Lista 2

Dutos e Muffler

Aeroacústica Computacional

Aluno:
José Pedro de Santana Neto - 201505394

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

5 de novembro de 2015

1 Condições de Contorno - Duto

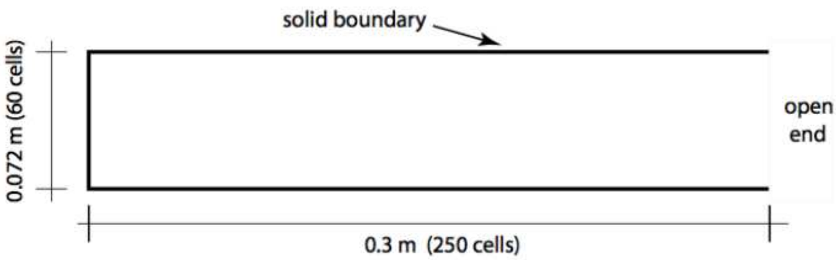

Figura 1: Ilustração do duto abordado com a parede a direita aberta.

1.1 Tubo Fechado-Fechado

Nesse primeiro caso de simulação haverá um pulso de densidades em forma de uma linha de células de lattice na extremidade esquerda do tubo. Esse pulso irá se propagar até encontrar a condição fechada (barreira) ao final do duto. Ao encontrar essa condição a frente de onda de pressão é refletida com fase positiva. E assim a onda fará 8 vezes o caminho de ida e volta desse percurso.

1.1.1 Códigos

```
clear all;
clc;
close all;

%% 1 - Set lattice sizes
number_lines_lattice = 60 + 2; % cells in the y direction
number_columns_lattice = 250 + 2; % cells in the x direction

%% 2 - Set physical parameters (macro)
physical_sound_velocity = 340; % [m/s]
```

```
physical_density = 1.2; % [kg/m^3]
physical_dimension_max_x = 0.3; % [m]
physical_dimension_max_y = 0.072; % [m]
% voxel is a term to express a volume decribed in a pixel: volume + pixel = voxel
dimension_x_voxel = physical_dimension_max_x/number_columns_lattice; % defining dimension x in voxel
lattice_time_step = (1/sqrt(3))*dimension_x_voxel/physical_sound_velocity;

%% 3 - Set lattice parameters (meso - lattice unities)
frequency_relaxation = 1.9; % to 1.5e-5 physcosity 1.9998; 860e-5 = 1.9
time_relaxation = 1/frequency_relaxation;
lattice_average_density = 1;
lattice_sound_speed = 1/sqrt(3);
lattice_sound_speed_pow_2 = lattice_sound_speed^2;
lattice_viscosity = lattice_sound_speed_pow_2*(1/frequency_relaxation-0.5);
physical_viscosity = lattice_viscosity*(dimension_x_voxel^2)/lattice_time_step; % [m^2/s]

% 4 - Build lattice struct with D2Q9 (lattice = Y x X)
lattice = build_lattice_D2Q9(number_lines_lattice, number_columns_lattice, lattice_average_density);

% 4.1 - Setting conditions of wall
wall_points{1} = [2 2 251 251 2]; % set points in horizontal
wall_points{2} = [2 61 61 2 2]; % set points in vertical
conditions_wall = crossing3( number_lines_lattice, ...
number_columns_lattice, wall_points);

% 5 - Set initial disturbance
initial_disturbance_density = 0.001;
points_lattice{2} = [3:60]; % set point along y
points_lattice{1} = linspace(3,3,length(points_lattice{2})); % set all the points in x = 3
lattice = set_initial_disturbances(lattice, initial_disturbance_density, points_lattice);
```

```

%% 6 - Begin the iteractive process
time_final = round(4*2*250*sqrt(3))
pressure_final_tube(1:time_final) = 0;
particule_velocity_final_tube(1:time_final) = 0;
for ta = 1 : time_final

    %% 6.1 - Propagation (streaming)
    lattice = stream_lattice(lattice);

    %% 6.1.2 - Setting conditions of walls
    lattice = set_conditions_wall(lattice, conditions_wall);

    %% 6.2 - Recalculating density and velocities
    density = sum(lattice{1},3);
    pressures_input(ta) = mean(density(26:31, 60) - 1)*lattice_sound_speed^2;
    pressures_output(ta) = mean(density(26:31, number_columns_lattice - 60) - 1)*lattice_sound_speed^2;
    lattice = calculate_velocities(lattice, density);

    %% 6.3 - Collide
    lattice = collide_lattice(lattice, frequency_relaxation);

    % % Ploting the results in real time
    pressure_final_tube(ta) = mean(density(3:60,250) - 1)*lattice_sound_speed^2;
    horizontal_velocity = lattice{2};
    particule_velocity_final_tube(ta) = mean(horizontal_velocity(3:60,250));
    %grid off
    imshow(mat2gray(density - 1));
    %imagesc(density-1)
    pause(.0000001)
    (ta/time_final)*100
end % End main time Evolution Loop


frequency_pressure_final_tube = fft(pressure_final_tube);
```

```
frequency_particule_velocity_final_tube = fft(
    particule_velocity_final_tube);
impedance = frequency_pressure_final_tube./
    frequency_particule_velocity_final_tube;
frequencies = linspace(0,1/lattice_time_step,length(
    frequency_pressure_final_tube))*2*pi*physical_dimension_max_y/
    physical_sound_velocity;
figure(2);
plot(frequencies, imag(impedance));
ylabel('Impedancia','FontSize',20);
xlabel('Numero de Helmholtz','FontSize',20);
title('Parte Imaginaria da Impedancia','FontSize',20);
axis([0 frequencies(end) -2000 2000]);
figure(3);
plot(frequencies, real(impedance), 'r');
ylabel('Impedancia','FontSize',20);
xlabel('Numero de Helmholtz','FontSize',20);
title('Parte Real da Impedancia','FontSize',20);
axis([0 frequencies(end) -2000 2000]);
```

code_matlab/code_refactored/closed–closed/main_lbgk.m

```
%% functionname: function description
function lattice = build_lattice_D2Q9(number_lines_lattice,
    number_columns_lattice, lattice_average_density)

% 1
number_directions_D2Q9 = 9;
lattice_distribution = zeros(number_lines_lattice,
    number_columns_lattice, number_directions_D2Q9);

% 2 - Filling the initial distribution function (at t=0) with initial
     values
lattice_distribution(:,:,:) = lattice_average_density/9;
lattice_velocity_x = zeros(number_lines_lattice,
    number_columns_lattice);
lattice_velocity_y = zeros(number_lines_lattice,
    number_columns_lattice);

```

```
% 3
lattice{1} = lattice_distribution;
lattice{2} = lattice_velocity_x;
lattice{3} = lattice_velocity_y;
```

code_matlab/code_refactored/closed–closed/build_lattice_D2Q9.m

```
function lattice = set_initial_disturbances(lattice,
    initial_disturbance_density, points_lattice)

  lattice_distribution = lattice{1};
  size_lattice = size(lattice_distribution(:, :, 1));
  number_lines_lattice = size_lattice(1);
  number_columns_lattice = size_lattice(2);

  horizontal_points = points_lattice{2};
  vertical_points = points_lattice{1};

  if length(horizontal_points) ~= length(vertical_points)
    disp('Quantity points not equal in X and Y axis.');
  else
    quantity_points = length(horizontal_points);
    for point = 1:quantity_points
      lattice_distribution(horizontal_points(point), vertical_points(
    point), 9) = initial_disturbance_density/9;
    end
  end

  lattice{1} = lattice_distribution;
```

code_matlab/code_refactored/closed–closed/set_initial_disturbances.m

```
function lattice = calculate_velocities(lattice, rho)

    % Determining lattice_time_stephe velocities according to Eq.() (
    see slides)
    f = lattice{1};
```

```
    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
    lattice{2} = (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  ;
    lattice{3} = (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  ;
```

code_matlab/code_refactored/closed–closed/calculate_velocities.m

```
function lattice = collide_lattice(lattice, frequency_relaxation);

    density = sum(lattice{1},3);
    w0=16/36. ; w1=4/36. ; w2=1/36.;                          % lattice weights
    rt0= w0*density;
    rt1= w1*density;
    rt2= w2*density;
    velocity_x_pow_2 = lattice{2}.^2;
    velocity_y_pow_2 = lattice{3}.^2;
    velocity_pow_2 = velocity_x_pow_2 + velocity_y_pow_2;
    f1=3.;
    f2=4.5;
    f3=1.5;                                                    % coef. of the f equil.
    feq(:,:,1)= rt1 .*(1 +f1*lattice{2} +f2.*velocity_x_pow_2 -f3*velocity_pow_2);
    feq(:,:,2)= rt1 .*(1 +f1*lattice{3} +f2*velocity_y_pow_2 -f3*velocity_pow_2);
    feq(:,:,3)= rt1 .*(1 -f1*lattice{2} +f2*velocity_x_pow_2 -f3*velocity_pow_2);
    feq(:,:,4)= rt1 .*(1 -f1*lattice{3} +f2*velocity_y_pow_2 -f3*velocity_pow_2);
    feq(:,:,5)= rt2 .*(1 +f1*(+lattice{2}+lattice{3}) +f2*(+lattice{2}+lattice{3}).^2 -f3.*velocity_pow_2);
```

```
    feq(:,:,6)= rt2 .*(1 +f1*(-lattice{2}+lattice{3}) +f2*(-lattice
   {2}+lattice{3}).^2 -f3.*velocity_pow_2);
    feq(:,:,7)= rt2 .*(1 +f1*(-lattice{2}-lattice{3}) +f2*(-lattice
   {2}-lattice{3}).^2 -f3.*velocity_pow_2);
    feq(:,:,8)= rt2 .*(1 +f1*(+lattice{2}-lattice{3}) +f2*(+lattice
   {2}-lattice{3}).^2 -f3.*velocity_pow_2);
    feq(:,:,9)= rt0 .*(1 - f3*velocity_pow_2);

    f=lattice{1};
    lattice{1} = frequency_relaxation*feq +(1-frequency_relaxation)*f
   ;
```

code_matlab/code_refactored/closed–closed/collide_lattice.m

```
%% set_conditions_wall: function to set the conditions wall in each
    cell in lattice
function lattice = set_conditions_wall(lattice, conditions_wall)
  vec1 = conditions_wall{1};
  vec2 = conditions_wall{2};
  vec3 = conditions_wall{3};
  vec4 = conditions_wall{4};
  vec5 = conditions_wall{5};
  vec6 = conditions_wall{6};
  vec7 = conditions_wall{7};
  vec8 = conditions_wall{8};
  f=lattice{1};
    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);
    lattice{1} = f;
```

code_matlab/code_refactored/closed–closed/set_conditions_wall.m

```
function lattice = stream_lattice(lattice)

    % 1
    lattice_distribution = lattice{1};
    size_lattice = size(lattice_distribution(:, :, 1));
    number_lines_lattice = size_lattice(1);
    number_columns_lattice = size_lattice(2);

    % 2
    lattice_distribution(:,:,1) = [lattice_distribution(:,1:2,1) ...
    lattice_distribution(:,2:number_columns_lattice-1,1)];
    lattice_distribution(:,:,2) = [lattice_distribution(1:2,:,2); ...
    lattice_distribution(2:number_lines_lattice-1,:,2)];
    lattice_distribution(:,:,3) = [lattice_distribution(:,2:number_columns_lattice-1,3) ...
    lattice_distribution(:,number_columns_lattice-1:number_columns_lattice,3)];
    lattice_distribution(:,:,4) = [lattice_distribution(2:number_lines_lattice-1,:,4); ...
    lattice_distribution(number_lines_lattice-1:number_lines_lattice,:,4)];
    lattice_distribution(:,:,5) = [lattice_distribution(:,1:2,5) ...
    lattice_distribution(:,2:number_columns_lattice-1,5)];
    lattice_distribution(:,:,5) = [lattice_distribution(1:2,:,5); ...
    lattice_distribution(2:number_lines_lattice-1,:,5)];
    lattice_distribution(:,:,6) = [lattice_distribution(:,2:number_columns_lattice-1,6) ...
    lattice_distribution(:,number_columns_lattice-1:number_columns_lattice,6)];
    lattice_distribution(:,:,6) = [lattice_distribution(1:2,:,6); ...
    lattice_distribution(2:number_lines_lattice-1,:,6)];
    lattice_distribution(:,:,7) = [lattice_distribution(:,2:number_columns_lattice-1,7) ...
    lattice_distribution(:,number_columns_lattice-1:number_columns_lattice,7)];
    lattice_distribution(:,:,7) = [lattice_distribution(2:number_lines_lattice-1,:,7); ...
    lattice_distribution(number_lines_lattice-1:number_lines_lattice,:,7)];
```

```
    lattice_distribution(:,:,8) = [lattice_distribution(:,1:2,8) ...
    lattice_distribution(:,2:number_columns_lattice-1,8)];
    lattice_distribution(:,:,8) = [lattice_distribution(2:number_lines_lattice-1,:,8); ...
    lattice_distribution(number_lines_lattice-1:number_lines_lattice,:,8)];

    %% 4
    lattice{1} = lattice_distribution;
```

code_matlab/code_refactored/closed–closed/stream_lattice.m

1.1.2 Imagens da Simulação

Figura 2: Imagem da frente de onda indo em direção a parede.

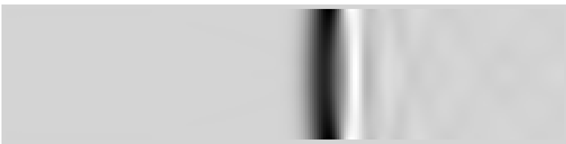

Figura 3: Imagem da frente de onda voltando depois da colisão com a parede do tubo fechado.

1.1.3 Gráficos de Impedância

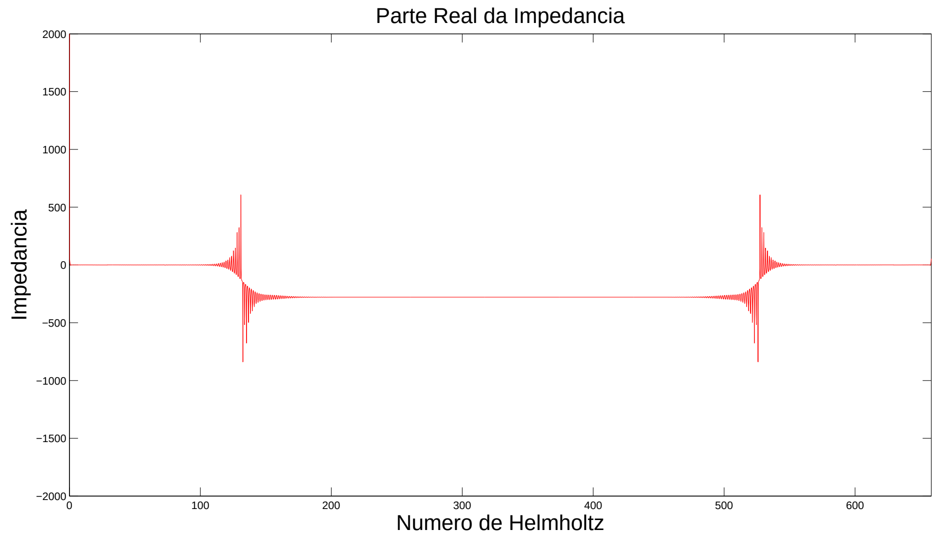

Figura 4: Gráfico da parte real da impedância, ou seja, parte ativa de energia que não permanece no sistema.

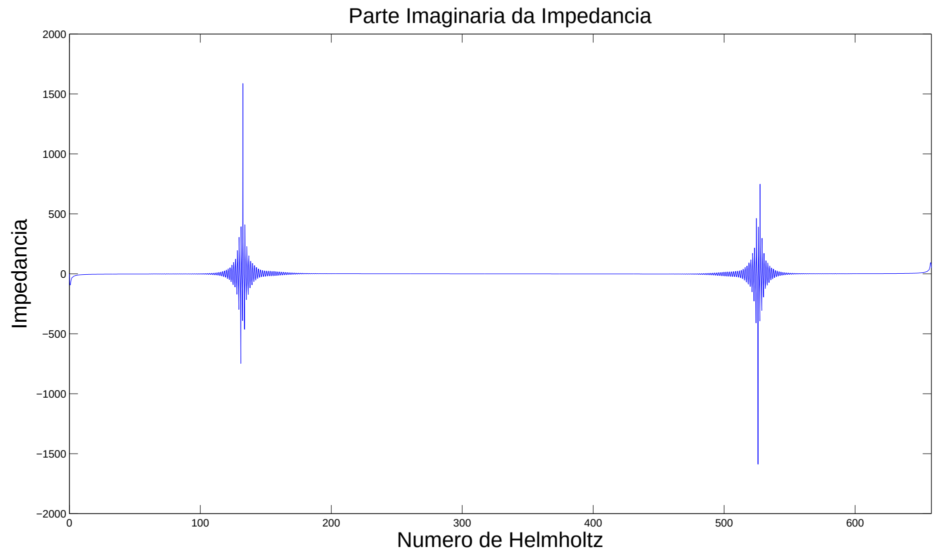

Figura 5: Gráfico da parte imaginária da impedância, ou seja, parte reativa de energia que permanece no sistema.

1.1.4 Análise

Em vista do que foi mostrado nas figuras 2 e 3, de fato a onda colidiu com a parede, voltou com fase positiva e com bastante energia conservada. Também há de se comentar que o gráfico de maior energia foi o da figura 5. Esse fato confirma a hipótese de que, num duto de paredes fechadas, há pouquíssima energia (bem próxima de zero) se dissipando ou indo para fora do sistema (representado no gráfico da figura 4) e bastante energia contida dentro do tubo representado em 5. Também é perceptível que a parte real da impedância (ativa) decresce se aproximando de zero e a parte imaginária (reativa) cresce.

1.2 Tubo Fechado-Aberto

Nesse segundo caso de simulação haverá um pulso de densidades em forma de uma linha de células de lattice na extremidade esquerda do tubo. Esse pulso irá se propagar até encontrar a condição aberta (sem nenhuma barreira ou condição de contorno) ao final do duto. Ao encontrar essa condição a frente de onda de pressão é refletida com fase negativa. E assim a onda fará 8 vezes o caminho de ida e volta desse percurso.

1.2.1 Códigos

```
clear all;
clc;
close all;

%% 1 - Set lattice sizes
number_lines_lattice = 60 + 2; % cells in the y direction
number_columns_lattice = 250 + 2; % cells in the x direction

%% 2 - Set physical parameters (macro)
physical_sound_velocity = 340; % [m/s]
physical_density = 1.2; % [kg/m^3]
physical_dimension_max_x = 0.3; % [m]
physical_dimension_max_y = 0.072; % [m]
% voxel is a term to express a volume decribed in a pixel: volume +
    pixel = voxel
dimension_x_voxel = physical_dimension_max_x/number_columns_lattice;
    % defining dimension x in voxel
lattice_time_step = (1/sqrt(3))*dimension_x_voxel/
    physical_sound_velocity;

%% 3 - Set lattice parameters (meso - lattice unities)
frequency_relaxation = 1.9; % to 1.5e-5 physcosity 1.9998; 860e-5 =
    1.9
time_relaxation = 1/frequency_relaxation;
lattice_average_density = 1;
lattice_sound_speed = 1/sqrt(3);
```

```
lattice_sound_speed_pow_2 = lattice_sound_speed^2;
lattice_viscosity = lattice_sound_speed_pow_2*(1/frequency_relaxation
    -0.5);
physical_viscosity = lattice_viscosity*(dimension_x_voxel^2)/
    lattice_time_step; % [m^2/s]

% 4 - Build lattice struct with D2Q9 (lattice = Y x X)
lattice = build_lattice_D2Q9(number_lines_lattice,
    number_columns_lattice, lattice_average_density);

% 4.1 - Setting conditions of wall
wall_points{1} = [251 2 2 251]; % set points in horizontal
wall_points{2} = [2 2 61 61]; % set points in vertical
conditions_wall = crossing3( number_lines_lattice, ...
number_columns_lattice, wall_points);

% 5 - Set initial disturbance
initial_disturbance_density = 0.001;
points_lattice{2} = [3:60]; % set point along y
points_lattice{1} = linspace(3,3,length(points_lattice{2})); % set
    all the points in x = 3
lattice = set_initial_disturbances(lattice,
    initial_disturbance_density, points_lattice);

%% 6 - Begin the iteractive process
time_final = round(4*2*250*sqrt(3))
pressure_final_tube(1:time_final) = 0;
particule_velocity_final_tube(1:time_final) = 0;
for ta = 1 : time_final

    %% 6.1 - Propagation (streaming)
    lattice = stream_lattice(lattice);

    %% 6.1.2 - Setting conditions of walls
    lattice = set_conditions_wall(lattice, conditions_wall);

    %% 6.2 - Recalculating density and velocities
    density = sum(lattice{1},3);
```

```
    pressures_input(ta) = mean(density(26:31, 60) - 1)*lattice_sound_speed^2;
    pressures_output(ta) = mean(density(26:31, number_columns_lattice - 60) - 1)*lattice_sound_speed^2;
    lattice = calculate_velocities(lattice, density);

    %% 6.3 - Collide
    lattice = collide_lattice(lattice, frequency_relaxation);

    % % Ploting the results in real time
    pressure_final_tube(ta) = mean(density(3:60,250) - 1)*lattice_sound_speed^2;
    horizontal_velocity = lattice{2};
    particule_velocity_final_tube(ta) = mean(horizontal_velocity(3:60,250));
    %grid off
    imshow(mat2gray(density - 1));
    %imagesc(density-1)
    pause(.0000001)
    (ta/time_final)*100
end % End main time Evolution Loop

frequency_pressure_final_tube = fft(pressure_final_tube);
frequency_particule_velocity_final_tube = fft(particule_velocity_final_tube);
impedance = frequency_pressure_final_tube./frequency_particule_velocity_final_tube;
frequencies = linspace(0,1/lattice_time_step,length(frequency_pressure_final_tube))*2*pi*physical_dimension_max_y/physical_sound_velocity;
figure(2);
plot(frequencies, imag(impedance));
ylabel('Impedancia','FontSize',20);
xlabel('Numero de Helmholtz','FontSize',20);
title('Parte Imaginaria da Impedancia','FontSize',20);
axis([0 frequencies(end) -25 25]);
figure(3);
plot(frequencies, real(impedance), 'r');
ylabel('Impedancia','FontSize',20);
```

```
xlabel('Numero de Helmholtz','FontSize',20);
title('Parte Real da Impedancia','FontSize',20);
axis([0 frequencies(end) -25 25]);
```

code_matlab/code_refactored/closed–opened/main_lbgk.m

1.2.2 Imagens da Simulação

Figura 6: Imagem da frente de onda indo em direção a parede.

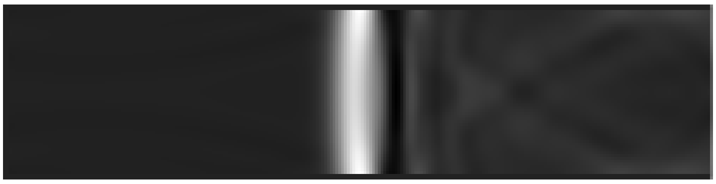

Figura 7: Imagem da frente de onda voltando depois da colisão com a parede do tubo aberto.

1.2.3 Gráficos de Impedância

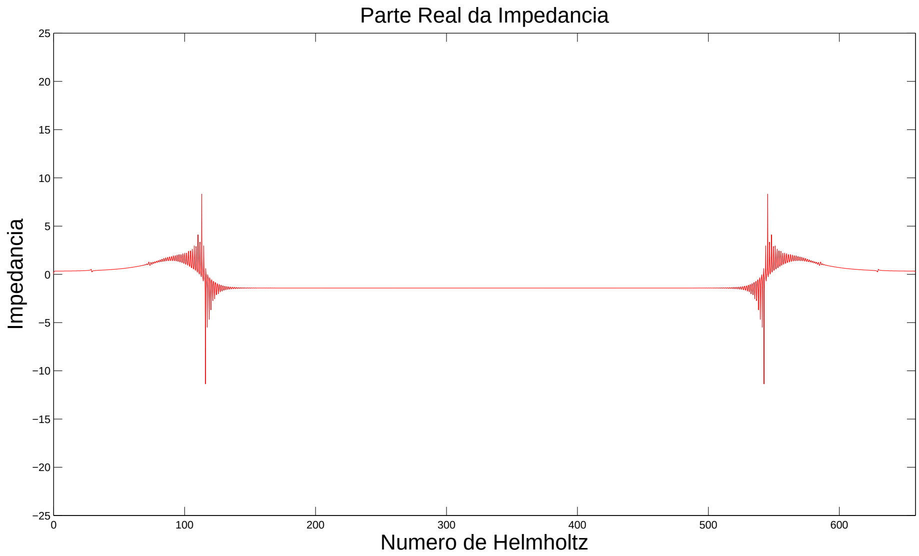

Figura 8: Gráfico da parte real da impedância, ou seja, parte ativa de energia que não permanece no sistema.

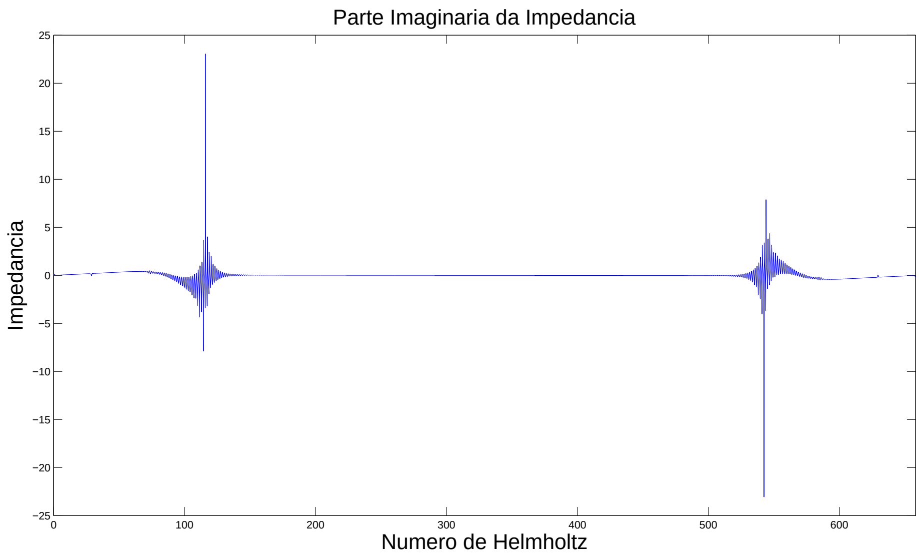

Figura 9: Gráfico da parte imaginária da impedância, ou seja, parte reativa de energia que permanece no sistema.

1.2.4 Análise

Em vista do que foi mostrado nas figuras 6 e 7, de fato a onda colidiu com a parede e voltou com fase negativa, através de uma reflexão que não é de natureza propriamente física. Também há de se comentar que o gráfico de maior energia foi o da figura 9. Esse fato confirma a hipótese de que, devido as reflexões não físicas com a camada livre do duto, há pouquíssima energia se dissipando ou indo para fora do sistema (representado no gráfico da figura 9) e bastante energia se contida dentro do tubo representado em 8, porém a quantidade de energia ativa é maior do que é mostrado no mesmo caso para um duto fechado nas duas extremidades abordado anteriormente (figura 4).

1.3 Tubo Fechado-Aberto com tratamento ABC

Nesse terceiro caso de simulação haverá um pulso de densidades em forma de uma linha de células de lattice na extremidade esquerda do tubo. Esse pulso irá se propagar até encontrar a condição aberta de tratamento anecóico (condição de contorno ABC) ao final do duto. Ao encontrar essa condição a frente de onda de pressão é absorvida.

1.3.1 Códigos

```
clear all;
clc;
close all;

%% 1 - Set lattice sizes
number_lines_lattice = 60 + 2; % cells in the y direction
number_columns_lattice = 250 + 2; % cells in the x direction

%% 2 - Set physical parameters (macro)
physical_sound_velocity = 340; % [m/s]
physical_density = 1.2; % [kg/m^3]
physical_dimension_max_x = 0.3; % [m]
physical_dimension_max_y = 0.072; % [m]
% voxel is a term to express a volume decribed in a pixel: volume +
    pixel = voxel
dimension_x_voxel = physical_dimension_max_x/number_columns_lattice;
    % defining dimension x in voxel
lattice_time_step = (1/sqrt(3))*dimension_x_voxel/
    physical_sound_velocity;

%% 3 - Set lattice parameters (meso - lattice unities)
frequency_relaxation = 1.9; % to 1.5e-5 physcosity 1.9998; 860e-5 =
    1.9
time_relaxation = 1/frequency_relaxation;
lattice_average_density = 1;
lattice_sound_speed = 1/sqrt(3);
lattice_sound_speed_pow_2 = lattice_sound_speed^2;
```

```
lattice_viscosity = lattice_sound_speed_pow_2*(1/frequency_relaxation -0.5);
physical_viscosity = lattice_viscosity*(dimension_x_voxel^2)/lattice_time_step; % [m^2/s]

% 4 - Build lattice struct with D2Q9 (lattice = Y x X)
lattice = build_lattice_D2Q9(number_lines_lattice, number_columns_lattice, lattice_average_density);

% 4.0.1 - Adding conditions anechoic
distance = 30;
growth_delta = 1;
[sigma_mat9 Ft] = build_anechoic_condition(number_lines_lattice, ...
number_columns_lattice, distance, growth_delta);

% 4.1 - Setting conditions of wall
wall_points{1} = [2 2 251 251 2]; % set points in horizontal
wall_points{2} = [2 61 61 2 2]; % set points in vertical
conditions_wall = crossing3( number_lines_lattice, ...
number_columns_lattice, wall_points);

% 5 - Set initial disturbance
initial_disturbance_density = 0.001;
points_lattice{2} = [3:60]; % set point along y
points_lattice{1} = linspace(3,3,length(points_lattice{2})); % set all the points in x = 3
lattice = set_initial_disturbances(lattice, initial_disturbance_density, points_lattice);

%% 6 - Begin the iteractive process
time_final = round(4*2*250*sqrt(3))
pressure_final_tube(1:time_final) = 0;
particule_velocity_final_tube(1:time_final) = 0;
for ta = 1 : time_final

    %% 6.1 - Propagation (streaming)
    lattice = stream_lattice(lattice);

    %% 6.1.2 - Setting conditions of walls
```

```
    lattice = set_conditions_wall(lattice, conditions_wall);

    %% 6.2 - Recalculating density and velocities
    density = sum(lattice{1},3);
    pressures_input(ta) = mean(density(26:31, 60) - 1)*
   lattice_sound_speed^2;
    pressures_output(ta) = mean(density(26:31, number_columns_lattice
    - 60) - 1)*lattice_sound_speed^2;
    lattice = calculate_velocities(lattice, density);

    %% 6.3 - Collide
    lattice = collide_lattice(lattice, frequency_relaxation, ...
    sigma_mat9, Ft);

    % % Ploting the results in real time
    pressure_final_tube(ta) = mean(density(3:60,250) - 1)*
   lattice_sound_speed^2;
    horizontal_velocity = lattice{2};
    particule_velocity_final_tube(ta) = mean(horizontal_velocity
   (3:60,250));
    %grid off
    imshow(mat2gray(density - 1));
    %imagesc(density-1)
    pause(.0000001)
    (ta/time_final)*100
end %  End main time Evolution Loop

frequency_pressure_final_tube = fft(pressure_final_tube);
frequency_particule_velocity_final_tube = fft(
    particule_velocity_final_tube);
impedance = frequency_pressure_final_tube./
    frequency_particule_velocity_final_tube;
frequencies = linspace(0,1/lattice_time_step,length(
    frequency_pressure_final_tube))*2*pi*physical_dimension_max_y/
    physical_sound_velocity;
figure(2);
plot(frequencies, imag(impedance));
ylabel('Impedancia','FontSize',20);
xlabel('Numero de Helmholtz','FontSize',20);
```

```
title('Parte Imaginaria da Impedancia','FontSize',20);
axis([0 frequencies(end) -20 20]);
figure(3);
plot(frequencies, real(impedance), 'r');
ylabel('Impedancia','FontSize',20);
xlabel('Numero de Helmholtz','FontSize',20);
title('Parte Real da Impedancia','FontSize',20);
axis([0 frequencies(end) -20 20]);
```

code_matlab/code_refactored/closed–anechoic/main_lbgk.m

```
function lattice = collide_lattice(lattice, frequency_relaxation, sigma_mat9, Ft);

    density = sum(lattice{1},3);
    w0=16/36. ; w1=4/36. ; w2=1/36.;                    % lattice weights
    rt0= w0*density;
    rt1= w1*density;
    rt2= w2*density;
    velocity_x_pow_2 = lattice{2}.^2;
    velocity_y_pow_2 = lattice{3}.^2;
    velocity_pow_2 = velocity_x_pow_2 + velocity_y_pow_2;
    f1 =3.;
    f2 =4.5;
    f3 =1.5;                                            % coef. of the f equil.
    feq(:,:,1)= rt1 .*(1 +f1*lattice{2} +f2.*velocity_x_pow_2 -f3*velocity_pow_2);
    feq(:,:,2)= rt1 .*(1 +f1*lattice{3} +f2*velocity_y_pow_2 -f3*velocity_pow_2);
    feq(:,:,3)= rt1 .*(1 -f1*lattice{2} +f2*velocity_x_pow_2 -f3*velocity_pow_2);
    feq(:,:,4)= rt1 .*(1 -f1*lattice{3} +f2*velocity_y_pow_2 -f3*velocity_pow_2);
    feq(:,:,5)= rt2 .*(1 +f1*(+lattice{2}+lattice{3}) +f2*(+lattice{2}+lattice{3}).^2 -f3.*velocity_pow_2);
    feq(:,:,6)= rt2 .*(1 +f1*(-lattice{2}+lattice{3}) +f2*(-lattice{2}+lattice{3}).^2 -f3.*velocity_pow_2);
```

```
    feq(:,:,7)= rt2 .*(1 +f1*(-lattice{2}-lattice{3}) +f2*(-lattice{2}-lattice{3}).^2 -f3.*velocity_pow_2);
    feq(:,:,8)= rt2 .*(1 +f1*(+lattice{2}-lattice{3}) +f2*(+lattice{2}-lattice{3}).^2 -f3.*velocity_pow_2);
    feq(:,:,9)= rt0 .*(1 - f3*velocity_pow_2);

    f=lattice{1};
    lattice{1} = frequency_relaxation*feq +(1-frequency_relaxation)*f - sigma_mat9.*(feq- Ft);
```

code_matlab/code_refactored/closed–anechoic/collide_lattice.m

1.3.2 Imagens da Simulação

Figura 10: Imagem da frente de onda indo em direção a parede.

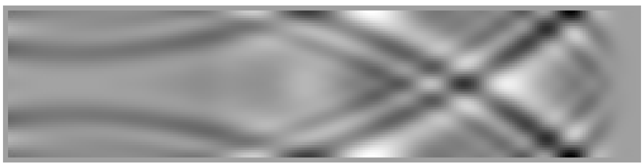

Figura 11: Imagem da frente de onda voltando depois da colisão com condição anecóica ao final.

1.3.3 Gráficos de Impedância

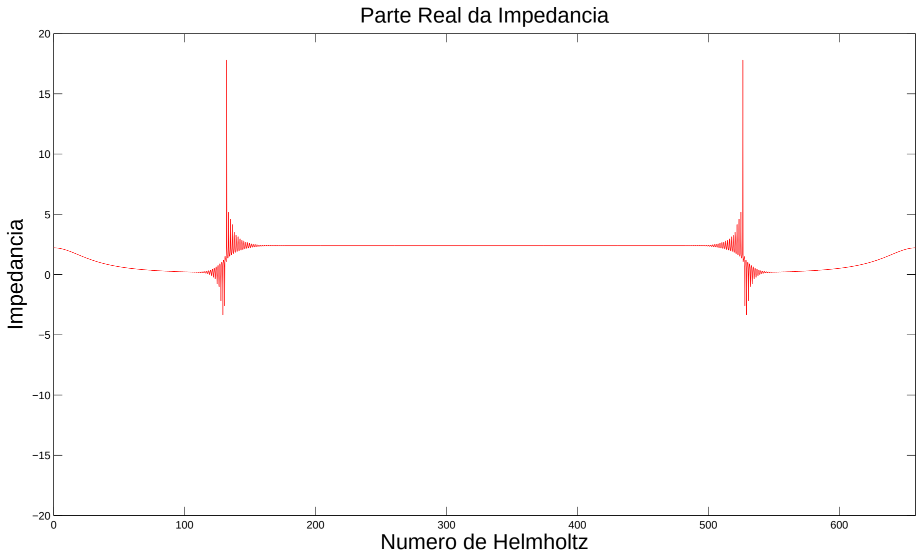

Figura 12: Gráfico da parte real da impedância, ou seja, parte ativa de energia que não permanece no sistema.

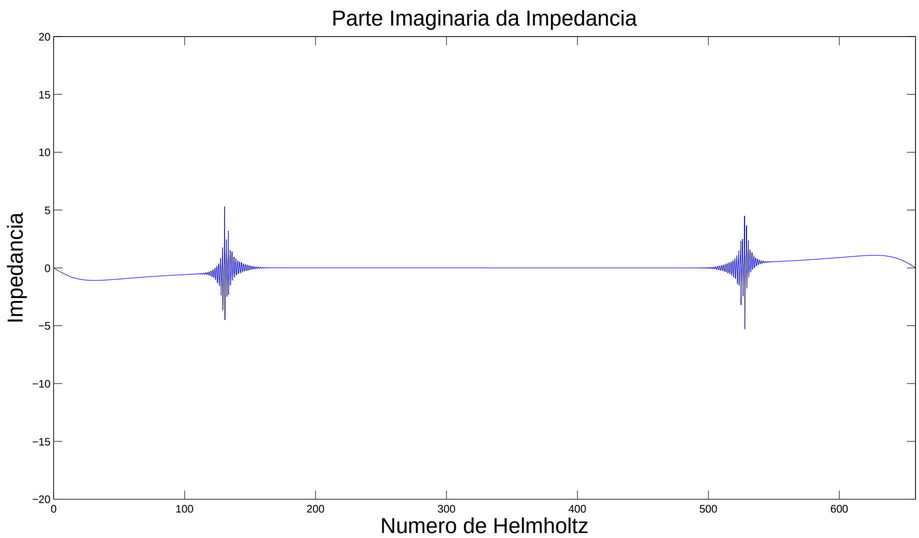

Figura 13: Gráfico da parte imaginária da impedância, ou seja, parte reativa de energia que permanece no sistema.

1.4 Análise

Em vista do que foi mostrado nas figuras 10 e 11, de fato a onda colidiu com a condição anecóica ABC e foi absorvida. Também há de se comentar que o gráfico de maior energia foi o da figura 12. Esse fato confirma a hipótese de que, devido a condição anecóica no final do duto, há bastante energia se dissipando ou indo para fora do sistema (representado no gráfico da figura 12) e pouca energia contida dentro do tubo representado em 11.

2 Filtro Acústico - Muffler

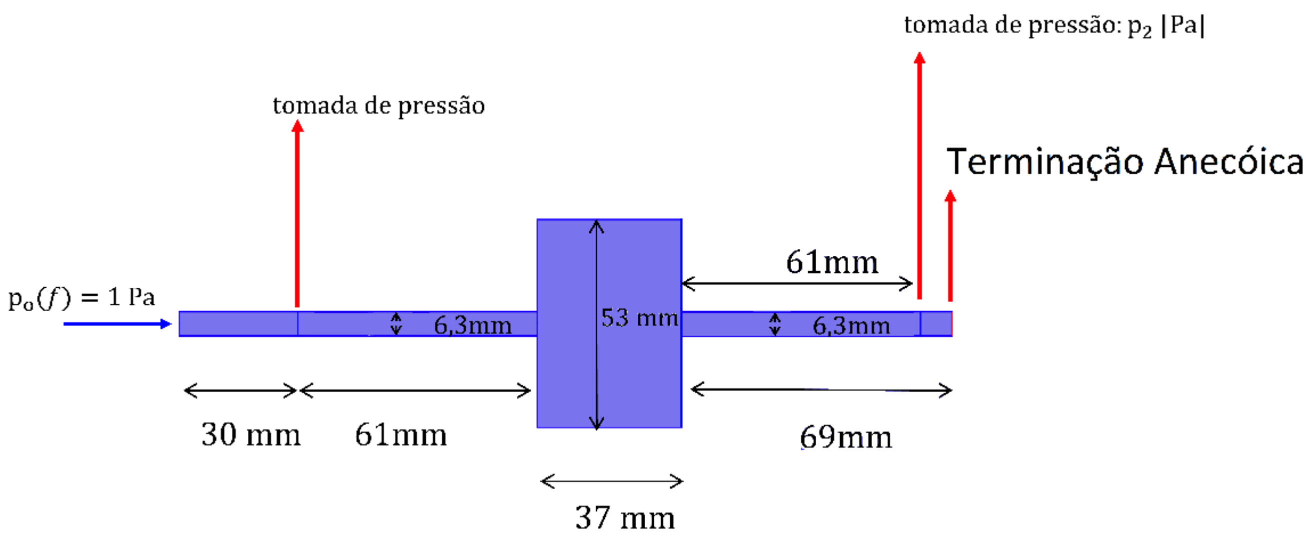

Figura 14: Ilustração esquemática do muffler.

Nessa simulação foi implementado um muffler de acordo com o que é mostrado em 14. Além da estrutura de paredes e condições anecóicas nas extremidades, foi colocado uma perturbação *chirp* na extremidade esquerda através de inserção de massa na implementação de contorno ABC. Essa perturbação *chirp* vai de 0 até 16kHz em unidades físicas ao longo da metade do tempo de simulação, deixando outra metade para propagação das ondas e excitação das frequências de ressonâncias do tubo.

2.1 Códigos

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 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

% Block 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Lattice size
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Lx = .198;
Ly = 0.053;
Dx = 0.0005

Nr = 53+4                    % Number of lines   (cells in the y direction)
Mc = 197                    % 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]
                     % Maximun dimenssion in the x direction [m]
                   % Maximun dimenssion on th y direction  [m]
                    % Lattice space (pitch)
Dt = (1/sqrt(3))*Dx/c_p        % lattice time step


% Block 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Lattice parameters (micro - lattice unities)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
omega = 1.9;                              % 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.

D_t=30;  % em numero de celulas
sigma_t=0.3;
delta_t=0:D_t;

% 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);

%rho_l = 0.01;   % initial disturbance



%funcoes target - saida
Ux_t=0;
Uy_t=0;
U_t=Ux_t^2+Uy_t^2;
rho_t=rho_l;

coef1=  1/(2*cs2^2); %para uso na relaxacao
coef2= -1/(2*cs2);


Ft=zeros(Nr,Mc,9);
Fe=zeros(Nr,Mc,9);

Ft(:,:,9)= w0*rho_t.*(1+coef2*U_t);

Ft(:,:,1)= w1*rho_t.*(1 +Ux_t/cs2 +coef1*(Ux_t.^2 )+coef2*U_t);
Ft(:,:,2)= w1*rho_t.*(1 +Uy_t/cs2 +coef1*(Uy_t.^2 )+coef2*U_t);
Ft(:,:,3)= w1*rho_t.*(1 -Ux_t/cs2 +coef1*(Ux_t.^2 )+coef2*U_t);
Ft(:,:,4)= w1*rho_t.*(1 -Uy_t/cs2 +coef1*(Uy_t.^2 )+coef2*U_t);

Ft(:,:,5)= w2*rho_t.*(1 +(+Ux_t+Uy_t)/cs2 +coef1*((+Ux_t+Uy_t).^2) + coef2*U_t);
Ft(:,:,6)= w2*rho_t.*(1 +(-Ux_t+Uy_t)/cs2 +coef1*((-Ux_t+Uy_t).^2) + coef2*U_t);
Ft(:,:,7)= w2*rho_t.*(1 +(-Ux_t-Uy_t)/cs2 +coef1*((-Ux_t-Uy_t).^2) + coef2*U_t);
Ft(:,:,8)= w2*rho_t.*(1 +(+Ux_t-Uy_t)/cs2 +coef1*((+Ux_t-Uy_t).^2) + coef2*U_t);
%
```

```
sigma=sigma_t*(delta_t/D_t).^2;
sigma_mat=[];
for i=1:Nr  % ver se tem jeito melhor de concatenar as matrizes
     sigma_mat=cat(1,sigma,sigma_mat);
end

sigmat=sigma_mat;

sigma_mat=[zeros(Nr,Mc-D_t-1)  sigmat];
sigma_mat2=[sigmat  zeros(Nr,Mc-D_t-1)];

% Condicao anecoica no final
inside= (round((53-6.3)/2)+3):(round((53+6.3)/2)+1);
outside=1:Nr;
for i=1:length(inside)
     outside=outside(outside~=inside(i));
end


sigma_mat(outside  ,:,:)=0;
sigma_mat2(outside  ,:,:)=0;

sigma_mat9=[];
for i=1:9
sigma_mat9=cat(3,sigma_mat,sigma_mat9);
end

sigma_mat9e=[];
for i=1:9
sigma_mat9e=cat(3,sigma_mat2,sigma_mat9e);
end
% sigma_mat9e=zeros(Nr,Mc,9);
%      sigma_mat9e=fliplr(sigma_mat9);


%Block 5
%% Begin the iteractive process
% REVER
```

```
xl=[    0             (30+61)       (30+61)  (30+61+37)  (30+61+37)  (30+61+37+69)];
yl=[   (53-6.3)/2   (53-6.3)/2     0            0          (53-6.3)/2  (53-6.3)/2   ]+2;
% xl=[];
% yl=[];
[vec1,vec2,vec3,vec4,vec5,vec6,vec7,vec8] = crossing3(Nr,Mc,xl,yl);
%
yl2=[(53+6.3)/2 (53+6.3)/2     53        53  (53+6.3)/2   (53+6.3)/2   ]+2;
[vec12,vec22,vec32,vec42,vec52,vec62,vec72,vec82] = crossing3(Nr,Mc,xl,yl2);

taf= 10*2*Mc*sqrt(3);
%% Build a chirp source sound
times_chirp = 1:taf/2;
frequency_lattice_max = 16e3*Dx/(340/cs);
chirp_source_sound = chirp(times_chirp, 0, times_chirp(end)*2, frequency_lattice_max);
source_sound = 1 + chirp_source_sound*0.01;
for ta = 1 : taf

    % 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(vec12)=G(vec32);
    f(vec32)=G(vec12);
    f(vec22)=G(vec42);
    f(vec42)=G(vec22);
    f(vec52)=G(vec72);
    f(vec72)=G(vec52);
    f(vec62)=G(vec82);
    f(vec82)=G(vec62);

    % Block 5.2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % recalculating rho and u
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    rho=sum(f,3);

%%      Perturbacao
%%      rho(150,150)=rho_l+A*sin(2*pi*freq*(ta-1));

    rt0= w0*rho;
    rt1= w1*rho;
    rt2= w2*rho;

    % Determining dthe 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 ;

% 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);

Ux_e=0;
Uy_e=0;
U_e=Ux_e^2+Uy_e^2;
lambda=25/ta;
freq=cs/lambda;
A=0.001;
%rho_e = rho_l+A*sin(2*pi*freq*(ta-1));
rho_e = 1;
if ta <= length(source_sound)
    rho_e = source_sound(ta);
end

Fe(:,:,9)= w0*rho_e.*(1+coef2*U_e);

Fe(:,:,1)= w1*rho_e.*(1 +Ux_e/cs2 +coef1*(Ux_e.^2 )+coef2*U_e);
Fe(:,:,2)= w1*rho_e.*(1 +Uy_e/cs2 +coef1*(Uy_e.^2 )+coef2*U_e);
Fe(:,:,3)= w1*rho_e.*(1 -Ux_e/cs2 +coef1*(Ux_e.^2 )+coef2*U_e);
Fe(:,:,4)= w1*rho_e.*(1 -Uy_e/cs2 +coef1*(Uy_e.^2 )+coef2*U_e);

```

```
    Fe(:,:,5)= w2*rho_e.*(1 +(+Ux_e+Uy_e)/cs2 +coef1*((+Ux_e+Uy_e).^2) +coef2*U_e);
    Fe(:,:,6)= w2*rho_e.*(1 +(-Ux_e+Uy_e)/cs2 +coef1*((-Ux_e+Uy_e).^2) +coef2*U_e);
    Fe(:,:,7)= w2*rho_e.*(1 +(-Ux_e-Uy_e)/cs2 +coef1*((-Ux_e-Uy_e).^2) +coef2*U_e);
    Fe(:,:,8)= w2*rho_e.*(1 +(+Ux_e-Uy_e)/cs2 +coef1*((+Ux_e-Uy_e).^2) +coef2*U_e);

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

%       f= (1-omega)*f + omega*feq;

    f=omega*feq +(1-omega)*f-sigma_mat9.*(feq-Ft) -sigma_mat9e.*(feq-Fe);

    % % Ploting the results in real time
    %surf(rho-1), view(2), shading flat, axis equal,axis off, caxis([-.00001 .00001]),
    %imagesc(rho - 1)
    %grid off
    imshow(mat2gray(rho - 1));
    pause(.0000001)
    % %

    pre(ta)=mean(rho(inside,32)-1)/3;
    prs(ta)=mean(rho(inside,Mc-D_t-3)-1)/3;

    (ta/taf)*100
end % End main time Evolution Loop
%%

Prs=abs(fft(prs));
Pre=abs(fft(pre));
freq1=linspace(0,1/Dt,length(Prs));
Z=20*log10(Prs./Pre);
```

```

open CurvaMufflerMareze.fig
hold on
plot(freq1,Z, 'r');
ylabel('Perda de transmissao [dB]');
xlabel('Frequencia [Hz]');
legend('Curva com Elementos Finitos', 'Curva da Simulacao');
%axis([0 10000 -20 20])
hold off
```

/home/gepeto/jose_pedro/muffler/main_lbgk.m

2.2 Imagens da Simulação

Figura 15: Muffler sendo excitado ainda no começo antes da metade do tempo de simulação.

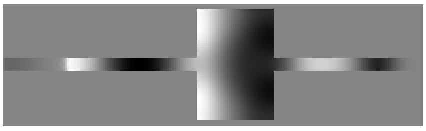

Figura 16: Muffler depois de excitado ressonando para a primeira frequência.

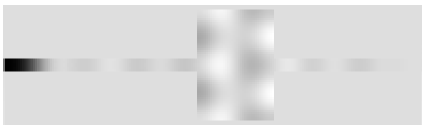

Figura 17: Muffler depois de excitado ressonando para a segunda frequência.

2.3 Gráfico da Atenuação

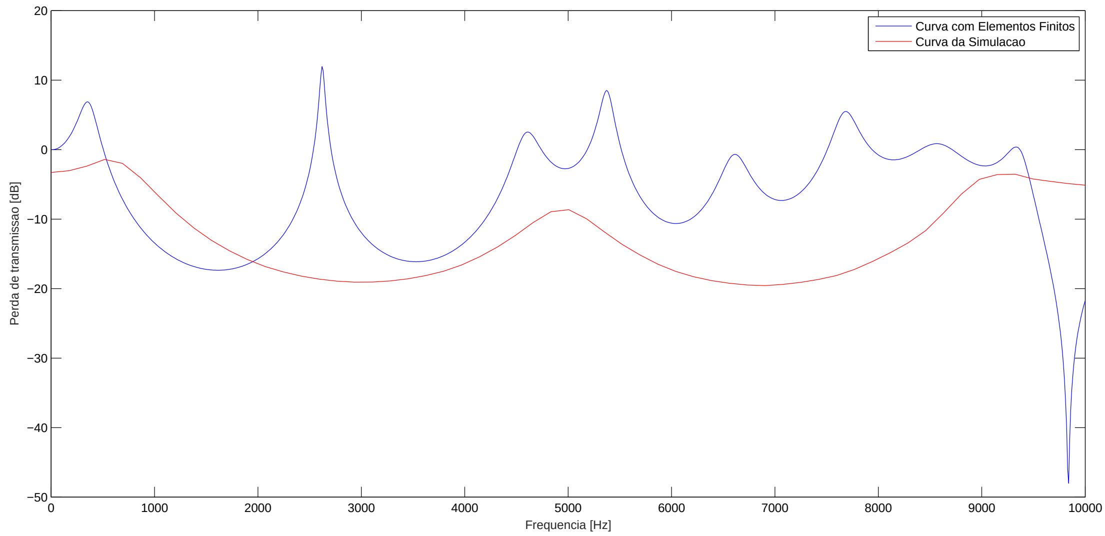

Figura 18: Comparação da curva obtida com elementos finitos e com a curva da simulação.

2.4 Análise

Em vista do que foi mostrado nas figuras 15, 16 e 17, a excitação do tipo *chirp* realmente estava mandando pulsos cada vez mais frequentes e rápidos para o muffler sem o estouro da *lattice* por emissão de massa em excesso e também é observado que depois da metade do tempo de simulação, depois que o *chirp* termina o processo de perturbação, dois modos de ressonância surgem com energia acumulada. Também há de se comentar que o gráfico 18 de perda de transmissão realmente mostrou dois picos para os dois modos excitados vistos nas imagens da simulação, porém o mesmo divergiu significamente da curva de referência feita em elementos finitos. Visivelmente percebe que outros modos não foram excitados e que os modos excitados estão deslocados, no entanto, a curva segue uma tendência grosseira e sem muito ruído.

Referências