UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA POLITÉCNICA SUPERIOR DE GANDIA

Máster en Ingeniería Acústica

Anexos a la memoria Documento nº 1.

Propagación transcraneal de ultrasonidos

TRABAJO FINAL DE MASTER

Autor/a:

Sergio Jiménez Gambín

Tutor/a:

D. Francisco Camarena Femenía

Cotutor/a:

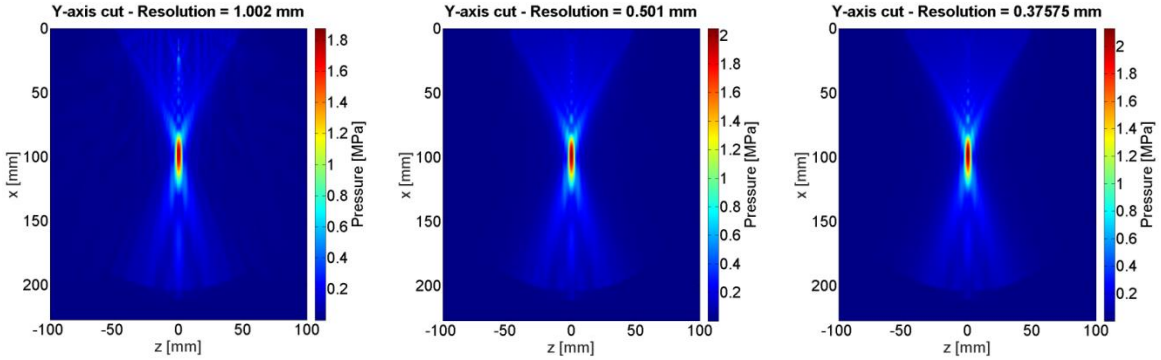
D. Noé Jiménez González

GANDIA, 09/2016

Índice

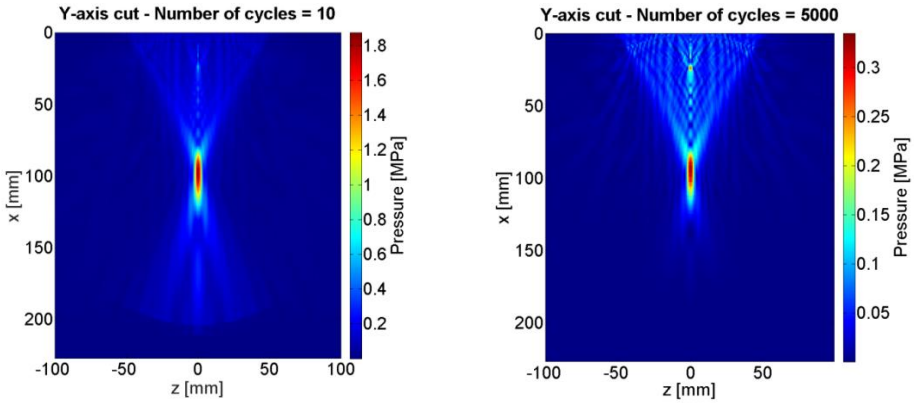
1. Anexos.

1.1. Resultados.



a) b) c)

Fig. 54. Simulación en agua; efecto de la resolución espacial; cortes en el eje Y.



a) b)

Fig. 55. Simulación en agua fijando el tiempo de propagación de la onda a 135 µs y variando el número de ciclos; cortes en el eje Y.

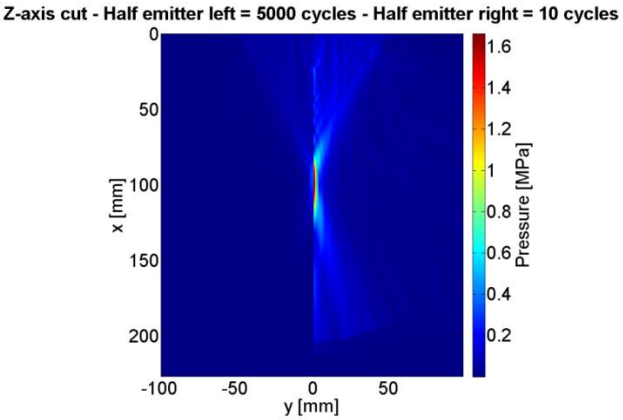


Fig. 56. Simulación en agua fijando el tiempo de propagación de la onda a 135 µs; señal emitida con 5000 ciclos (mitad izquierda), señal emitida con 10 ciclos (mitad derecha).

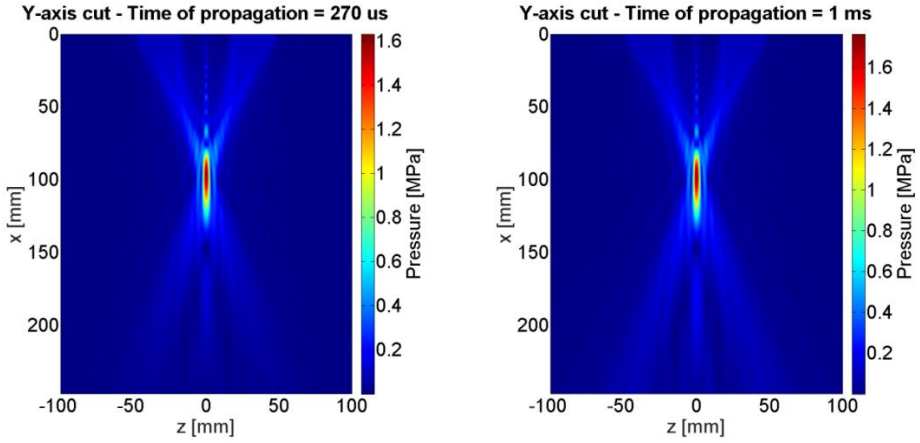


Fig. 57. Simulación en agua fijando el número de ciclos a 5000 y variando el tiempo de propagación; cortes en el eje Y.

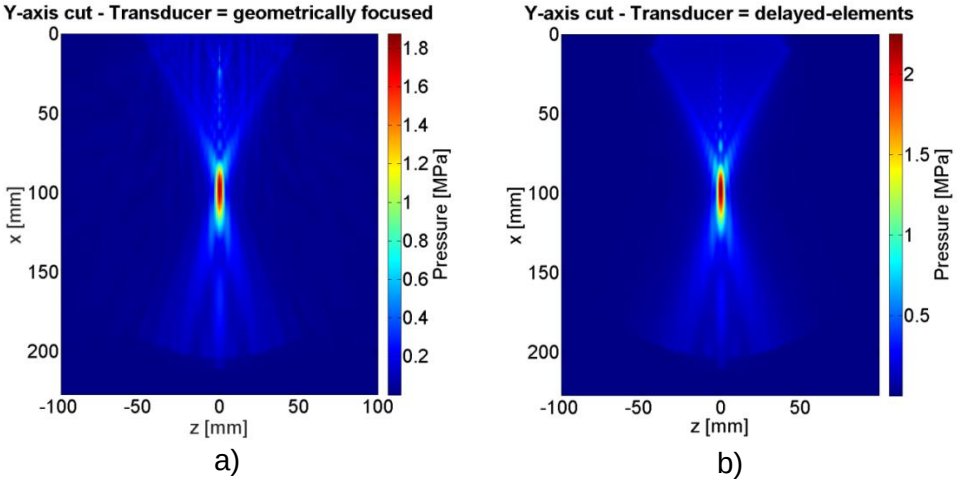


Fig. 58. Simulación en agua con transductor focalizado geométricamente o de forma natural (a) o con transductor plano con elementos retardados temporalmente (b); cortes en el eje Y.

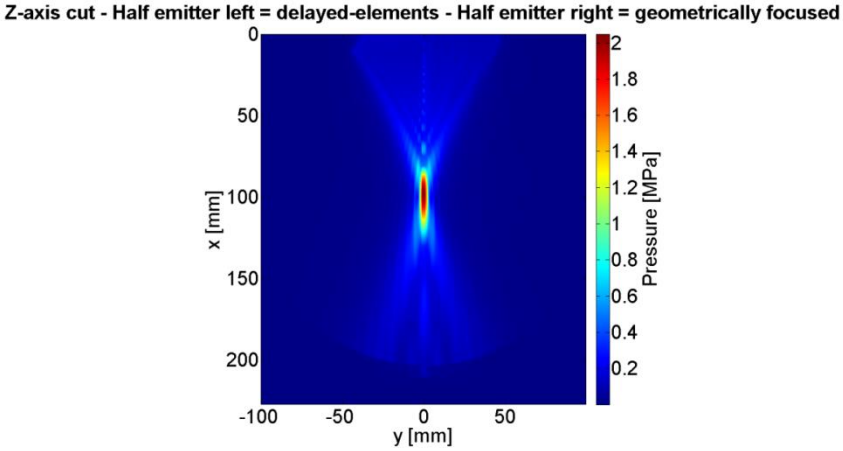
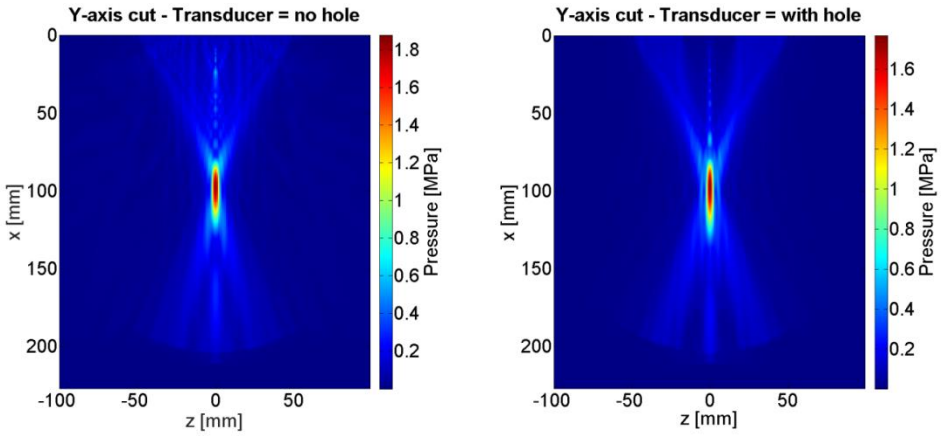


Fig. 59. Simulación en agua con transductor plano focalizado mediante retardos temporales (mitad izquierda)y con transductor focalizado geométricamente (mitad derecha).



a) b)

Fig. 60. Simulación en agua con transductor focalizado geométricamente sin hueco (a) o con hueco (b); cortes en el eje Y.

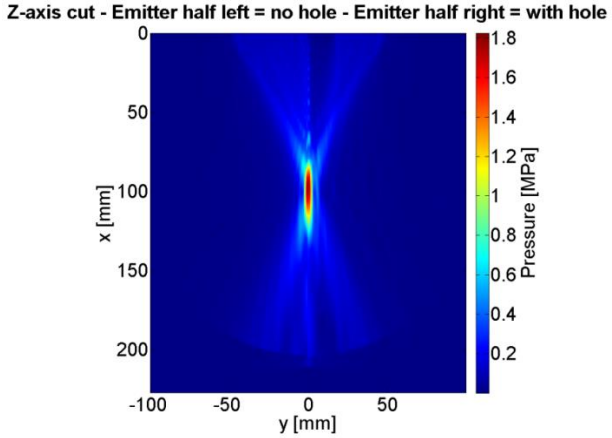
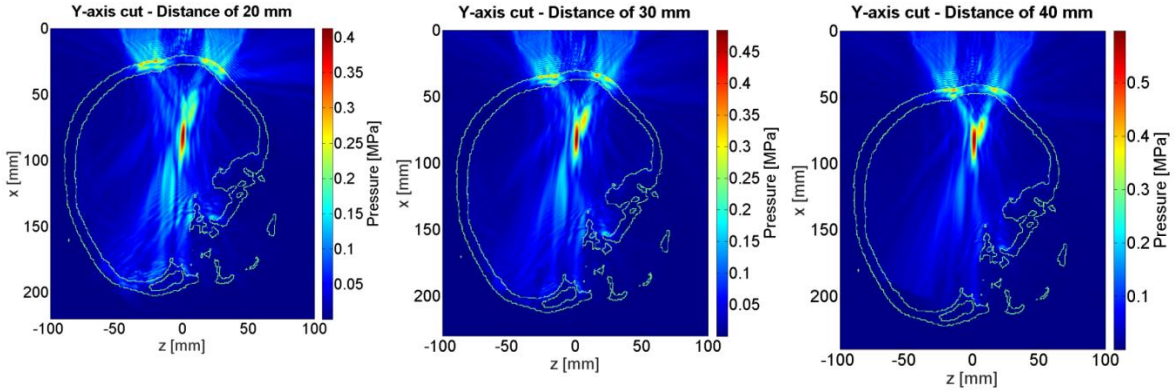
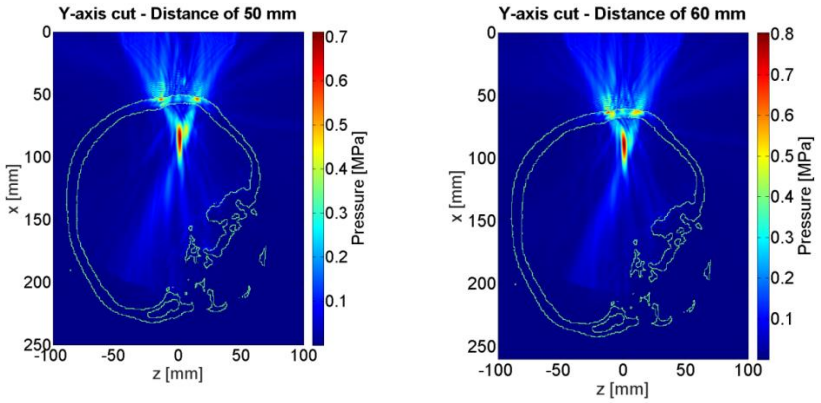


Fig. 61. Simulación en agua con transductor focalizado geométricamente sin hueco (mitad izquierda) y con hueco (mirad derecha).

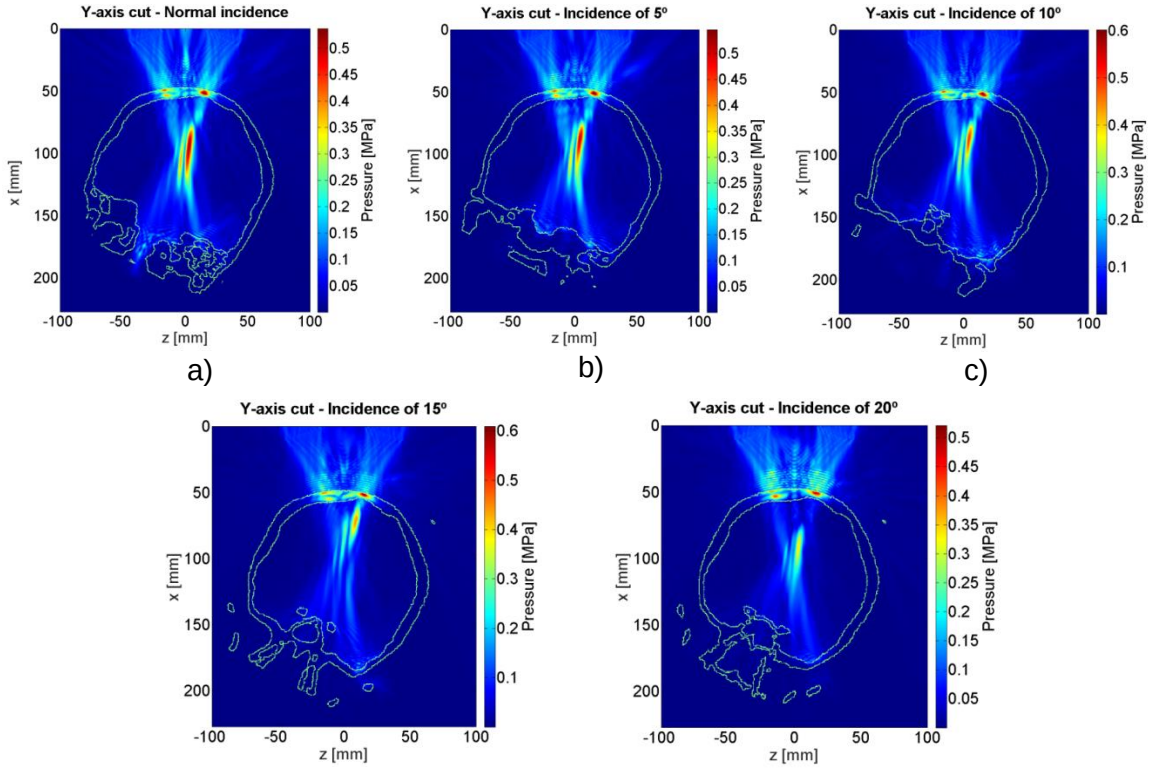


a) b) c)



d) e)

Fig. 62. Simulación propagando a través de la zona occipital central variando la distancia de separación transductor-cráneo; cortes en el eje Y.



a) b) c)

d) e)

Fig. 63. Simulación propagando a través de la zona parietal superior derecha variando el ángulo de incidencia; cortes en el eje Y.

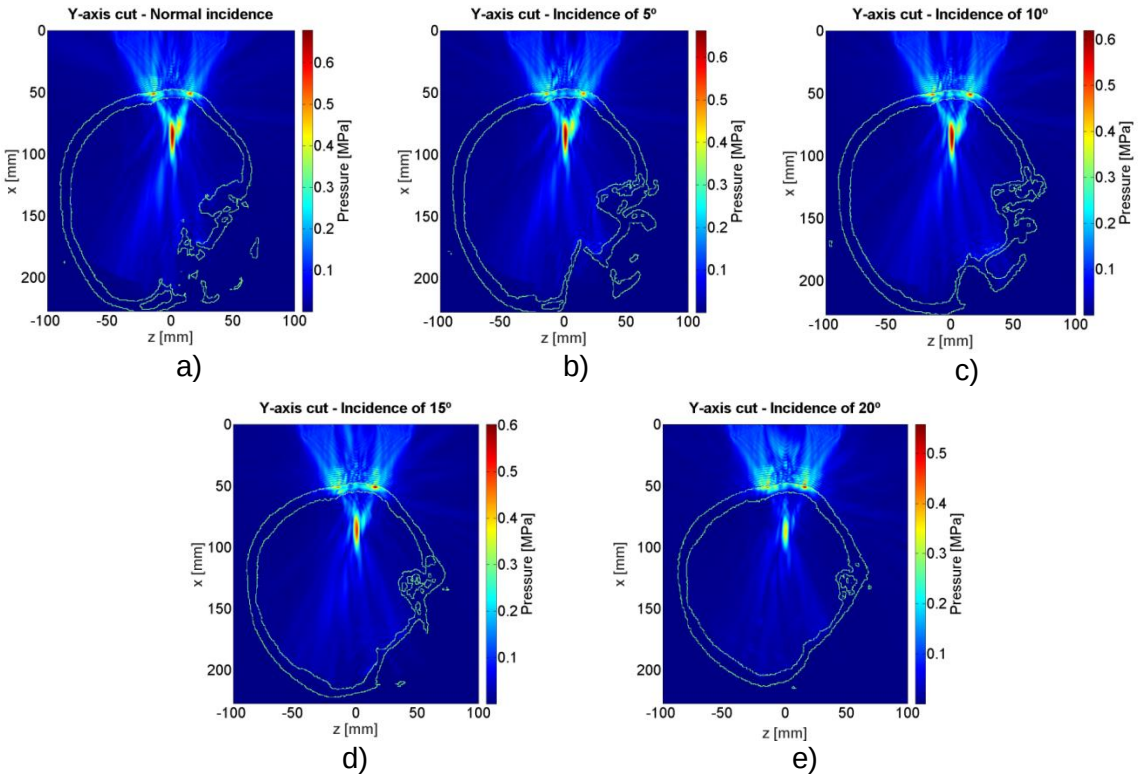


Fig. 64. Simulación propagando a través de la zona occipital central variando el ángulo de incidencia; cortes en el eje Y

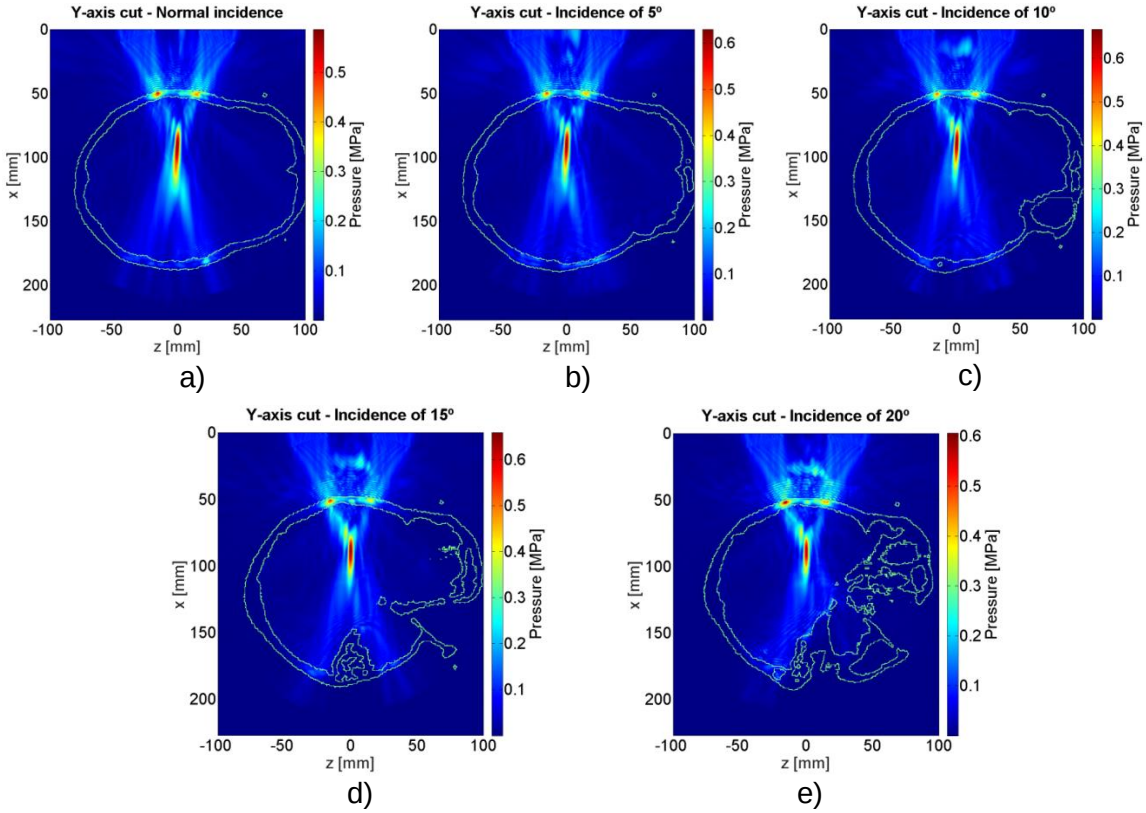


Fig. 65. Simulación propagando a través de la zona temporal derecha variando el ángulo de incidencia; cortes en el eje Y.

1.2. Código.

Script principal para lectura y ajuste para incidencia normal de la CT:

data_adq.m

```
%% Data adquisition

clc; clear; close all;

zone = 1;

dev1 = '0.00';

dev2 = '0.00';

targetP = [str2double(dev1),str2double(dev2)]; % Targeting zone n in coordinate (c1,c2)
[mm] ([0,0] case is the center)

[rho,c,alpha,mask,ds] = AdjustCT(zone,targetP);

save(['CT_SY_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].mat'], 'rho', 'c',
'alpha', 'mask', 'ds');

Plot3D(c);
```

Funciones secundarias del script ***data_adq.m***:

AdjustCT.m

```
function [rho,c,alpha,mask,hip_mask,ds] = AdjustCT(zone,targetP)

    %% Open directory of CT data

    load('CT_SY.mat');

    %load('CT_SY_zone=2_targetP=[-10.00,0.00].mat');

    c_water = min(c(:));

    rho_water = min(rho(:));

    c = c.*mask;

    rho = rho.*mask;

    %% Permute rows, columns or layers depending on zone of incidence

    if(zone~=1)

        [mask,rho,c] = permuteSkull(zone,mask,rho,c);

    end

    %% Locate the skull in order to have normal incidende beam in specified point

    D = 100; % Diameter of the incidence area of the beam onto skull [grid points]

    [rho,c,mask] = zoneAnalyzer(targetP,rho,c,mask,ds,D);

    alpha = coeff(1)*mask+coeff(2)*(1-mask);

    c(mask==0) = c_water;

    rho(mask==0) = rho_water;
```

```
end
```

permuteSkull.m

```
function [skull_mask2, skull_density2, skull_sound_speed2] =
permuteSkull(zone,skull_mask,skull_density,skull_sound_speed)

    N = size(skull_mask,1);

    skull_mask2 = zeros(N,N,N);

    skull_density2 = zeros(N,N,N);

    skull_sound_speed2 = zeros(N,N,N);

    cut3Axes(skull_mask);

    if(zone==2)

        for ii=1:N

            for kk=1:N

                skull_mask2(:,ii,kk) = skull_mask(end-ii+1,:,kk);

                skull_density2(:,ii,kk) = skull_density(end-ii+1,:,kk);

                skull_sound_speed2(:,ii,kk) = skull_sound_speed(end-ii+1,:,kk);

            end

        end

    elseif(zone==3)

        for jj=1:N

            for kk=1:N

                aux_mask(1,:) = skull_mask(:,jj,kk);

                skull_mask2(kk,jj,:) = fliplr(aux_mask);

                aux_c(1,:) = skull_sound_speed(:,jj,kk);

                skull_sound_speed2(kk,jj,:) = fliplr(aux_c);

                aux_rho(1,:) = skull_density(:,jj,kk);

                skull_density2(kk,jj,:) = fliplr(aux_rho);

            end

        end

    elseif(zone==4)

        for jj=1:N

            for kk=1:N

                skull_mask2(end-kk+1,jj,:) = skull_mask(:,jj,kk);

                skull_sound_speed2(end-kk+1,jj,:) = skull_sound_speed(:,jj,kk);

                skull_density2(end-kk+1,jj,:) = skull_density(:,jj,kk);

            end

        end
```

```
    end

    cut3Axes(skull_mask2);

end
```

cut3Axes.m

```
function cut3Axes(mask)

    load kgrid.mat;

    figure;

    %% Y-axis cut

    cut(:,:) = mask(:,round(end/2),:);

    subplot(221);

    imagesc(kgrid.z_vec*1e3,(kgrid.x_vec-kgrid.x_vec(1))*1e3,cut);

    xlabel('z [mm]'),ylabel('x [mm]'),title('Y-axis cut'),axis image;

    %cb = colorbar; ylabel(cb, 'Sound speed [m/s]'),drawnow;

    %cb = colorbar; ylabel(cb, 'Density [kg/m^{3}]'),drawnow;

    %cb = colorbar; ylabel(cb, 'Absorption coefficient [dB/(MHz^{1.1}cm)]'),drawnow;

    cb = colorbar; ylabel(cb, 'Skull mask'),drawnow;

    %% Z-axis cut

    clear cut;

    cut(:,:) = mask(:,:,round(end/2));

    subplot(222);

    imagesc(kgrid.y_vec*1e3,(kgrid.x_vec-kgrid.x_vec(1))*1e3,cut);

    xlabel('y [mm]'),ylabel('x [mm]'),title('Z-axis cut'),axis image;

    %cb = colorbar; ylabel(cb, 'Sound speed [m/s]'),drawnow;

    %cb = colorbar; ylabel(cb, 'Density [kg/m^{3}]'),drawnow;

    %cb = colorbar; ylabel(cb, 'Absorption coefficient [dB/(MHz^{1.1}cm)]'),drawnow;

    cb = colorbar; ylabel(cb, 'Skull mask'),drawnow;

    %% X-axis cut

    clear cut;

    cut(:,:) = mask(round(end/2),:,:);

    subplot(223);

    imagesc(kgrid.z_vec*1e3,kgrid.y_vec*1e3,cut);

    xlabel('z [mm]'),ylabel('y [mm]'),title('X-axis cut'),axis image;

    %cb = colorbar; ylabel(cb, 'Sound speed [m/s]'),drawnow;

    %cb = colorbar; ylabel(cb, 'Density [kg/m^{3}]'),drawnow;

    %cb = colorbar; ylabel(cb, 'Absorption coefficient [dB/(MHz^{1.1}cm)]'),drawnow;
```

```
    cb = colorbar; ylabel(cb, 'Skull mask'),drawnow;

end
```

zoneAnalyzer.m

```
function [rho,c,mask,ds] =
zoneAnalyzer(targetP,skull_density,skull_sound_speed,skull_mask,ds,D)

    %% Rotation in YZ plane

    step = 1;

    % Get angle (theta)

    N = size(skull_mask,1);

    Yc = round(N/2);

    Zc = round(N/2);

    centerP = [Yc,Zc];

    theta = SSNIA(step,targetP,centerP,skull_mask,ds,D)

    F = 1;

    skull_mask2 = redim(skull_mask,1/F,1/F,1/F);

    skull_density2 = redim(skull_density,1/F,1/F,1/F);

    skull_sound_speed2 = redim(skull_sound_speed,1/F,1/F,1/F);

    clear skull_mask skull_density skull_sound_speed;

    skull_mask = skull_mask2;

    skull_density = skull_density2;

    skull_sound_speed = skull_sound_speed2;

    % Grid expansion in order not to lose data when rotating (adding "2*add" grid
points to 3 axis)

    N = size(skull_mask,1);

    add = round(N/2);

    aux_skull_mask = zeros(N+2*add,N+2*add,N+2*add);

    aux_skull_density = zeros(N+2*add,N+2*add,N+2*add);

    aux_skull_sound_speed = zeros(N+2*add,N+2*add,N+2*add);

    aux_skull_mask(add+1:end-add,add+1:end-add,add+1:end-add) = skull_mask;

    aux_skull_density(add+1:end-add,add+1:end-add,add+1:end-add) = skull_density;

    aux_skull_sound_speed(add+1:end-add,add+1:end-add,add+1:end-add) =
skull_sound_speed;

    cut3AxesExpanded(aux_skull_mask);

    % Apply rotation of "theta" degrees from reference point (yTar,zTar) displaced to
the center of the beam

    yc_beam = round(N/2)+add;   % Y-axis point where the beam incides onto skull

    zc_beam = round(N/2)+add;   % Z-axis point where the beam incides onto skull
```

```
    yTar = round(targetP(1)/(1000*ds*F))+round(centerP(1)/F)+add;

    zTar = round(targetP(2)/(1000*ds*F))+round(centerP(2)/F)+add;

    Z = zeros(N+2*add,N+2*add);

    for ii=1+add:N+add

        aux_skull_mask(ii,:,:) =
rotate(Z,aux_skull_mask(ii,:,:),yTar,zTar,theta,yc_beam-yTar,zc_beam-zTar);

        aux_skull_density(ii,:,:) =
rotate(Z,aux_skull_density(ii,:,:),yTar,zTar,theta,yc_beam-yTar,zc_beam-zTar);

        aux_skull_sound_speed(ii,:,:) =
rotate(Z,aux_skull_sound_speed(ii,:,:),yTar,zTar,theta,yc_beam-yTar,zc_beam-zTar);

    end

    % Fill empty gaps due to rotation (interpolation)

    T = 1;

    M = 2;

    [aux_skull_mask, aux_skull_density, aux_skull_sound_speed] =
fillGaps(T,M,add,step,aux_skull_mask,aux_skull_density,aux_skull_sound_speed);

    cut3AxesExpanded(aux_skull_mask);

    %% Rotation in XZ plane

    step = 2;

    % Get angle (theta)

    targetP = [0,0]; % It's zero because in step 1 skull was aligned to beam incidence
point

    theta = SSNIA(step,targetP,round(centerP/F)+add,aux_skull_mask,F*ds,round(D/F))

    % Apply rotation of "theta" degrees from reference point (xTar,zTar) displaced to
the center of the beam

    N = size(aux_skull_mask,1);

    xc_beam = 1;  % X-axis point where the beam incides onto skull

    xc_beam = xc_beam+add;

    xTar = find(aux_skull_mask(:,round(Yc/F)+add,round(Zc/F)+add),1);

    zTar = round(Zc/F)+add;

    Z = zeros(N,N);

    for ii=1:N

        aux_skull_mask(:,ii,:) =
rotate(Z,aux_skull_mask(:,ii,:),xTar,zTar,theta,xc_beam-xTar,zc_beam-zTar);

        aux_skull_density(:,ii,:) =
rotate(Z,aux_skull_density(:,ii,:),xTar,zTar,theta,xc_beam-xTar,zc_beam-zTar);

        aux_skull_sound_speed(:,ii,:) =
rotate(Z,aux_skull_sound_speed(:,ii,:),xTar,zTar,theta,xc_beam-xTar,zc_beam-zTar);

    end

    % Grid reduction in order to obtain original size of CT data after rotating
(cutting the central NxNxN grid points)
```

```
    skull_mask = aux_skull_mask(add+1:end-add,add+1:end-add,add+1:end-add);

    skull_density = aux_skull_density(add+1:end-add,add+1:end-add,add+1:end-add);

    skull_sound_speed = aux_skull_sound_speed(add+1:end-add,add+1:end-add,add+1:end-
add);

    % Fill empty gaps due to rotation (interpolation)

    add = 0;

    T = 1;

    M = 2;

    [mask,rho,c] = fillGaps(T,M,add,step,skull_mask,skull_density,skull_sound_speed);

    N = size(mask,1);

    add = round(N/2);

    aux_skull_mask(:) = 0;

    aux_skull_mask(add+1:end-add,add+1:end-add,add+1:end-add) = mask;

    cut3AxesExpanded(aux_skull_mask);

end
```

SSNIA.m

```
function theta = SSNIA(step,targetP,centerP,skull_mask,ds,D)

    N = size(skull_mask,1);

    mask = zeros(N,N);

    R = round(D/2);

    yc = centerP(1)+round(targetP(1)/(1000*ds));

    zc = centerP(2)+round(targetP(2)/(1000*ds));

    for y=yc-R:yc+R-1

        z = sqrt((R^2)-((y-yc)^2))+zc;

        mask(y,round(zc+(z-zc))) = 1;

        mask(y,round(zc-(z-zc))) = 1;

    end

    for ii=1:N

        row = mask(ii,:);

        if(sum(row)>0)

            ind = find(row==1);

            if(length(ind)>1)

                mask(ii,ind(1):ind(2)) = 1;

            end

        end

    end
```

```
    iter = 0;
    for ii=1:N
        for jj=1:N
            if((mask(ii,jj) == 1) && (sum(skull_mask(:,ii,jj)) > 0))
                iter = iter+1;
                x(iter) = ii;
                y(iter) = jj;
                aux = skull_mask(:,ii,jj);
                z(iter) = N-find(aux,1);
            end
        end
    end
    P = fit([x',y'],z','poly11')
    %cftool(x,y,z)
    if(step==1)
        v1 = P.p10;     % Coordinate in Y-axis of normal vector to skull surface
        v2 = P.p01;     % Coordinate in Z-axis of normal vector to skull surface
    elseif(step==2)
        v1 = P.p01;     % Coordinate in Y-axis of normal vector to skull surface
        v2 = -1;        % Coordinate in Z-axis of normal vector to skull surface
    else
        error('Error analyzing Zone: not valid chosen step.');
    end
    theta = -atan(v1/v2)*180/pi;
end
```

redim.m

```
function y = redim(x,x_scale,y_scale,z_scale)
    [Nx,Mx,Lx] = size(x);
    aux = imresize(x,[round(Nx*x_scale) round(Mx*y_scale)]);
    [N,M,L] = size(aux);
    y = zeros(N,M,round(L*z_scale));
    for ii=1:M
        aux2(:,:) = aux(:,ii,:);
        y(:,ii,:) = imresize(aux2,[N round(L*z_scale)]);
    end
```

```
end
```

rotate.m

```
function R = rotate(B,I,xr,yr,theta,xt,yt)
% This function rotates a 2D matrix.
%
% Usage:
%
%   R = rotate(B,E,xr,yr,theta,xt,yt)
%
%
% Inputs:
%
%   - B: background image (double format) nedded to locate the image E
%     which we want to rotate.
%
%   - E: image (double format) we want to rotate.
%
%   - xr: X-axis coordinate where we want image E to rotate
%
%   - yr: Y-axis coordinate where we want image E to rotate
%
%   - theta: angle (in degrees) of rotation.
%
%   - xt: X-axis coordinate where we want image E to traslate
%
%   - yt: Y-axis coordinate where we want image E to traslate
%
    E(:,:) = I;
    % Conversion from degrees to radians
    theta = theta*pi/180;
    % Get sizes of both images
    [Em,En] = size(E);
    [Fm,Fn] = size(B);
    % Traslation image
    TT1 = [1,0,xt;0,1,yt;0,0,1];
```

```
    % Translation matrix from center of E image to origin (0,0) (TT2):
    TT2 = [1,0,-xr;0,1,-yr;0,0,1];
    % Rotation matrix (TR):
    TR = [cos(theta),sin(theta),0;-sin(theta),cos(theta),0;0,0,1];
    % Translation matrix from center of E image to its center (xc,yc) (TT3):
    TT3 = [1,0,xr;0,1,yr;0,0,1];
    % General matrix (Mg):
    Mg = TT1*TT3*TR*TT2;
    % Application of transformation
    R = B;
    for i=1:Em
        for j=1:En
            w = Mg*[i;j;1];
            u = round(w(1));
            v = round(w(2));
            if((u<=Fm) && (u>=1) && (v<=Fn) && (v>=1))
                R(u,v) = E(i,j);
            end
        end
    end
end
```

fillGaps.m

```
function [mask,rho,c] = fillGaps(T,M,add,step,mask,rho,c)
% This function fills empty gaps created when rotating a 2D matrix.
%
% Usage:
%
%   [mask,rho,c] = fillGaps(T,M,add,step,mask,rho,c)
%
%
% Inputs:
%
%   -T: size (pixels) of the cutted region of (2T+1)*(2T+1) pixels.
%
%   -M: number of empty gaps in the region due to rotation. If the number
```

```
%   of empty gaps in that region is equal or smaller than M, mean procedure
%   will be applied to obtain values for empty density and sound speed,
%   also a value for the skull mask.
%
    [Nx,Ny,Nz] = size(mask);
    if(step==2)
        mask = permute(mask,[2 1 3]);
        rho = permute(rho,[2 1 3]);
        c = permute(c,[2 1 3]);
    elseif(step==3)
        mask = permute(mask,[3 2 1]);
        rho = permute(rho,[3 2 1]);
        c = permute(c,[3 2 1]);
    end
    for ii=1+add:Nx-add
        for jj=1+T:Ny-T
            for kk=1+T:Nz-T
                if(mask(ii,jj,kk) == 0)
                    aux = mask(ii,jj-T:jj+T,kk-T:kk+T);
                    Np = sum(1-aux(:))-1;     % number of empty pixels in this region
                    if(Np <= M)
                        mask(ii,jj,kk) = 1;
                        aux = rho(ii,jj-T:jj+T,kk-T:kk+T);
                        Np = length(aux(aux>0));    % number of pixels with non-zero
density in this region
                        rho(ii,jj,kk) = sum(aux(:))/Np;
                        aux = c(ii,jj-T:jj+T,kk-T:kk+T);
                        Np = length(aux(aux>0));    % number of pixels with non-zero
velocity in this region
                        c(ii,jj,kk) = sum(aux(:))/Np;
                    end
                end
            end
        end
    end
    if(step==2)
        mask = permute(mask,[2 1 3]);
```

```
        rho = permute(rho,[2 1 3]);
        c = permute(c,[2 1 3]);
    elseif(step==3)
        mask = permute(mask,[3 2 1]);
        rho = permute(rho,[3 2 1]);
        c = permute(c,[3 2 1]);
    end
end
```

cut3AxesExpanded.m

```
function cut3AxesExpanded(mask)
    load kgrid.mat;
    figure;
    %% Y-axis cut
    cut(:,:) = mask(:,round(end/2),:);
    subplot(221);
    imagesc(2*kgrid.z_vec*1e3,2*(kgrid.x_vec-kgrid.x_vec(1))*1e3,cut);
    xlabel('z [mm]'),ylabel('x [mm]'),title('Y-axis cut'),axis image;
    %cb = colorbar; ylabel(cb, 'Sound speed [m/s]'),drawnow;
    %cb = colorbar; ylabel(cb, 'Density [kg/m^{3}]'),drawnow;
    %cb = colorbar; ylabel(cb, 'Absorption coefficient [dB/(MHz^{1.1}cm)]'),drawnow;
    cb = colorbar; ylabel(cb, 'Skull mask'),drawnow;
    %% Z-axis cut
    clear cut;
    cut(:,:) = mask(:,:,round(end/2));
    subplot(222);
    imagesc(2*kgrid.y_vec*1e3,2*(kgrid.x_vec-kgrid.x_vec(1))*1e3,cut);
    xlabel('y [mm]'),ylabel('x [mm]'),title('Z-axis cut'),axis image;
    %cb = colorbar; ylabel(cb, 'Sound speed [m/s]'),drawnow;
    %cb = colorbar; ylabel(cb, 'Density [kg/m^{3}]'),drawnow;
    %cb = colorbar; ylabel(cb, 'Absorption coefficient [dB/(MHz^{1.1}cm)]'),drawnow;
    cb = colorbar; ylabel(cb, 'Skull mask'),drawnow;
    %% X-axis cut
    clear cut;
    cut(:,:) = mask(round(end/2),:,:);
    subplot(223);
```

```
    imagesc(2*kgrid.z_vec*1e3,2*kgrid.y_vec*1e3,cut);

    xlabel('z [mm]'),ylabel('y [mm]'),title('X-axis cut'),axis image;

    %cb = colorbar; ylabel(cb, 'Sound speed [m/s]'),drawnow;

    %cb = colorbar; ylabel(cb, 'Density [kg/m^{3}]'),drawnow;

    %cb = colorbar; ylabel(cb, 'Absorption coefficient [dB/(MHz^{1.1}cm)]'),drawnow;

    cb = colorbar; ylabel(cb, 'Skull mask'),drawnow;

end
```

Plot3D.mat

```
function Plot3D(data)

    figure;

    p = patch(isosurface(data));

    set(p, 'FaceColor', 'red', 'EdgeColor', 'none', 'FaceAlpha', 1);

    daspect([1 1 1])

    view(3)

    camlight

    lighting gouraud

    light('Position',[0 0 -1])

end
```

Script principal para lectura y ajuste para incidencia no normal del cráneo, empleando directamente el cráneo ajustado anteriormente para incidencia normal:

data_adq_2.m

```
clc; clear; close all;

zone = 3;

dev1 = '0.00';

dev2 = '0.00';

load(['CT_SY_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].mat']);

rho_water = min(rho(:));

c_water = min(c(:));

plane = 'XY';

theta = 20;

[rho,c,mask,hip_mask] = RNNI(plane,theta,rho.*mask,c.*mask,mask,hip_mask,ds);

rho(rho==0) = rho_water;

c(c==0) = c_water;

alpha = max(alpha(:))*mask+min(alpha(:))*(1-mask);
```

```
save(['CT_SY_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2str(theta),'
_plane=',plane,'.mat'], 'rho', 'c', 'alpha', 'mask', 'hip_mask', 'ds');

Plot3D(c(:,:,1:end-1));
```

Funciones secundarias del script ***data_adq_2.m*** (se emplean también algunas de las utilizadas en el ajuste de incidencia normal):

RNNI.m

```
%% Rotation for Not Normal Incidence (RNNI)

function [rho,c,mask,hip_mask] =
RNNI(plane,theta,skull_density,skull_sound_speed,skull_mask,hip_mask,ds)

    F = 1;

    N = size(skull_mask,1);

    Yc = round(N/(F*2));

    Zc = round(N/(F*2));

    skull_mask2 = redim(skull_mask,1/F,1/F,1/F);

    skull_density2 = redim(skull_density,1/F,1/F,1/F);

    skull_sound_speed2 = redim(skull_sound_speed,1/F,1/F,1/F);

    clear skull_mask skull_density skull_sound_speed hip_mask;

    skull_mask = skull_mask2;

    skull_density = skull_density2;

    skull_sound_speed = skull_sound_speed2;

    cut3Axes(skull_mask);

    % Apply rotation of "theta" degrees from reference point (xTar,zTar) displaced to
the center of the beam

    N = size(skull_mask,1);

    xc_beam = 20;  % X-axis point where the beam incides onto skull [mm]

    xc_beam = round(xc_beam/(1000*ds*F));

    xTar = find(skull_mask(:,Yc,Zc),1);

    Z = zeros(N,N);

    % Rotation in XZ plane

    if(strcmp(plane,'XZ'))

        zc_beam = Zc;

        zTar = Zc;

        for ii=1:N

            skull_mask(:,ii,:) = rotate(Z,skull_mask(:,ii,:),xTar,zTar,theta,xc_beam-
xTar,zc_beam-zTar);

            skull_density(:,ii,:) =
rotate(Z,skull_density(:,ii,:),xTar,zTar,theta,xc_beam-xTar,zc_beam-zTar);

            skull_sound_speed(:,ii,:) =
```

```
rotate(Z,skull_sound_speed(:,ii,:),xTar,zTar,theta,xc_beam-xTar,zc_beam-zTar);

        end

        step = 2;

    % Rotation in XZ plane

    elseif(strcmp(plane,'XY'))

        yc_beam = Yc;

        yTar = Yc;

        for ii=1:N

            skull_mask(:,:,ii) = rotate(Z,skull_mask(:,:,ii),xTar,yTar,theta,xc_beam-
xTar,yc_beam-yTar);

            skull_density(:,:,ii) =
rotate(Z,skull_density(:,:,ii),xTar,yTar,theta,xc_beam-xTar,yc_beam-yTar);

            skull_sound_speed(:,:,ii) =
rotate(Z,skull_sound_speed(:,:,ii),xTar,yTar,theta,xc_beam-xTar,yc_beam-yTar);

        end

        step = 3;

    end

    % Fill empty gaps due to rotation (interpolation)

    add = 0;

    T = 1;

    M = 2;

    [mask,rho,c] = fillGaps(T,M,add,step,skull_mask,skull_density,skull_sound_speed);

    cut3Axes(mask);

end
```

Script principal lanzar la simulación:

Transcranial.m

```
%% General cleaning

clear; close all; clc;



%% Simulation settings

DATA_CAST = 'off';      % 'off' to work with 'double' precision

use_skull = 1;

source_freq = 0.5e6;    % emitted frequency



%% Data adquisition (water + skull)

zone = 2;
```

```
dev1 = '-10.00';

dev2 = '0.00';

normal_inc = 1;

if(~normal_inc)

    d = 27;

    plane = 'XY';

    theta = 15;


load(['CT_SY_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2str(theta),'
_plane=',plane,'.mat']);

else

    d = 47;

    load(['CT_SY_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].mat']);

end

medium.alpha_power = 1.1;


%% Obtain grid

fd = 100;       % focal length [mm]

addNx = round(d/(1000*ds));

Nx = size(c,1)+addNx;

Ny = size(c,2);

Nz = size(c,3);

kgrid = makeGrid(Nx,ds, Ny,ds, Nz,ds);

PML_size = 10;  % PML size [grid points]


%% Obtain total medium properties

medium.density = min(rho(:))*ones(Nx,Ny,Nz);

medium.sound_speed = min(c(:))*ones(Nx,Ny,Nz);

medium.alpha_coeff = min(alpha(:))*ones(Nx,Ny,Nz);

if(use_skull)

    medium.density(addNx+1:end,:,:) = rho;

    medium.sound_speed(addNx+1:end,:,:) = c;

    medium.alpha_coeff(addNx+1:end,:,:) = alpha;

end


%% Testing if working between 5 or 6 points for every wavelength

c_min = min(medium.sound_speed(:)); % [m/s]
```

```
lambda_min = c_min/source_freq; % [m]

if(ds>lambda_min/5)

    error('Employed point spacing is not enough to obtain precise results (5 or 6 points per shorthest wavelength)');

end



%% Emitter design

focus = fd*1e-3/ds;

out_dim = 65e-3/ds;

height = (2*focus-sqrt(4*focus^2 - 2*out_dim^2))/2;

ss = makeSphericalSection(round(focus), round(height), [], false);

source.p_mask = zeros(Nx,Ny,Nz);

[a_mask,b_mask,c_mask] = size(ss);

source.p_mask(1:a_mask,round((Ny-b_mask+1)/2+(1:b_mask)),round((Nz-c_mask+1)/2+(1:c_mask))) = ss;

gap = 17e-3;    % Radius of the gap in the transducer (P/E transducer in the middle)[mm]

gap = gap/ds;

disk = makeDisc(Ny, Nz, round(Ny/2), round(Nz/2), gap);

solid_cylinder = zeros(size(source.p_mask));

for ii = 1:Nx

    solid_cylinder(ii,:,:) = disk;

end

source.p_mask(solid_cylinder==1) = 0;



%% Create the time array

c_max = max(medium.sound_speed(:));

cfl = 0.2;

t_end = 135e-6;     % [s]

kgrid.t_array = makeTime(kgrid, c_max, cfl, t_end);



%% Create the input signal using toneBurst

tone_burst_cycles = 10;

magnitude = 0.1e6;

input_signal = magnitude*toneBurst(1/kgrid.dt, source_freq, tone_burst_cycles,'Envelope','Rectangular');

source.p = input_signal;



%% Define a sensor mask in the whole medium
```

```
sensor.mask = ones(Nx,Ny,Nz);

sensor.record = {'p_max'};  % Peak value is stored



%% Start simulation

input_args = {'PMLInside', false, 'PMLSize', PML_size, 'DataCast', DATA_CAST,
'DisplayMask', source.p_mask};

sensor_data = kspaceFirstOrder3DC(kgrid, medium, source, sensor, input_args{:});



%% Plot results

% Reshape the returned max field to its original position (conversion from vector to
matrix)

sensor_data.p_max = reshape(sensor_data.p_max, [Nx,Ny,Nz]);

mask = medium.density;

mask(mask>1000) = 1;

mask(mask<=1000) = 0;

% Plot the two beam patterns using the pressure max

if(use_skull)

    aux_mask(:,:) = mask(:,:,round(Nz/2));

    aux_mask(aux_mask<0.5) = 0;

    aux_mask(aux_mask>=0.5) = 1;

end

xy_plane(:,:) = sensor_data.p_max(:,:,round(Nz/2));

if(use_skull)

    xy_plane = includeSkullEdge(aux_mask,xy_plane,use_skull);

    clear aux_mask;

    aux_mask(:,:) = mask(:,round(Ny/2),:);

    aux_mask(aux_mask<0.5) = 0;

    aux_mask(aux_mask>=0.5) = 1;

end

xz_plane(:,:) = sensor_data.p_max(:,round(Ny/2),:);

if(use_skull)

    xz_plane = includeSkullEdge(aux_mask,xz_plane,use_skull);

end

XY = figure; % XY plane

imagesc(kgrid.y_vec*1e3, (kgrid.x_vec - min(kgrid.x_vec(:)))*1e3, xy_plane/1e6);

xlabel('y [mm]');

ylabel('x [mm]');
```

```
title('Total Beam Pattern Using PEAK Of Recorded Pressure - XY Plane');

colormap(jet(256));

c = colorbar;

ylabel(c, 'Pressure [MPa]');

axis image;

XZ = figure; % XZ plane

imagesc(kgrid.z_vec*1e3, (kgrid.x_vec - min(kgrid.x_vec(:)))*1e3, xz_plane/1e6);

xlabel('z [mm]');

ylabel('x [mm]');

title('Total Beam Pattern Using PEAK Of Recorded Pressure - XZ Plane');

colormap(jet(256));

c = colorbar;

ylabel(c, 'Pressure [MPa]');

axis image;


%% Save results

dir = 'D:\Sergio\Universidad\UPV\Máster\Apuntes\Curso 1º\Semestre 2\TFM\3 -
Simulation\Images';

if(~normal_inc)

    if(use_skull)


saveas(XY,[dir,'\XYplane_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2
str(theta),'_plane=',plane,'.tiffn']);


saveas(XY,[dir,'\XYplane_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2
str(theta),'_plane=',plane,'.fig']);


saveas(XZ,[dir,'\XZplane_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2
str(theta),'_plane=',plane,'.tiffn']);


saveas(XZ,[dir,'\XZplane_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2
str(theta),'_plane=',plane,'.fig']);


save(['Simulation_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2str(the
ta),'_plane=',plane,'.mat']);

    else

        saveas(XY,[dir,'\XYplane_only_water.tiffn']);

        saveas(XY,[dir,'\XYplane_only_water.fig']);

        saveas(XZ,[dir,'\XZplane_only_water.tiffn']);

        saveas(XZ,[dir,'\XZplane_only_water.fig']);

        save('Simulation_only_water_.mat');

    end
```

```
elseif(normal_inc)

    if(use_skull)

saveas(XY,[dir,'\XYplane_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].tiffn']);

saveas(XY,[dir,'\XYplane_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].fig']);

saveas(XZ,[dir,'\XZplane_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].tiffn']);

saveas(XZ,[dir,'\XZplane_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].fig']);

        save(['Simulation_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].mat']);

    else

        saveas(XY,[dir,'\XYplane_only_water.tiffn']);

        saveas(XY,[dir,'\XYplane_only_water.fig']);

        saveas(XZ,[dir,'\XZplane_only_water.tiffn']);

        saveas(XZ,[dir,'\XZplane_only_water.fig']);

        save('Simulation_only_water_.mat');

    end

end
```

Función secundaria del script ***Transcranial.m*** (se emplean también algunas de las utilizadas en el ajuste de incidencia normal y otras que son las proporcionadas por la herramienta k-Wave):

inculdeSkullEdge.m

```
function Pout = includeSkullEdge(skull_mask,Pin,use_skull)

    [Nx,Ny] = size(skull_mask);

    if(use_skull)

        eroded_skull_mask = skull_mask;

        T = 1;

        M = 1;

        for ii=1+T:Nx-T

            for jj=1+T:Ny-T

                if(skull_mask(ii,jj) == 1)

                    aux = skull_mask(ii-T:ii+T,jj-T:jj+T);

                    Np = sum(sum(1-aux)); % number of zeros pixels (no skull) in this
region

                    if(Np >= M)

                        eroded_skull_mask(ii,jj) = 0;
```

```
                    end
                end
            end
        end
        skullEdge = skull_mask-eroded_skull_mask;
        Pout = (1-skullEdge).*Pin+skullEdge*max(Pin(:))/2;
    else
        Pout = Pin;
    end
end
```

Script auxiliar para calcular la atenuación sufrida por el foco y su localización cuando la propagación es a través de diferentes posiciones de cráneo:

Analyze_results.m

```
%% General cleaning
clear; close all; clc;


%% Configuration parameters
zone = 2;
dev1 = '-10.00';
dev2 = '0.00';
use_angle = 1;
if(use_angle)
    theta = 15;
end


%% Water maximum pressure at focus
load('Pmax_water');


%% Load simulation data
if(use_angle)

load(['Simulation_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2str(the
ta),'_plane=XY.mat']);
else
    load(['Simulation_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].mat']);
end
```

```
posX = 63; % [mm]

posX = round(posX/(ds*1000));

[pos_mm,pos_gp,Pmax_MPa] = focusPos(sensor_data,ds,posX);

pos_mm

Attenuation_dB = 20*log10(Pmax_MPa_water/Pmax_MPa)



%% Plot results

% Reshape the returned max field to its original position (conversion from vector to
matrix)

sensor_data.p_max = reshape(sensor_data.p_max, [Nx,Ny,Nz]);

mask = medium.density;

mask(medium.density>1000) = 1;

mask(medium.density<=1000) = 0;

% Plot the two beam patterns using the pressure max

aux_mask(:,:) = mask(:,:,pos_gp(3));

aux_mask(aux_mask<0.5) = 0;

aux_mask(aux_mask>=0.5) = 1;

xy_plane(:,:) = sensor_data.p_max(:,:,pos_gp(3));

xy_plane = includeSkullEdge(aux_mask,xy_plane,1);

clear aux_mask;

aux_mask(:,:) = mask(:,pos_gp(2),:);

aux_mask(aux_mask<0.5) = 0;

aux_mask(aux_mask>=0.5) = 1;

xz_plane(:,:) = sensor_data.p_max(:,pos_gp(2),:);

xz_plane = includeSkullEdge(aux_mask,xz_plane,1);

XY = figure; % XY plane

imagesc(kgrid.y_vec*1e3, (kgrid.x_vec - min(kgrid.x_vec(:)))*1e3, xy_plane/1e6);

xlabel('y [mm]');

ylabel('x [mm]');

title('Total Beam Pattern Using PEAK Of Recorded Pressure - XY Plane');

colormap(jet(256));

c = colorbar;

ylabel(c, 'Pressure [MPa]');

axis image;

XZ = figure; % XZ plane

imagesc(kgrid.z_vec*1e3, (kgrid.x_vec - min(kgrid.x_vec(:)))*1e3, xz_plane/1e6);

xlabel('z [mm]');
```

```
ylabel('x [mm]');

title('Total Beam Pattern Using PEAK Of Recorded Pressure - XZ Plane');

colormap(jet(256));

c = colorbar;

ylabel(c, 'Pressure [MPa]');

axis image;



%% Save results

dir = 'D:\Sergio\Universidad\UPV\Máster\Apuntes\Curso 1º\Semestre 2\TFM\3 -
Simulation\Images';

if(use_angle)


saveas(XY,[dir,'\Zcut_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2str
(theta),'.tif']);


saveas(XY,[dir,'\Zcut_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2str
(theta),'.fig']);


saveas(XZ,[dir,'\Ycut_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2str
(theta),'.tif']);


saveas(XZ,[dir,'\Ycut_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,']_theta=',num2str
(theta),'.fig']);

else

    saveas(XY,[dir,'\Zcut_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].tif']);

    saveas(XY,[dir,'\Zcut_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].fig']);

    saveas(XZ,[dir,'\Ycut_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].tif']);

    saveas(XZ,[dir,'\Ycut_zone=',num2str(zone),'_targetP=[',dev1,',',dev2,'].fig']);

end
```

Función secundaria del script ***Analyze_results.m*** (se emplean también la utilizada en la simulación y otras que son las proporcionadas por la herramienta k-Wave):

focusPos.m

```
function [pos_mm,pos_grid_points,Pmax] = focusPos(sensor_data,ds,posX)

    P = sensor_data.p_max;

    aux = P(posX:end,:,:);

    Pmax1 = max(aux(:));

    ind = find(P==Pmax1);

    ind = ind(round(end/2));

    Pmax = P(ind)*1e-6;
```

```
    [Nx,Ny,Nz] = size(P);

    Z1 = ceil(ind/(Nx*Ny));

    layerP = P(:,:,Z1);

    [X1,Y1] = find(layerP==Pmax1);

    pos_mm = [X1*ds*1000, (Y1-Ny/2-1)*ds*1000, (Z1-Nz/2-1)*ds*1000];

    pos_grid_points = [X1, Y1, Z1];

end
```