Lista 5

Campo Distante pela Superfície de Ffowcs Williams and Hawkings

Aeroacústica Computacional

Aluno:
José Pedro de Santana Neto - 201505394

Professor: **Andrey Ricardo da Silva, PhD.**

10 de dezembro de 2015

1 Problema

No contexto de simulações de fenômenos aeroacústicos usando Lattice Boltzmann, normalmente o campo de pressão ao longo do espaço é referente ao campo próximo e, por limitações de recursos computacionais, criar uma malha grande o suficiente para englobar o campo distante inviabiliza de forma integral a simulação. O presente problema a ser estudado e analisado diz respeito a um sólido quadrado que está sofrendo a influência de um escoamento de velocidade horizontal. O presente sistema é retratado de acordo com a figura 1.

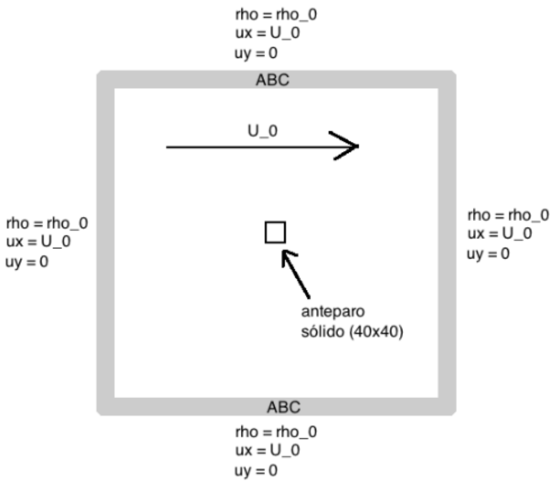

Figura 1: Esquemático do problema.

Diante do contexto exposto, propõe-se submeter esse sistema a um escomento de velocides Mach = 0.07 e 0.1. Além disso a superfície de FW-H será variada de 1 célula até 100 células de Lattice e análise da diretividade em campo distante há 15 metros da fonte.

2 Solução e Implementação

Para mitigar o problema da captação do campo distante dado as limitações computacionais, há a proposta de (LOCKARD, 2000) que utiliza da superfície de Ffowcs Williams and Hawkings para extrapolar resultados para o campo distante usando a composição do espectro de frequências do campo acústico próximo. Para a aplicação dessa técnica é preciso delimitar uma região para essa superfície e aplicar a equação da figura 2.

$$
\begin{aligned}
H(f)c_o^2\rho'(\mathbf{y},\omega) = & -\oint_{f=0} F_i(\xi,\omega)\frac{\partial G(\mathbf{y};\xi)}{\partial \xi_i}\mathrm{d}l \\
& -\oint_{f=0} \mathrm{i}\omega Q(\xi,\omega)G(\mathbf{y};\xi)\,\mathrm{d}l \\
& -\int_{f>0} T_{ij}(\xi,\omega)H(f)\frac{\partial^2 G(\mathbf{y};\xi)}{\partial \xi_i \partial \xi_j}\mathrm{d}\xi.
\end{aligned}
$$

Figura 2: Equação da superfície de FW-H.

Para as determinações das forças de dipolo e monopolo usa-se as equações da figura 2 somente, de tal forma que a contribuição das fontes quadripolos seja redirecionada para as fontes originadas a partir da inserção da superfície de FW-H.

$$F_i = (p\delta_{ij} + \rho(u_i - 2U_i)u_j + \rho_o U_i U_j)\frac{\partial f}{\partial y_j}$$

$$Q = (\rho u_i - \rho_o U_i)\frac{\partial f}{\partial y_i}.$$

Figura 3: Equações de dipolo e monopolo respectivamente.

Para a implementação da função de Green foi utilizada uma solução específica obtida a partir da transformação de Prandtl-Glauert. Os valores de x e y são respectivamente a posição na abscissa e ordenada do observador e os valores de ξ

e η são os valores de cada ponto da superfície na abscissa e ordenada respectivamente. A figura 2 mostra a solução da função de Green implementada e a figura 2 mostra o esquemático apresentado de forma visual.

$$G(x, y; \xi, \eta) = \frac{\mathrm{i}}{4\beta} \exp^{(Mk\bar{x}/\beta^2)} H_o^{(2)}\left(\frac{k}{\beta^2}\sqrt{\bar{x}^2 + \beta^2 \bar{y}^2}\right).$$

Figura 4: Equação da função de Green implementada.

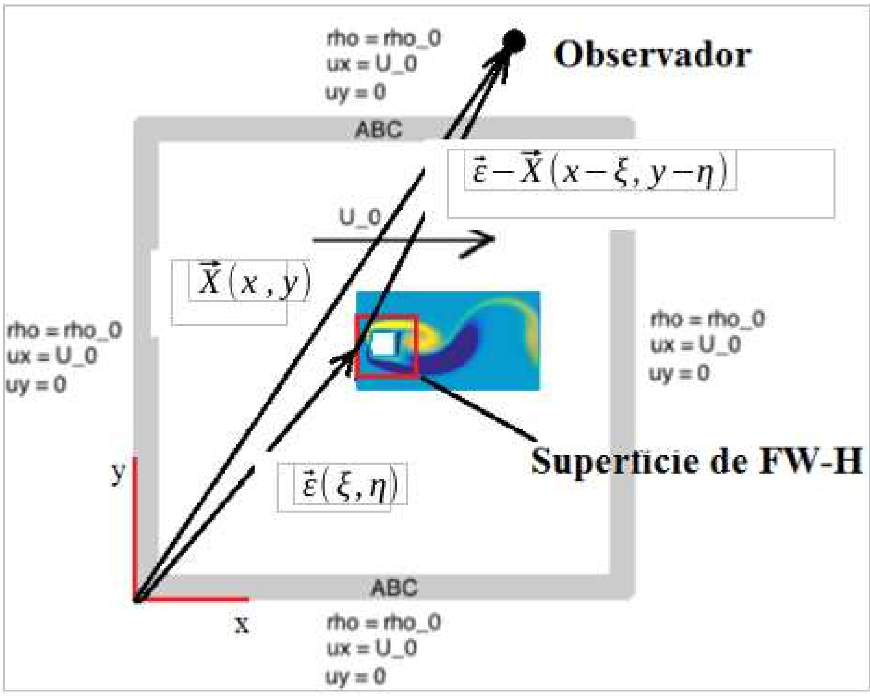

Figura 5: Esquemático visual da implementação.

No intuito de filtrar a frequência do dipolo e apresentá-lo em diretividade no campo distante, foi realizada uma captura de histórico de pressões e feito a transformada de fourier para a verificação da frequência mais forte, caracterizada no dipolo. Eis que na figura 2 mostra os resultados desse processo.

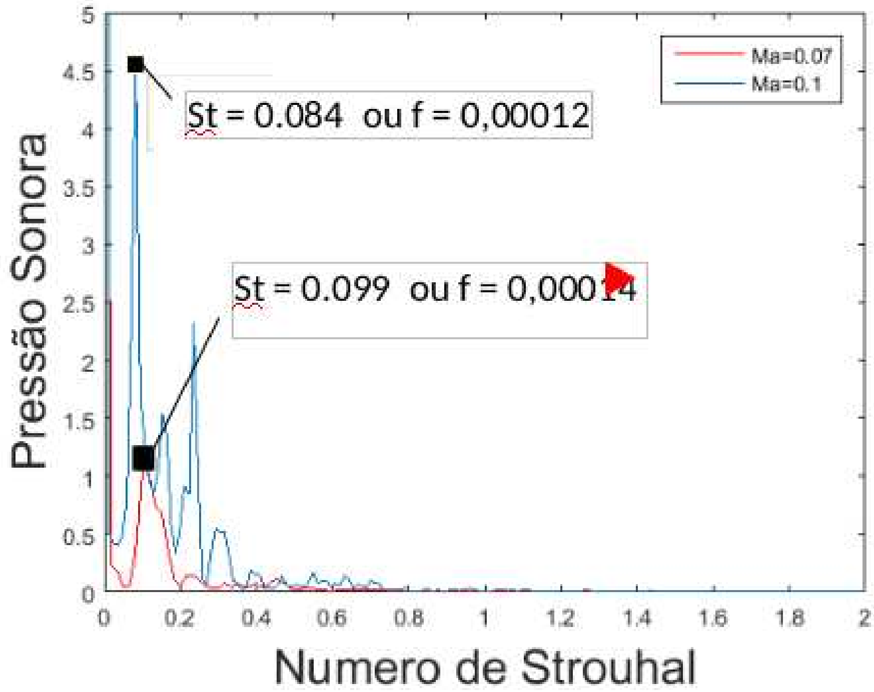

Figura 6: Frequências e números de strouhal de pico.

E para implementação foram desenvolvidos 2 *scripts*: simulação e pós-processamento. O *script* da simulação foi consolidado a partir do trabalho anterior que abordava o mesmo tema porém foram implementadas extração dos valores de densidade e velocidades para cada incremento de tempo. Tais valores foram armazenados no formato binário do MATLAB do tipo **.mat**. Para o *script* de pós-processamento foram considerados os seguintes procedimentos:

1. Abrir o arquivo de dados salvo a partir da simulação realizada;

2. Para cada incremento de tempo calcular os valores das fontes da superfície de FW-H: dipolo (F_i) e monopolo (Q);

3. Realizar a transformada de fourier (FFT) nos vetores calculados F_i e Q para obter a variação das componentes frequenciais ao longo da superfície;

4. Calcular a função de Green a partir de variáveis simbólicas a partir dos valores de campo distante;

5. Multiplicar as variáveis dipolo (F_i) e monopolo (Q) com a função de Green;

6. Derivar a multiplicação do dipolo (F_i);

7. Realizar esse processo para vários valores de campo de distante de tal forma a captar a diretividade.

Segue *script* de simulação:

```
%% This is a simple implementation of the LBGK model.
%% By Andrey R. da Silva, August 2010
%%
%% The code does not take into account eny specific boundaru
    condiition.

clear all, clc
close all
tic
% Block 1

%%% Lattice size

Nr = 502;                    % Number of lines    (cells in the y
    direction)
Mc = 502;                    % Number of columns (cells in the x
    direction)
% Block 2

%%% Physical parameters (macro)

c_p = 340;                    % Sound velocity on the fluid [m/s]
rho_p = 1.2;                  % physical density [kg/m^3]
rho_p = 1;                    % Fluid density  [kg/m^3]
```

```
Lx = .5;                    % Maximun dimenssion in the x direction [m]
Ly = 0.0833;                % Maximun dimenssion on th y direction [m]
Dx = Lx/Mc                  % Lattice space (pitch)
Dt = (1/sqrt(3))*Dx/c_p     % lattice time step
% Block 3
%%% Lattice parameters (micro - lattice unities)
omega = 1.93;                                    % Relaxation frequency
tau = 1/omega;                                   % Relaxation time
rho_l = 1;                                       % avereged fluid density (latice density
cs = 1/sqrt(3);                                  % lattice speed of sound
cs2 = cs^2;                                      % Squared speed of sound cl^2
visc = cs2*(1/omega-0.5);                        % lattice viscosity
visc_phy = visc*(Dx^2)/Dt;                       % physical kinematic viscosity
% Block 4
%%%%%%%%
%% Lattice properties for the D2Q9 model
%%%%%%%%
N_c=9 ;                                            % number of directions of the D2Q9 model
C_x=[1 0 -1  0 1 -1 -1  1 0];                      % velocity vectors in x
C_y=[0 1  0 -1 1  1 -1 -1 0];                      % velocity vectors in y
w0=16/36. ; w1=4/36. ; w2=1/36.;                   % lattice weights
W = [w1 w1 w1 w1 w2 w2 w2 w2 w0];
f1=3.;
f2=4.5;
f3=1.5;                                            % coef. of the f equil.
% Array of distribution and relaxation functions
f=zeros(Nr,Mc,N_c);
feq=zeros(Nr,Mc,N_c);
```

```
% Filling the initial distribution function (at t=0) with initial
    values
f(:,:,:)=rho_l/9;
ux = zeros(Nr, Mc);
uy = zeros(Nr, Mc);
% Calculando a condicao anecoica
% 4.0.1 - Adding conditions anechoic
distance = 30;
% condicao anecoica para cima
growth_delta = 0.5;
[sigma_mat9_cima Ft_cima] = build_anechoic_condition(Mc, ...
Nr, distance, growth_delta);
% condicao anecoica para baixo
growth_delta = -0.5;
[sigma_mat9_baixo Ft_baixo] = build_anechoic_condition(Mc, ...
Nr, distance, growth_delta);
% condicao anecoica para esquerda
growth_delta = -1;
[sigma_mat9_esquerda Ft_esquerda] = build_anechoic_condition(Mc, ...
Nr, distance, growth_delta);
% condicao anecoica para direita
 growth_delta = 1;
 [sigma_mat9_direito Ft_direito] = build_anechoic_condition(Mc, ...
 Nr, distance, growth_delta);
% Vendo pontos de barreira da xirizuda
xl = [2 2 501 501 2];
yl = [2 501 501 2 2];
[vec1,vec2,vec3,vec4,vec5,vec6,vec7,vec8] = crossing3(Nr,Mc,xl,yl);
%xl = [250 250];
%yl = [1 50];
xl = [231 231 271 271 231];
yl = [231 271 271 231 231];
[vec1_duto,vec2_duto,vec3_duto,vec4_duto,vec5_duto,vec6_duto,
    vec7_duto,vec8_duto] = ...
crossing3(Nr,Mc,xl,yl);
%Block 5
%%%%%%
%% Begin the iteractive process
%%%%%%
```

```
% Construindo chirp
total_time = 80000;
rho_save = [];
ux_save =  [];
uy_save =  [];
for ta = 1 : total_time

    % Block 5.1
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% propagation (streaming)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    f(:,:,1) = [f(:,1:2,1) f(:,2:Mc-1,1)];
    f(:,:,2) = [f(1:2,:,2);f(2:Nr-1,:,2)];
    f(:,:,3) = [f(:,2:Mc-1,3) f(:,Mc-1:Mc,3)];
    f(:,:,4) = [f(2:Nr-1,:,4);f(Nr-1:Nr,:,4)];
    f(:,:,5) = [f(:,1:2,5) f(:,2:Mc-1,5)];
    f(:,:,5) = [f(1:2,:,5);f(2:Nr-1,:,5)];
    f(:,:,6) = [f(:,2:Mc-1,6) f(:,Mc-1:Mc,6)];
    f(:,:,6) = [f(1:2,:,6);f(2:Nr-1,:,6)];
    f(:,:,7) = [f(:,2:Mc-1,7) f(:,Mc-1:Mc,7)];
    f(:,:,7) = [f(2:Nr-1,:,7);f(Nr-1:Nr,:,7)];
    f(:,:,8) = [f(:,1:2,8) f(:,2:Mc-1,8)];
    f(:,:,8) = [f(2:Nr-1,:,8);f(Nr-1:Nr,:,8)];
    G=f;
    f(vec1)=G(vec3);
    f(vec3)=G(vec1);
    f(vec2)=G(vec4);
    f(vec4)=G(vec2);
    f(vec5)=G(vec7);
    f(vec7)=G(vec5);
    f(vec6)=G(vec8);
    f(vec8)=G(vec6);
    G=f;
    f(vec1_duto)=G(vec3_duto);
    f(vec3_duto)=G(vec1_duto);
    f(vec2_duto)=G(vec4_duto);
    f(vec4_duto)=G(vec2_duto);
    f(vec5_duto)=G(vec7_duto);
```

```
    f(vec7_duto)=G(vec5_duto);
    f(vec6_duto)=G(vec8_duto);
    f(vec8_duto)=G(vec6_duto);
    % Block 5.2
    % recalculating rho and u
    rho=sum(f,3);
    %% Calculando uma fonte ABC dentro do dominio
    Ma = [0.03 0.07 0.1];
    density_source = rho_l;

    rt0= w0*rho;
    rt1= w1*rho;
    rt2= w2*rho;
    % Determining the velocities according to Eq.() (see slides)
    ux = (C_x(1).*f(:,:,1)+C_x(2).*f(:,:,2)+C_x(3).*f(:,:,3)+C_x(4).*
f(:,:,4)+C_x(5).*f(:,:,5)+C_x(6).*f(:,:,6)+C_x(7).*f(:,:,7)+C_x(8)
.*f(:,:,8))./rho ;
    uy = (C_y(1).*f(:,:,1)+C_y(2).*f(:,:,2)+C_y(3).*f(:,:,3)+C_y(4).*
f(:,:,4)+C_y(5).*f(:,:,5)+C_y(6).*f(:,:,6)+C_y(7).*f(:,:,7)+C_y(8)
.*f(:,:,8))./rho ;
    pressoes(ta) = (rho(450, 250) - 1)*cs2;

        numero_quadrados = 1;
        ponto_1_superficie = [(231 - numero_quadrados) (231 -
numero_quadrados)];
        y1 = ponto_1_superficie(1);
        x1 = ponto_1_superficie(2);
        ponto_2_superficie = [(271 + numero_quadrados) (271 +
numero_quadrados)];
        y2 = ponto_2_superficie(1);
        x2 = ponto_2_superficie(2);
        shift = 1;
        rho_linha = [rho(y1:y1, x1:(x2-shift)) rho(y1:(y2-shift), x2:
x2)' rho(y2:y2, (x2-shift):-1:x1) rho((y2-shift):-1:y1,x1:x1)'];
        rho_save(:, ta) = rho_linha;
        ux_linha = [ux(y1:y1, x1:(x2-shift)) ux(y1:(y2-shift), x2:x2)
' ux(y2:y2, (x2-shift):-1:x1) ux((y2-shift):-1:y1,x1:x1)'];
        ux_save(:, ta) = ux_linha;
        uy_linha = [uy(y1:y1, x1:(x2-shift)) ux(y1:(y2-shift), x2:x2)
```

```
' uy(y2:y2, (x2-shift):-1:x1) uy((y2-shift):-1:y1,x1:x1) '];
        uy_save(:, ta) = uy_linha;

    % Block 5.3
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Determining the relaxation functions for each direction
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    uxsq=ux.^2;
    uysq=uy.^2;
    usq=uxsq+uysq;

    feq(:,:,1)= rt1 .*(1 +f1*ux +f2.*uxsq -f3*usq);
    feq(:,:,2)= rt1 .*(1 +f1*uy +f2*uysq -f3*usq);
    feq(:,:,3)= rt1 .*(1 -f1*ux +f2*uxsq -f3*usq);
    feq(:,:,4)= rt1 .*(1 -f1*uy +f2*uysq -f3*usq);
    feq(:,:,5)= rt2 .*(1 +f1*(+ux+uy) +f2*(+ux+uy).^2 -f3.*usq);
    feq(:,:,6)= rt2 .*(1 +f1*(-ux+uy) +f2*(-ux+uy).^2 -f3.*usq);
    feq(:,:,7)= rt2 .*(1 +f1*(-ux-uy) +f2*(-ux-uy).^2 -f3.*usq);
    feq(:,:,8)= rt2 .*(1 +f1*(+ux-uy) +f2*(+ux-uy).^2 -f3.*usq);
    feq(:,:,9)= rt0 .*(1 - f3*usq);

    % Block 5.4
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Collision (relaxation) step
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % colidindo tudo
    f = (1-omega)*f + omega*feq ...
    - sigma_mat9_cima.*(feq- Ft_cima) ...
    - sigma_mat9_direito.*(feq- Ft_direito)...
    - sigma_mat9_esquerda.*(feq- Ft_esquerda)...
    - sigma_mat9_baixo.*(feq- Ft_baixo);

    % Ploting the results in real time

    %%
    if mod(ta, 100) == 0
        %vorticidade = curl(ux, uy);
      % velocity = sqrt(ux.^2 + uy.^2);
```

```
        %imagesc(flip(vorticidade))
        disp('Progresso: ');
        disp((ta/total_time*100));
    %   imagesc(flip(velocity), [.00001 Ma(3)*cs]);
        %axis equal
    %   pause(.000001);
    end

    % gravando dados
    % Inserindo dimensoes da superficie (y, x) quadrada de FW-H
end % End main time Evolution Loop
nome_arquivo = 'dados';
%nome_arquivo = ['dados/' num2str(ta) '.mat'];
save(nome_arquivo, 'rho_save', 'ux_save', 'uy_save','pressoes','
    total_time');
%Analise Freq.
p=pressoes(30000:end);
f=linspace(0,1,length(p));
fft=abs(fft(p));
%Getting slot
figure;plot(fft); axis([2 20 0 4])
% Getting frequency
figure;plot(f,fft);axis([0.01 0.2 0 4])
```

../lista_5.m

Segue *script* de pós-procecssamento:

```
%% Post-processing

clear variables; close all; clc

file = 'dados';

load([file '.mat']);

% Correction to include the smallest FW surface
%if exist('Nx','var') == false
%    Nx = 43;
```

```
%end
%if exist('Ny','var') == false
%     Ny = 43;
%end

%% DATA FROM PREVIOUS SIMULATIONS
%peak.velocity =      [0.03,   0.07,    0.10];
%peak.frequency =     [35,      79,      113];
%peak.index =         [10,      21,      30  ];

%I = find(M == peak.velocity);
%freq_peak = peak.frequency(I);
I_peak = 7;

%% Directivity analysis points
theta = (0:3:360)*pi/180;
radius = 15;
Dx = 0.004177e-3; % metros
M = 0.1;
cs = 1/sqrt(3);
cs2 = cs^2;
radius_lat = radius/Dx;
[probe_x,probe_y] = pol2cart(theta,radius_lat);

%% Green function
syms x y xi eta
x_bar = x - xi;
y_bar = y - eta;

beta = sqrt(1-M^2);
%freq_lat = freq_peak*Dx/zeta;
freq_lat = 0.00012;

k = 2*pi*freq_lat/cs;

z = k/beta^2*sqrt(x_bar^2 + beta^2*y_bar^2);
H = besselj(0,z) - 1i*bessely(0,z);
Green = 1i/(4*beta)*exp(1i*M*k*x_bar/beta^2)*H;

```

```
dGdxi = diff(Green, xi);
dGdeta = diff(Green, eta);
% Removes transient
%hist_p(:,1:50000) = [];
%hist_ux(:,1:50000) = [];
%hist_uy(:,1:50000) = [];
%hist_rho(:,1:50000) = [];

% Frequency vector
%Fs = 1/Dt;                    % Sampling frequency
%L = length(hist_p);
L = total_time;
Fs = 1;
Nx = 42;
Ny = 42;
frequency = Fs*(0:(L/2))/L; % frequency vector

%% SURFACE NORMALS
% Surface derivatives
hist_rho = rho_save(:,30000:end);
hist_ux = ux_save(:,30000:end);
hist_uy = uy_save(:,30000:end);
hist_p = (hist_rho - 1)*cs2;
%hist_p = hist_p - mean(hist_p);
dfdx = [repelem(0,Nx), repelem(1,Nx), repelem(0,Nx), repelem(-1,Nx)];
dfdx = repmat(dfdx',1,length(hist_p));
dfdy = [repelem(1,Ny), repelem(0,Ny), repelem(-1,Ny), repelem(0,Ny)];
dfdy = repmat(dfdy',1,length(hist_p));

%% MONOPOLE
% Monopole source for each cell at each time step
Q_point = hist_rho.*hist_ux.*dfdx + hist_rho.*hist_uy.*dfdy;

% FFT on monopole source time history
fft_Q = zeros(size(Q_point));
tamanho = size(hist_rho);
tamanho = length(theta);
for i = 1:tamanho
    fft_Q(i,:) = fft(Q_point(i,:));%/L; % row = angle, column =
```

```
    frequency
end
fft_Q(:,1) = []; % removes first line
Q = fft_Q(:, I_peak);

monopole = zeros(1,tamanho);
for j = 1:length(probe_x)
    G = subs(Green, [x,y,xi,eta], [probe_x(j),probe_y(j),0,0]);
    monopole(j) = trapz(1i*2*pi*freq_lat*Q*G);
end

figure
polar(theta,abs(monopole))

%% DIPOLE
% Dipole source for each cell at each time step
F1_point = (hist_p + hist_rho.*hist_ux.^2).*dfdx + hist_rho.*hist_ux.*hist_uy.*dfdy;
F2_point = hist_rho.*hist_ux.*hist_uy.*dfdx + (hist_p + hist_rho.*hist_uy.^2).*dfdy;

% FFT on dipole source time history
fft_F1 = zeros(size(F1_point));
fft_F2 = zeros(size(F2_point));
for i = 1:length(theta)
    fft_F1(i,:) = fft(F1_point(i,:));%/L; % row = angle, column = frequency
    fft_F2(i,:) = fft(F2_point(i,:));%/L; % row = angle, column = frequency
end
fft_F1(:,1) = []; % removes first line
fft_F2(:,1) = []; % removes first line

F1 = fft_F1(:,I_peak);
F2 = fft_F2(:,I_peak);

F_i_peak = sqrt(abs(F1).^2 + abs(F2).^2);

dipole = zeros(1,length(probe_x));
```

```
for j = 1:length(probe_x)
    G1 = subs(dGdxi, [x,y,xi,eta], [probe_x(j),probe_y(j),0,0]);
    G2 = subs(dGdeta, [x,y,xi,eta], [probe_x(j),probe_y(j),0,0]);
    dipole(j) = trapz(F1*G1 + F2*G2);
end

figure
polar(theta,abs(dipole))

%% TOTAL PRESSURE
pressure = - monopole - dipole;
figure
polar(theta,abs(pressure))

data = [theta; monopole; dipole];
save([file '_results.mat'],'data')
```

../post_processing.m

3 Resultados

Referências

LOCKARD, D. P. An efficient, two-dimensional implementation of the ffowcs williams and hawkings equation. *Journal of Sound and Vibration*, Elsevier, v. 229, n. 4, p. 897–911, 2000. Citado na página 2.