : Name of the scan files
%paramfile : Name of the parameter file
%paramflag : if equal to 0, use the hardcoded values
%focus : Depth of focus in meter
%realsim : if equal to 0 then real data, else simulated data
% (Slightly chnage the reading procedure, not the processing)
```

```
if (paramflag==0)
Ndx=4;
Ndy=2;     %Number of transducers or transducer locations
Nd=Ndx*Ndy;
Nz=20; %Number of reflection points in z direction
Nx=20; %and in x direction
Ny=20;
Nt=1024; %signal length
Np=10; %Number of point sources on the transducer surface
Ndeep=5; %Number of different depths of the buried obj.
Nr=1; %Number of realizations
Na=2;

PSS=200; %Propagation speed of sound [M/s]
freq=2000; %Center frequency of input signal
CycleCount=2;       % Number of cycles of input signal
alpha=1.0; %Attenuation constant

DiamTrans=0.05;
DiamRec=0.07;
Lx=0.038;  %Size of the rectangular transducer piston
Ly=Lx;

xxt=0.0;
yyt=0;    %Coordinates of the circular transmitting element

xx1=-0.5;
xx2=0.5; %Bounds of area and time of interest
yy1=-0.5;
yy2=0.5;
zz1=0.0;
zz2=0.5;
tt1=0.0;
ox1=-10.0;
ox2=0.0;
oy1=-10.0;
oy2=0.0;
oz1=20.0;
oz2=30.0;
```

```
ya1=0;                %starting point of the SAR track

SamplingFreq=60060;
T=1/freq; %Pulse width is now 1 wavelength

da=0.01; %azimuthal sapmling distance
dt=1/SamplingFreq;       %time interval between samples
tt2=tt1+dt*(Nt-1);
dx=(xx2-xx1)/(Nx-1); %separation of scattering points
dy=(yy2-yy1)/(Ny-1);
dz=(zz2-zz1)/(Nz-1);
dpx=DiamRec/Np; %transducer surface discretization
dpy=DiamRec/Np;

SensorPositionx0=0.0;
SensorPositiony0=0.0;
SensorSpacingx=.05; %Transducer spacing in x direction
SensorSpacingy=.05;
MaxNoise=0.0001;


else


%
% Reads parameters from the file paramfile
%


FID = fopen(paramfile, 'r');
PSS=fscanf(FID, 'PSS=%f\n', 1)%
freq=fscanf(FID, 'freq=%f\n', 1)%
CycleCount=fscanf(FID, 'CycleCount=%f\n', 1)%
alpha=fscanf(FID, 'alpha=%f\n', 1)%
DiamTrans=fscanf(FID, 'DiamTrans=%f\n', 1)%
DiamRec=fscanf(FID, 'DiamRec=%f\n', 1)%
xxt=fscanf(FID, 'xxt=%f\n', 1)%
```

```
yyt=fscanf(FID, 'yyt=%f\n', 1)%
Ndx=fscanf(FID, 'Ndx=%d\n', 1)%
Ndy=fscanf(FID, 'Ndy=%d\n', 1)%
Nx=fscanf(FID, 'Nx=%d\n', 1)%
Ny=fscanf(FID, 'Ny=%d\n', 1)%
Nz=fscanf(FID, 'Nz=%d\n', 1)%
Nt=fscanf(FID, 'Nt=%d\n', 1)%
Np=fscanf(FID, 'Np=%d\n', 1)%
Ndeep=fscanf(FID, 'Ndeep=%d\n', 1)%
Nr=fscanf(FID, 'Nr=%d\n', 1)%
Na=fscanf(FID, 'Na=%d\n', 1)%
da=fscanf(FID, 'da=%f\n', 1)%
ya1=fscanf(FID, 'ya1=%f\n', 1)%
xx1=fscanf(FID, 'xx1=%f\n', 1)%
xx2=fscanf(FID, 'xx2=%f\n', 1)%
yy1=fscanf(FID, 'yy1=%f\n', 1)%
yy2=fscanf(FID, 'yy2=%f\n', 1)%
zz1=fscanf(FID, 'zz1=%f\n', 1)%
zz2=fscanf(FID, 'zz2=%f\n', 1)%
tt1=fscanf(FID, 'tt1=%f\n', 1)%
ox1=fscanf(FID,'ox1=%f\n',1)%
ox2=fscanf(FID,'ox2=%f\n',1)%
oy1=fscanf(FID,'oy1=%f\n',1)%
oy2=fscanf(FID,'oy2=%f\n',1)%
oz1=fscanf(FID,'oz1=%f\n',1)%
oz2=fscanf(FID,'oz2=%f\n',1)%
SensorPositionx0=fscanf(FID, 'SensorPositionx0=%f\n', 1)%
SensorPositiony0=fscanf(FID, 'SensorPositiony0=%f\n', 1)%
SensorSpacingx=fscanf(FID, 'SensorSpacingx=%f\n', 1)%
SensorSpacingy=fscanf(FID, 'SensorSpacingy=%f\n', 1)%
MaxNoise=fscanf(FID, 'MaxNoise=%f\n', 1)%
SamplingFreq=fscanf(FID, 'SamplingFreq=%f\n', 1)%
object_exists=fscanf(FID, 'object_exists=%d\n', 1)%
object_mode=fscanf(FID, 'object_mode=%d\n', 1)%
wrt_file=fscanf(FID, 'wrt_file=%d\n', 1)%
noisy=fscanf(FID, 'noisy=%d\n', 1)%

end
```

```
N=Na*Ndx*Ndy;  % total number of files is Na*Ndx*Ndy
N

limite=2*(zz2-zz1)/PSS*SamplingFreq
limite=floor(limite+1); % Limit in samples to reach the max depth


% Read data :




for i=1:N
filename = [name, int2str(i-1)];
 fid = fopen(filename,'r');

if realsim==0
 fread(fid,12*16+4,'char');                  % get rid of the header
 sarace = fread(fid, [Nt, 1], 'short');
else
 sarace=fscanf(fid,'%f',[ Nt 1 ]);
end

%size(sarace);
Data(:,i)=sarace;
Data(:,i)=Data(:,i)-mean(Data(:,i));    % Remove DC

fclose(fid);
end

%Low Pas Filtering of raw data.

f = [0 2 4 30]/30;                % 80 kHz sampling rate
m = [1 1 0 0];                    % low pass filter
w = [1 1];                        % equal ripple in passband
                                  % and stopband
g = remez(51, f, m, w);           % create filter
```

```
temp = [Data; zeros(size(Data))];
                    % zero pad to compensate for circular conv.
out = fftfilt(g, temp);
                    % filter columns of "temp" with filter "g"
out = out(26:1024+25,:);           % keep relevant data
clear temp;
temp = mean(out);
for(i=1:N)
   out(:,i) = out(:,i) - temp(i);  % remove DC component
end

% Delay and Sum
%
% Signals from the receiver array are delayed to arrive
% in phase at the focus depth and then summed to increase
% the Signal to Noise Ratio.

if (focus>0)

%Computation of the delays

delays=zeros(Ndx,Ndy);
   for ix=1:Ndx % For each element of the receiver array
     for iy=1:Ndy

      d1=1/2*sqrt(yyt^2+xxt^2);
      d2=sqrt(d1^2+focus^2);

      d3y=-yyt/2+(iy-Ndy/2-0.5)*SensorSpacingy;
      d3x=-xxt/2+(ix-Ndx/2-0.5)*SensorSpacingx;
      d3=sqrt(d3x^2+d3y^2+focus^2);

      d=d2+d3-2*focus;
      delays(ix,iy)=floor(d/PSS*SamplingFreq)+1;

end %iy
end %ix
```

```
delays

% Sum of delayed signals.

temp=zeros(Nt,Na);

for s=1:Na    % For each sample
   for ix=1:Ndx % For each element of the receiver array
     for iy=1:Ndy
   D=delays(ix,iy);

     temp(:,s)=temp(:,s)+[ out(D:Nt,Ndx*Ndy*(s-1)+(ix-1)*
                     Ndy+iy)/(Ndx*Ndy);zeros((D-1),1)];
end %s
end %iy
end %ix

out=temp;
end



% Log, Enveloppe detection and display

temp = 20*log10(abs(hilbert(out)));  % log and enveloppe detection
temp = temp - max(max(temp));% just to have a nice visual result

% for the scale of the picture
rangee = (0:(Nt-1))/(2*SamplingFreq)*PSS;
azimuth = (0:(Na-1))*da;


% displays the picture.
% Dynamic range is 30 dB.

imagesc(azimuth, rangee(1:limite), temp(1:limite,:), [-30 0]);
axis('image');
xlabel('Horizontal distance, [m]');
ylabel('Range, [m]');
```

```
colorbar;
```

A.2 BeamformerTGC.m Script

```
% beamformerTGC.m
%
% Jonathan Mamou
% January 2002
%
% Form a B-mode image by delay and sum beamforming
%
% Can be used for both real data and simulated data
%
% Version 1: Jan 2002
%
% Simple Delay and Sum, no Time Gain Compensation,
% no dynamic focusing.
% All signals are hence focused at a fixed depth.
%
% Version 2: Feb 2002
%
% TGC has been added
%


function [z1, z2] =
     beamformerTGC(name, paramfile, paramflag, focus, realsim )

%name : Name of the scan files
%paramfile : Name of the parameter file
%paramflag : if equal to 0, use the hardcoded values
%focus : Depth of focus in meter
%realsim : if equal to 0 then real data, else simulated data
% (Slightly chnage the reading procedure, not the processing)


if (paramflag==0)
Ndx=4;
```

```
Ndy=2;     %Number of transducers or transducer locations
Nd=Ndx*Ndy;
Nz=20; %Number of reflection points in z direction
Nx=20; %and in x direction
Ny=20;
Nt=1024; %signal length
Np=10; %Number of point sources on the transducer surface
Ndeep=5; %Number of different depths of the buried obj.
Nr=1; %Number of realizations
Na=2;

PSS=200; %Propagation speed of sound [M/s]
freq=2000; %Center frequency of input signal
CycleCount=2;       % Number of cycles of input signal
alpha=1.0; %Attenuation constant

DiamTrans=0.05;
DiamRec=0.07;
Lx=0.038;  %Size of the rectangular transducer piston
Ly=Lx;

xxt=0.0;
yyt=0;    %Coordinates of the circular transmitting element

xx1=-0.5;
xx2=0.5; %Bounds of area and time of interest
yy1=-0.5;
yy2=0.5;
zz1=0.0;
zz2=0.5;
tt1=0.0;
ox1=-10.0;
ox2=0.0;
oy1=-10.0;
oy2=0.0;
oz1=20.0;
oz2=30.0;
```

```
ya1=0;                %starting point of the SAR track

SamplingFreq=60060;
T=1/freq; %Pulse width is now 1 wavelength

da=0.01; %azimuthal sapmling distance
dt=1/SamplingFreq;       %time interval between samples
tt2=tt1+dt*(Nt-1);
dx=(xx2-xx1)/(Nx-1); %separation of scattering points
dy=(yy2-yy1)/(Ny-1);
dz=(zz2-zz1)/(Nz-1);
dpx=DiamRec/Np; %transducer surface discretization
dpy=DiamRec/Np;

SensorPositionx0=0.0;
SensorPositiony0=0.0;
SensorSpacingx=.05; %Transducer spacing in x direction
SensorSpacingy=.05;
MaxNoise=0.0001;


else


%
% Reads parameters from the file paramfile
%


FID = fopen(paramfile, 'r');
PSS=fscanf(FID, 'PSS=%f\n', 1)%
freq=fscanf(FID, 'freq=%f\n', 1)%
CycleCount=fscanf(FID, 'CycleCount=%f\n', 1)%
alpha=fscanf(FID, 'alpha=%f\n', 1)%
DiamTrans=fscanf(FID, 'DiamTrans=%f\n', 1)%
DiamRec=fscanf(FID, 'DiamRec=%f\n', 1)%
xxt=fscanf(FID, 'xxt=%f\n', 1)%
yyt=fscanf(FID, 'yyt=%f\n', 1)%
Ndx=fscanf(FID, 'Ndx=%d\n', 1)%
```

```
Ndy=fscanf(FID, 'Ndy=%d\n', 1)%
Nx=fscanf(FID, 'Nx=%d\n', 1)%
Ny=fscanf(FID, 'Ny=%d\n', 1)%
Nz=fscanf(FID, 'Nz=%d\n', 1)%
Nt=fscanf(FID, 'Nt=%d\n', 1)%
Np=fscanf(FID, 'Np=%d\n', 1)%
Ndeep=fscanf(FID, 'Ndeep=%d\n', 1)%
Nr=fscanf(FID, 'Nr=%d\n', 1)%
Na=fscanf(FID, 'Na=%d\n', 1)%
da=fscanf(FID, 'da=%f\n', 1)%
ya1=fscanf(FID, 'ya1=%f\n', 1)%
xx1=fscanf(FID, 'xx1=%f\n', 1)%
xx2=fscanf(FID, 'xx2=%f\n', 1)%
yy1=fscanf(FID, 'yy1=%f\n', 1)%
yy2=fscanf(FID, 'yy2=%f\n', 1)%
zz1=fscanf(FID, 'zz1=%f\n', 1)%
zz2=fscanf(FID, 'zz2=%f\n', 1)%
tt1=fscanf(FID, 'tt1=%f\n', 1)%
ox1=fscanf(FID,'ox1=%f\n',1)%
ox2=fscanf(FID,'ox2=%f\n',1)%
oy1=fscanf(FID,'oy1=%f\n',1)%
oy2=fscanf(FID,'oy2=%f\n',1)%
oz1=fscanf(FID,'oz1=%f\n',1)%
oz2=fscanf(FID,'oz2=%f\n',1)%
SensorPositionx0=fscanf(FID, 'SensorPositionx0=%f\n', 1)%
SensorPositiony0=fscanf(FID, 'SensorPositiony0=%f\n', 1)%
SensorSpacingx=fscanf(FID, 'SensorSpacingx=%f\n', 1)%
SensorSpacingy=fscanf(FID, 'SensorSpacingy=%f\n', 1)%
MaxNoise=fscanf(FID, 'MaxNoise=%f\n', 1)%
SamplingFreq=fscanf(FID, 'SamplingFreq=%f\n', 1)%
object_exists=fscanf(FID, 'object_exists=%d\n', 1)%
object_mode=fscanf(FID, 'object_mode=%d\n', 1)%
wrt_file=fscanf(FID, 'wrt_file=%d\n', 1)%
noisy=fscanf(FID, 'noisy=%d\n', 1)%

end

N=Na*Ndx*Ndy;  % total number of files is Na*Ndx*Ndy
N
```

```
limite=2*(zz2-zz1)/PSS*SamplingFreq
limite=floor(limite+1);  % Limit in samples to reach the max depth


% Read data :


 % For TGC later on

TGC=[0:(Nt-1)];
%TGC=exp((alpha*PSS/SamplingFreq)*TGC);
 TGC=exp((alpha*PSS/SamplingFreq)*TGC).*TGC.*TGC*PSS*
                                PSS/SamplingFreq/SamplingFreq;


%figure(1)
%plot(TGC(1:limite));

%TGC(limite:Nt)=TGC(limite-1);

Data=zeros(Nt,N);
for i=1:N
filename = [name, int2str(i-1)];
 fid = fopen(filename,'r');

if realsim==0
 fread(fid,12*16+4,'char');                  % get rid of the header
 sarace = fread(fid, [Nt, 1], 'short');
else
 sarace=fscanf(fid,'%f',[ Nt 1 ]);
end

%size(sarace);
Data(:,i)=sarace;
% TGC

if i==26
Signal=Data(:,26);
```

```
end

Data(:,i)=Data(:,i).*TGC';
Data(:,i)=Data(:,i)-mean(Data(:,i));    % Remove DC

SignalTGC=Data(:,26);

fclose(fid);
end



%Low Pas Filtering of raw data.

f = [0 2 4 30]/30;                  % 80 kHz sampling rate
m = [1 1 0 0];                      % low pass filter
w = [1 1];                          % equal ripple in passband
                                    % and stopband
g = remez(51, f, m, w);             % create filter


temp = [Data; zeros(size(Data))];
                      % zero pad to compensate for circular conv.
out = fftfilt(g, temp);
                      % filter columns of "temp" with filter "g"
out = out(26:1024+25,:);            % keep relevant data
clear temp;
temp = mean(out);
for(i=1:N)
   out(:,i) = out(:,i) - temp(i);  % remove DC component
end

SignalF=out(:,26);

figure
plot(20*log10(abs(Signal)))
figure
plot(20*log10(abs(SignalTGC)))
figure
```

```
plot(SignalF)

% Delay and Sum
%
% Signals from the receiver array are delayed to arrive
% in phase at the focus depth and then summed to increase
% the Signal to Noise Ratio.

if (focus>0)

%Computation of the delays

delays=zeros(Ndx,Ndy);
   for ix=1:Ndx % For each element of the receiver array
     for iy=1:Ndy

      d1=1/2*sqrt(yyt^2+xxt^2);
      d2=sqrt(d1^2+focus^2);

      d3y=-yyt/2+(iy-Ndy/2-0.5)*SensorSpacingy;
      d3x=-xxt/2+(ix-Ndx/2-0.5)*SensorSpacingx;
      d3=sqrt(d3x^2+d3y^2+focus^2);

      d=d2+d3-2*focus;
      delays(ix,iy)=floor(d/PSS*SamplingFreq)+1;

end %iy
end %ix

delays

% Sum of delayed signals.

temp=zeros(Nt,Na);

for s=1:Na    % For each sample
   for ix=1:Ndx % For each element of the receiver array
     for iy=1:Ndy
   D=delays(ix,iy);
```

```
     temp(:,s)=temp(:,s)+[ out(D:Nt,Ndx*Ndy*(s-1)+(ix-1)*
                                    Ndy+iy)/(Ndx*Ndy); zeros((D-1),1)];
end %s
end %iy
end %ix

out=temp;
end


figure
% Log, Enveloppe detection and display

temp = 20*log10(abs(hilbert(out)));  % log and enveloppe detection
temp = temp - max(max(temp));% just to have a nice visual result

% for the scale of the picture
rangee = (0:(Nt-1))/(2*SamplingFreq)*PSS;
azimuth = (0:(Na-1))*da;


% displays the picture.
% Dynamic range is 30 dB.

figure
imagesc(azimuth, rangee(1:limite), temp(1:limite,:), [-30 0]);
axis('image');
xlabel('Horizontal distance, [m]');
ylabel('Range, [m]');
colorbar;
```

A.3 BeamformerDFTGC.m Script

```
% beamformerDFTGC.m
%
% Jonathan Mamou
% January 2002
%
```

```
% Form a B-mode image by delay and sum beamforming
%
% Can be used for both real data and simulated data
%
% Version 1:
%
% Simple Delay and Sum, no Time Gain Compensation,
% no dynamic focusing.
% All signals are hence focused at a fixed depth.
%
% Version TGC:
%
% Time Gain Compensation was added Feb 2002
%
% Version DF:
%
% Dynamic Focusing was added: Feb 2002
%
%


function [z1, z2] = beamformerDFTGC(name, paramfile, paramflag, ...
    focus, realsim ,tgcflag)

%name : Name of the scan files
%paramfile : Name of the parameter file
%paramflag : if equal to 0, use the hardcoded values
%focus : Depth of focus in meter
%realsim : if equal to 0 then real data, else simulated data
% (Slightly chnage the reading procedure, not the processing)
%tgcflag if > 0 TGC is achieved.

if (paramflag==0)
Ndx=4;
Ndy=2;     %Number of transducers or transducer locations
Nd=Ndx*Ndy;
Nz=20; %Number of reflection points in z direction
Nx=20; %and in x direction
Ny=20;
```

```
Nt=1024; %signal length
Np=10; %Number of point sources on the transducer surface
Ndeep=5; %Number of different depths of the buried obj.
Nr=1; %Number of realizations
Na=2;

PSS=200; %Propagation speed of sound [M/s]
freq=2000; %Center frequency of input signal
CycleCount=2;       % Number of cycles of input signal
alpha=1.0; %Attenuation constant

DiamTrans=0.05;
DiamRec=0.07;
Lx=0.038;  %Size of the rectangular transducer piston
Ly=Lx;

xxt=0.0;
yyt=0;    %Coordinates of the circular transmitting element

xx1=-0.5;
xx2=0.5; %Bounds of area and time of interest
yy1=-0.5;
yy2=0.5;
zz1=0.0;
zz2=0.5;
tt1=0.0;
ox1=-10.0;
ox2=0.0;
oy1=-10.0;
oy2=0.0;
oz1=20.0;
oz2=30.0;


ya1=0;               %starting point of the SAR track

SamplingFreq=60060;
T=1/freq; %Pulse width is now 1 wavelength
```

```
da=0.01; %azimuthal sapmling distance
dt=1/SamplingFreq;       %time interval between samples
tt2=tt1+dt*(Nt-1);
dx=(xx2-xx1)/(Nx-1); %separation of scattering points
dy=(yy2-yy1)/(Ny-1);
dz=(zz2-zz1)/(Nz-1);
dpx=DiamRec/Np; %transducer surface discretization
dpy=DiamRec/Np;

SensorPositionx0=0.0;
SensorPositiony0=0.0;
SensorSpacingx=.05; %Transducer spacing in x direction
SensorSpacingy=.05;
MaxNoise=0.0001;


else


%
% Reads parameters from the file paramfile
%


FID = fopen(paramfile, 'r');
PSS=fscanf(FID, 'PSS=%f\n', 1)%
freq=fscanf(FID, 'freq=%f\n', 1)%
CycleCount=fscanf(FID, 'CycleCount=%f\n', 1)%
alpha=fscanf(FID, 'alpha=%f\n', 1)%
DiamTrans=fscanf(FID, 'DiamTrans=%f\n', 1)%
DiamRec=fscanf(FID, 'DiamRec=%f\n', 1)%
xxt=fscanf(FID, 'xxt=%f\n', 1)%
yyt=fscanf(FID, 'yyt=%f\n', 1)%
Ndx=fscanf(FID, 'Ndx=%d\n', 1)%
Ndy=fscanf(FID, 'Ndy=%d\n', 1)%
Nx=fscanf(FID, 'Nx=%d\n', 1)%
Ny=fscanf(FID, 'Ny=%d\n', 1)%
Nz=fscanf(FID, 'Nz=%d\n', 1)%
Nt=fscanf(FID, 'Nt=%d\n', 1)%
```

```
Np=fscanf(FID, 'Np=%d\n', 1)%
Ndeep=fscanf(FID, 'Ndeep=%d\n', 1)%
Nr=fscanf(FID, 'Nr=%d\n', 1)%
Na=fscanf(FID, 'Na=%d\n', 1)%
da=fscanf(FID, 'da=%f\n', 1)%
ya1=fscanf(FID, 'ya1=%f\n', 1)%
xx1=fscanf(FID, 'xx1=%f\n', 1)%
xx2=fscanf(FID, 'xx2=%f\n', 1)%
yy1=fscanf(FID, 'yy1=%f\n', 1)%
yy2=fscanf(FID, 'yy2=%f\n', 1)%
zz1=fscanf(FID, 'zz1=%f\n', 1)%
zz2=fscanf(FID, 'zz2=%f\n', 1)%
tt1=fscanf(FID, 'tt1=%f\n', 1)%
ox1=fscanf(FID,'ox1=%f\n',1)%
ox2=fscanf(FID,'ox2=%f\n',1)%
oy1=fscanf(FID,'oy1=%f\n',1)%
oy2=fscanf(FID,'oy2=%f\n',1)%
oz1=fscanf(FID,'oz1=%f\n',1)%
oz2=fscanf(FID,'oz2=%f\n',1)%
SensorPositionx0=fscanf(FID, 'SensorPositionx0=%f\n', 1)%
SensorPositiony0=fscanf(FID, 'SensorPositiony0=%f\n', 1)%
SensorSpacingx=fscanf(FID, 'SensorSpacingx=%f\n', 1)%
SensorSpacingy=fscanf(FID, 'SensorSpacingy=%f\n', 1)%
MaxNoise=fscanf(FID, 'MaxNoise=%f\n', 1)%
SamplingFreq=fscanf(FID, 'SamplingFreq=%f\n', 1)%
object_exists=fscanf(FID, 'object_exists=%d\n', 1)%
object_mode=fscanf(FID, 'object_mode=%d\n', 1)%
wrt_file=fscanf(FID, 'wrt_file=%d\n', 1)%
noisy=fscanf(FID, 'noisy=%d\n', 1)%

end

N=Na*Ndx*Ndy;  % total number of files is Na*Ndx*Ndy
N

limite=2*(zz2-zz1)/PSS*SamplingFreq
limite=floor(limite+1); % Limite in samples to reach the max depth
```

```
% For TGC later on

TGC=[0:(Nt-1)];
%TGC=exp((alpha*PSS/SamplingFreq)*TGC);
 TGC=exp((alpha*PSS/SamplingFreq)*TGC).*TGC.*TGC*PSS*
                                        PSS/SamplingFreq/SamplingFreq;


% Read data :

for i=1:N
filename = [name, int2str(i-1)];
 fid = fopen(filename,'r');

if realsim==0
 fread(fid,12*16+4,'char');                  % get rid of the header
 sarace = fread(fid, [Nt, 1], 'short');
else
 sarace=fscanf(fid,'%f',[ Nt 1 ]);
end

%size(sarace);
Data(:,i)=sarace;

% TGC
if (tgcflag>0)
  Data(:,i)=Data(:,i).*TGC';
end

Data(:,i)=Data(:,i)-mean(Data(:,i));    % Remove DC

fclose(fid);
end

%Low Pas Filtering of raw data.

f = [0 2 4 30]/30;                 % 80 kHz sampling rate
m = [1 1 0 0];                     % low pass filter
```

```
w = [1 1];                          % equal ripple in passband
                                    % and stopband
g = remez(51, f, m, w);             % create filter


temp = [Data; zeros(size(Data))];
                      % zero pad to compensate for circular conv.
out = fftfilt(g, temp);
                      % filter columns of "temp" with filter "g"
out = out(26:1024+25,:);            % keep relevant data
clear temp;
temp = mean(out);
for(i=1:N)
   out(:,i) = out(:,i) - temp(i);  % remove DC component
end

% Delay and Sum
%
% Signals from the receiver array are delayed to arrive
% in phase at the focus depth and then summed to increase
% the Signal to Noise Ratio.

if (focus>0)

%Computation of the delays


delays=zeros(Ndx,Ndy,Ndeep);
FocusDepth=zeros(1,Ndeep);
FocusDepth=[1:Ndeep];
FocusDepth=(zz2-zz1)*(FocusDepth-0.5)/Ndeep
FocusChunk=floor(2*(SamplingFreq*(zz2-zz1)/PSS*[0:Ndeep])/Ndeep+1)
   for ix=1:Ndx % For each element of the receiver array
     for iy=1:Ndy
       for iD=1:Ndeep
 focus=FocusDepth(iD);
 d1=1/2*sqrt(yyt^2+xxt^2);
 d2=sqrt(d1^2+focus^2);
```

```
 d3y=-yyt/2+(iy-Ndy/2-0.5)*SensorSpacingy;
 d3x=-xxt/2+(ix-Ndx/2-0.5)*SensorSpacingx;
 d3=sqrt(d3x^2+d3y^2+focus^2);

 d=d2+d3-2*focus;
 delays(ix,iy,iD)=floor(d/PSS*SamplingFreq)+1;

       end %iD
     end %iy
   end %ix

   delays

% Computation of delayed signals.

temp=zeros(Nt,Na,Ndeep);

for s=1:Na    % For each sample
   for ix=1:Ndx % For each element of the receiver array
     for iy=1:Ndy
       for iD=1:Ndeep
 D=delays(ix,iy,iD);
 temp(:,s,iD)=temp(:,s,iD)+[ out(D:Nt,Ndx*Ndy*
               (s-1)+(ix-1)*Ndy+iy)/(Ndx*Ndy); zeros((D-1),1)];
       end %iD
     end %s
   end %iy
end %ix

% Concatenation of delayed signals to have image in focus
% at Ndeep depths.

temp2=zeros(Nt,Na);

for s=1:Na    % For each sample
   for ix=1:Ndx % For each element of the receiver array
     for iy=1:Ndy
       for iD=1:Ndeep
 temp2(FocusChunk(iD):FocusChunk(iD+1),s)=
```

```
                    temp(FocusChunk(iD):FocusChunk(iD+1),s,iD);
        end %iD
      end %s
    end %iy
end %ix

out=temp2;
end



% Log, Enveloppe detection and display

temp = 20*log10(abs(hilbert(out)));  % log and enveloppe detection
temp = temp - max(max(temp));% just to have a nice visual result

% for the scale of the picture
rangee = (0:(Nt-1))/(2*SamplingFreq)*PSS;
azimuth = (0:(Na-1))*da;


% displays the picture.
% Dynamic range is 30 dB.

imagesc(azimuth, rangee(1:limite), temp(1:limite,:), [-30 0]);
axis('image');
xlabel('Horizontal distance, [m]');
ylabel('Range, [m]');
colorbar;
```

A.4 BeamformerDFTGCsmooth.m Script

```
% beamformerDFTGCsmooth.m
%
% Jonathan Mamou
% January 2002
%
% Form a B-mode image by delay and sum beamforming
%
```

```
% Can be used for both real data and simulated data
%
% Version 1:
%
% Simple Delay and Sum, no Time Gain Compensation,
% no dynamic focusing.
% All signals are hence focused at a fixed depth.
%
% Version TGC:
%
% Time Gain Compensation was added Feb 2002
%
% Version DF:
%
% Dynamic Focusing was added: Feb 2002
%
% Version smooth
%
% Windowing and wheigthed averaging was added to smooth the DF
% process: Mar 2002
%
%

function [z1, z2] = beamformerDFTGCsmooth(name, paramfile, paramflag,
    focus, realsim ,tgcflag)

%name : Name of the scan files
%paramfile : Name of the parameter file
%paramflag : if equal to 0, use the hardcoded values
%focus : Depth of focus in meter
%realsim : if equal to 0 then real data, else simulated data
% (Slightly chnage the reading procedure, not the processing)
%tgcflag if > 0 TGC is achieved.

if (paramflag==0)
Ndx=4;
Ndy=2;     %Number of transducers or transducer locations
Nd=Ndx*Ndy;
Nz=20; %Number of reflection points in z direction
```

```
Nx=20; %and in x direction
Ny=20;
Nt=1024; %signal length
Np=10; %Number of point sources on the transducer surface
Ndeep=5; %Number of different depths of the buried obj.
Nr=1; %Number of realizations
Na=2;

PSS=200; %Propagation speed of sound [M/s]
freq=2000; %Center frequency of input signal
CycleCount=2;       % Number of cycles of input signal
alpha=1.0; %Attenuation constant

DiamTrans=0.05;
DiamRec=0.07;
Lx=0.038;  %Size of the rectangular transducer piston
Ly=Lx;

xxt=0.0;
yyt=0;    %Coordinates of the circular transmitting element

xx1=-0.5;
xx2=0.5; %Bounds of area and time of interest
yy1=-0.5;
yy2=0.5;
zz1=0.0;
zz2=0.5;
tt1=0.0;
ox1=-10.0;
ox2=0.0;
oy1=-10.0;
oy2=0.0;
oz1=20.0;
oz2=30.0;


ya1=0;                %starting point of the SAR track

SamplingFreq=60060;
```

```
T=1/freq; %Pulse width is now 1 wavelength

da=0.01; %azimuthal sapmling distance
dt=1/SamplingFreq;       %time interval between samples
tt2=tt1+dt*(Nt-1);
dx=(xx2-xx1)/(Nx-1); %separation of scattering points
dy=(yy2-yy1)/(Ny-1);
dz=(zz2-zz1)/(Nz-1);
dpx=DiamRec/Np; %transducer surface discretization
dpy=DiamRec/Np;

SensorPositionx0=0.0;
SensorPositiony0=0.0;
SensorSpacingx=.05; %Transducer spacing in x direction
SensorSpacingy=.05;
MaxNoise=0.0001;


else


%
% Reads parameters from the file paramfile
%


FID = fopen(paramfile, 'r');
PSS=fscanf(FID, 'PSS=%f\n', 1)%
freq=fscanf(FID, 'freq=%f\n', 1)%
CycleCount=fscanf(FID, 'CycleCount=%f\n', 1)%
alpha=fscanf(FID, 'alpha=%f\n', 1)%
DiamTrans=fscanf(FID, 'DiamTrans=%f\n', 1)%
DiamRec=fscanf(FID, 'DiamRec=%f\n', 1)%
xxt=fscanf(FID, 'xxt=%f\n', 1)%
yyt=fscanf(FID, 'yyt=%f\n', 1)%
Ndx=fscanf(FID, 'Ndx=%d\n', 1)%
Ndy=fscanf(FID, 'Ndy=%d\n', 1)%
Nx=fscanf(FID, 'Nx=%d\n', 1)%
Ny=fscanf(FID, 'Ny=%d\n', 1)%
```

```
Nz=fscanf(FID, 'Nz=%d\n', 1)%
Nt=fscanf(FID, 'Nt=%d\n', 1)%
Np=fscanf(FID, 'Np=%d\n', 1)%
Ndeep=fscanf(FID, 'Ndeep=%d\n', 1)%
Nr=fscanf(FID, 'Nr=%d\n', 1)%
Na=fscanf(FID, 'Na=%d\n', 1)%
da=fscanf(FID, 'da=%f\n', 1)%
ya1=fscanf(FID, 'ya1=%f\n', 1)%
xx1=fscanf(FID, 'xx1=%f\n', 1)%
xx2=fscanf(FID, 'xx2=%f\n', 1)%
yy1=fscanf(FID, 'yy1=%f\n', 1)%
yy2=fscanf(FID, 'yy2=%f\n', 1)%
zz1=fscanf(FID, 'zz1=%f\n', 1)%
zz2=fscanf(FID, 'zz2=%f\n', 1)%
tt1=fscanf(FID, 'tt1=%f\n', 1)%
ox1=fscanf(FID,'ox1=%f\n',1)%
ox2=fscanf(FID,'ox2=%f\n',1)%
oy1=fscanf(FID,'oy1=%f\n',1)%
oy2=fscanf(FID,'oy2=%f\n',1)%
oz1=fscanf(FID,'oz1=%f\n',1)%
oz2=fscanf(FID,'oz2=%f\n',1)%
SensorPositionx0=fscanf(FID, 'SensorPositionx0=%f\n', 1)%
SensorPositiony0=fscanf(FID, 'SensorPositiony0=%f\n', 1)%
SensorSpacingx=fscanf(FID, 'SensorSpacingx=%f\n', 1)%
SensorSpacingy=fscanf(FID, 'SensorSpacingy=%f\n', 1)%
MaxNoise=fscanf(FID, 'MaxNoise=%f\n', 1)%
SamplingFreq=fscanf(FID, 'SamplingFreq=%f\n', 1)%
object_exists=fscanf(FID, 'object_exists=%d\n', 1)%
object_mode=fscanf(FID, 'object_mode=%d\n', 1)%
wrt_file=fscanf(FID, 'wrt_file=%d\n', 1)%
noisy=fscanf(FID, 'noisy=%d\n', 1)%

end

N=Na*Ndx*Ndy;  % total number of files is Na*Ndx*Ndy
N

limite=2*(zz2-zz1)/PSS*SamplingFreq
limite=floor(limite+1);  % Limit in samples to reach the max depth
```

```
% For TGC later on

TGC=[0:(Nt-1)];
%TGC=exp((alpha*PSS/SamplingFreq)*TGC);
 TGC=exp((alpha*PSS/SamplingFreq)*TGC).*TGC.
                  *TGC*PSS*PSS/SamplingFreq/SamplingFreq;


% Read data :

for i=1:N
filename = [name, int2str(i-1)];
 fid = fopen(filename,'r');

if realsim==0
 fread(fid,12*16+4,'char');                  % get rid of the header
 sarace = fread(fid, [Nt, 1], 'short');
else
 sarace=fscanf(fid,'%f',[ Nt 1 ]);
end

%size(sarace);
Data(:,i)=sarace;

% TGC
if (tgcflag>0)
  Data(:,i)=Data(:,i).*TGC';
end

Data(:,i)=Data(:,i)-mean(Data(:,i));    % Remove DC

fclose(fid);
end

%Low Pas Filtering of raw data.
```

```
f = [0 2 4 30]/30;                % 80 kHz sampling rate
m = [1 1 0 0];                    % low pass filter
w = [1 1];                        % equal ripple in passband
                                  % and stopband
g = remez(51, f, m, w);           % create filter


temp = [Data; zeros(size(Data))];
                      % zero pad to compensate for circular conv.
out = fftfilt(g, temp);
                      % filter columns of "temp" with filter "g"
out = out(26:1024+25,:);          % keep relevant data
clear temp;
temp = mean(out);
for(i=1:N)
   out(:,i) = out(:,i) - temp(i);  % remove DC component
end

% Delay and Sum
%
% Signals from the receiver array are delayed to arrive
% in phase at the focus depth and then summed to increase
% the Signal to Noise Ratio.

if (focus>0)

%Computation of the delays


delays=zeros(Ndx,Ndy,Ndeep);
FocusDepth=zeros(1,Ndeep);
FocusDepth=[1:Ndeep];
FocusDepth=(zz2-zz1)*(FocusDepth-0.5)/Ndeep
FocusChunk=floor(2*(SamplingFreq*(zz2-zz1)/PSS*[0:Ndeep])/Ndeep+1)
   for ix=1:Ndx % For each element of the receiver array
     for iy=1:Ndy
       for iD=1:Ndeep
 focus=FocusDepth(iD);
 d1=1/2*sqrt(yyt^2+xxt^2);
```

```
 d2=sqrt(d1^2+focus^2);

 d3y=-yyt/2+(iy-Ndy/2-0.5)*SensorSpacingy;
 d3x=-xxt/2+(ix-Ndx/2-0.5)*SensorSpacingx;
 d3=sqrt(d3x^2+d3y^2+focus^2);

 d=d2+d3-2*focus;
 delays(ix,iy,iD)=floor(d/PSS*SamplingFreq)+1;

       end %iD
     end %iy
   end %ix

   delays

% Computation of delayed signals.

temp=zeros(Nt,Na,Ndeep);

for s=1:Na    % For each sample
   for ix=1:Ndx % For each element of the receiver array
     for iy=1:Ndy
       for iD=1:Ndeep
 D=delays(ix,iy,iD);
 temp(:,s,iD)=temp(:,s,iD)+[ out(D:Nt,Ndx*Ndy*
               (s-1)+(ix-1)*Ndy+iy)/(Ndx*Ndy); zeros((D-1),1)];
       end %iD
     end %s
   end %iy
end %ix

% Concatenation of delayed signals to have image in focus
% at Ndeep depths.

temp2=zeros(Nt,Na);


% Weigthing vectors for smoothing
```

```
Poids2=(0:1:(FocusChunk(2)-FocusChunk(1)))./
                                    (FocusChunk(2)-FocusChunk(1))
Poids1=1-Poids2;

Poids2=Poids2';
Poids1=Poids1';
HalfChunk=floor(size(Poids2)/2);

for s=1:Na    % For each sample

  temp2(FocusChunk(1):FocusChunk(1)+HalfChunk,s)= ...
      temp(FocusChunk(1):FocusChunk(1)+HalfChunk,s,1);

  temp2(FocusChunk(Ndeep+1)-HalfChunk:FocusChunk(Ndeep+1),s)= ...
      temp(FocusChunk(Ndeep+1)-HalfChunk:FocusChunk(Ndeep+1),s,Ndeep);

  for iD=2:(Ndeep)
%  size (temp(FocusChunk(iD)-HalfChunk:FocusChunk(iD+1)-HalfChunk,s,iD))
    temp2(FocusChunk(iD)-HalfChunk:FocusChunk(iD+1)-HalfChunk,s)= ...
Poids1.*temp(FocusChunk(iD)-HalfChunk:FocusChunk(iD+1)
        -HalfChunk,s,iD-1)+Poids2.*temp(FocusChunk(iD)
        -HalfChunk:FocusChunk(iD+1)-HalfChunk,s,iD);
  end %iD
end %s


out=temp2;
end



% Log, Enveloppe detection and display

temp = 20*log10(abs(hilbert(out)));  % log and enveloppe detection
temp = temp - max(max(temp));% just to have a nice visual result

% for the scale of the picture
rangee = (0:(Nt-1))/(2*SamplingFreq)*PSS;
azimuth = (0:(Na-1))*da;
```

```
% displays the picture.
% Dynamic range is 30 dB.

imagesc(azimuth, rangee(1:limite), temp(1:limite,:), [-30 0]);
axis('image');
xlabel('Horizontal distance, [m]');
ylabel('Range, [m]');

colorbar;
```

APPENDIX B: SIMSIGNAL PROGRAM CODE

```
//=================================================================
//
//  C++ code soil subsurface
// data collection simulation.
//      Produces received signals.
//
// Nail Cadalli, December 1997
//      Modified : April, 1999
//      Modified : January 2002 By Jonathan Mamou
//
//      The january 2002 modification generates received signals.
//      Single element transmitter.
//      Any rectangular array can be simulated and receive.
//
//=================================================================
#include "SIM.h"
#include "Cmat_simple.h"
#include "SIM_Routines.h"

#define USAGE "\
*-------------------------------------------------------*\n\
|      Soil Subsurface SAR Data Generation |\n\
*-------------------------------------------------------*\n\
Argument(s):  >> <transmitted signal filename> \n\
              >> <received signal filename> \n\
              >> [<reflection coeff. filename>] \n\n"
#define HEADER "\
*-------------------------------------------------------*\n\
|      Soil Subsurface SAR Data Generation  |\n\
*-------------------------------------------------------*\n\n"
```

```
//=============================================================
// MAIN
//=============================================================
void main(int argc, char *argv[])
{
  FILE *logf;

  //logf=fopen("monitor_file","w");
  //fclose(logf);
  printf("Simulate source OK");
  char
    *str1=new char[100],
    *str2=new char[100],
    *str3=new char[100],
    *str4=new char[100];

  printf(">>");
  for (int kk=0;kk<argc;kk++) printf("%s ",argv[kk]);
  printf("\n");

  if (argc < 3) {printf(USAGE); exit(1);}
  else
    {
      strcpy(str1,argv[1]);
      strcpy(str2,argv[2]);
      printf(HEADER);
    }

  if (argc > 3)
    strcpy(str3,argv[3]);
  else
    strcpy(str3,"");
  if (argc > 4)
     strcpy(str4,argv[4]);
  else
    strcpy(str4,"");

  printf("param file : %s\n",str3);
```

```
read_param(str3);   //reads parameters from file defined in SIM.h
//  read_param2();


int
  ki,kj,li,lj,
  k,l,
  it, ix, iy, iz, ipx, ipy, is, isk, isl, ir, ia;



onofftype
  displayRRcoefficients;

double
  x,y,z,t,xp,yp,xp0, yp0, ya, xxxt, yyyt,
  dist1, dist2,
  centerx, centery,
  lag,factor,var1,var2,var3,var4,var5,var6,
  dummy1,dummy2,dummy3,
  sourcedist,
  time, expfactor2,
  test;

complex
  gxz,
  expfactor1;

matrix
  g(Nd*Na,Nt), gg(Nt,1),
  s(Nt,1),
  SensorPosition(Nd,1); //transducer position



complex ***RRR=new complex** [Nx];
for (k=0;k<Nx;k++)
  {
```

```
      RRR[k]=new complex* [Ny];
      for (l=0;l<Ny;l++) RRR[k][l]=new complex [Nz];
    }

  double ***Distance=new double** [Nd];
  for (k=0;k<Nd;k++)
    {
      Distance[k]=new double* [Np];
      for (l=0;l<Np;l++) Distance[k][l]=new double [Np];
    }

  Simulate_RealSource(s);

  printf("Simulate source OK\n");

  Assign_RelativeArrayPositions(SensorPosition);
  //pointing the lower left corner of each transducer
  //for later simplicity.
  SensorPosition = SensorPosition +
                 complex(-(DiamRec/2)+dpx/2,-(DiamRec/2)+dpy/2);

 printf("Assign_RelativeArrayPositions OK");
  for (ir=0;ir<Nr;ir++)             //realizations
    {
      printf("**** Realization # %d *****\n", ir);
      Assign_Reflection(RRR,object_mode,wrt_file,noisy,"Ref.dat",str4);
      for (ya=ya1,ia=0; ia < Na; ia++,ya+=da) //position of array center
{
  printf("Array is at y=%lf\n",ya);

  //monitor the ia steps by a file
  logf=fopen("monitor_file","a");
  fprintf(logf,"%d",ia);
  fclose(logf);

  //transmitter's  absolute coordinates
  xxxt=0+xxt;
  yyyt=ya+yyt;
```

```
  printf("Receiver is at (%lf,%lf)\n",xxxt,yyyt);

  for (x=xx1,ix=0;ix<Nx;ix++,x+=dx)  //x-axis
    {
      //printf("ix=%d, x=%lf\n",ix,x);
      dummy1=(x-xxxt)*(x-xxxt);

      for (y=yy1,iy=0;iy<Ny;iy++,y+=dy)  //y-axis
{
  //printf("  iy=%d, y=%lf\n",iy,y);
  dummy2=dummy1 + (y-yyyt)*(y-yyyt);

  for (z=zz1,iz=0;iz<Nz;iz++,z+=dz) //z-axis
    {
      //printf("    iz=%d, z=%lf\n",iz,z);
      var3=z*z;
      sourcedist = sqrt(dummy2 + var3);

      //make a lookup table for scatterer-rec distances
      for (isk=0; isk<Nd; isk++)
{
  for (xp=real(SensorPosition(isk)),
 ipx=0;ipx<Np;ipx++,xp+=dpx)
    for (yp=imag(SensorPosition(isk))+ya,
   ipy=0;ipy<Np;ipy++,yp+=dpy)
      {
test=(ipx+0.5-Np/2.0)*(ipx+0.5-Np/2.0)+(ipy+0.5-Np/2.0)*
                                  (ipy+0.5-Np/2.0);
if (test<=(float) (Np*Np/4.0))   //Circular
  {
    if (ix==0 && iy==0 && iz==0)
      printf("In the circle: (%lf,%lf)\n",xp,yp);
    var1=x-xp;
    var2=y-yp;
    Distance[isk][ipx][ipy]=
      sqrt(var1*var1 + var2*var2 + var3);
  }
      }
} //isk
```

```
      for (isk=0; isk<Nd; isk++)
{
  gg.resize(Nt,1);
  //INTEGRATION ON THE TRANSDUCER SURFACE
  for (ki=0; ki<Np; ki++)
    for (kj=0; kj< Np; kj++)
      {
test=(ki+0.5-Np/2.0)*(ki+0.5-Np/2.0)+(kj+0.5-Np/2.0)*(kj+0.5-Np/2.0);
if (test<=(float) (Np*Np/4.0))   //Circular
  {
    dist1=Distance[isk][ki][kj]+sourcedist;
    dist2=Distance[isk][ki][kj]*sourcedist;
    lag=dist1/PSS;


    for (t=tt1,it=0; it<Nt; it++, t+=dt)
      gg(it) += RealSignal(t-lag,lag,dist1,dist2);
  }
      } // ki, kj
  for (it=0;it<Nt;it++)
    g(isk+Nd*ia,it) += gg(it)*RRR[ix][iy][iz];
}//isk
    }//iz
}//iy
    }//ix
}// ia
    }// ir

  g *= dpx * dpx * dpy * dpy / Nr;
  s.display(str1);
  g.display(str2);
  g.saveas(str2,Nd,Na,Nt);
  printf("\nTransmitted signal is recorded in file %s\n",str1);
  printf("Received signal is recorded in file %s\n",str2);

}//main
```

APPENDIX C: SIMULATION PARAMETER AND POINT SCATTERER FILES

C.1 Single Target in a Pulse-Echo Configuration

C.1.1 Parameter file

```
PSS=200
freq=2000
CycleCount=2
alpha=13.8
DiamTrans=0.078
DiamRec=0.051
xxt=0.0
yyt=0.0
Ndx=1
Ndy=1
Nx=51
Ny=51
Nz=51
Nt=1024
Np=1
Ndeep=20
Nr=1
Na=51
da=0.01
ya1=-0.25
xx1=-0.5
xx2=0.5
yy1=-0.5
```

```
yy2=0.5
zz1=0
zz2=0.5
tt1=0
ox1=0
ox2=0.1
oy1=0
oy2=0.1
oz1=0.1
oz2=0.7
SensorPositionx0=0.0
SensorPositiony0=0.0
SensorSpacingx=0.01
SensorSpacingy=0.01
MaxNoise=0.0000005
SamplingFreq=60060
object_exists=1
object_mode=2
wrt_file=0
noisy=0
```

C.1.2 Point scatterer file

```
(0.0,0.0,0.25)=1.0
```

C.2 Time Gain Compensation Validation Simulation

C.2.1 Parameter file

```
PSS=200
freq=2000
CycleCount=2
alpha=13.8
DiamTrans=0.078
DiamRec=0.051
xxt=0.0
yyt=0.0
```

```
Ndx=1
Ndy=1
Nx=51
Ny=51
Nz=51
Nt=1024
Np=1
Ndeep=20
Nr=1
Na=51
da=0.01
ya1=-0.25
xx1=-0.5
xx2=0.5
yy1=-0.5
yy2=0.5
zz1=0
zz2=0.5
tt1=0
ox1=0
ox2=0.1
oy1=0
oy2=0.1
oz1=0.1
oz2=0.7
SensorPositionx0=0.0
SensorPositiony0=0.0
SensorSpacingx=0.01
SensorSpacingy=0.01
MaxNoise=0.0000005
SamplingFreq=60060
object_exists=1
object_mode=2
wrt_file=0
noisy=0
```

C.2.2 Point scatterer file

```
(0.0,0.0,0.05)=1.0
(0.0,0.0,0.15)=1.0
(0.0,0.0,0.25)=1.0
(0.0,0.0,0.35)=1.0
(0.0,0.0,0.45)=1.0
```

C.3 Delay-and-Sum Beamforming Simulation

C.3.1 Parameter file

```
PSS=200
freq=2000
CycleCount=2
alpha=13.8
DiamTrans=0.078
DiamRec=0.051
xxt=0.20
yyt=0.0
Ndx=8
Ndy=1
Nx=51
Ny=51
Nz=51
Nt=1024
Np=4
Ndeep=20
Nr=1
Na=50
da=0.01
ya1=-0.25
xx1=-0.5
xx2=0.5
yy1=-0.5
yy2=0.5
zz1=0
zz2=0.5
tt1=0
ox1=0
```

```
ox2=0.1
oy1=0
oy2=0.1
oz1=0.1
oz2=0.7
SensorPositionx0=0.0
SensorPositiony0=0.0
SensorSpacingx=0.018
SensorSpacingy=0.018
MaxNoise=0.0000005
SamplingFreq=60060
object_exists=1
object_mode=2
wrt_file=0
noisy=0
```

C.3.2 Point scatterer file

```
(0.0,0.0,0.05)=1.0
(0.0,0.0,0.15)=1.0
(0.0,0.0,0.25)=1.0
(0.0,0.0,0.35)=1.0
(0.0,0.0,0.45)=1.0
```