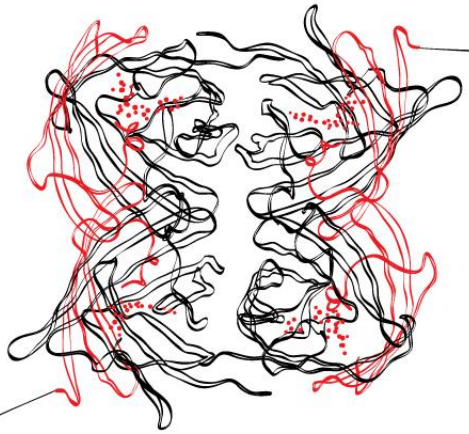
# IMPLEMENTATION OF ACOUSTIC BOUNDARY ELEMENT METHOD FOR THE PURPOSE OF BEAMFORMING

A numerical and experimental study



UNIVERSITY OF TWENTE.

Master Internship Report

Internship performed at the National Research Council of Canada from March-June 2012

# Implementation of acoustic Boundary Element Method for the purpose of beamforming: A numerical and experimental study


| | |
|---|---|
| Author: | T.J. van der Meer<br>s0148792 |
| UT supervisor: | Prof. dr. H.W.M. Hoeijmakers |
| NRC supervisor: | Dr. Jerry Syms |


Amsterdam, July 2013

# Preface

An important part of the curriculum of the Mechanical Engineering studies at the University of Twente consists of an internship outside the walls of the university. This serves as an opportunity for the student to put his scientific knowledge into practice in an industrial environment. As such the student gets a sneak preview of working life. Many students choose to carry out this 15 to 20 ECTS assignment abroad to add another experience to the list, namely that of living in another culture.

With these goals in mind I spent the period between February and June 2012 at the National Research Council of Canada (NRC), located in Ottawa. Specifically I was employed by the Aerodynamics Laboratory (AL) of the NRC's Institute for Aerospace Research (IAR). This report describes my assignment, consisting of the preparation of an acoustic boundary element method (BEM), ultimately to be used for optimization of the beamforming analysis method. The developed model was validated in a few basic tests using a simple experimental setup.

The project has been made possible using the help of many others, and some words of thanks are in order. First of all I want to thank my university supervisor, professor Hoeijmakers of the Engineering Fluid Dynamics group for arranging my placement with the NRC. The assignment itself came from dr. Jerry Syms who also was my supervisor during the internship itself. I am grateful for the start-up material he provided and the many valuable discussions during the project. Most of all however I want to thank Jerry for the warm welcome I received and the enjoyable time I have experienced during my placement at the NRC.

Thomas van der Meer

7 July 2013, Amsterdam

# Contents

# 1 Introduction

In the past 50 years air travel has experienced an incredible growth. According to the International Air Transport Association (IATA) the air transport industry has continuously grown with double-digit rates from 1945 until the first oil crisis in 1973 [1]. Since then the growth has slowed down, but is still considerable and consistent. IATA predicts a total of 3.6 billion air passengers in 2016 [2]. The increased mobility of ever more air transport is not without negative side effects. Especially in the developed world, the problem of noise pollution is becoming ever more serious. In recent years the airline industry has begun a general move to reduce this noise pollution. In fact the International Civil Aviation Organisation (ICAO) has set regulations to reduce noise generated by commercial aircraft by 32 decibels relative to the current standard [3]. To this end the NRC is researching noise production of airplane parts in its experimental facilities.

## 1.1 Modified wind tunnel

One of the research fields of the Aerodynamics Laboratory (AL) of the Institute for Aerospace Research (IAR) at the National Research Council of Canada (NRC) is the field of aero-acoustics. Specifically, the topic of noise reduction in airplanes is studied experimentally in a modified two-by-three meters wind tunnel in Ottawa. The walls of this wind tunnel are covered with acoustic foam (resembling cardboard egg cartons) which is itself covered in a fine mesh to create a smooth surface [3]. Figure 1.1 shows this wind tunnel at the Ottawa branch of the NRC.

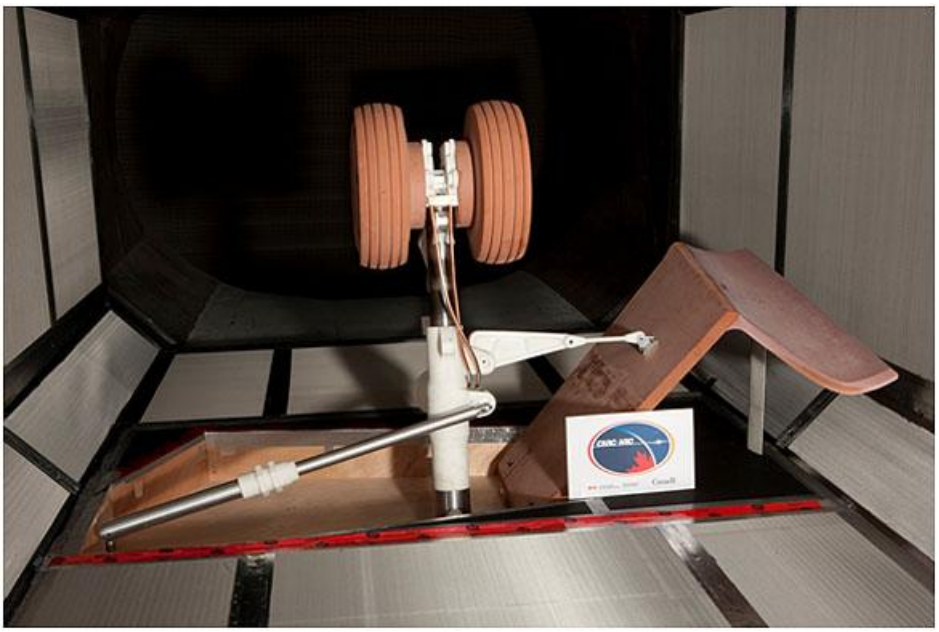


**Figure 1.1: Modified wind tunnel for acoustic testing, with a landing gear installed as test object**

The air velocities generated in this wind tunnel can reach almost 500 km/h meaning that any commercially interesting situation (most notably planes landing or taking off) can be replicated. Airplane parts of interest are then placed in this wind tunnel. The noise generated by an airplane during take-off or landing is mostly caused by the non-aerodynamic parts like flaps, slats and landing gear.

An array of 64 microphones is mounted inside one of the wind tunnel walls. These microphones record the sound generated by the air flowing over the object. From the test data of these microphones sound maps are produced that show the location and strength of sound sources in the wind tunnel. Ultimately, the sound source information can be used by airplane manufacturers to optimize their parts for noise reduction.

In order to produce the three-dimensional sound source maps from the data of the microphone array a technique called beamforming is used at the NRC. When locating the position of sound sources this technique assumes three-dimensional point sources in free space. In other words, the presence of the model and the sound reflection that comes with it are not accounted for in the beamforming analysis. Especially when the model is located between the source and the microphone this may yield to very poor results, i.e. sound sources in locations far behind the model which are clearly unphysical. Optimization of the beamforming process using numerical simulations is the subject of this research. More specifically this research is a proof-of-concept study towards the use of the Boundary Element Method (BEM) for acoustic simulation.

The beamforming technique and the problems it contains are explained in chapter 2. A possible solution to this problem is proposed in the form of numerical simulations that can correct for the presence of the object. Chapter 3 introduces the numerical BEM. The implementation of this method for the present purpose and some verification cases are also discussed. The validation experiments for this model are discussed in chapter 4. Chapter 5 contains the results of the model and the experiments as well as the comparison thereof. Chapter 6 ends with the conclusions and recommendations for future research.

# 2 Problem description

In chapter 1 the localisation of sound sources in 3D space by using a phased array of microphones (mounted in a 2D plane) was introduced. The method that produces these sound maps from the microphone data is called beamforming. This chapter explains how this technique works in section 2.1. The fundamental problem which is the presence of the model object is outlined in section 2.2. A solution to this problem in the form of corrections from numerical simulations is proposed in section 2.3.

## 2.1 Beamforming principles

To introduce the concept of beamforming consider a uniformly spaced array of $M$ microphones along a line as sketched in Figure 2.1. Sound waves can reach the array from the entire half-space in front of the array. For simplicity all sound waves are assumed to be plane waves for now, which implies the microphone array is sufficiently far away from the source. That is, the sound sources can be assumed to lie infinitely far away. The output from the array consists of a summation of the individual pressure signals $p_m$:

$$b(t) = \sum_{m=1}^{M} w_m p_m(t) \tag{2.1}$$

where $w_m$ is a weighting coefficient applied to the microphone $m$ signal (not further pursued here). As illustrated in Figure 2.1, waves perpendicular to the array are correlated in time and will result in an amplified output signal, whereas waves from other directions will arrive at each microphone at different times causing a lesser output amplitude. Therefore, if a wave actually is present perpendicular to the array this will be instantly noticeable from the amplified output signal. The array is said to be directionally sensitive in the direction normal to the array.

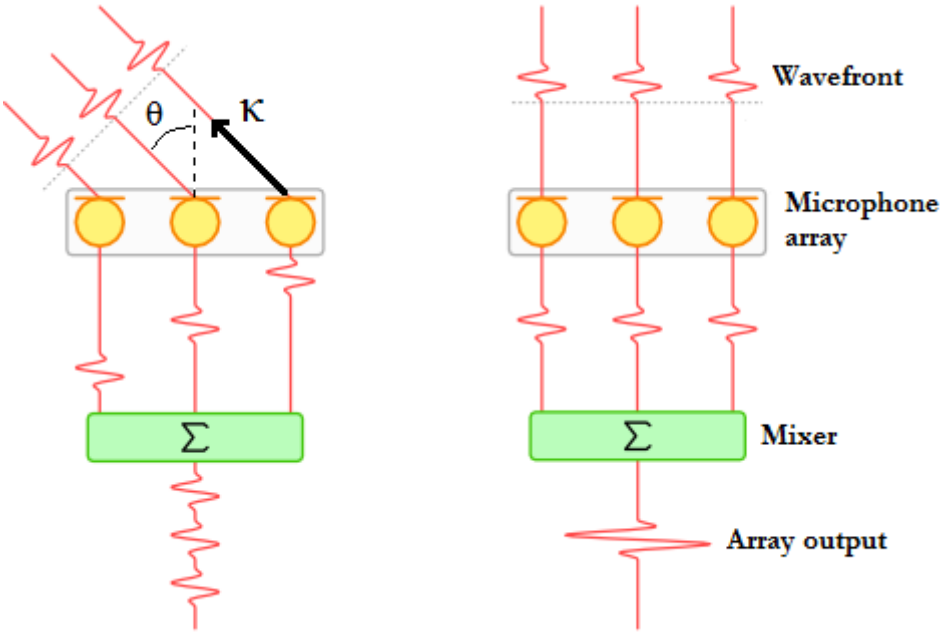



**Figure 2.1: Sketch of the output signal from the summation of microphone signals from plane waves coming from different directions (modified from Greensted [4])**

The main concept of beamforming is to achieve directional sensitivity of the array to any direction denoted by unit vector $\boldsymbol{\kappa} = (\sin\theta, \cos\theta)^T$, not just perpendicular to the array. This is done by delaying in time the individual microphone signals before they are summed:

$$b(\boldsymbol{\kappa}, t) = \sum_{m=1}^{M} w_m p_m\big(t - \delta t_m(\boldsymbol{\kappa})\big) \tag{2.2}$$

where $\delta t_m(\boldsymbol{\kappa})$ is the time delay associated with microphone $m$ and direction $\boldsymbol{\kappa}$ relative to a reference point, usually the middle microphone. From geometrical considerations:

$$\delta t_m(\boldsymbol{\kappa}) = \frac{\boldsymbol{\kappa} \cdot \boldsymbol{r}_m}{c} = \frac{r_m \sin\theta}{c} = \frac{\delta s_m}{c} \tag{2.3}$$

where $r_m$ is the distance of the microphone to the reference point and $c$ is the speed of sound. Equation (2.3) is illustrated in Figure 2.2. This time delay operation aligns the signal from direction $\boldsymbol{\kappa}$ so that now the waves coming from this direction will be amplified.

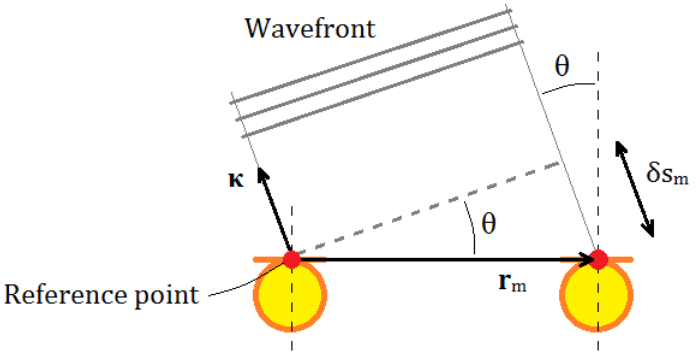



**Figure 2.2: Sketch of delay in arrival of a wavefront at microphone $m$ compared to the reference microphone**

When this time delay operation is performed over, say, the half-space given by $-\pi < \theta < \pi$ equation (2.2) yields a polar plot of which an example is shown in Figure 2.3. Clearly around $\theta = 40°$ an amplified signal is found, implying a sound source in that direction.

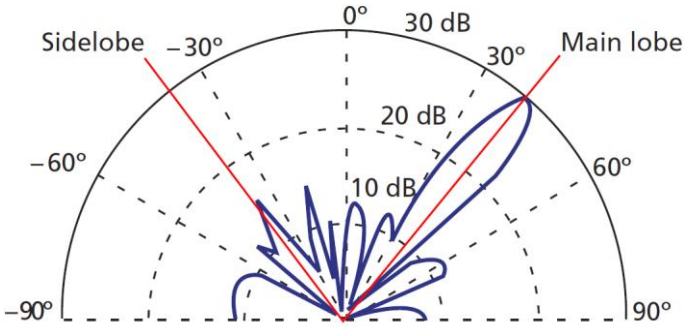



**Figure 2.3: Example polar plot showing a possible sound source at $\theta = 40°$ (from Christensen and Hald [5])**

Up to this point the procedure can only resolve plane waves coming from a certain direction, hence corresponding to point sources at infinite distance. To focus on point sources at a finite distance (such as on objects in a wind tunnel), the microphone delays $\delta t_m$ should be chosen so that they align in time the signals of a spherical wave radiating from the focus point, instead of a plane wave coming from infinity. This is achieved by writing the time delays as:

$$\delta t_m(\boldsymbol{r}) = \frac{|\boldsymbol{r}| - |\boldsymbol{r} - \boldsymbol{r}_m|}{c} \tag{2.4}$$

where $\boldsymbol{r}$ is the position of the focus point. This extension to point sources in 3D space is illustrated in Figure 2.4.

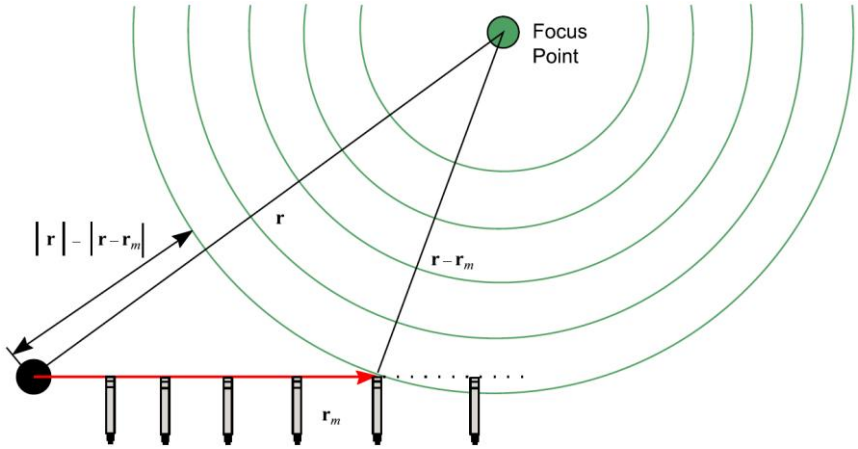



**Figure 2.4: Beamforming extended to point sources in 3D space (from Christensen and Hald [5])**

This section is a summary of parts of Christensen and Hald [5] and Greensted [4].

## 2.2 Signal obstruction by model object

In principle the beamforming method is a precise technique to localize sound sources in 3D free space. Now imagine the 3D space is not free but contains one or more objects. This is the case in the NRC wind tunnel, where the test section is naturally occupied by airplane parts. The assumption of free 3D monopoles as made in beamforming now does not hold anymore. Especially for sound sources on the backside of the object (as seen from the array point of view) this leads to unphysical source localization. The presence of the object obstructs the wave path to the microphones which delays the signal. This leads the beamforming algorithm to 'think' that the source is located further away than it actually is. Hence sources are predicted in the free space behind the object, while they logically come from the object itself. The problem is sketched conceptually in Figure 2.5.

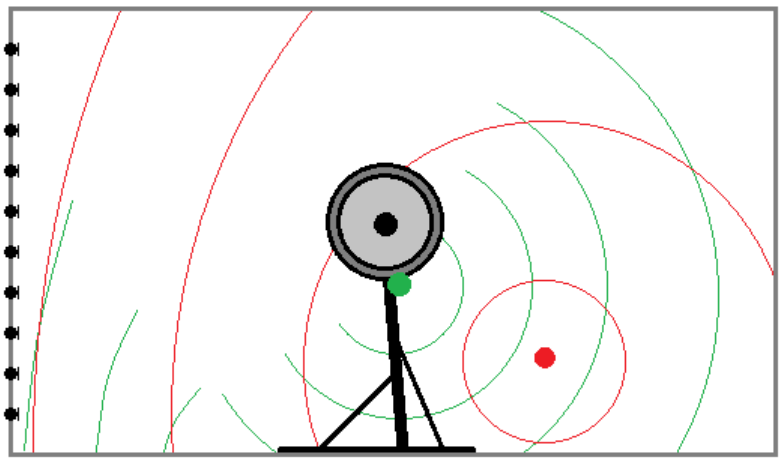


**Figure 2.5: Cross-sectional view of a wind tunnel with a landing gear as test object. The real sound source (green dot) is obstructed by the object, causing the microphone array to receive a signal that the beamforming algorithm reconstructs to be caused by a non-existing source (red dot)**

For accurate sound source predictions it is evident this inaccuracy must be solved. A possible solution is to use numerical simulations to calculate the extra delay produced by the object. This information can then be used to modify the artificial time-delay that the beamforming method uses in each direction. The idea is presented in the next section.

## 2.3 Optimization method

To solve the stated problem numerical simulations may be used. On each point of the surface of the object a sound source can be simulated, and the delay on the positions of the 64 microphones can be stored and used to correct the beamforming method. However, one can imagine that on a complex object like a landing gear this requires thousands -if not millions- of simulations to get the delay information for each relevant possible point source on the object geometry.

Luckily the physics underlying sound wave propagation allow for a very elegant simplification. Sound waves are essentially small disturbances in the pressure field. This means the non-linear governing equations simplify to linear equations following the use of perturbation theory. Rayleigh [6] reasoned that then the reciprocity principle by Stokes [7] and Helmholtz [8] is valid. The reciprocity principle means that for such linear situations the positions of the source and the receiver (in this case the microphone) can be swapped to produce the exact same result. Therefore, using this principle, simulating sound sources at the location of each of the microphones produces the delay information for all points on the mesh of the object! Hence 64 simulations suffice to generate all information needed to correct the beamforming method for any object geometry.

The numerical method used is the boundary element method (BEM). The powerful advantage of this method is that it solves for the pressure and velocity fields in the entire 3D space by solving just the equations in the boundary elements. For a mesh of $n \times n$ elements this cuts computational time by a factor $n$. As with the reciprocity principle this is made possible by the linearity of the perturbation problem.

# 3 Numerical study

The numerical method used is the boundary element method (BEM). The theory underlying this method is given in section 3.1, after which the numerical implementation is discussed in section 3.2. Both the theory and numerical implementations are adapted from Boundary Element Acoustics by Wu [9]. The results of several simulation cases are presented in chapter 5.

## 3.1 BEM theory

The starting point of the development of the BEM equations is a review of the fundamentals of linear acoustics which results in the Helmholtz equation (section 3.1.1). This equation is used to derive the boundary integral equations with the use of Green's second identity in section 3.1.2. Section 3.1.3 discusses the possible boundary conditions and their physical meaning.

### 3.1.1 Linear acoustics

When considering an inviscid, quiescent and homogenous medium the pressure $P$ and velocity $V$ are decomposed into the mean and fluctuating values:

$$P = p_0 + p' \qquad \boldsymbol{V} = \boldsymbol{v}' \tag{3.1}$$

Since the fluid is initially at rest, $v_0 = 0$, the fluctuating component $v$ is the only component (the particle velocity). Using equation (3.1) the continuity and momentum transport equation become:

$$\frac{\partial p'}{\partial t} + \rho_0 c^2 \boldsymbol{\nabla} \cdot \boldsymbol{v}' = 0 \tag{3.2}$$

$$\rho_0 \frac{\partial \boldsymbol{v}'}{\partial t} = -\boldsymbol{\nabla} p' \tag{3.3}$$

Equation (3.3) is recognized as the Euler equation. Substitution of the Euler equation in the time derivative of equation (3.2) yields

$$\rho_0 \boldsymbol{\nabla} \cdot \left( \frac{1}{\rho_0} \boldsymbol{\nabla} p' \right) - \frac{1}{c^2} \frac{\partial^2 p'}{\partial t^2} = 0 \tag{3.4}$$

which for a homogeneous medium, i.e. $\rho_0 =$ constant reduces to

$$\boldsymbol{\nabla}^2 p' - \frac{1}{c^2} \frac{\partial^2 p'}{\partial t^2} = 0 \tag{3.5}$$

Many sound waves of practical interest are time-harmonic, i.e. continuous waves at constant frequency $\omega$. In the presence of such waves the perturbation variables $p'$ and $\boldsymbol{v}'$ fluctuate sinusoidally as

$$p' = p e^{i\omega t} \qquad \boldsymbol{v}' = \boldsymbol{v} e^{i\omega t} \tag{3.6}$$

Here $p$ and $\boldsymbol{v}$ are the complex amplitude of the pressure and velocity fluctuation, respectively. Substitution of the pressure in equation (3.5) yields

$$\boldsymbol{\nabla}^2 p + k^2 p = 0 \tag{3.7}$$

where $k = \omega/c$ is the wave number. Equation (3.7) is known as the Helmholtz equation. It is in fact the sole governing equation for time-harmonic linear acoustics, as substitution of (3.6) in the Euler equation (3.3) results in an expression for the velocity amplitude:

$$\boldsymbol{v} = -\frac{1}{i\omega\rho_0}\boldsymbol{\nabla}p \tag{3.8}$$

Hence solving the Helmholtz equation is sufficient to find both the pressure and velocity fields.

### 3.1.2 BEM formulation

The basis is formed by the Helmholtz equation (3.7). The first step in the BEM formulation is to find the fundamental solution of the adjoint operator. Since the Laplacian in equation (3.7) is self-adjoint, the equation for the fundamental solution becomes

$$\boldsymbol{\nabla}^2\psi + k^2\psi = -\delta(Q - P) \tag{3.9}$$

where $\delta(Q - P)$ is the Dirac delta function at any point $Q$ in the domain with a singular point at $P$, and $\psi$ is the fundamental solution at point $Q$. When adopting a spherical coordinate system $(r, \theta, \phi)$ centred at $P$ equation (3.9) becomes

$$\frac{d^2\psi}{dr^2} + \frac{2}{r}\frac{d\psi}{dr} + k^2\psi = 0 \qquad \text{everywhere except } P \tag{3.10}$$

The general solution to this equation is

$$\psi = A\frac{e^{-ikr}}{r} + B\frac{e^{ikr}}{r} \tag{3.11}$$

where $A$ and $B$ are unknown coefficients. Physically, the first term in the solution represents an outgoing wave and the second term represents an incoming wave. Therefore $B = 0$ since $P$ is a point source instead of a sink. To find $A$ consider a small spherical volume $\Omega_\varepsilon$ with radius $\varepsilon$ enclosing the point $P$, and integrate equation (3.9) over this volume to find

$$\lim_{\varepsilon\to 0}\int_{\Omega_\varepsilon}(\boldsymbol{\nabla}^2\psi + k^2\psi)dV = -1 \tag{3.12}$$

In the limit of $\varepsilon \to 0$ $dV$ goes to zero much faster than $\psi$ goes to infinity, this is why the second term is dropped. Applying the divergence theorem to the remainder converts it to a surface integral where $\partial\Omega_\varepsilon$ is the boundary surface with $n$ as its outward unit normal:

$$\lim_{\varepsilon\to 0}\int_{\partial\Omega_\varepsilon}\frac{\partial\psi}{\partial n}dS = -1 \tag{3.13}$$

Carrying out this integration using the notion that $n = r$ for $\varepsilon \to 0$ one finds $A = \pi/4$ and the fundamental solution in a 3D free space is

$$\psi = \frac{e^{-ikr}}{4\pi r} \tag{3.14}$$

Let $P$ be the singular point associated with the fundamental solution located in the acoustic domain $V$. A tiny spherical volume $V_\varepsilon$ enclosing $P$ (radius $\varepsilon$) is excluded from this domain since it is singular. The domain is sketched in Figure 3.1.

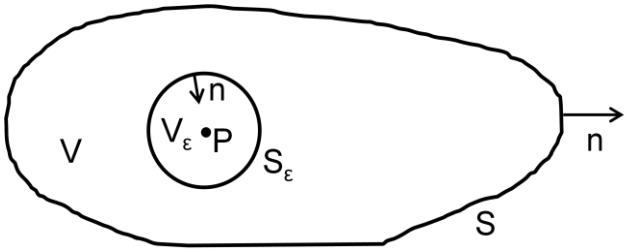



**Figure 3.1: Acoustic domain $V$ with exclusion of $V_\varepsilon$ enclosing $P$**

The next step is to apply Green's second identity to the two functions $p$ and $\psi$ to obtain

$$\int_{V-V_\varepsilon} (\psi \boldsymbol{\nabla}^2 p - p \boldsymbol{\nabla}^2 \psi) dV = \int_{S+S_\varepsilon} \left(\psi \frac{\partial p}{\partial n} - p \frac{\partial \psi}{\partial n}\right) dS \tag{3.15}$$

where $n$ is the normal vector pointing away from the domain and $S_\varepsilon$ is the boundary of $V_\varepsilon$. Equation (3.7) gives $\boldsymbol{\nabla}^2 p = -k^2 p$, and since $P$ is not in the domain equation (3.9) yields $\boldsymbol{\nabla}^2 \psi = -k^2 p$. Hence the left-hand side of the above equation vanishes since the integrand is zero. Splitting the integral on the right-hand according to the boundaries $S$ and $S_\varepsilon$ and taking the limit of $\varepsilon \to 0$ then gives

$$-\lim_{\varepsilon \to 0} \int_{S_\varepsilon} \psi \frac{\partial p}{\partial n} dS + \lim_{\varepsilon \to 0} \int_{S_\varepsilon} p \frac{\partial \psi}{\partial n} dS = \int_S \left(\psi \frac{\partial p}{\partial n} - p \frac{\partial \psi}{\partial n}\right) dS \tag{3.16}$$

The first term vanishes since $\psi$ is in the order $\mathcal{O}(1/r)$ and $dS$ is in the order $\mathcal{O}(r^2)$ in the limit of $\varepsilon \to 0$. That is, $dS$ goes to zero much faster than $\psi$ goes to infinity. Similarly the second term becomes

$$\lim_{\varepsilon \to 0} \int_{S_\varepsilon} p \frac{\partial \psi}{\partial n} dS = p(P) \int_{S_\varepsilon} \lim_{\varepsilon \to 0} \frac{\partial \psi}{\partial n} dS = p(P) \int_{S_\varepsilon} \lim_{r \to 0} \left(-\frac{\partial \psi}{\partial r}\right) dS \tag{3.17}$$

where the last step is obtained by realising that, in the chosen limit, $n$ is in the negative $r$ direction. The integrand can be calculated using the equation (3.14) and the quotient rule:

$$\lim_{r \to 0} \left(-\frac{\partial \psi}{\partial r}\right) = \lim_{r \to 0} \left(\frac{ike^{-ikr} 4\pi r + e^{-ikr} 4\pi}{(4\pi r^2)}\right) = \frac{1}{4\pi r^2} \tag{3.18}$$

Since the surface of a sphere is $4\pi r^2$ this means equation (3.17) becomes

$$p(P) \int_{S_\varepsilon} \lim_{r \to 0} \left(-\frac{\partial \psi}{\partial r}\right) dS = p(P) \tag{3.19}$$

Using these results together with equation (3.8) to rewrite the pressure derivative, equation (3.16) becomes

$$p(P) = -\int_S \left(i\rho_0 \omega v_n \psi + p \frac{\partial \psi}{\partial n}\right) dS \qquad \text{for } P \text{ in the domain.} \tag{3.20}$$

where $v_n$ is the normal velocity on the boundary. This equation states that the sound pressure $p$ at any point $P$ inside the domain can be found by integrating the equation on the boundary of the domain. However, for a well-posed problem $p$ or $v_n$ is known from the boundary condition, not both. Only half the necessary data is known. To obtain the other half the point $P$ is located on the boundary itself and the above procedure is repeated. After some algebra which is out of scope of this report (but can be found in chapter 2 in Wu [9]) one obtains an extension of equation (3.20):

$$C^0(P)p(P) = -\int_S \left(i\rho_0\omega v_n\psi + p\frac{\partial\psi}{\partial n}\right)dS \qquad \begin{aligned} C^0(P) &= 1 \text{ for } P \text{ in the domain} \\ C^0(P) &= \frac{1}{2} \text{ for } P \text{ on smooth boundary} \\ C^0(P) &= -\int_S \frac{\partial\psi_L}{\partial n}dS \text{ for } P \text{ on an edge} \end{aligned} \tag{3.21}$$

In this equation $\psi_L = 1/4\pi r$ is the fundamental solution to the Laplace equation. The above equation is called the Helmholtz integral equation. The BEM now consists of solving this equation for each point on the boundary to obtain the 'missing' boundary information. Then, with both $p$ and $v_n$ known on the entire boundary mesh, the sound pressure $p$ can be calculated for any point in the domain. Finally the velocity fluctuation $v$ can then be found through equation (3.8).

Some notes regarding this derivation must be made. First, the derivation holds strictly for interior problems, meaning the domain is entirely enclosed by a boundary, unlike an exterior domain which is semi-infinite. The domain may however contain objects inside of it in the form of an inner boundary surface. This is the way in which the object under research is modelled in the next section. The second remark is to emphasize that this derivation holds strictly for an inviscid, quiescent and homogeneous medium. The first and last condition are quite acceptable, however the second essentially prohibits these equations for use in a running wind tunnel, since the medium (air) will be anything but quiescent. However for the present purpose of a proof of concept of the BEM this approach suffices.

### 3.1.3 Boundary conditions

To be able to solve the Helmholtz integral equation on the boundary, conditions on that boundary must be prescribed. For a well-posed problem three possible boundary conditions (BC) may exist, where the dependent variable is the pressure $p$. A Dirichlet condition implies a prescribed value for $p$:

$$p = p_e \tag{3.22}$$

Physically this BC implies the presence of a sound source with pressure amplitude $p_e$ at the location where this BC is applied. The second type is a Neumann condition which prescribes the normal derivative of $p$:

$$\frac{\partial p}{\partial n} = -i\omega\rho_0 v_n \tag{3.23}$$

Note that this is equation (3.8) expressed in the normal direction. Physically this BC implies a vibrating surface with amplitude $v_n$ (for example a car window). Note also that when $v_n = 0$ the BC implies a sound-reflecting surface. The third type is a Robin condition which combines the above two conditions into:

$$z = p/v_n \tag{3.24}$$

This is physically an impedance condition and means that the surface is not fully reflective but not fully absorbent of the sound either. In the special case of $z = \rho_0 c$ all sound is absorbed on the surface. In that case the boundary is called an anechoic termination (for example a wall lined with acoustic foam such as in the NRC wind tunnel).

## 3.2 Numerical implementation

A finite element code describing the BEM as formulated in equation (3.21) has been developed by Wu [9] and is described briefly in section 3.2.1. The pre- and post-processing routines developed in Matlab for this project are discussed in section 3.2.2. The model structure is illustrated in the flow chart in Figure 3.2.

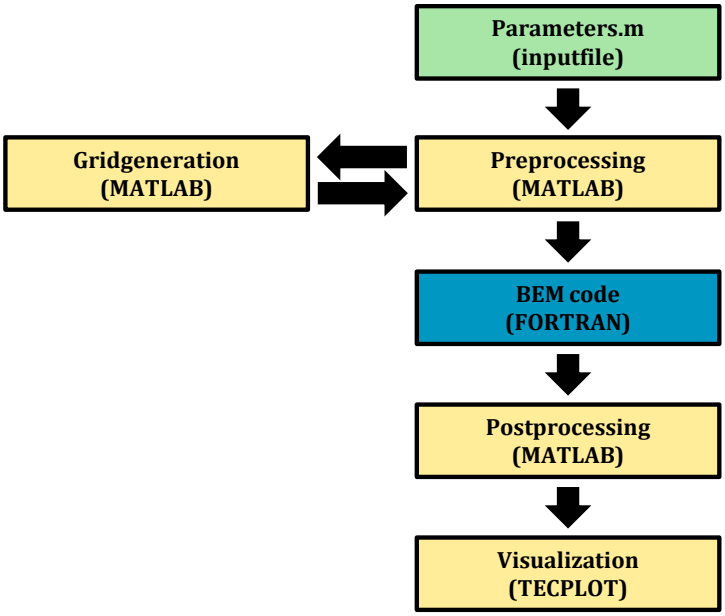



**Figure 3.2: Flow chart of the numerical model structure**

### 3.2.1 BEM code

This section discusses the relevant numerical aspects of the FORTRAN 77 code developed by Wu [9]. The cased discussed is three-dimensional.

To solve the Helmholtz integral equation (3.21) on the boundary numerically, the boundary of the domain is first discretized using linear four-node quadrilateral elements. The resulting linear shape functions are denoted by $N_i$ where $i$=1,...,4 are the indices of the element nodes. Using isoparametric elements, $p$ and $v_n$ are represented by the same shape functions used to represent the geometry:

$$p = \sum_{i=1}^{4} p_i N_i(\xi_1, \xi_2) \tag{3.25}$$

$$v_n = \sum_{i=1}^{4} v_{ni} N_i(\xi_1, \xi_2) \tag{3.26}$$

Here $p_i$ and $v_{ni}$ are the nodal values of the sound pressure and normal velocity, respectively, and $(\xi_1, \xi_2)$ is the local coordinate system in the element. Each combination of a boundary node $i$ and element $j$ produces two coefficient vectors from equation (3.21):

$$h_i = \int_{S_j} \frac{\partial \psi}{\partial n} N_i \, d\Gamma \tag{3.27}$$

$$g_i = -i\rho_0 \omega \int_{S_j} \psi N_i \, d\Gamma \tag{3.28}$$

When assembling $h_i$ and $g_i$ into the global matrices $\boldsymbol{H}^*$ and $\boldsymbol{G}$ the system of equations reads

$$\boldsymbol{Cp} + \boldsymbol{H}^*\boldsymbol{p} = \boldsymbol{G}\boldsymbol{v_n} \tag{3.29}$$

where $\boldsymbol{C}$ is a diagonal matrix containing the $C_i^0$ coefficients for each node. Combining $\boldsymbol{C}$ and $\boldsymbol{H}^*$ into a single matrix yields

$$\boldsymbol{Hp} = \boldsymbol{G}\boldsymbol{v_n} \tag{3.30}$$

Since only half the boundary conditions are known, i.e. $p$ or $v_n$ is known on each node, the unknowns in vectors $\boldsymbol{p}$ and $\boldsymbol{v_n}$ are collected into a single unknown vector $\boldsymbol{x}$, the known values are collected in the right-hand side vector $\boldsymbol{b}$, and the matrices $\boldsymbol{H}$ and $\boldsymbol{G}$ are collected in $\boldsymbol{A}$, so the final system of equations is

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{3.31}$$

The FORTRAN code solves this system of equations using a standard least-squares solver contained in the LINPACK FORTRAN package. The integrals in equations (3.27) and (3.28) are approximated using standard Gaussian quadrature. Once the system is solved all boundary values of pressure and velocity are known. Then the value of $p$ and/or $v_n$ at any point inside the domain can be found using standard Gaussian quadrature on the integrals in equation (3.21).

### 3.2.2 Pre- and post-processing routines

The BEM code as described in the previous sections needs modification to suit the present purpose. The model will be explained according to the subroutines in Figure 3.2. All Matlab files are found in Appendix 1.

*Parameters (Matlab)*

The first pre-processing file *parameters.m* contains the necessary parameters needed to generate the input data files for the FORTRAN code. The static pressure $p_0$ and temperature $T$, sound frequency $f$ and the physical type of each of the cube walls (anechoic, reflecting, or lined with sound sources) can be specified in this file. Furthermore, the dimensions and number of nodes on each wall is specified here. From $p_0$ and $T$ the density $\rho_0$ and speed of sound $c$ are calculated via the ideal gas law:

$$\rho_0 = \frac{p_0 M}{RT} \tag{3.32}$$

$$c = \sqrt{\frac{\lambda_c p_0}{\rho_0}} \tag{3.33}$$

where $M \approx 28.9645 \cdot 10^{-3}$ kg/mol is the molar mass of dry air, $R = 8.3144621$ J/(K · mol) is the universal gas constant and $\lambda_c \approx 1.4$ is the specific heat ratio for air. The signal frequency for all simulations was taken to be $f = 1000$ Hz. The model however has the option of solving for a range of signal frequencies.

*Pre-processor (Matlab)*

When running the *sim_preprocessor.m* file, the parameter file is called. The dimension and mesh size information is used when calling the grid generator (explained below). The grid generator returns the nodal information (coordinates and element connectivity) to the pre-processor. Then, the pre-processor assigns boundary conditions to each element according to the wall type specified in the parameters file. The boundary conditions are implemented by specifying the three constants $C_A$, $C_B$ and $C_C$ in

$$C_A p + C_B v_n = C_C \tag{3.34}$$

The data files generated by the pre-processor serve as input for the FORTRAN BEM code.

*Grid generator (Matlab)*

The grid generator file *sim_gridgen.m* produces a mesh of quadrilateral elements of the geometry sketched in Figure 3.3. The validation geometry to be tested on consists of a cubic box with a cylindrical object somewhere in its interior. The sides of the box have dimension $L_{box} = 6$ ft $=$ 1.8 m. Using the empirical engineering rule that one wavelength $\lambda$ must be represented by a minimum of 6 elements [10] this means the coarsest possible mesh consists of

$$N_{box} = \frac{L_{box}}{\lambda/6} = \frac{6 L_{box} f}{c} = 31.5 \rightarrow 32 \tag{3.35}$$

elements per side. Furthermore since the BEM produces fully populated matrices, the computational time is proportional to $N^4$ (again according to Gunda [10]). Therefore, based on these considerations, throughout this study 48 elements per side are used. The mesh of the cylinder is taken to be much finer given that it is this object of which the amplitude and phase information is eventually of interest. The mesh consists of $N_{cr} = 6$ elements in radial direction, $N_{cc} = 48$ elements around the circumference and $N_{ca} = 36$ elements in axial direction.

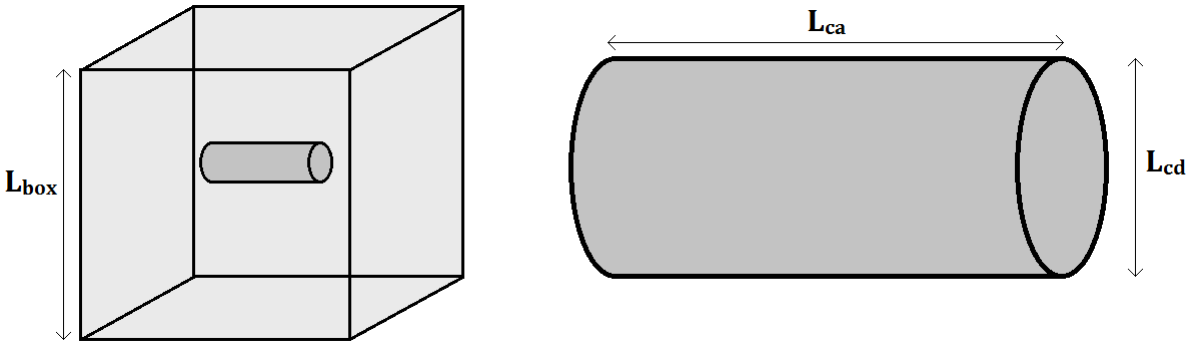



**Figure 3.3: Sketch of the test and modelling geometry**

The cubic box and the cylinder it contains are plotted for visual inspection. If one of the walls contains an array pattern (to replicate the microphone pattern on the wall in the wind tunnel) this mesh is also plotted since the algorithm is prone to producing erroneous meshes when the mesh is too coarse. Examples are given in Figure 3.4. The blue dots in the bottom left picture represent the locations of 32 microphones on one of the box walls. This is explained in detail in chapter 4 on the experimental study. The output of the grid generator routine consists of the node numbers and their coordinates in 3D space as well as their connectivity into quadrilateral elements.

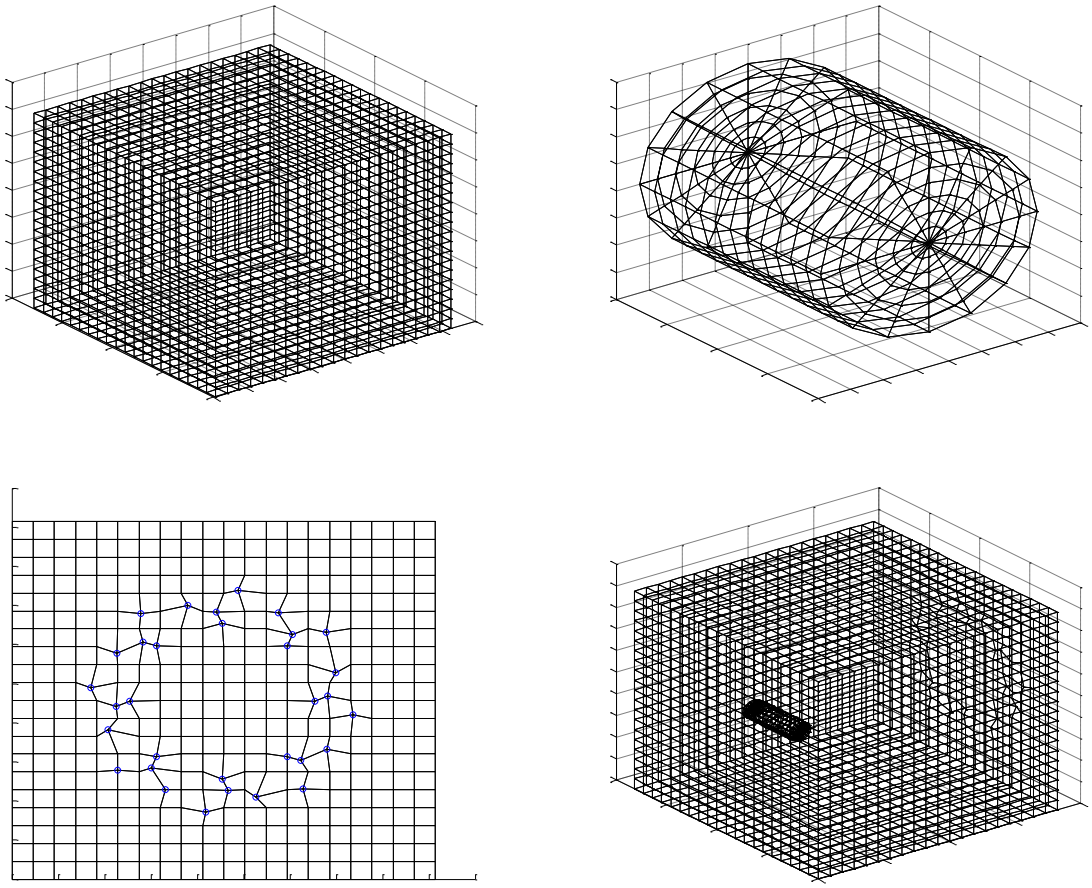


**Figure 3.4: Examples of a cube mesh with 20x20 element sides (top left), a 6x12x16 element mesh of the cylinder (top right), a visual inspection of the wall with microphone array nodes (bottom left) and the complete mesh (bottom right)**

*Post-processor (Matlab)*

The output data files of the BEM code consist of nodal values of the complex pressure and its amplitude. The post-processor routine *sim_postprocessor.m* converts the complex pressure into the phase angle $\theta$ according to:

$$\theta = \text{atan}\frac{\text{Im}(p)}{\text{Re}(p)} \tag{3.36}$$

The pressure amplitude and phase angle are stored in input files that can be used in the visualization program Tecplot.

*Visualization (Tecplot)*

Tecplot is a commercial visualization package, especially powerful in producing flexible colour plots. Most results of the numerical simulations are analyzed using the Tecplot output.

# 4 Experimental study

The objective of the experimental study is to validate the BEM for acoustic simulations. Specifically, the goal is to replicate amplitude and phase data predicted by the BEM.

The setup and equipment used are presented in section 4.1. Then in section 4.2 the actual design of the experiments is discussed. As with the numerical study, the results are postponed until chapter 5.

## 4.1 Setup and equipment

### 4.1.1 Acoustic test section

As mentioned in chapter 3, the test geometry consists of a closed box with sides of 1.8 m. This box is constructed using simple retail plywood plates that are characterized by good sound reflection (an absorption coefficient of $\alpha \approx 0.09$ at $f = 1000$ Hz). To be able to enter the box to adjust the inside walls or the test object, a door is mounted in one of the sides. This door consists of a 1.2 by 1.2 m steel plate in which the 32 microphone array can be mounted. This plate has been used in the past for wind tunnel experiments prior to the arrival of the 64 microphone array currently in use by the NRC. Figure 4.1 is a SolidWorks overview of the box with the microphone array doorplate on the left and the test cylinder in the middle cut-out.

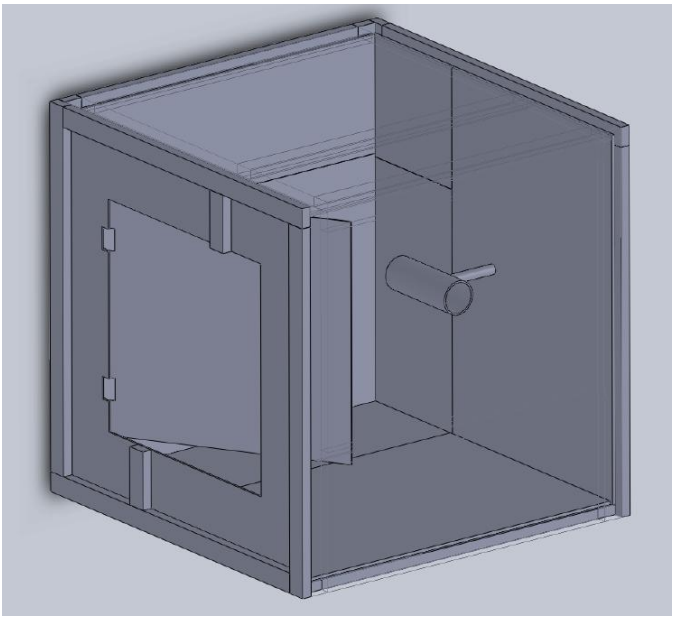


**Figure 4.1: SolidWorks overview of the box geometry, including the microphone array plate and the mounted test object (cylinder)**

The reciprocity principle stated in chapter 1 means that the 32 array positions in the door will be lined with sound sources and the signal will be measured on the test object. The sound sources are simple cell phone speakers. To also obtain information on the box walls, the back-wall (the wall directly opposite to the door) is lined with 21 microphones. The remaining three visible walls contain five microphones each. Some pictures of the acoustic box are given below.

**Figure 4.2: Picture of the acoustic box**

**Figure 4.3: Close-up of the microphone array plate. The plate acts as a door and can open to the inside. The door is closed tightly by the clamps seen on the right of the picture. The groove in which the door fits is lined with sealant to create a sealed sound environment. The array holes are lined with simple cell phone speakers.**

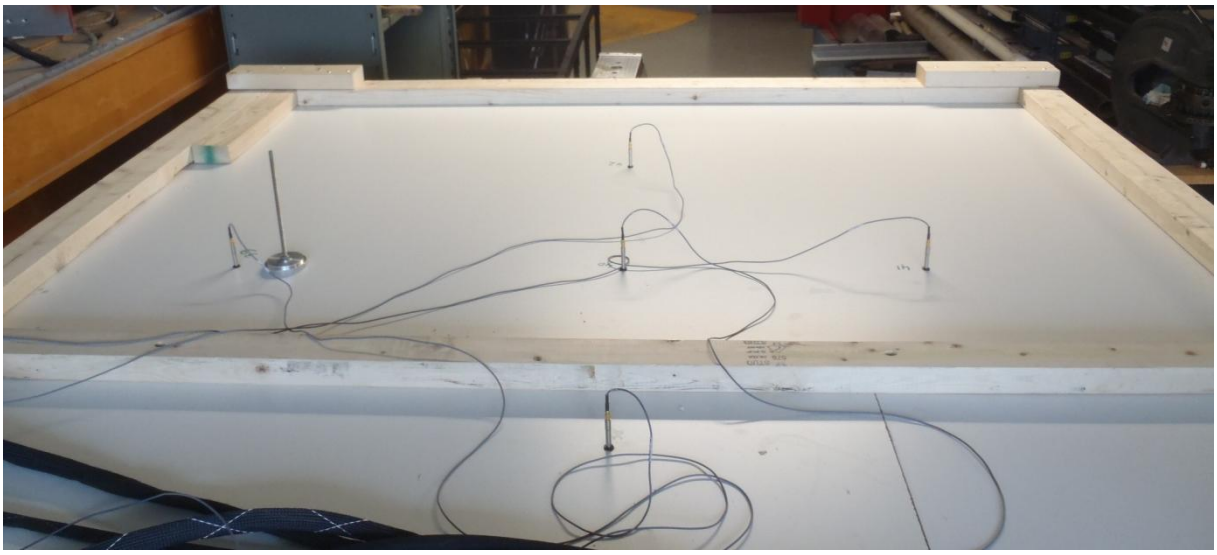


**Figure 4.4: Topside of the box, lined with five microphones**

### 4.1.2 Cylinder test object

The object representing the wind tunnel object is a cylinder with diameter $L_{cd} = 16.6$ cm and length $L_{ca} = 45$ cm. It consists of a piece of PVC pipe of which both ends are sealed with a round metal plate. To mount the microphones, twenty-six holes are drilled in the cylinder in the form of a double helix. Furthermore, a line of five holes is drilled in one of the metal end plates. The purpose of this hole layout will be explained in section 4.2. Some pictures of the cylinder are given below.


**Figure 4.5: Cylinder with double helix microphone array (left), cylinder top plate with five microphones (middle), cylinder insides with microphone montage (right)**

The cylinder must be mounted inside the box without the mounting equipment itself creating too much disturbance for the acoustic waves. Furthermore the mounting must be able to position the cylinder accurately in the order of millimetres. Therefore, the cylinder is hung from the ceiling of the box by a finely threaded rod. The rod is attached to the ceiling by a round metal plate with internal threading (see Figure 4.4). Turning this plate controls the height of the

cylinder inside the box. The other side of the rod is attached to a mounting device inside the cylinder (see Figure 4.6).

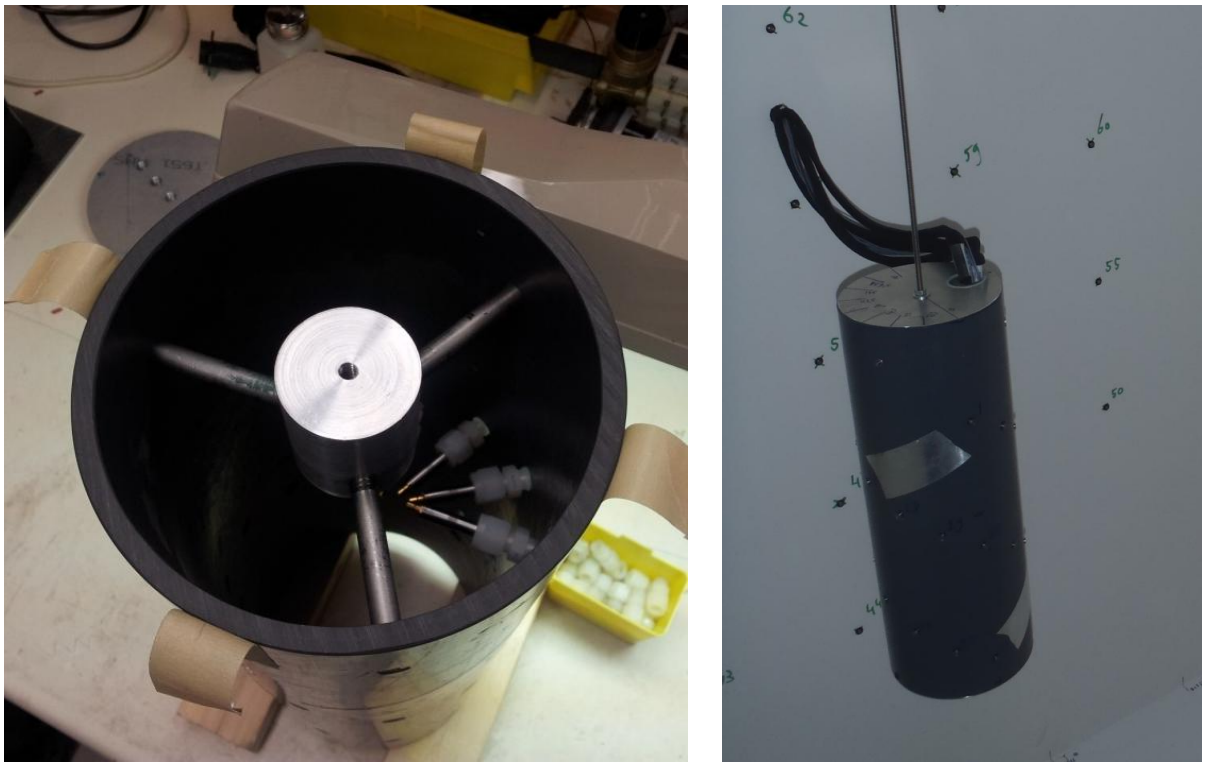


**Figure 4.6: Mounting device inside the cylinder (left), cylinder mounted inside the box (right). Behind the cylinder the microphones in the back-wall can be seen. The cables coming out of the cylinder top contain the wiring for the microphones.**

### 4.1.3 Acoustic array console

The NRC's acoustic array console is used to control the generated sound signal and to process the microphone signals. It consists of a waveform generator, hardware gathering the microphone signals and control software that generates output data files with the frequency information per microphone. The control software also provides a user interface used to monitor the generated and measured signals. Some pictures are given below.

The control software requires the coordinates of the speakers and microphones. The input files for this operation are created by the Matlab file *exp_preprocessor.m*. Since the data files from the console consist of the data in the frequency domain, the Matlab routine *exp_postprocessor.m* carries out a fast Fourier transform (FFT) to convert the data to amplitude and phase data. These routines are found in Appendix 1.

Pictures of the entire setup, the waveform generator and the control environment are given below.


**Figure 4.7: Overview of the test setup with the waveform generator on the left, the acoustic array console in the middle and the acoustic box on the right**

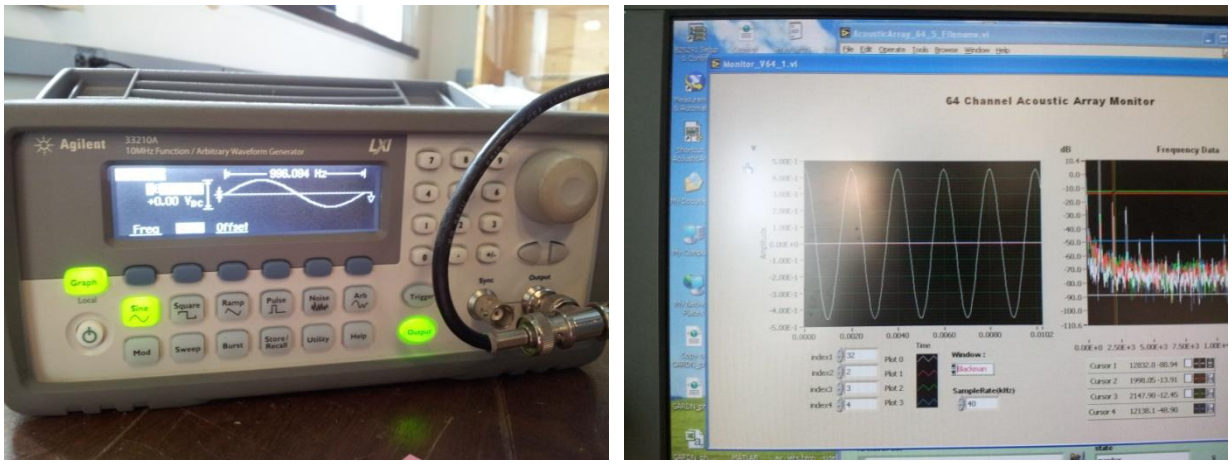



**Figure 4.8: Waveform generator (left) and control environment (right). The control environment shows the individual amplitude-time plots of selected sources (white is the signal from the waveform generator) and the gain-frequency plot**

## 4.2 Experiment design

### 4.2.1 Procedure for one sound source

For each experiment one single sound source on the array mounted on the box door is provided with a 1000 Hz signal. For several microphones on the cylinder and on the back-wall the signal strength is inspected from the control software environment. When the signal strength is confirmed, data collection from all microphones is performed for 60 seconds. The sample frequency is set at 40000 Hz, i.e. the each wavelength for the 1000 Hz signal is reconstructed with 40 data points.

In order to get a sufficiently fine mesh of information on the test object (the cylinder), the microphones on the cylinder are mounted in the form of a double helix of 2 times 13 microphones around the circumference and one line of 5 microphones along the diameter of the bottom endplate (as mentioned in section 4.1.2). For each experiment, the cylinder is rotated 360/16 = 22.5° after each measurement for 7 consecutive times (see Figure 4.9). This procedure yields a final data mesh of $(2 * 13 + 4) * 8 + 1 = 241$ nodes on the cylinder.

### 4.2.2 Wall layout

The experiments are performed for three acoustic situations:

Case 1: All walls of the box are of simple plywood (fully reflecting).
Case 2: The four walls adjacent to the door are lined with acoustic foam (anechoic termination).
Case 3: All walls except the one with the door are lined with acoustic foam.

The purpose of these three different cases is to test the model performance for different acoustic situations. The acoustic foam attachment to the walls is illustrated in Figure 4.9. The result of this procedure is a total of 3 cases * 32 speakers = 96 experiments.

**Figure 4.9: Cylinder with rotation marks on the topside (left), picture of one of the sidewalls lined with acoustic foam (right). The cylinder mounted to the ceiling is shown on the right.**

# 5 Results & comparison

This chapter presents the results of the BEM simulations and experiments described in the previous sections. Some simple verification cases modelled using the BEM are presented in section 5.1. Then, section 5.2 presents the results for the three validation cases using the BEM (section 5.2.1) and from the experiments (section 5.2.2). These results are compared in section 5.3.

## 5.1 BEM verification

### 5.1.1 Point source in free space

Figure 5.1 shows the phase on the boundary for a point source with $f = 1000$ Hz located in the middle of the wall on plane $x = L_{box}$. The phase data reflects the periodic character of sound waves, i.e. the phase $\phi$ varies between $-2\pi$ and 0. Almost 6 wavelengths are represented on the plane in which the source lies, which reflects the frequency: $L_{box}/\lambda = L_{box}/(c/f) = 1.8/0.343 = 5.25$.

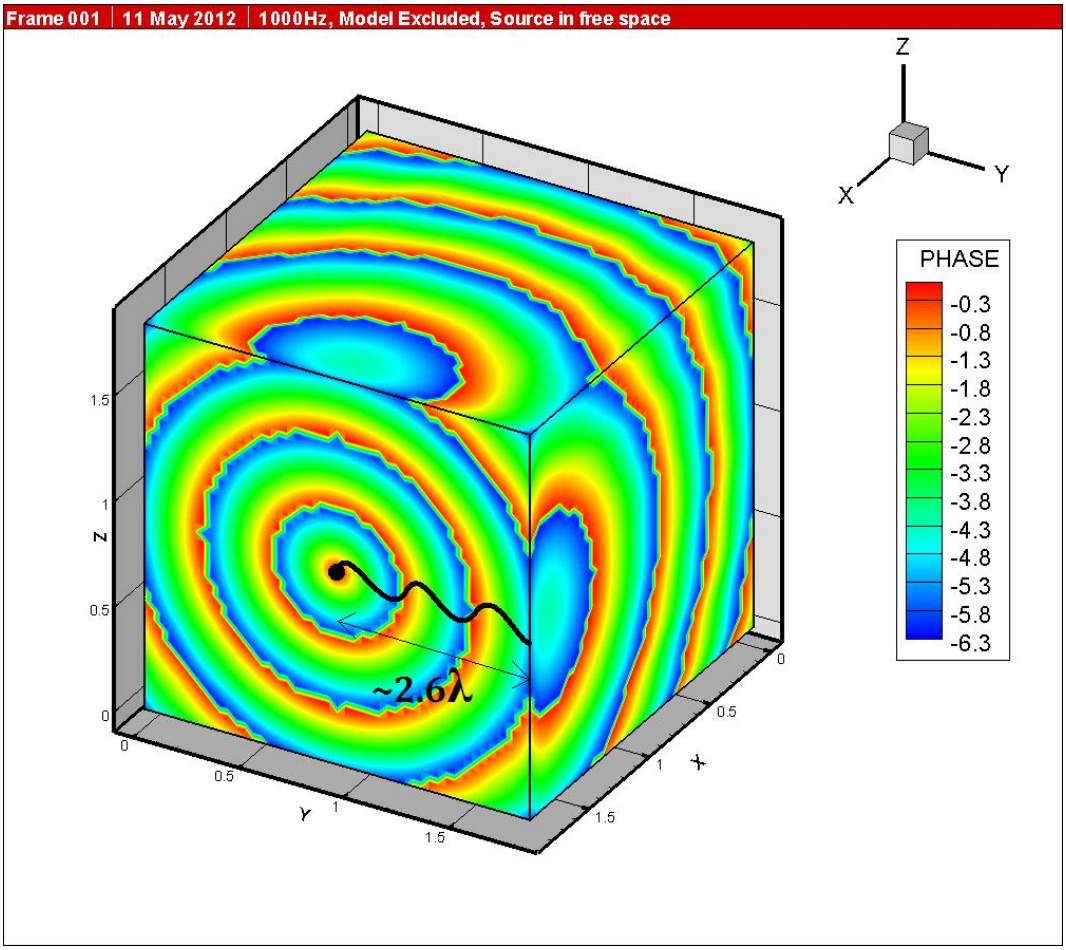



**Figure 5.1: Phase results for a point source (black dot) in free space. Circa 2.6 wavelengths are represented on a half-side of one wall**

The phase result for a point source at the same location, with a solid opposite wall, is depicted in Figure 5.2. In comparison with the point source in free space the reflection off of the solid wall is noted.

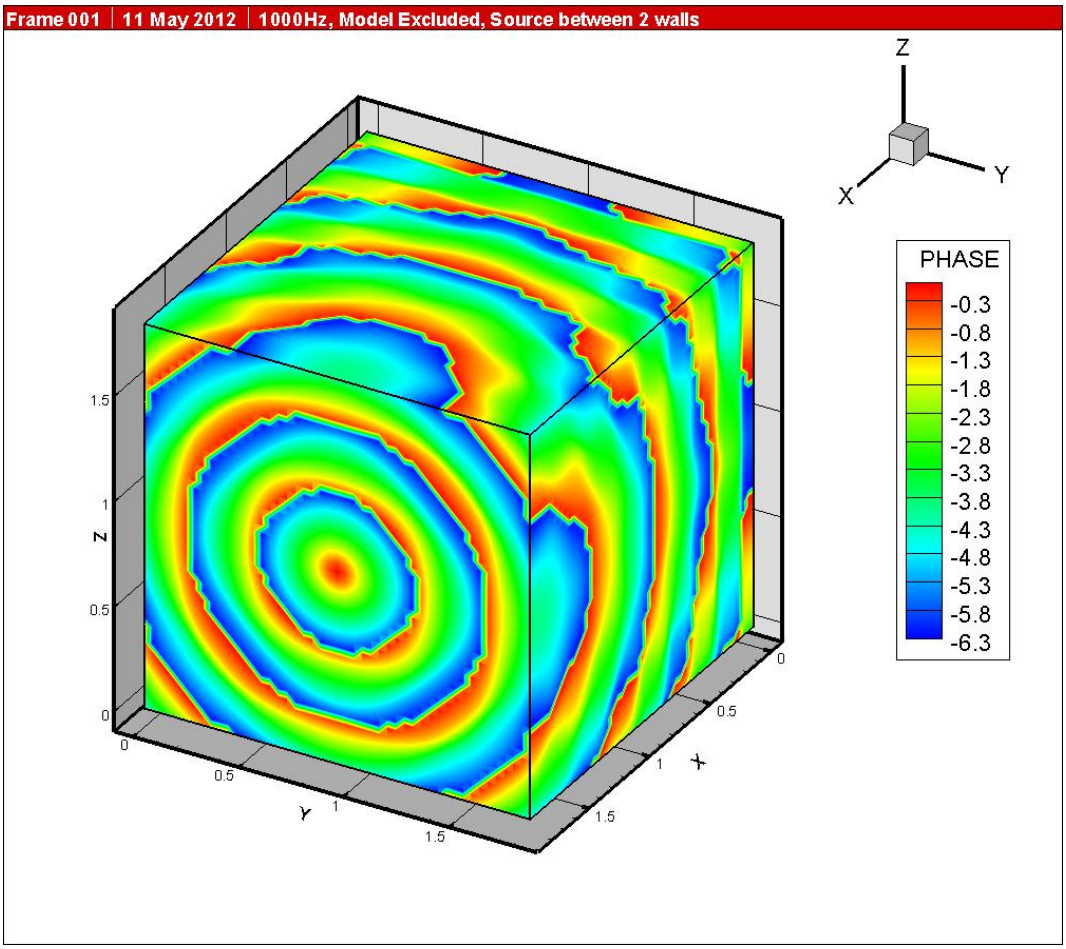



**Figure 5.2: Phase results for a point source with a solid wall on the opposite side (plane $x = 0$).**

The following figures show phase results of several simulations including the cylinder object. The point source locations are $(x = L_{box}, y = 0.5L_{box}, z = 0.5L_{box})$ in Figure 5.3, $(x = L_{box}, y = 0.75L_{box}, z = 0.5L_{box})$ in Figure 5.4 and $(x = L_{box}, y = 0.5L_{box}, z = 0.75L_{box})$ in Figure 5.5. The left figures are simulations in free space whereas the right figures simulate a solid wall at $x = 0$.

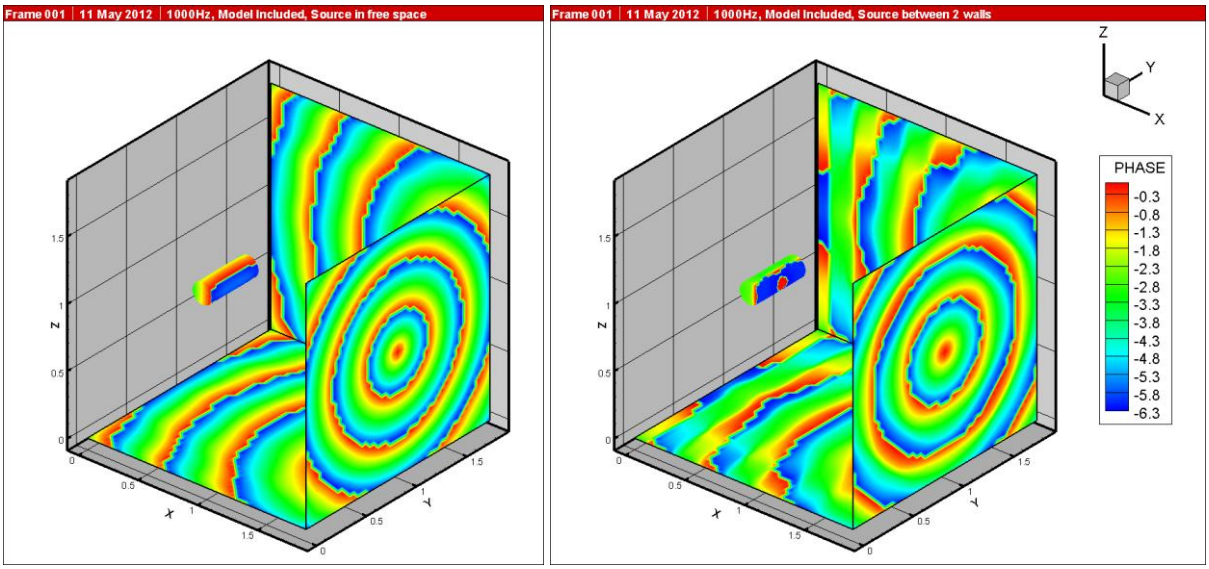



**Figure 5.3: Phase results for a point source at $(x = L_{box}, y = 0.5L_{box}, z = 0.5L_{box})$ in free space (left) and with a solid wall at $x = 0$ (right)**

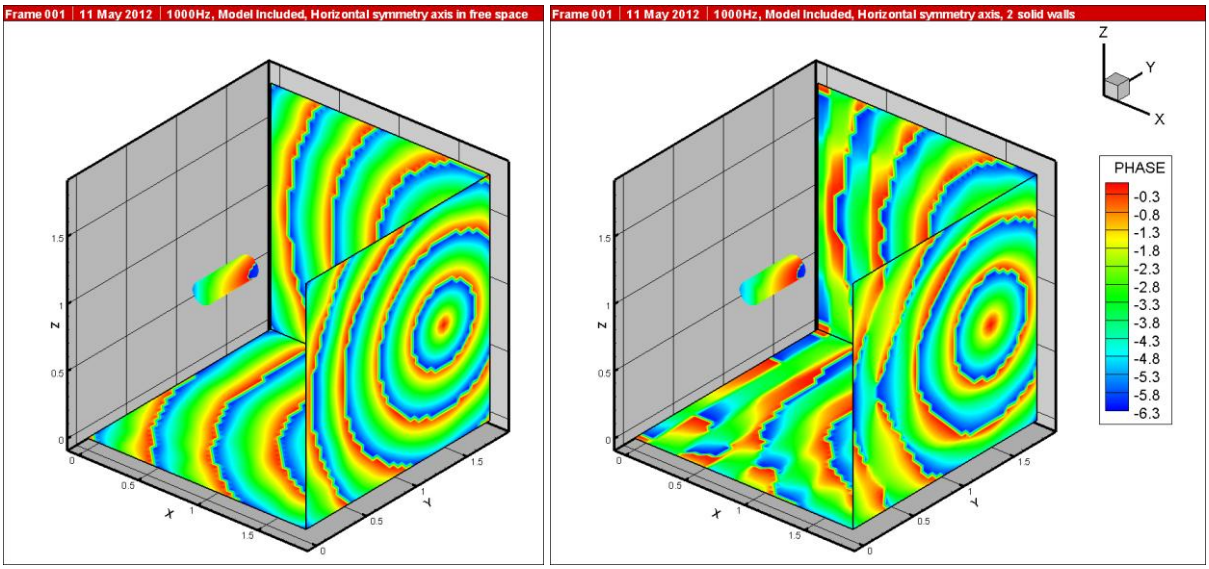



**Figure 5.4: Phase results for a point source at $(x = L_{box}, y = 0.75L_{box}, z = 0.5L_{box})$ in free space (left) and with a solid wall at $x = 0$ (right)**

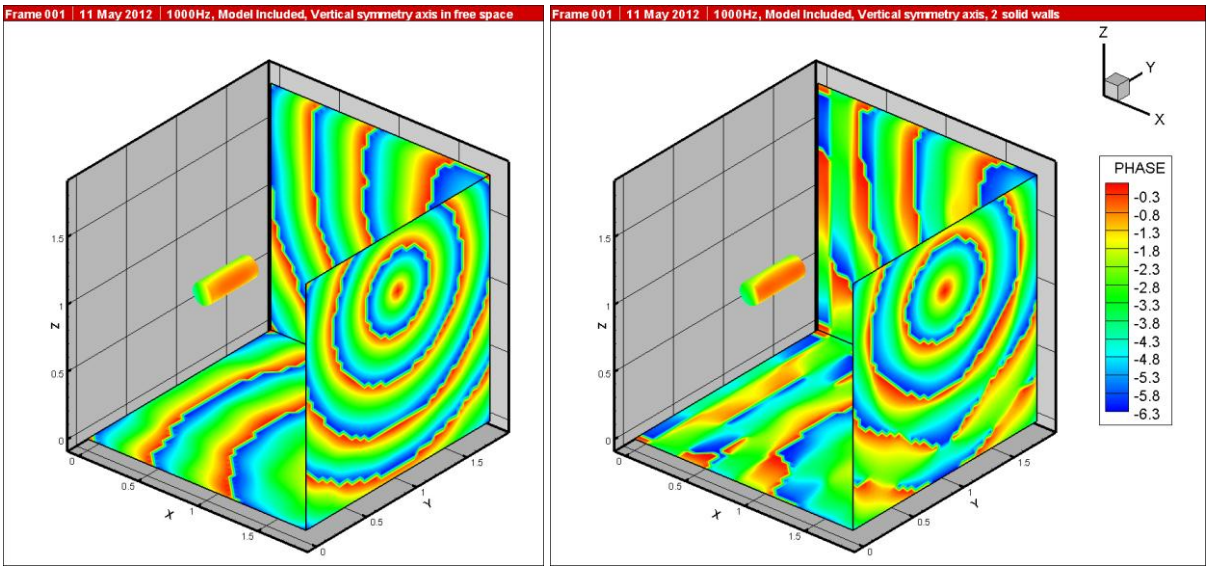



**Figure 5.5: Phase results for a point source at $(x = L_{box}, y = 0.5L_{box}, z = 0.75L_{box})$ in free space (left) and with a solid wall at $x = 0$ (right)**

## 5.2 Validation cases

The results for the three validation cases as described in section 4.2.2 are given in this section. The information on the walls cannot be presented since this would require hundreds of microphones on the walls. This is not an issue however, since the walls have been treated in the previous section in the verification simulations. The results on the surface of the cylinder are presented here, as this is the object of interest.

The results presented here are for a point source (speaker) at array location 1. The speaker position is depicted in Figure 5.6. While experiments and simulations have been carried out for all 32 speaker positions, they all yield similar results. Therefore, the results for just one speaker are presented here.

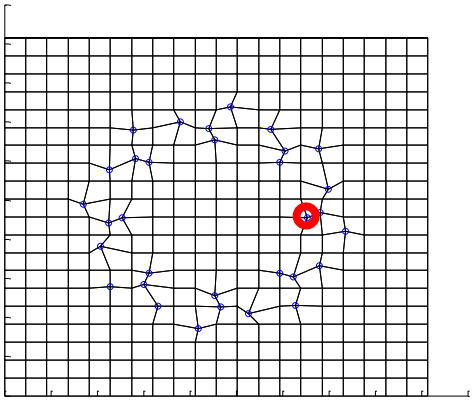


**Figure 5.6: Speaker wall with location of speaker node 1 (red circle)**

### 5.2.1 BEM results

The simulation results for the three cases are presented in the figures below. The amplitude is given in the left figures while the phase data is in the right figures.

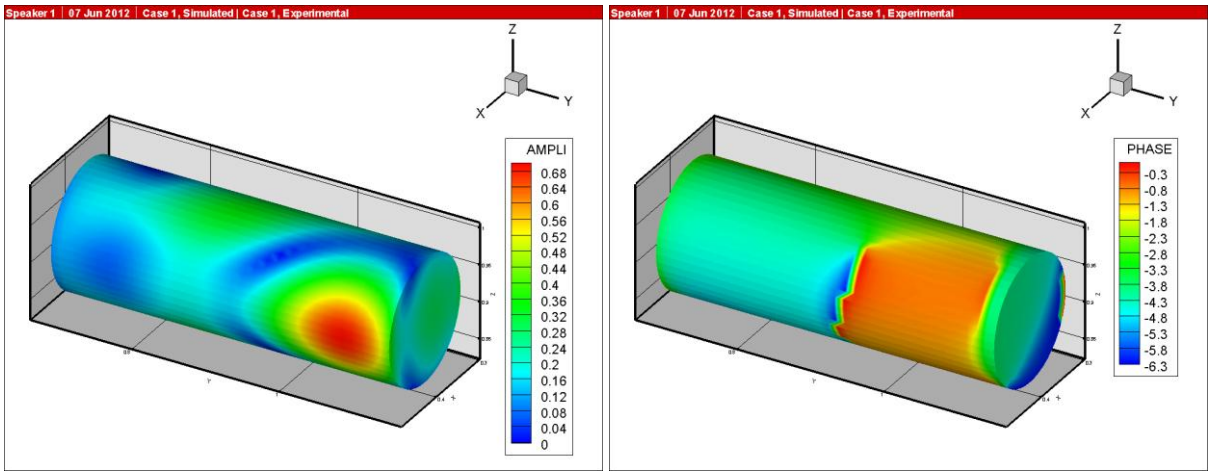



**Figure 5.7: BEM results for amplitude (left) and phase (right) results for case 1: all solid walls**

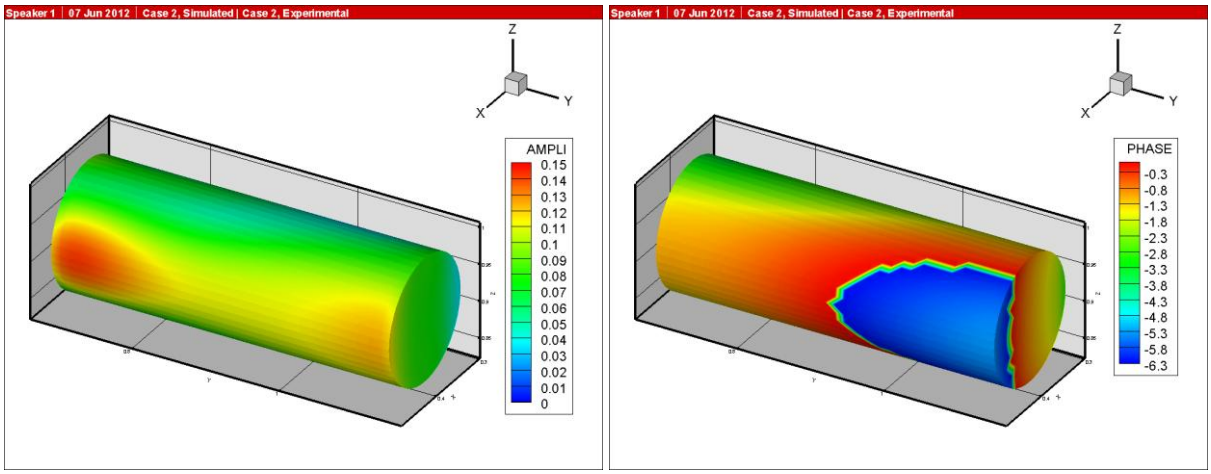



**Figure 5.8: BEM results for amplitude (left) and phase (right) results for case 2: Four anechoic walls, one solid wall**

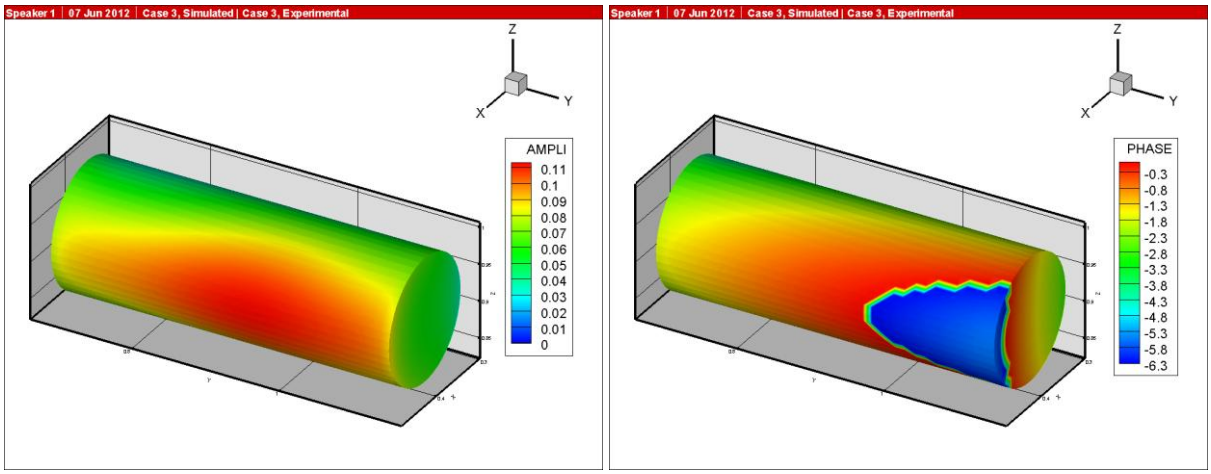



**Figure 5.9: BEM results for amplitude (left) and phase (right) results for case 3: Five anechoic walls**

### 5.2.2 Experimental results

The experimental results for the three cases are presented in the figures below. The amplitude is given in the left figures while the phase data is in the right figures.

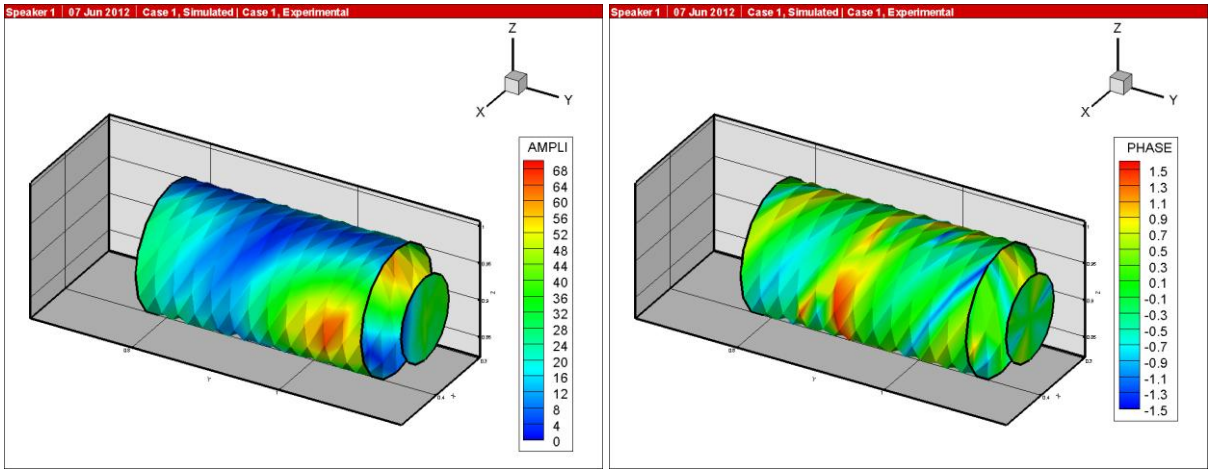



**Figure 5.10: Experimental results for amplitude (left) and phase (right) results for case 1: all solid walls**

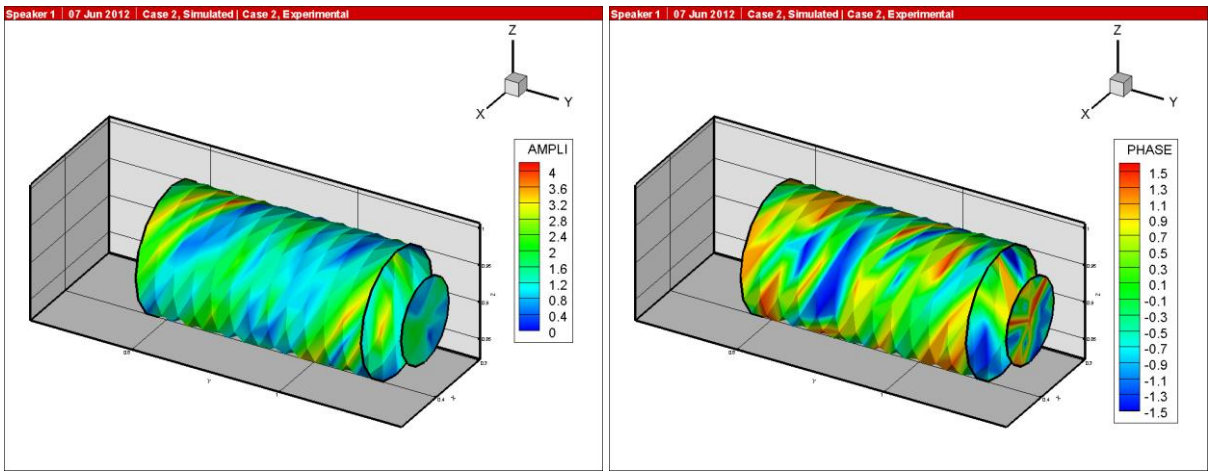



**Figure 5.11: Experimental results for amplitude (left) and phase (right) results for case 2: Four anechoic walls, one solid wall**

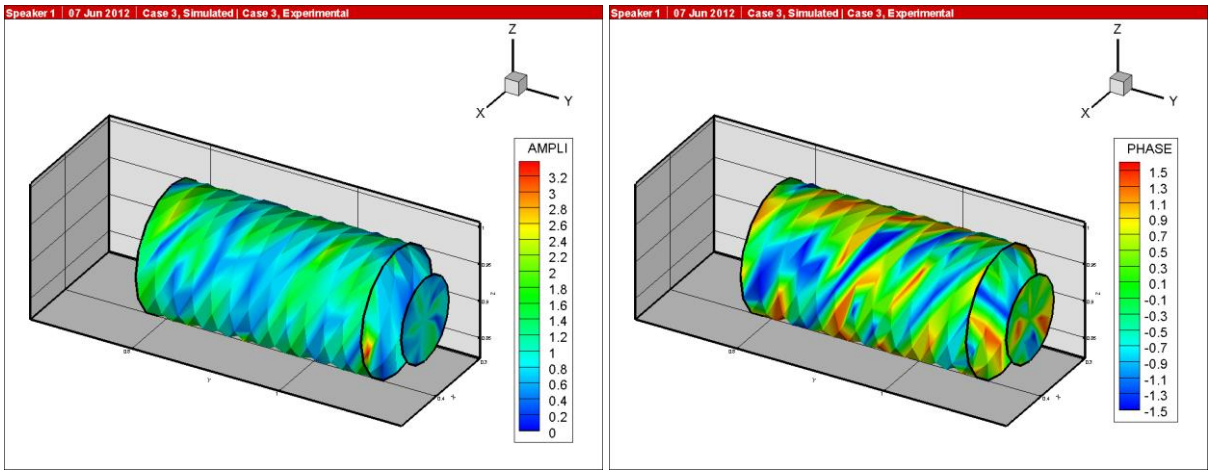



**Figure 5.12: Experimental results for amplitude (left) and phase (right) results for case 3: Five anechoic walls**

## 5.3 Comparison

In the comparison of the numerical and experimental results there is only one situation that yields satisfactory results. This is the amplitude data for case 1: All solid walls. Figure 5.13 shows the amplitude data on the cylinder for case 1 using the BEM and the experimental results (this is merely a copy of the left figures in Figure 5.7 and Figure 5.10).

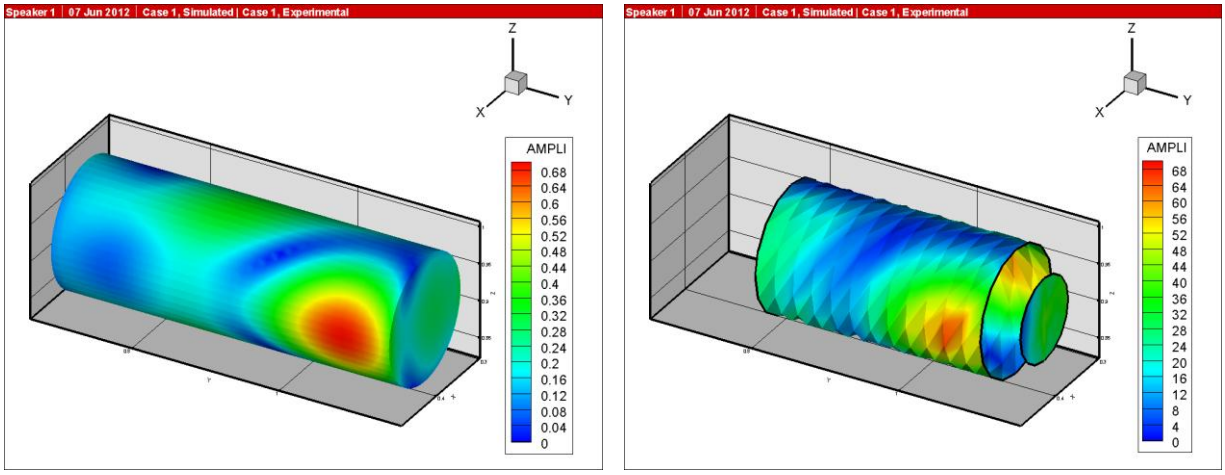



**Figure 5.13: Amplitude results for case 1 using the BEM (left) and experimental data (right)**

The amplitude results for case 2 and case 3, as well as all phase data do not compare well at all. It is hypothesized that one or more of the following might be the cause of the poor amplitude comparison for case 2 and case 3:

1. The acoustic foam used to cover four (case 2) or five (case 3) walls may not provide a fully anechoic termination.
2. When four or five walls are covered in foam most of the signal is absorbed. The signal may have become too weak for the microphones to pick it up properly.

One possible cause for the overall poor phase comparison may lie in the error introduced by the finite sample frequency of 40000 Hz. This sample frequency translates into an allowed phase angle uncertainty of $2\pi/(40000/1000) = 0.15$ rad around the points of zero amplitude [11].

# 6 Conclusions & recommendations

## 6.1 Conclusions

The results in chapter 5 give rise to the following conclusions.

1. The verification results and amplitude results for case 1 show that the Boundary Element Method can be a suitable modelling method for acoustic problems. The computational demands are lower than fully 3D FEM and the results are promisingly accurate.
2. Given the good results for amplitude for case 1, the poor comparison results for the other cases are concluded to be caused by experimental factors such as absorption coefficient of the acoustic foam or measurement inaccuracies stemming from the rotation of the cylinder in between measurements.

Given these conclusions, the proof-of-concept study is positively concluded and the NRC will continue to pursue the topic of employing the BEM to optimize the beamforming method.

## 6.2 Recommendations

The recommendations following the results and conclusions are the following.

1. A suitable BEM code needs to be developed that can include an external velocity field, since the intended use is wind tunnel experiments.
2. Accurate validation experiments are needed to ensure the BEM is suitable to obtain valid phase data. Therefore, it is recommended to line the test object with some orders of magnitude more microphones. This makes it possible to perform entire experiments without having to turn the cylinder in between measurements, therefore minimizing the introduction of human error.
3. The wiring coming out of the cylinder (see Figure 4.6) and the wavelength are of the same order of magnitude and therefore the wiring will influence the acoustic field. The wiring of the microphones must therefore be drastically reduced in size to obtain more accurate experiments.

# Bibliography


[1] International Air Transport Association, “Growth and Development,” 2013. [Online]. Available: http://www.iata.org/about/Pages/history_3.aspx. [Accessed 26 June 2013].

[2] International Air Transport Association, “Airlines to Welcome 3.6 Billion Passengers in 2016,” 6 December 2012. [Online]. Available: http://www.iata.org/pressroom/pr/pages/2012-12-06-01.aspx. [Accessed 26 June 2013].

[3] J. Syms, “Reducing noise pollution near airports,” 1 December 2011. [Online]. Available: http://archive.nrc-cnrc.gc.ca/eng/news/nrc/2011/12/01/iar-wind-tunnel.html. [Accessed 26 June 2013].

[4] A. Greensted, “Microphone Array Beamforming - The Lab Book Pages,” 29 November 2010. [Online]. Available: http://www.labbookpages.co.uk/audio/beamforming.html. [Accessed 28 June 2013].

[5] J. Christensen and J. Hald, “Beamforming Technical Review,” Brüel & Kjær Sound & Vibration Measurement A/S, Nærum, 2004.

[6] L. Rayleigh, “On the Application of the Principle of Reciprocity to Acoustics,” *Proc. R. Soc. Lond.,* vol. 25, no. 171-178, pp. 118-122, 1876.

[7] G. G. Stokes, “On the perfect Blackness of the Central Spot in Newton's Rings, and on the Verification of Fresnel's Formula for the intensities of Reflected and Reflacted Rays,” *The Cambridge and Dublin mathematical journal,* vol. 4, pp. 1-14, 1849.

[8] H. v. Helmholtz, Handbuch der physiologischen Optik, Leipzig: Leopold Voss, 1856, p. 169.

[9] T. Wu, Boundary Element Acoustics - Fundamentals and Computer Codes, Southampton: WIT Press, 2005.

[10] R. Gunda, “Boundary Element Acoustics and the Fast Multipole Method (FMM),” *Sound and Vibrations,* pp. 12-16, 2008.

[11] J. Syms, *Private correspondence,* 2012.

# Appendix 1. Matlab files

## A1.1 parameters.m

```
%% Input: Problem parameters

int = 1;            % 0 = exterior problem, 1 = interior problem
isc = 0;            % 0 = no scattering, 1 = scattering
isym = 0;           % 0 = no symmetry, 1 = symmetry
p = 100900;         % Pressure in Pa
T = 23.5;           % Temperature in degrees C
freq1 = 996.0938;   % First frequency in Hz
freq2 = 0;          % Second frequency in Hz
df = 0;             % Frequency stepsize (if freq2=df=0 only freq1 is used)
tol = 0.001;        % Tolerance for comparison purposes because of floating point error
inch = 0.0254;

%%% Calculation of rho and c (assumption of ideal gas)
M=28.9645*10^-3;        % Molar mass of dry air in kg/mol
R=8.3144621;            % Universal gas constant in J/(K*mol)
lambda=1.4;             % Specific heat ratio for air
rho=p*M/(R*(T+273.15)); % Density in m/s from ideal gas law
c=sqrt(lambda*p/rho);   % Speed of sound in m/s

%% Input: Simulation number

simulation = 3;
% 1 = All solid walls, with cylinder (Case 1)
% 2 = Four anechoic sidewalls, with cylinder (Case 2)
% 3 = Five anechoic sidewalls, with cylinder (Case 3)

% 4 = All solid walls, without cylinder
% 7 = Four anechoic sidewalls, without cylinder
% 8 = Five anechoic sidewalls, without cylinder

% 5 = Testcase: Wu page 64
% 6 = Testcase: Adjustable (in preprocessor.m)

%% Input: Geometry and mesh parameters (in inches)

xlow = 20.7;        % x-coordinate of lower box boundary
xlim = 92.7;        % x-coordinate of upper box boundary
ylow = 0;           % y-coordinate of lower box boundary
ylim = 72;          % y-coordinate of upper box boundary
zlow = 0;           % z-coordinate of lower box boundary
zlim = 72;          % z-coordinate of upper box boundary
N = 20;             % Number of elements per side
ccx = 36;           % x-coordinate of the center of the cylinder
ccy = 36;           % y-coordinate of the center of the cylinder
ccz = 36;           % z-coordinate of the center of the cylinder
ccl = 18;           % Length of the cylinder
ccr = 3.3;          % Radius of the cylinder
Nrad = 6;           % Number of elements in radial direction
Ncir = 16;          % Number of elements in circumferential direction
Naxi = 12;          % Number of elements in axial direction

sample_freq = 40000;   % Sample frequency in Hz
fft_steps = 2048;

%% Optional input: Only used in testcases (simulation = 5 or 6)

title = 'Vertical symmetry axis in free space';
inputfilename = 'input_node99.dat';
outputfilename = 'output_node99.dat';

%% Determination of simulation variables

% micplate: 0 = no micplate, 1 = micplate
% cylinder: 0 = no cylinder, 1 = cylinder, 2 = no cylinder, but included as field points
% walls: 1 = solid walls, 2 = anechoic walls, 3 = Four anechoic walls, 4 = Five anechoic walls
if simulation==1
    micplate=1;
    cylinder=1;
```

```
    walls=1;
elseif simulation==2
    micplate=1;
    cylinder=1;
    walls=3;
elseif simulation==3
    micplate=1;
    cylinder=1;
    walls=4;
elseif simulation==4
    micplate=1;
    cylinder=2;
    walls=1;
elseif simulation==5
    micplate=0;
    cylinder=0;
elseif simulation==6
    micplate=0;
    cylinder=1;
    walls=2;
elseif simulation==7
    micplate=1;
    cylinder=2;
    walls=3;
elseif simulation==8
    micplate=1;
    cylinder=2;
    walls=4;
else
    disp('ERROR: Simulation number not recognized');
    return
end
```

## A1.2 sim_preprocessor.m

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% SIM_PREPROCESSOR
%-----------------------------------------------------------------------------
%
% Author:       T.J. van der Meer
%
% Modified:     14-5-2012
%
% Description:  Writes the inputfile(s) for the FORTRAN BEM-model based on
%               the problem parameters, boundary conditions and grid
%               generation.
%
% Inputs:       - parameters.m
%               - sim_gridgen.m
%
% Outputs:      - input_node*.dat
%
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%------ Initialization --------------------------------------------------------

clear
clc
bcs=zeros(1,9);
abs_coeff=0.09;   % abs_coeff = 0.09 for plywood at 1000 Hz
impedance=(1+sqrt(1-abs_coeff))/(1-sqrt(1-abs_coeff));

%------ Load problem parameters -----------------------------------------------

run('parameters');

%------ Grid generation -------------------------------------------------------

[nodes,elems,field,displnodes]=sim_gridgen(cylinder,micplate,xlow,xlim,ylow,ylim,zlow,zlim,N,c
cx,ccy,ccz,ccl,ccr,Nrad,Ncir,Naxi,tol,inch);
nnodes=size(nodes,1);    % Number of nodes
nelems=size(elems,1);    % Number of elements
nfield=size(field,1);    % Number of field points

%------ Set boundary conditions and write output ------------------------
```

```
if simulation==5   % Testcase: Wu page 64

    % Boundary condition generation
    bcs(1,:)=[1 nelems 0 1 0 -impedance*rho*c 0 0 0];   % Plywood walls
    j=2;
    for i=1:nelems
        if abs(nodes(elems(i,1),1))<tol && abs(nodes(elems(i,1),2))>tol &&
abs(nodes(elems(i,1),3))>tol
            bcs(j,:)=[i i 0 0 0 1 0 -1 0];   % Plane x=0 moving with 1 m/s
            j=j+1;
        elseif abs(nodes(elems(i,1),1)-(xlim-xlow)*inch)<tol && abs(nodes(elems(i,1),2)-(ylim-
ylow)*inch)>tol && abs(nodes(elems(i,1),3)-(zlim-zlow)*inch)>tol
            bcs(j,:)=[i i 0 1 0 -rho*c 0 0 0];   % Anechoic termination on plane x=xlim
            j=j+1;
        end
    end
    nbcs=size(bcs,1);   % Number of BC's

    % Write output
    fid = fopen(inputfilename,'w');   % Open file
    fprintf(fid,'%s\nGENERAL
INFORMATION\n%i,%i,%i\n%f,%f\n%f,0,0\n',title,int,isc,isym,rho,c,freq1);   % General info
    fprintf(fid,'NODES\n%i\n',nnodes);   % Nodal numbers and coordinates
    for i=1:nnodes
        fprintf(fid,'%i,%f,%f,%f\n',i,nodes(i,1),nodes(i,2),nodes(i,3));
    end
    fprintf(fid,'ELEMENTS\n%i\n',nelems);   % Element numbers and nodes
    for i=1:nelems
        fprintf(fid,'%i,%i,%i,%i,%i\n',i,elems(i,1),elems(i,2),elems(i,3),elems(i,4));
    end
    fprintf(fid,'BOUNDARY CONDITIONS\n%i\n',nbcs);   % Boundary conditions
    for i=1:nbcs

fprintf(fid,'%i,%i,%f,%f,%f,%f,%f,%f,%f\n',bcs(i,1),bcs(i,2),bcs(i,3),bcs(i,4),bcs(i,5),bcs(i,
6),bcs(i,7),bcs(i,8),bcs(i,9));
    end
    fprintf(fid,'FIELD POINTS\n%i\n%f,%f,%f\n%f,%f,%f\n',2,0.25,0.5,0.5,0.5,0.5,0.5);   %
Field points
    fprintf(fid,'SOLVE\nEND');   % Closing statements
    fclose(fid);   % Close file

elseif simulation==6   % Testcase: adjustable

    % Boundary condition generation
    j=1;
    bcs(j,:)=[1 nelems 0 1 0 -rho*c 0 0 0];   % All anechoic walls
    j=j+1;
    if walls==4
        for i=1:nnodes
            if abs(nodes(i,1)-(xlim-xlow)*inch)<tol && abs(nodes(i,2)-(ylim-ylow)*inch)>tol &&
abs(nodes(i,3)-(zlim-zlow)*inch)>tol
                bcelem=find(ismember(elems(:,1),i)'==1);
                bcs(j,:)=[bcelem bcelem 0 1 0 -impedance*rho*c 0 0 0];   % Speakerplate =
Plywood wall
                j=j+1;
            end
        end
    elseif walls==3
        for i=1:nnodes
            if abs(nodes(i,1)-(xlim-xlow)*inch)<tol && abs(nodes(i,2)-(ylim-ylow)*inch)>tol &&
abs(nodes(i,3)-(zlim-zlow)*inch)>tol
                bcelem=find(ismember(elems(:,1),i)'==1);
                bcs(j,:)=[bcelem bcelem 0 1 0 -impedance*rho*c 0 0 0];   % Speakerplate =
Plywood wall
                j=j+1;
            elseif abs(nodes(i,1))<tol && abs(nodes(i,2))>tol && abs(nodes(i,3))>tol
                bcelem=find(ismember(elems(:,1),i)'==1);
                bcs(j,:)=[bcelem bcelem 0 1 0 -impedance*rho*c 0 0 0];   % Mounting plate =
Plywood wall
                j=j+1;
            end
        end
    end
    for i=1:nnodes
        if abs(nodes(i,1)-(xlim-xlow)*inch)<tol && abs(nodes(i,2)-0.5*(ylim-ylow)*inch)<tol &&
abs(nodes(i,3)-0.75*(zlim-zlow)*inch)<tol
            bcelem=find(ismember(elems(:,1),i)'==1);
```

```
            bcs(j,:)=[bcelem bcelem 1 1 0 0 0 10 0];   % Source on [xlim, ylim*0.5, zlim*0.75]
            bcelem=find(ismember(elems(:,2),i)'==1);
            bcs(j+1,:)=[bcelem bcelem 2 1 0 0 0 10 0];
            bcelem=find(ismember(elems(:,3),i)'==1);
            bcs(j+2,:)=[bcelem bcelem 3 1 0 0 0 10 0];
            bcelem=find(ismember(elems(:,4),i)'==1);
            bcs(j+3,:)=[bcelem bcelem 4 1 0 0 0 10 0];
        end
    end
    nbcs=size(bcs,1);   % Number of BC's

    % Write output
    fid = fopen(inputfilename,'w');   % Open file
    fprintf(fid,'%s\nGENERAL
INFORMATION\n%i,%i,%i\n%f,%f\n%f,0,0\n',title,int,isc,isym,rho,c,freq1);   % General info
    fprintf(fid,'NODES\n%i\n',nnodes);   % Nodal numbers and coordinates
    for i=1:nnodes
        fprintf(fid,'%i,%f,%f,%f\n',i,nodes(i,1),nodes(i,2),nodes(i,3));
    end
    fprintf(fid,'ELEMENTS\n%i\n',nelems);   % Element numbers and nodes
    for i=1:nelems
        fprintf(fid,'%i,%i,%i,%i,%i\n',i,elems(i,1),elems(i,2),elems(i,3),elems(i,4));
    end
    fprintf(fid,'BOUNDARY CONDITIONS\n%i\n',nbcs);   % Boundary conditions
    for i=1:nbcs

fprintf(fid,'%i,%i,%f,%f,%f,%f,%f,%f,%f\n',bcs(i,1),bcs(i,2),bcs(i,3),bcs(i,4),bcs(i,5),bcs(i,
6),bcs(i,7),bcs(i,8),bcs(i,9));
    end
    fprintf(fid,'FIELD POINTS\n%i\n',nfield);   % Field points
    for i=1:nfield
        fprintf(fid,'%f,%f,%f\n',field(i,1),field(i,2),field(i,3));
    end
    fprintf(fid,'SOLVE\nEND');   % Closing statements
    fclose(fid);   % Close file
else   % All other simulations
    for mic=1:32

        % Boundary condition generation
        j=1;
        if walls==1
            bcs(j,:)=[1 nelems 0 1 0 -impedance*rho*c 0 0 0];   % Plywood walls
            j=j+1;
        elseif walls==2
            bcs(j,:)=[1 nelems 0 1 0 -rho*c 0 0 0];   % Anechoic walls
            j=j+1;
        elseif walls==3
            bcs(j,:)=[1 nelems 0 1 0 -rho*c 0 0 0];   % Four anechoic walls
            j=j+1;
            for i=1:nnodes
                if abs(nodes(i,1))<tol && abs(nodes(i,2))>tol && abs(nodes(i,3))>tol
                    bcelem=find(ismember(elems(:,1),i)'==1);
                    bcs(j,:)=[bcelem bcelem 0 1 0 -impedance*rho*c 0 0 0];   % Mounting plate
= Plywood wall
                    j=j+1;
                elseif abs(nodes(i,1)-(xlim-xlow)*inch)<tol && abs(nodes(i,2)-(ylim-
ylow)*inch)>tol && abs(nodes(i,3)-(zlim-zlow)*inch)>tol
                    bcelem=find(ismember(elems(:,1),i)'==1);
                    bcs(j,:)=[bcelem bcelem 0 1 0 -impedance*rho*c 0 0 0];   % Speakerplate =
Plywood wall
                    j=j+1;
                end
            end
        elseif walls==4
            bcs(j,:)=[1 nelems 0 1 0 -rho*c 0 0 0];   % Five anechoic walls
            j=j+1;
            for i=1:nnodes
                if abs(nodes(i,1)-(xlim-xlow)*inch)<tol && abs(nodes(i,2)-(ylim-
ylow)*inch)>tol && abs(nodes(i,3)-(zlim-zlow)*inch)>tol
                    bcelem=find(ismember(elems(:,1),i)'==1);
                    bcs(j,:)=[bcelem bcelem 0 1 0 -impedance*rho*c 0 0 0];   % Speakerplate =
Plywood wall
                    j=j+1;
                end
            end
        end
```

```
        bcelem=find(ismember(elems(:,1),displnodes(mic))'==1);   % Source at current
speakernode
        bcs(j,:)=[bcelem bcelem 1 1 0 0 0 10 0];
        bcelem=find(ismember(elems(:,2),displnodes(mic))'==1);
        bcs(j+1,:)=[bcelem bcelem 2 1 0 0 0 10 0];
        bcelem=find(ismember(elems(:,3),displnodes(mic))'==1);
        bcs(j+2,:)=[bcelem bcelem 3 1 0 0 0 10 0];
        bcelem=find(ismember(elems(:,4),displnodes(mic))'==1);
        bcs(j+3,:)=[bcelem bcelem 4 1 0 0 0 10 0];
        nbcs=size(bcs,1);   % Number of BC's

        %%% Write 32 inputfiles
        title = sprintf('Speaker %d',mic);
        inputfilename = sprintf('input_node%2.2d.dat',mic);   % Determination of inputfile
        fid = fopen(inputfilename,'w');   % Open file
        fprintf(fid,'%s\nGENERAL
INFORMATION\n%i,%i,%i\n%f,%f\n%f,0,0\n',title,int,isc,isym,rho,c,freq1);   % General info
        fprintf(fid,'NODES\n%i\n',nnodes);   % Nodal numbers and coordinates
        for i=1:nnodes
            fprintf(fid,'%i,%f,%f,%f\n',i,nodes(i,1),nodes(i,2),nodes(i,3));
        end
        fprintf(fid,'ELEMENTS\n%i\n',nelems);   % Element numbers and nodes
        for i=1:nelems
            fprintf(fid,'%i,%i,%i,%i,%i\n',i,elems(i,1),elems(i,2),elems(i,3),elems(i,4));
        end
        fprintf(fid,'BOUNDARY CONDITIONS\n%i\n',nbcs);   % Boundary conditions
        for i=1:nbcs

fprintf(fid,'%i,%i,%f,%f,%f,%f,%f,%f,%f\n',bcs(i,1),bcs(i,2),bcs(i,3),bcs(i,4),bcs(i,5),bcs(i,
6),bcs(i,7),bcs(i,8),bcs(i,9));
        end
        fprintf(fid,'FIELD POINTS\n%i\n',nfield);   % Field points
        for i=1:nfield
            fprintf(fid,'%f,%f,%f\n',field(i,1),field(i,2),field(i,3));
        end
        fprintf(fid,'SOLVE\nEND');   % Closing statements
        fclose(fid);   % Close file
    end
end
```

## A1.3 sim_gridgen.m

```
function [nodes,elems,field,displnodes] =
sim_gridgen(cylinder,micplate,xlow,xlim,ylow,ylim,zlow,zlim,N,ccx,ccy,ccz,ccl,ccr,Nrad,Ncir,Na
xi,tol,inch)
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% SIM_GRIDGEN
%-----------------------------------------------------------------------------
%
% Author:       T.J. van der Meer
%
% Modified:     14-5-2012
%
% Description:  Creates the quadrilateral 3D boundary grid for BEM-model.
%
% Inputs:       - geometrical and simulation parameters
%
% Outputs:      - [nodes]                   (nodal coordinates)
%               - [elems]                   (element connectivity)
%               - [field]                   (nodal coordinates of fieldpoints)
%               - displnodes                (microphone node numbers)
%               - posproc_conn_*.dat        (element connectivity for postprocessor)
%               - posproc_micnodenrs.dat%   (microphone node numbers for postprocessor)
%
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%------ Initialization --------------------------------------------------------

    nodesbox=zeros(1,3);
    nodescil=zeros(1,5);
    elemsbox=zeros(1,4);
    elemscil=zeros(1,4);
    field=zeros(1,3);
    dist=zeros(1,2);
    dx=(xlim-xlow)/N;
```

```
    dy=(ylim-ylow)/N;
    dz=(zlim-zlow)/N;
    [x,y]=meshgrid(xlow:(xlim-xlow)/N:xlim,ylow:(ylim-ylow)/N:ylim);
    z=zlow:(zlim-zlow)/N:zlim;
    dr=ccr/Nrad;
    da=ccl/Naxi;
    dc=2*pi/Ncir;
    r=0:ccr/Nrad:ccr;
    a=0:ccl/Naxi:ccl;
    c=0:2*pi/Ncir:2*pi;
    displnodes=1;

%------ Box part ------------------------------------------------------------

    % Construction of box-part of the nodal matrix
    nnr=1;
    for w=1:N+1
        if w==1||w==N+1   % Creation of nodal roster on z=0 and z=zlim
            for u=1:N+1
                for v=1:N+1
                    nodesbox(nnr,:)=[x(v,u) y(v,u) z(w)];
                    nnr=nnr+1;
                end
            end
        else
            for u=1:N+1   % Creation of nodal circumferential roster on 0<z<zlim
                if u==1||u==N+1
                    for v=1:N+1
                        nodesbox(nnr,:)=[x(v,u) y(v,u) z(w)];
                        nnr=nnr+1;
                    end
                else
                    nodesbox(nnr,:)=[x(1,u) y(1,u) z(w)];
                    nnr=nnr+1;
                    nodesbox(nnr,:)=[x(N+1,u) y(N+1,u) z(w)];
                    nnr=nnr+1;
                end
            end
        end
    end
    nnodesbox=size(nodesbox,1);   % Number of box nodes

    % Construction of box-part of the element matrix
    felxlow = fopen('postproc_conn_xlow.dat','w');   % Connectivity files for the purpose of
Tecplot postprocessing
    felxlim = fopen('postproc_conn_xlim.dat','w');
    felylow = fopen('postproc_conn_ylow.dat','w');
    felylim = fopen('postproc_conn_ylim.dat','w');
    felzlow = fopen('postproc_conn_zlow.dat','w');
    felzlim = fopen('postproc_conn_zlim.dat','w');
    for nnr=1:nnodesbox   % Loop over all nodes, each node except 2 are used to
        % create an element with its surrounding nodes.
        % This is done by determining on what plane the node
        % lies and finding the surrounding nodes in counter-
        % clockwise direction. The resulting connectivity is
        % stored in elemsbox(nnr,:) and the above files.
        if nodesbox(nnr,1)==xlow && nodesbox(nnr,2)~=ylow && nodesbox(nnr,3)~=zlow   % Plane
x=xlow
            node1=nnr;
            j=0;
            for i=1:nnodesbox   % Loops over all nodes to find the counter-clockwise nodes
around node nnr
                if abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-(nodesbox(nnr,3)-dz))<tol
                    node2=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
(nodesbox(nnr,2)-dy))<tol && abs(nodesbox(i,3)-(nodesbox(nnr,3)-dz))<tol
                    node3=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
(nodesbox(nnr,2)-dy))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node4=i;
                    j=j+1;
                end
                if j==3   % Terminates the search when the three counter-clockwise nodes are
found
```

```
                    break
                end
            end
            elemsbox(nnr,:)=[node1 node2 node3 node4];   % Stores connectivity for element nnr
            fprintf(felxlow,'%i %i %i %i\n',node1,node2,node3,node4);   % Stores connectivity
in .dat file
        elseif nodesbox(nnr,1)==xlim && nodesbox(nnr,2)~=ylim && nodesbox(nnr,3)~=zlim   %
Plane x=xlim
            node1=nnr;
            j=0;
            for i=1:nnodesbox
                if abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
(nodesbox(nnr,2)+dy))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node2=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
(nodesbox(nnr,2)+dy))<tol && abs(nodesbox(i,3)-(nodesbox(nnr,3)+dz))<tol
                    node3=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-(nodesbox(nnr,3)+dz))<tol
                    node4=i;
                    j=j+1;
                end
                if j==3
                    break
                end
            end
            elemsbox(nnr,:)=[node1 node2 node3 node4];
            fprintf(felxlim,'%i %i %i %i\n',node1,node2,node3,node4);
        elseif nodesbox(nnr,2)==ylow && nodesbox(nnr,1)~=xlim && nodesbox(nnr,3)~=zlim   %
Plane y=0
            node1=nnr;
            j=0;
            for i=1:nnodesbox
                if abs(nodesbox(i,1)-(nodesbox(nnr,1)+dx))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node2=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-(nodesbox(nnr,1)+dx))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-(nodesbox(nnr,3)+dz))<tol
                    node3=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-(nodesbox(nnr,3)+dz))<tol
                    node4=i;
                    j=j+1;
                end
                if j==3
                    break
                end
            end
            elemsbox(nnr,:)=[node1 node2 node3 node4];
            fprintf(felylow,'%i %i %i %i\n',node1,node2,node3,node4);
        elseif nodesbox(nnr,2)==ylim && nodesbox(nnr,1)~=xlow && nodesbox(nnr,3)~=zlow   %
Plane y=ylim
            node1=nnr;
            j=0;
            for i=1:nnodesbox
                if abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-(nodesbox(nnr,3)-dz))<tol
                    node2=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-(nodesbox(nnr,1)-dx))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-(nodesbox(nnr,3)-dz))<tol
                    node3=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-(nodesbox(nnr,1)-dx))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node4=i;
                    j=j+1;
                end
                if j==3
                    break
                end
            end
            elemsbox(nnr,:)=[node1 node2 node3 node4];
```

```
            fprintf(felylim,'%i %i %i %i\n',node1,node2,node3,node4);
        elseif nodesbox(nnr,3)==zlow && nodesbox(nnr,1)~=xlim && nodesbox(nnr,2)~=ylow   %
Plane z=0
            node1=nnr;
            j=0;
            for i=1:nnodesbox
                if abs(nodesbox(i,1)-(nodesbox(nnr,1)+dx))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node2=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-(nodesbox(nnr,1)+dx))<tol && abs(nodesbox(i,2)-
(nodesbox(nnr,2)-dy))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node3=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
(nodesbox(nnr,2)-dy))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node4=i;
                    j=j+1;
                end
                if j==3
                    break
                end
            end
            elemsbox(nnr,:)=[node1 node2 node3 node4];
            fprintf(felzlow,'%i %i %i %i\n',node1,node2,node3,node4);
        elseif nodesbox(nnr,3)==zlim && nodesbox(nnr,1)~=xlow && nodesbox(nnr,2)~=ylim   %
Plane z=zlim
            node1=nnr;
            j=0;
            for i=1:nnodesbox
                if abs(nodesbox(i,1)-nodesbox(nnr,1))<tol && abs(nodesbox(i,2)-
(nodesbox(nnr,2)+dy))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node2=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-(nodesbox(nnr,1)-dx))<tol && abs(nodesbox(i,2)-
(nodesbox(nnr,2)+dy))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node3=i;
                    j=j+1;
                elseif abs(nodesbox(i,1)-(nodesbox(nnr,1)-dx))<tol && abs(nodesbox(i,2)-
nodesbox(nnr,2))<tol && abs(nodesbox(i,3)-nodesbox(nnr,3))<tol
                    node4=i;
                    j=j+1;
                end
                if j==3
                    break
                end
            end
            elemsbox(nnr,:)=[node1 node2 node3 node4];
            fprintf(felzlim,'%i %i %i %i\n',node1,node2,node3,node4);
        end
    end
    fclose('all');   % Close the connectivity files
    elemsbox(find(ismember(elemsbox,[0 0 0 0],'rows')'==1),:)=[];   % Removal of zero lines
    nelemsbox=size(elemsbox,1);   % Number of box elements

    % Plot of the box grid
    scrsz = get(0,'ScreenSize');
    figure('Position',[1 scrsz(4)/2 scrsz(3)/2 0.8*scrsz(3)/2])
    view(3)
    set(gca,'XTickLabel',[])
    set(gca,'YTickLabel',[])
    set(gca,'ZTickLabel',[])
    grid on
    hold on
    for i=1:nelemsbox
        elemsboxplot=[elemsbox(i,:) elemsbox(i,1)];
        plot3(nodesbox(elemsboxplot,1),nodesbox(elemsboxplot,2),nodesbox(elemsboxplot,3),'k');
    end

%------ Cylinder-part ----------------------------------------------------

    if cylinder==1||cylinder==2   % Skip if cylinder = 0

        % Construction of cylinder-part of the nodal matrix
        nnr=1;
        for t=1:Ncir   % Circumferential loop
            xloc=ccx+ccr*cos(c(t));
```

```
            zloc=ccz+ccr*sin(c(t));
            for j=1:Naxi+1   % Axial loop per circumferential position
                nodescil(nnr,:)=[xloc ccy-0.5*ccl+a(j) zloc c(t) 0];
                nnr=nnr+1;
            end
            for k=2:Nrad   % Radial loop (sides) per circumferential loop
                nodescil(nnr,:)=[ccx+r(k)*cos(c(t)) ccy-0.5*ccl ccz+r(k)*sin(c(t)) c(t) r(k)];
% Left side
                nnr=nnr+1;
                nodescil(nnr,:)=[ccx+r(k)*cos(c(t)) ccy+0.5*ccl ccz+r(k)*sin(c(t)) c(t) r(k)];
% Right side
                nnr=nnr+1;
            end
        end
        nodescil(nnr,:)=[ccx ccy-0.5*ccl ccz 0 0];   % Node on left centrepoint
        nrleftnode=nnr;   % For element creation purposes
        nnr=nnr+1;
        nodescil(nnr,:)=[ccx ccy+0.5*ccl ccz 0 0];   % Node on right centrepoint
        nrrightnode=nnr;   % For element creation purposes
        nnodescil=size(nodescil,1);   % Number of cylinder nodes

        % Construction of cylinder-part of the element matrix
        felcyl = fopen('postproc_conn_cyl.dat','w');   % Connectivity file for the purpose of
tecplot postprocessing
        for nnr=1:nnodescil
            if abs((nodescil(nnr,1)-ccx)^2+(nodescil(nnr,3)-ccz)^2-ccr^2)<tol &&
abs(nodescil(nnr,2)-(ccy+0.5*ccl))>tol   % Circumference of cylinder
                node1=nnr+nnodesbox;
                j=0;
                for i=1:nnodescil
                    if abs(nodescil(i,1)-(ccx+ccr*cos(nodescil(nnr,4)+dc)))<tol &&
abs(nodescil(i,2)-nodescil(nnr,2))<tol && abs(nodescil(i,3)-
(ccz+ccr*sin(nodescil(nnr,4)+dc)))<tol
                        node2=i+nnodesbox;
                        j=j+1;
                    elseif abs(nodescil(i,1)-(ccx+ccr*cos(nodescil(nnr,4)+dc)))<tol &&
abs(nodescil(i,2)-(nodescil(nnr,2)+da))<tol && abs(nodescil(i,3)-
(ccz+ccr*sin(nodescil(nnr,4)+dc)))<tol
                        node3=i+nnodesbox;
                        j=j+1;
                    elseif abs(nodescil(i,1)-nodescil(nnr,1))<tol && abs(nodescil(i,2)-
(nodescil(nnr,2)+da))<tol && abs(nodescil(i,3)-nodescil(nnr,3))<tol
                        node4=i+nnodesbox;
                        j=j+1;
                    end
                    if j==3
                        break
                    end
                end
                elemscil(nnr,:)=[node1 node2 node3 node4];
                fprintf(felcyl,'%i %i %i %i\n',node1,node2,node3,node4);
            elseif abs((nodescil(nnr,1)-ccx)^2+(nodescil(nnr,3)-ccz)^2-ccr^2)>tol &&
nodescil(nnr,5)~=0   % Sides of the cylinder (except nodes on r=0)
                node1=nnr+nnodesbox;
                j=0;
                if abs(nodescil(nnr,2)-(ccy-0.5*ccl))<tol   % Left side of cylinder
                    for i=1:nnodescil
                        if abs(nodescil(i,2)-nodescil(nnr,2))<tol
                            if abs(nodescil(i,1)-
(ccx+nodescil(nnr,5)*cos(nodescil(nnr,4)+dc)))<tol && abs(nodescil(i,3)-
(ccx+nodescil(nnr,5)*sin(nodescil(nnr,4)+dc)))<tol
                                node2=i+nnodesbox;
                                j=j+1;
                            elseif abs(nodescil(i,1)-
(ccx+(nodescil(nnr,5)+dr)*cos(nodescil(nnr,4)+dc)))<tol && abs(nodescil(i,3)-
(ccx+(nodescil(nnr,5)+dr)*sin(nodescil(nnr,4)+dc)))<tol
                                node3=i+nnodesbox;
                                j=j+1;
                            elseif abs(nodescil(i,1)-
(ccx+(nodescil(nnr,5)+dr)*cos(nodescil(nnr,4))))<tol && abs(nodescil(i,3)-
(ccx+(nodescil(nnr,5)+dr)*sin(nodescil(nnr,4))))<tol
                                node4=i+nnodesbox;
                                j=j+1;
                            end
                            if j==3
                                break
                            end
```

```
                        end
                    end
                elseif abs(nodescil(nnr,2)-(ccy+0.5*ccl))<tol   % Right side of cylinder
                    for i=1:nnodescil
                        if abs(nodescil(i,2)-nodescil(nnr,2))<tol
                            if abs(nodescil(i,1)-(ccx+nodescil(nnr,5)*cos(nodescil(nnr,4)-
dc)))<tol && abs(nodescil(i,3)-(ccx+nodescil(nnr,5)*sin(nodescil(nnr,4)-dc)))<tol
                                node2=i+nnodesbox;
                                j=j+1;
                            elseif abs(nodescil(i,1)-
(ccx+(nodescil(nnr,5)+dr)*cos(nodescil(nnr,4)-dc)))<tol && abs(nodescil(i,3)-
(ccx+(nodescil(nnr,5)+dr)*sin(nodescil(nnr,4)-dc)))<tol
                                node3=i+nnodesbox;
                                j=j+1;
                            elseif abs(nodescil(i,1)-
(ccx+(nodescil(nnr,5)+dr)*cos(nodescil(nnr,4))))<tol && abs(nodescil(i,3)-
(ccx+(nodescil(nnr,5)+dr)*sin(nodescil(nnr,4))))<tol
                                node4=i+nnodesbox;
                                j=j+1;
                            end
                            if j==3
                                break
                            end
                        end
                    end
                end
                elemscil(nnr,:)=[node1 node2 node3 node4];
                fprintf(felcyl,'%i %i %i %i\n',node1,node2,node3,node4);
            end
        end
        nelemscil=size(elemscil,1);   % Intermediate number of elements in order to get
correct numbering for the last circles on left and right sides (see below)
        for nnr=1:nnodescil
            if abs(nodescil(nnr,5)-dr)<tol   % Nodes on r=dr
                node1=nnr+nnodesbox;
                if abs(nodescil(nnr,2)-(ccy-0.5*ccl))<tol   % Left side of the cylinder
                    for i=1:nnodescil
                        if abs(nodescil(i,1)-(ccx+nodescil(nnr,5)*cos(nodescil(nnr,4)-
dc)))<tol && abs(nodescil(i,2)-nodescil(nnr,2))<tol && abs(nodescil(i,3)-
(ccx+nodescil(nnr,5)*sin(nodescil(nnr,4)-dc)))<tol
                            node2=i+nnodesbox;
                            node3=nrleftnode+nnodesbox;
                            node4=node3;
                            break
                        end
                    end
                else   % Right side of the cylinder
                    for i=1:nnodescil
                        if abs(nodescil(i,1)-
(ccx+nodescil(nnr,5)*cos(nodescil(nnr,4)+dc)))<tol && abs(nodescil(i,2)-nodescil(nnr,2))<tol
&& abs(nodescil(i,3)-(ccx+nodescil(nnr,5)*sin(nodescil(nnr,4)+dc)))<tol
                            node2=i+nnodesbox;
                            node3=nrrightnode+nnodesbox;
                            node4=node3;
                            break
                        end
                    end
                end
                elemscil(nelemscil+1,:)=[node1 node2 node3 node4];
                fprintf(felcyl,'%i %i %i %i\n',node1,node2,node3,node4);
                nelemscil=nelemscil+1;
            end
        end
        fclose('all');   % Close the connectivity file
        elemscil(find(ismember(elemscil,[0 0 0 0],'rows')'==1),:)=[];   % Removal of zero
lines
        nelemscil=size(elemscil,1);   % Number of cylinder elements

        % Plot of the cylinder grid
        figure('Position',[1 scrsz(4)/2 scrsz(3)/2 0.8*scrsz(3)/2])
        view(3)
        set(gca,'XTickLabel',[])
        set(gca,'YTickLabel',[])
        set(gca,'ZTickLabel',[])
        grid on
        hold on
        for i=1:nelemscil
```

```
            elemscilplot=[elemscil(i,:)-nnodesbox elemscil(i,1)-nnodesbox];

plot3(nodescil(elemscilplot,1),nodescil(elemscilplot,2),nodescil(elemscilplot,3),'k');
        end

        % Assembly of global nodal and element matrices
        if cylinder==1   % Including cylinder
            nodes=[nodesbox;nodescil(:,1:3)];
            elems=[elemsbox;elemscil];
        else   % Excluding cylinder (implemented later as field points)
            nodes=nodesbox;
            elems=elemsbox;
        end

    else   % Excluding cylinder
        nodes=nodesbox;
        elems=elemsbox;
    end
    nnodes=size(nodes,1);   % Number of nodes
    nelems=size(elems,1);   % Number of elements

%------ Fix ---------------------------------------------------------------

    % Fix because of the error in the creation of the cylinder-part of the mesh
    % (it breaks down with certain input).

    % Description: The input is given in inches (see parameters.m) with a
    % shifted x-axis. Here the output is transformed back to the intended
    % origin and then converted to meters (1 inch = 0.0254 m)

    nodes(:,1)=nodes(:,1)-xlow;
    nodes(:,2)=nodes(:,2)-ylow;
    nodes(:,3)=nodes(:,3)-zlow;
    nodes=inch*nodes;

%------ Creation of field points matrix if cylinder = 2 (including fix) ---

    if cylinder==2
        field=nodescil(:,1:3);
        field(:,1)=field(:,1)-xlow;
        field(:,2)=field(:,2)-ylow;
        field(:,3)=field(:,3)-zlow;
        field=inch*field;
    end

%------ Microphone/Speaker plate ------------------------------------------

    if micplate==1   % Skips if micplate = 0

        % Relocation of nodes on plane x=xlim to incorporated microphone nodes
        fid=fopen('mic_pattern_low_frequency.txt','r');   % Open microphone pattern file
        nodesmic=fscanf(fid,'%g',[3,inf])';   % Store miccoordinates [y,z,x]
        fclose(fid);   % Close microphone pattern file
        nodesmic(:,3)=[];   % Delete 3rd column (x=0)
        nodesmic(:,1)=nodesmic(:,1)+0.5*(ylim-ylow)*inch;   % Transform y-miccoordinates to
global y-axis
        nodesmic(:,1)=nodesmic(:,1)-5*inch/16;   % Move plate to exact location in
experimental setup
        nodesmic(:,2)=nodesmic(:,2)*-1+0.5*(zlim-zlow)*inch;   % Transform z-miccoordinates to
global z-axis
        nodesmic(:,2)=nodesmic(:,2)-3*inch/16;   % Move plate to exact location in
experimental setup
        nnodesmic=size(nodesmic,1);   % Number of micnodes
        for nnr=1:nnodesmic
            j=1;
            for i=1:nnodes   % Calculates the distance from a micnode to each node on the
plane x=xlim and
                % stores it in local array dist(nnodes)
                if abs(nodes(i,1)-(xlim-xlow)*inch)<tol
                    dist(j,:)=[i norm([nodes(i,2)-nodesmic(nnr,1) nodes(i,3)-
nodesmic(nnr,2)])];
                    j=j+1;
                end
            end
            displnodes(nnr)=dist(find(ismember(dist(:,2),min(dist(:,2)))'==1),1);   % Finds
closest node
```

```
            nodes(displnodes(nnr),2)=nodesmic(nnr,1);   % Relocates closest node to microphone
y-position
            nodes(displnodes(nnr),3)=nodesmic(nnr,2);   % Relocates closest node to microphone
z-position
        end
        if max(size(unique(displnodes)))~=nnr   % Checks for multiple displacements of the
same node
            disp('ERROR: One or more nodes displaced multiple times');
            return
        end
        fid=fopen('postproc_micnodenrs.dat','w');   % Write micnodenrs for postprocessing
purposes
        for i=1:nnodesmic
            fprintf(fid,'%i\n',displnodes(i));
        end
        fclose(fid);

        % Plot of the microphone-plate grid (x=xlim)
        figure('Position',[1 scrsz(4)/2 scrsz(3)/2 0.8*scrsz(3)/2])
        set(gca,'XTickLabel',[])
        set(gca,'YTickLabel',[])
        hold on
        for i=1:nelems
            if abs(nodes(elems(i,1),1)-(xlim-xlow)*inch)<tol
                elemsplot=[elems(i,:) elems(i,1)];
                plot(nodes(elemsplot,2),nodes(elemsplot,3),'k');
            end
        end
        plot(nodesmic(:,1),nodesmic(:,2),'o');
    end

%------ Plot of the complete grid ----------------------------------------

    figure('Position',[1 scrsz(4)/2 scrsz(3)/2 0.8*scrsz(3)/2])
    view(3)
    set(gca,'XTickLabel',[])
    set(gca,'YTickLabel',[])
    set(gca,'ZTickLabel',[])
    grid on
    hold on
    for i=1:nelems
        elemsplot=[elems(i,:) elems(i,1)];
        plot3(nodes(elemsplot,1),nodes(elemsplot,2),nodes(elemsplot,3),'k');
    end
end
```

## A1.4 sim_postprocessor.m

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% SIM_POSTPROCESSOR
%-----------------------------------------------------------------------------
%
% Author:       T.J. van der Meer
%
% Modified:     14-5-2012
%
% Description:  Reads in the pressure solution of the BEM-model and creates
%               as output the inputfile for Tecplot and a datafile with all
%               nodal results.
%
% Inputs:       - parameters.m
%               - output_node*.dat          (solution from BEM-model)
%               - posproc_conn_*.dat        (element connectivity from gridgenerator)
%               - posproc_micnodenrs.dat%   (microphone element numbers from gridgenerator)
%
% Outputs:      - sim_tecplot_speaker*.dat
%               - sim_solution_speaker*.dat
%
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%------ Initialization --------------------------------------------------

clear
clc
```

```
%------ Load problem parameters ------------------------------------------

run('parameters');

%------ Test cases (single source) ---------------------------------------

if simulation==5||simulation==6

    % Read in the pressure solution
    fid=fopen(outputfilename,'r');   % Open pressure solution
    fgetl(fid);   % Discard row with variable names
    p=fscanf(fid,'%g',[4,inf])';   % Store solution [nnr re(p) im(p) ampl]
    if cylinder==2   % Reads in field point solution if cylinder = 2
        fgetl(fid);   % Discard title row
        fgetl(fid);   % Discard row with variable names
        pfield=fscanf(fid,'%g',[6,inf])';   % Store field point pressure [x y z re(p) im(p)
ampl]
    end
    fclose(fid);   % Close pressure solution
    np=size(p,1);   % Number of nodal solutions

    % Read in nodal coordinates
    fid=fopen(inputfilename,'r');   % Open nodal coordinates
    title=fgetl(fid);   % Read simulation title
    fgetl(fid);   % Discard 5 rows (general information)
    fgetl(fid);
    fgetl(fid);
    fgetl(fid);
    fgetl(fid);
    nnodes=fscanf(fid,'%i',[1 1]);   % Read number of nodes
    if nnodes~=np   % Checks for simulation problems
        disp('Error: Number of nodes in input and output do not match');
        return
    end
    nodes=fscanf(fid,'%g,%g,%g,%g',[4,nnodes])';   % Store nodes [nnr x y z]
    fgetl(fid);   % Discard blank row
    fgetl(fid);   % Discard title row
    nelems=fscanf(fid,'%i',[1 1]);   % Read number of elements
    elems=fscanf(fid,'%g,%g,%g,%g,%g',[5,nelems])';   % Store elements [enr node1 node2 node3
node4]
    fclose(fid);   % Close nodal coordinates

    % Combine nodal coordinates and pressure solution
    phase=atan2(p(:,3),p(:,2));   % Compute phase from -pi to pi
    if cylinder==2   % Add fielpoint values to the phase array if cylinder = 2
        fieldphase=atan2(pfield(:,5),pfield(:,4));
        xcoords=[nodes(:,2);pfield(:,1)];
        ycoords=[nodes(:,3);pfield(:,2)];
        zcoords=[nodes(:,4);pfield(:,3)];
        phase=[phase;fieldphase];
        ampl=[p(:,4);pfield(:,6)];
    else
        xcoords=nodes(:,2);
        ycoords=nodes(:,3);
        zcoords=nodes(:,4);
        ampl=p(:,4);
    end
    nodalp=[xcoords ycoords zcoords phase ampl];   % Nodal solutions [x y z phase amplitude]
    nnodes=size(nodalp,1);   % Number of nodal solutions

    % Fix phase values
    for i=1:nnodes   % Fix phase to range from -2*pi to 0
        if phase(i)>0
            phase(i)=phase(i)-2*pi;
        end
    end
    nodalp(:,4)=phase;

    % Write general info
    if cylinder==1
        model='Included';
    else
        model='Excluded';
    end
    input_tecplot=sprintf('sim_tecplot_%s.dat',title);
    ftec=fopen(input_tecplot,'w');   % Open inputfile
    solution_file=sprintf('sim_solution_%s.dat',title);
```

```
    fsol=fopen(solution_file,'w');   % Open solutionfile
    fprintf(ftec,'TITLE="%.0fHz, Model %s, %s"\nVARIABLES="X" "Y" "Z" "PHASE"
"AMPLI"\n',freq1,model,title);

    % Write zone X_low
    fid=fopen('postproc_conn_xlow.dat','r');   % Open connectivity file (with GLOBAL node
numbering!)
    conn=fscanf(fid,'%g',[4,inf])';   % Store connectivity
    fclose(fid);   % Close connectivity file
    fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT ET=QUADRILATERAL\n','X_low',(N+1)*(N+1),N*N);
    j=1;
    for nnr=1:nnodes
        if nodalp(nnr,1)<tol   % Store every node on the plane in the inputfile
            fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            for i=1:N*N   % Adjust connectivity info to local values (i.e. starts from 1 in
every zone)
                for k=1:4
                    if conn(i,k)==nnr
                        conn(i,k)=j;
                    end
                end
            end
            j=j+1;
        end
    end
    for i=1:N*N   % Store local connectivity info in the inputfile
        fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
    end

    % Write zone X_lim
    fid=fopen('postproc_conn_xlim.dat','r');
    conn=fscanf(fid,'%g',[4,inf])';
    fclose(fid);
    fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT ET=QUADRILATERAL\n','X_lim',(N+1)*(N+1),N*N);
    j=1;
    for nnr=1:nnodes
        if abs(nodalp(nnr,1)-(xlim-xlow)*inch)<tol
            fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            for i=1:N*N
                for k=1:4
                    if conn(i,k)==nnr
                        conn(i,k)=j;
                    end
                end
            end
            j=j+1;
        end
    end
    for i=1:N*N
        fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
    end

    % Write zone Y_low
    fid=fopen('postproc_conn_ylow.dat','r');
    conn=fscanf(fid,'%g',[4,inf])';
    fclose(fid);
    fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT ET=QUADRILATERAL\n','Y_low',(N+1)*(N+1),N*N);
    j=1;
    for nnr=1:nnodes
        if nodalp(nnr,2)<tol
            fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            for i=1:N*N
                for k=1:4
                    if conn(i,k)==nnr
                        conn(i,k)=j;
                    end
                end
            end
```

```
            j=j+1;
        end
    end
    for i=1:N*N
        fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
    end

    % Write zone Y_lim
    fid=fopen('postproc_conn_ylim.dat','r');
    conn=fscanf(fid,'%g',[4,inf])';
    fclose(fid);
    fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT ET=QUADRILATERAL\n','Y_lim',(N+1)*(N+1),N*N);
    j=1;
    for nnr=1:nnodes
        if abs(nodalp(nnr,2)-(ylim-ylow)*inch)<tol
            fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            for i=1:N*N
                for k=1:4
                    if conn(i,k)==nnr
                        conn(i,k)=j;
                    end
                end
            end
            j=j+1;
        end
    end
    for i=1:N*N
        fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
    end

    % Write zone Z_low
    fid=fopen('postproc_conn_zlow.dat','r');
    conn=fscanf(fid,'%g',[4,inf])';
    fclose(fid);
    fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT ET=QUADRILATERAL\n','Z_low',(N+1)*(N+1),N*N);
    j=1;
    for nnr=1:nnodes
        if nodalp(nnr,3)<tol
            fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            for i=1:N*N
                for k=1:4
                    if conn(i,k)==nnr
                        conn(i,k)=j;
                    end
                end
            end
            j=j+1;
        end
    end
    for i=1:N*N
        fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
    end

    % Write zone Z_lim
    fid=fopen('postproc_conn_zlim.dat','r');
    conn=fscanf(fid,'%g',[4,inf])';
    fclose(fid);
    fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT ET=QUADRILATERAL\n','Z_lim',(N+1)*(N+1),N*N);
    j=1;
    for nnr=1:nnodes
        if abs(nodalp(nnr,3)-(zlim-zlow)*inch)<tol
            fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
            for i=1:N*N
                for k=1:4
                    if conn(i,k)==nnr
                        conn(i,k)=j;
                    end
                end
            end
```

```
            end
            j=j+1;
        end
    end
    for i=1:N*N
        fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
    end

    % Write zone cylinder
    if cylinder==1||cylinder==2   % Skips if cylinder = 0
        fid=fopen('postproc_conn_cyl.dat','r');
        conn=fscanf(fid,'%g',[4,inf])';
        fclose(fid);
        fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT
ET=QUADRILATERAL\n','Cylinder',2*Nrad*Ncir+(Naxi-1)*Ncir+2,2*Nrad*Ncir+Naxi*Ncir);
        j=1;
        for nnr=1:nnodes
            if nodalp(nnr,1)>tol && abs(nodalp(nnr,1)-(xlim-xlow)*inch)>tol &&
nodalp(nnr,2)>tol && abs(nodalp(nnr,2)-(ylim-ylow)*inch)>tol && nodalp(nnr,3)>tol &&
abs(nodalp(nnr,3)-(zlim-zlow)*inch)>tol
                fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                for i=1:2*Nrad*Ncir+Naxi*Ncir
                    for k=1:4
                        if conn(i,k)==nnr
                            conn(i,k)=j;
                        end
                    end
                end
                j=j+1;
            end
        end
        for i=1:2*Nrad*Ncir+Naxi*Ncir
            fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
        end
    end
    fclose(ftec);   % Close inputfile
    fclose(fsol);   % Close solutionfile

%------ All other simulations (32 source array) ---------------------------
else

    % Read in microphone nodenumbers
    fid=fopen('postproc_micnodenrs.dat','r');   % Open micnodenrs
    micnodenrs=fscanf(fid,'%i',[1,inf])';
    fclose(fid);   % Close micnodenrs

    for mic=1:32

        % Read in the pressure solution
        pfile=sprintf('output_node%2.2d.dat',mic);
        fid=fopen(pfile,'r');   % Open pressure solution
        fgetl(fid);   % Discard row with variable names
        p=fscanf(fid,'%g',[4,inf])';   % Store solution [nnr re(p) im(p) ampl]
        if cylinder==2   % Reads in field point solution if cylinder = 2
            fgetl(fid);   % Discard title row
            fgetl(fid);   % Discard row with variable names
            pfield=fscanf(fid,'%g',[6,inf])';   % Store field point pressure [x y z re(p)
im(p) ampl]
        end
        fclose(fid);   % Close pressure solution
        np=size(p,1);   % Number of nodal solutions

        % Read in nodal coordinates
        nfile=sprintf('input_node%2.2d.dat',mic);
        fid=fopen(nfile,'r');   % Open nodal coordinates
        title=fgetl(fid);   % Read simulation title
        fgetl(fid);   % Discard 5 rows (general information)
        fgetl(fid);
        fgetl(fid);
        fgetl(fid);
        fgetl(fid);
        nnodes=fscanf(fid,'%i',[1 1]);   % Number of nodes
        if nnodes~=np   % Checks for simulation problems
            disp('Error: Number of nodes in input and output do not match');
```

```
            return
        end
        nodes=fscanf(fid,'%g,%g,%g,%g',[4,nnodes])';   % Store nodes [nnr x y z]
        fgetl(fid);   % Discard blank row
        fgetl(fid);   % Discard title row
        nelems=fscanf(fid,'%i',[1 1]);   % Number of elements
        elems=fscanf(fid,'%g,%g,%g,%g,%g',[5,nelems])';   % Store elements [enr node1 node2
node3 node4]
        fclose(fid);   % Close nodal coordinates

        % Combine nodal coordinates and pressure solution
        phase=atan2(p(:,3),p(:,2));   % Compute phase from -pi to pi
        if cylinder==2   % Add fielpoint values to the phase array if cylinder = 2
            fieldphase=atan2(pfield(:,5),pfield(:,4));
            xcoords=[nodes(:,2);pfield(:,1)];
            ycoords=[nodes(:,3);pfield(:,2)];
            zcoords=[nodes(:,4);pfield(:,3)];
            phase=[phase;fieldphase];
            ampl=[p(:,4);pfield(:,6)];
        else
            xcoords=nodes(:,2);
            ycoords=nodes(:,3);
            zcoords=nodes(:,4);
            ampl=p(:,4);
        end
        nodalp=[xcoords ycoords zcoords phase ampl];   % Nodal solutions [x y z phase
amplitude]
        nnodes=size(nodalp,1);   % Number of nodal solutions

        % Fix phase values
        for i=1:nnodes   % Fix phase to range from -2*pi to 0
            if phase(i)>0
                phase(i)=phase(i)-2*pi;
            end
        end
        nodalp(:,4)=phase;

        % Write general info
        input_tecplot=sprintf('sim_tecplot_speaker%2.2d.dat',mic);
        ftec=fopen(input_tecplot,'w');   % Open inputfile
        solution_file=sprintf('sim_solution_speaker%2.2d.dat',mic);
        fsol=fopen(solution_file,'w');   % Open solutionfile
        fprintf(ftec,'TITLE="Case %i, Simulated"\nVARIABLES="X" "Y" "Z" "PHASE"
"AMPLI"\n',simulation);

        % Write zone X_low
        fid=fopen('postproc_conn_xlow.dat','r');
        conn=fscanf(fid,'%g',[4,inf])';
        fclose(fid);
        fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT
ET=QUADRILATERAL\n','X_low',(N+1)*(N+1),N*N);
        j=1;
        for nnr=1:nnodes
            if nodalp(nnr,1)<tol
                fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                for i=1:N*N
                    for k=1:4
                        if conn(i,k)==nnr
                            conn(i,k)=j;
                        end
                    end
                end
                j=j+1;
            end
        end
        for i=1:N*N
            fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
        end

        % Write zone X_lim
        fid=fopen('postproc_conn_xlim.dat','r');
        conn=fscanf(fid,'%g',[4,inf])';
        fclose(fid);
```

```
        fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT
ET=QUADRILATERAL\n','X_lim',(N+1)*(N+1),N*N);
        j=1;
        for nnr=1:nnodes
            if abs(nodalp(nnr,1)-(xlim-xlow)*inch)<tol
                fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                for i=1:N*N
                    for k=1:4
                        if conn(i,k)==nnr
                            conn(i,k)=j;
                        end
                    end
                end
                j=j+1;
            end
        end
        for i=1:N*N
            fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
        end

        % Write zone Y_low
        fid=fopen('postproc_conn_ylow.dat','r');
        conn=fscanf(fid,'%g',[4,inf])';
        fclose(fid);
        fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT
ET=QUADRILATERAL\n','Y_low',(N+1)*(N+1),N*N);
        j=1;
        for nnr=1:nnodes
            if nodalp(nnr,2)<tol
                fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                for i=1:N*N
                    for k=1:4
                        if conn(i,k)==nnr
                            conn(i,k)=j;
                        end
                    end
                end
                j=j+1;
            end
        end
        for i=1:N*N
            fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
        end

        % Write zone Y_lim
        fid=fopen('postproc_conn_ylim.dat','r');
        conn=fscanf(fid,'%g',[4,inf])';
        fclose(fid);
        fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT
ET=QUADRILATERAL\n','Y_lim',(N+1)*(N+1),N*N);
        j=1;
        for nnr=1:nnodes
            if abs(nodalp(nnr,2)-(ylim-ylow)*inch)<tol
                fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                for i=1:N*N
                    for k=1:4
                        if conn(i,k)==nnr
                            conn(i,k)=j;
                        end
                    end
                end
                j=j+1;
            end
        end
        for i=1:N*N
            fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
        end
```

```
        % Write zone Z_low
        fid=fopen('postproc_conn_zlow.dat','r');
        conn=fscanf(fid,'%g',[4,inf])';
        fclose(fid);
        fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT
ET=QUADRILATERAL\n','Z_low',(N+1)*(N+1),N*N);
        j=1;
        for nnr=1:nnodes
            if nodalp(nnr,3)<tol
                fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                for i=1:N*N
                    for k=1:4
                        if conn(i,k)==nnr
                            conn(i,k)=j;
                        end
                    end
                end
                j=j+1;
            end
        end
        for i=1:N*N
            fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
        end

        % Write zone Z_lim
        fid=fopen('postproc_conn_zlim.dat','r');
        conn=fscanf(fid,'%g',[4,inf])';
        fclose(fid);
        fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT
ET=QUADRILATERAL\n','Z_lim',(N+1)*(N+1),N*N);
        j=1;
        for nnr=1:nnodes
            if abs(nodalp(nnr,3)-(zlim-zlow)*inch)<tol
                fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                for i=1:N*N
                    for k=1:4
                        if conn(i,k)==nnr
                            conn(i,k)=j;
                        end
                    end
                end
                j=j+1;
            end
        end
        for i=1:N*N
            fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
        end

        % Write zone cylinder
        if cylinder==1||cylinder==2   % Skips if cylinder = 0
            fid=fopen('postproc_conn_cyl.dat','r');
            conn=fscanf(fid,'%g',[4,inf])';
            fclose(fid);
            fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT
ET=QUADRILATERAL\n','Cylinder',2*Nrad*Ncir+(Naxi-1)*Ncir+2,2*Nrad*Ncir+Naxi*Ncir);
            j=1;
            for nnr=1:nnodes
                if nodalp(nnr,1)>tol && abs(nodalp(nnr,1)-(xlim-xlow)*inch)>tol &&
nodalp(nnr,2)>tol && abs(nodalp(nnr,2)-(ylim-ylow)*inch)>tol && nodalp(nnr,3)>tol &&
abs(nodalp(nnr,3)-(zlim-zlow)*inch)>tol
                    fprintf(ftec,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                    fprintf(fsol,'%f %f %f %f
%f\n',nodalp(nnr,1),nodalp(nnr,2),nodalp(nnr,3),nodalp(nnr,4),nodalp(nnr,5));
                    for i=1:2*Nrad*Ncir+Naxi*Ncir
                        for k=1:4
                            if conn(i,k)==nnr
                                conn(i,k)=j;
                            end
                        end
                    end
```

```
                    j=j+1;
                end
            end
            for i=1:2*Nrad*Ncir+Naxi*Ncir
                fprintf(ftec,'%i %i %i %i\n',conn(i,1),conn(i,2),conn(i,3),conn(i,4));
            end
        end
        fclose(ftec);   % Close inputfile
        fclose(fsol);   % Close solutionfile
    end
end
```

## A1.5 exp_preprocessor.m

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% GEN_POSTPROCESSOR
%-----------------------------------------------------------------------------
%
% Author:       T.J. van der Meer
%
% Modified:     23-5-2012
%
% Description:  Creates textfile with coordinates of the microphones on the
%               cylinder, waveform-signal (0,0,0), microphones on the walls
%               and speakers in the door (up to 63 signals). Also creates
%               the Tecplot grid for the experimental output.
%
% Inputs:       - parameters.m
%
% Outputs:      - exp_mic_locations_*.txt       (transducer locations for testing purposes)
%               - exp_cylinder_mic_nodes.dat    (microphone nodes for postprocessor)
%               - exp_sides_mic_nodes.dat       (microphone nodes for postprocessor)
%
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%------ Load problem parameters -----------------------------------------

clear
clc
run('parameters');
side=(xlim-xlow)*inch;   % Convert relevant dimensions to meters
ccx=(ccx-20.7)*inch;
ccy=ccy*inch;
ccz=ccz*inch;
ccl=ccl*inch;
ccr=ccr*inch;
alpha=2*pi/12;

%------ Initialization --------------------------------------------------

coords=zeros(64,3);   % Based on 64 channels in Cartesian coordinates
nodes=zeros(31*8,3);   % Based on 31 cylinder microphones times 8 rotations, all in Cartesian
coordinates
nnodes=size(nodes,1);
elems=zeros(24*8+4*8,4);   % See report for calculation of number of elements
nelems=size(elems,1);
fconn=fopen('exp_cylinder_mic_nodes.dat','w');

%------ Write the coordinates -------------------------------------------

k=1;
for angle=0:pi/8:7*pi/8
    j=1;

    % Write first spiral
    for i=0:12
        xloc=ccx+ccr*cos(angle+i*alpha);
        yloc=ccy-0.5*ccl+4.5*inch+i*inch;
        zloc=ccz+ccr*sin(angle+i*alpha);
        coords(j,:)=[xloc yloc zloc];
        fprintf(fconn,'%f %f %f\n',coords(j,1),coords(j,2),coords(j,3));
        nodes(k,:)=coords(j,:);
        j=j+1;
        k=k+1;
    end
```

```
% Write second spiral
for i=0:12
    xloc=ccx+ccr*cos(pi+angle+i*alpha);
    yloc=ccy-0.5*ccl+4.5*inch+i*inch;
    zloc=ccz+ccr*sin(pi+angle+i*alpha);
    coords(j,:)=[xloc yloc zloc];
    fprintf(fconn,'%f %f %f\n',coords(j,1),coords(j,2),coords(j,3));
    nodes(k,:)=coords(j,:);
    j=j+1;
    k=k+1;
end

% Write bottom plate
coords(j,:)=[ccx+2*ccr/3*cos(0.5*pi+angle) ccy+0.5*ccl ccz+2*ccr/3*sin(0.5*pi+angle)];
fprintf(fconn,'%f %f %f\n',coords(j,1),coords(j,2),coords(j,3));
nodes(k,:)=coords(j,:);
j=j+1;
k=k+1;
coords(j,:)=[ccx+ccr/3*cos(0.5*pi+angle) ccy+0.5*ccl ccz+ccr/3*sin(0.5*pi+angle)];
fprintf(fconn,'%f %f %f\n',coords(j,1),coords(j,2),coords(j,3));
nodes(k,:)=coords(j,:);
j=j+1;
k=k+1;
coords(j,:)=[ccx ccy+0.5*ccl ccz];
fprintf(fconn,'%f %f %f\n',coords(j,1),coords(j,2),coords(j,3));
nodes(k,:)=coords(j,:);
j=j+1;
k=k+1;
coords(j,:)=[ccx+ccr/3*cos(-0.5*pi+angle) ccy+0.5*ccl ccz+ccr/3*sin(-0.5*pi+angle)];
fprintf(fconn,'%f %f %f\n',coords(j,1),coords(j,2),coords(j,3));
nodes(k,:)=coords(j,:);
j=j+1;
k=k+1;
coords(j,:)=[ccx+2*ccr/3*cos(-0.5*pi+angle) ccy+0.5*ccl ccz+2*ccr/3*sin(-0.5*pi+angle)];
fprintf(fconn,'%f %f %f\n',coords(j,1),coords(j,2),coords(j,3));
nodes(k,:)=coords(j,:);
j=j+1;
k=k+1;

% Leave 32nd node zero (it's the waveform signal)
j=j+1;

% Write z=0
if walls==1   % Only relevant when wall is not covered with foam
    coords(j,:)=[0.5*side 0.75*side 0];
    coords(j+1,:)=[0.25*side 0.5*side 0];
    coords(j+2,:)=[0.5*side 0.5*side 0];
    coords(j+3,:)=[0.75*side 0.5*side 0];
    coords(j+4,:)=[0.5*side 0.25*side 0];
end
j=j+5;

% Write y=0
if walls==1   % Only relevant when wall is not covered with foam
    coords(j,:)=[0.5*side 0 0.25*side];
    coords(j+1,:)=[12*0.0254 0 0.5*side];
    coords(j+2,:)=[0.5*side 0 0.5*side];
    coords(j+3,:)=[0.75*side 0 0.5*side];
    coords(j+4,:)=[0.5*side 0 0.75*side];
end
j=j+5;

% Write x=0
if walls==1 || walls==3   % Only relevant when wall is not covered with foam
    coords(j,:)=[0 0.75*side 0.375*side];
    coords(j+1,:)=[0 0.75*side 0.5*side];
    coords(j+2,:)=[0 0.75*side 0.625*side];
    coords(j+3,:)=[0 0.625*side 0.25*side];
    coords(j+4,:)=[0 0.625*side 0.375*side];
    coords(j+5,:)=[0 0.625*side 0.5*side];
    coords(j+6,:)=[0 0.625*side 0.625*side];
    coords(j+7,:)=[0 0.625*side 0.75*side];
    coords(j+8,:)=[0 0.5*side 0.25*side];
    coords(j+9,:)=[0 0.5*side 0.375*side];
    coords(j+10,:)=[0 0.5*side 0.5*side];
    coords(j+11,:)=[0 0.5*side 0.625*side];
```

```
        coords(j+12,:)=[0 0.5*side 0.75*side];
        coords(j+13,:)=[0 0.375*side 0.25*side];
        coords(j+14,:)=[0 0.375*side 0.375*side];
        coords(j+15,:)=[0 0.375*side 0.5*side];
        coords(j+16,:)=[0 0.375*side 0.625*side];
        coords(j+17,:)=[0 0.375*side 0.75*side];
        coords(j+18,:)=[0 0.25*side 0.375*side];
        coords(j+19,:)=[0 0.25*side 0.5*side];
        coords(j+20,:)=[0 0.25*side 0.625*side];
    end
    j=j+21;

    % Plot microphone coordinates per rotation and write them into outputfile
    filename=sprintf('exp_mic_locations_%05.1fdegrees.txt',angle*180/pi);
    fid=fopen(filename,'w');
    figure
    view(3);
    grid on
    hold on
    axis([0 side 0 side 0 side]);
    for i=1:64
        plot3(coords(i,1),coords(i,2),coords(i,3),'ko');
        fprintf(fid,'%f,%f,%f\n',coords(i,1),coords(i,2),coords(i,3));
    end
    fclose(fid);
end

%------ Write coordinates of the microphones on the side ------------------
fmics=fopen('exp_sides_mic_nodes.dat','w');
for i=33:63
    fprintf(fmics,'%f %f %f\n',coords(i,1),coords(i,2),coords(i,3));
end
fclose(fmics);

%------ Create element connectivity ---------------------------------------
% NOTE: This is a very brute-force way of creating the connectivity, based
% on the knowledge of the node numbering, hence this is only applicable in
% this exact case.
k=1;
for i=0:7
    for j=1:12
        elems(k,1)=j+i*31;
        elems(k,2)=j+i*31+31;
        if elems(k,2)>nnodes
            elems(k,2)=elems(k,2)-8*31+13;
        end
        elems(k,3)=j+i*31+32;
        if elems(k,3)>nnodes
            elems(k,3)=elems(k,3)-8*31+13;
        end
        elems(k,4)=j+i*31+1;
        k=k+1;
    end
    for j=14:25
        elems(k,1)=j+i*31;
        elems(k,2)=j+i*31+31;
        if elems(k,2)>nnodes
            elems(k,2)=elems(k,2)-8*31-13;
        end
        elems(k,3)=j+i*31+32;
        if elems(k,3)>nnodes
            elems(k,3)=elems(k,3)-8*31-13;
        end
        elems(k,4)=j+i*31+1;
        k=k+1;
    end
end
for i=0:7
    elems(k,1)=27+i*31;
    elems(k,2)=27+i*31+31;
    if elems(k,2)>nnodes
        elems(k,2)=elems(k,2)-8*31+4;
    end
    elems(k,3)=28+i*31+31;
    if elems(k,3)>nnodes
        elems(k,3)=elems(k,3)-8*31+2;
    end
```

```
    elems(k,4)=28+i*31;
    k=k+1;
    elems(k,1)=28+i*31;
    elems(k,2)=28+i*31+31;
    if elems(k,2)>nnodes
        elems(k,2)=elems(k,2)-8*31+2;
    end
    elems(k,3)=29;
    elems(k,4)=29;
    k=k+1;
    elems(k,1)=30+i*31;
    elems(k,2)=30+i*31+31;
    if elems(k,2)>nnodes
        elems(k,2)=elems(k,2)-8*31-2;
    end
    elems(k,3)=29;
    elems(k,4)=29;
    k=k+1;
    elems(k,1)=31+i*31;
    elems(k,2)=31+i*31+31;
    if elems(k,2)>nnodes
        elems(k,2)=elems(k,2)-8*31-4;
    end
    elems(k,3)=30+i*31+31;
    if elems(k,3)>nnodes
        elems(k,3)=elems(k,3)-8*31-2;
    end
    elems(k,4)=30+i*31;
    k=k+1;
end

%------ Plot of the cylinder grid and write element connectivity ----------

figure
view(3);
grid on
hold on
for i=1:nelems
    elemsplot=[elems(i,:) elems(i,1)];
    plot3(nodes(elemsplot,1),nodes(elemsplot,2),nodes(elemsplot,3),'k');
    fprintf(fconn,'%i %i %i %i\n',elems(i,1),elems(i,2),elems(i,3),elems(i,4));
end
for i=1:248
    plot3(nodes(i,1),nodes(i,2),nodes(i,3),'ko');
end
fclose('all');
```

## A1.6 exp_postprocessor.m

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% EXP_POSTPROCESSOR
%-----------------------------------------------------------------------------
%
% Author:       T.J. van der Meer
%
% Modified:     31-5-2012
%
% Description:  Reads in the time-history of the signal at each microphone
%               on the cylinder, carries out a Fast Fourier Transform (FFT)
%               and creates as output the inputfile for Tecplot analysis.
%
% Inputs:       - parameters.m
%               - exp_cylinder_mic_nodes.dat    (microphone nodes for postprocessor)
%               - CASE*.dat                     (time-histories from experiment)
%
% Outputs:      - exp_tecplot_speaker*.dat
%
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%------ Load simulation parameters --------------------------------------

clear
clc
run('parameters');
```

```
%------ Initialization --------------------------------------------------

phase=zeros(31*8,1);
ampl=zeros(31*8,1);
freq_spectrum=0:sample_freq/fft_steps:sample_freq-sample_freq/fft_steps;
fourier_samples=floor(sample_freq/fft_steps);
array_entry=round(freq1/(sample_freq/fft_steps));

%------ Store nodal coordinates and elements ------------------------------

fid=fopen('exp_cylinder_mic_nodes.dat','r');
nodes=fscanf(fid,'%g',[3,31*8])';   % Nodal coordinates from preprocessor
nnodes=size(nodes,1);
elems=fscanf(fid,'%g',[4,inf])';   % Element connectivity from preprocessor
nelems=size(elems,1);
fclose(fid);

%------ Manipulation of experimental data, write Tecplot file -------------

for source=1:32
    if exist(sprintf('CASE%i_SP%2.2i_000.0_degrees.chans.dat',simulation,source),'file')==2
% Check for existence of experimental source data
        j=1;
        for angle=0:180/8:157.5   % Eight rotational datafiles

            % Read in time history

datafile=sprintf('CASE%i_SP%2.2i_%05.1f_degrees.chans.dat',simulation,source,angle);
            fid=fopen(datafile,'r');
            timehist=fread(fid,[sample_freq,inf],'float');
            fclose(fid);

            % Extract source-phase from waveform signal as reference
            fourier_data=zeros(fft_steps,1);
            for sample=1:fourier_samples
                fourier_data=fourier_data+fft(timehist((fourier_samples-
1)*fft_steps+1:fourier_samples*fft_steps,32),fft_steps);
            end
            fourier_data=fourier_data./fourier_samples;
            ref_phase=atan(imag(fourier_data)./real(fourier_data));

            % Test
%             figure(1)
%             plot(freq_spectrum,ref_phase);
%             ref_phase(array_entry)
%             return

            % Extract phase and amplitude from cylinder-microphones
            for mic=1:31
                fourier_data=zeros(fft_steps,1);
                for sample=1:fourier_samples
                    fourier_data=fourier_data+fft(timehist((fourier_samples-
1)*fft_steps+1:fourier_samples*fft_steps,mic),fft_steps);
                end
                fourier_data=fourier_data./fourier_samples;

phase(j)=atan(imag(fourier_data(array_entry))/real(fourier_data(array_entry)));
                ampl(j)=abs(fourier_data(array_entry));
                j=j+1;
            end
        end

%         % Fix phase values
%         for i=1:nnodes   % Fix phase to range from -2*pi to 0
%             if phase(i)>0
%                 phase(i)=phase(i)-2*pi;
%             elseif phase(i)<-2*pi
%                 phase(i)=phase(i)+2*pi;
%             end
%         end

        % Write Tecplot file
        input_tecplot=sprintf('exp_tecplot_speaker%2.2d.dat',source);
        ftec=fopen(input_tecplot,'w');   % Open inputfile
        fprintf(ftec,'TITLE="Case %i, Experimental"\nVARIABLES="X" "Y" "Z" "PHASE"
"AMPLI"\n',simulation);
```

```
        fprintf(ftec,'ZONE T=%s N=%i E=%i F=FEPOINT
ET=QUADRILATERAL\n','Cylinder',nnodes,nelems);
        for i=1:nnodes
            fprintf(ftec,'%f %f %f %f
%f\n',nodes(i,1),nodes(i,2),nodes(i,3),phase(i),ampl(i));
        end
        for i=1:nelems
            fprintf(ftec,'%i %i %i %i\n',elems(i,1),elems(i,2),elems(i,3),elems(i,4));
        end
        fclose('all');
    end
end
```