# SotonArray: Southampton University Wind Tunnel Microphone Array System Guide

Benjamin A. Fenech and Kenji Takeda[*]

February 2007

**Abstract**

This report shall serve as an introductory background to SotonArray, the microphone array system at the University of Southampton. It covers the practical aspects only; namely the instrumentation setup and the beamforming software. It aims to give a basic level of understanding of the system to the researcher who wants to use it, and especially to highlight both the capabilities and limitations of the system. Since the microphone array system is a new capability for the Aerodynamics and Flight Mechanics research group (AFM), the hardware and software is constantly being upgraded, and the details given here are correct only at the time of going to print.

[*]Corresponding author: ktakeda@soton.ac.uk

University of Southampton

ses School of Engineering Sciences

## Contents

# 1 Introduction

Carrying out acoustic measurements in closed-section hard-walled wind tunnels is a challenging task, given the nature of the measuring environment. Three major problems need to be addressed, namely the air flow creating turbulence over the microphone diaphragms, high background noise levels generated by the fan and the flow over the wind tunnel boundaries, and the hard walls giving rise to image noise sources. In most cases single microphone measurements can only be used to extract a very limited set of data. On the other hand an array of microphones can be used to yield useful information by making it act as a spatially-selective filter.[1] In this way a qualitative data set can be obtained to aid in source localisation; however accurate absolute levels are still not achievable. This is because of simplifying assumptions

[1]This processing technique is commonly referred to as beamforming.

in the beamforming analysis. Further limitations inherent in this processing technique include a poor resolution at low frequencies (typically below 1 kHz), and spatial noise at high frequencies.[2]

A microphone array system can be split up into hardware and software. The hardware is what actually measures the acoustic information and records it into a readily-accessible digital format. The software part involves the post processing of the measured data in order to obtain spatial noise distribution plots. The following sections will present a breakdown of the constituent parts of the system.

# 2 Microphone Array Hardware

Figure 1 shows a pictorial overview of a typical microphone array system. The array of microphones A is installed in the wind tunnel, typically flush-mounted in one of the walls of the test section. Signal conditioning occurs at the preamplifiers B, and the measured analog data is then converted to digital (C) and stored in a computer where post-processing is performed. In the following sections these four main stages are described in more detail. A more detailed schematic of the present array system at the School of Engineering Sciences (SES) can be found in Appendix A.

## 2.1 Microphone Array

### 2.1.1 Microphones

Traditionally, instrumentation-grade Type 1 condenser microphones from manufacturers such as Brüel and Kjær, Gras and PCB have been the primary choice for microphone array measurements. However the cost of these microphones is significant, and is often the limiting factor when deciding the channel-count of a particular system. It has been shown that arrays built using electret microphones (such as the Panasonic WM-61A/B), or MEMS microphones give similar results (within their operating frequency ranges[3]) as arrays employing instrumentation microphones[2], whereas their cost is around three orders of magnitude less. However in critical applications instrumentation microphones are still recommended (the cheaper microphones are more prone to errors and drift with changes in temperature, pressure and humidity). 1/4″ or 1/8″ instrumentation microphones should in theory also be used when the frequency of interest is higher than 20 kHz, for example

[2]For further details please refer to ref. [1].

[3]Typically up to 20 kHz for the electrets and 10 kHz for MEMS, as in 2005.

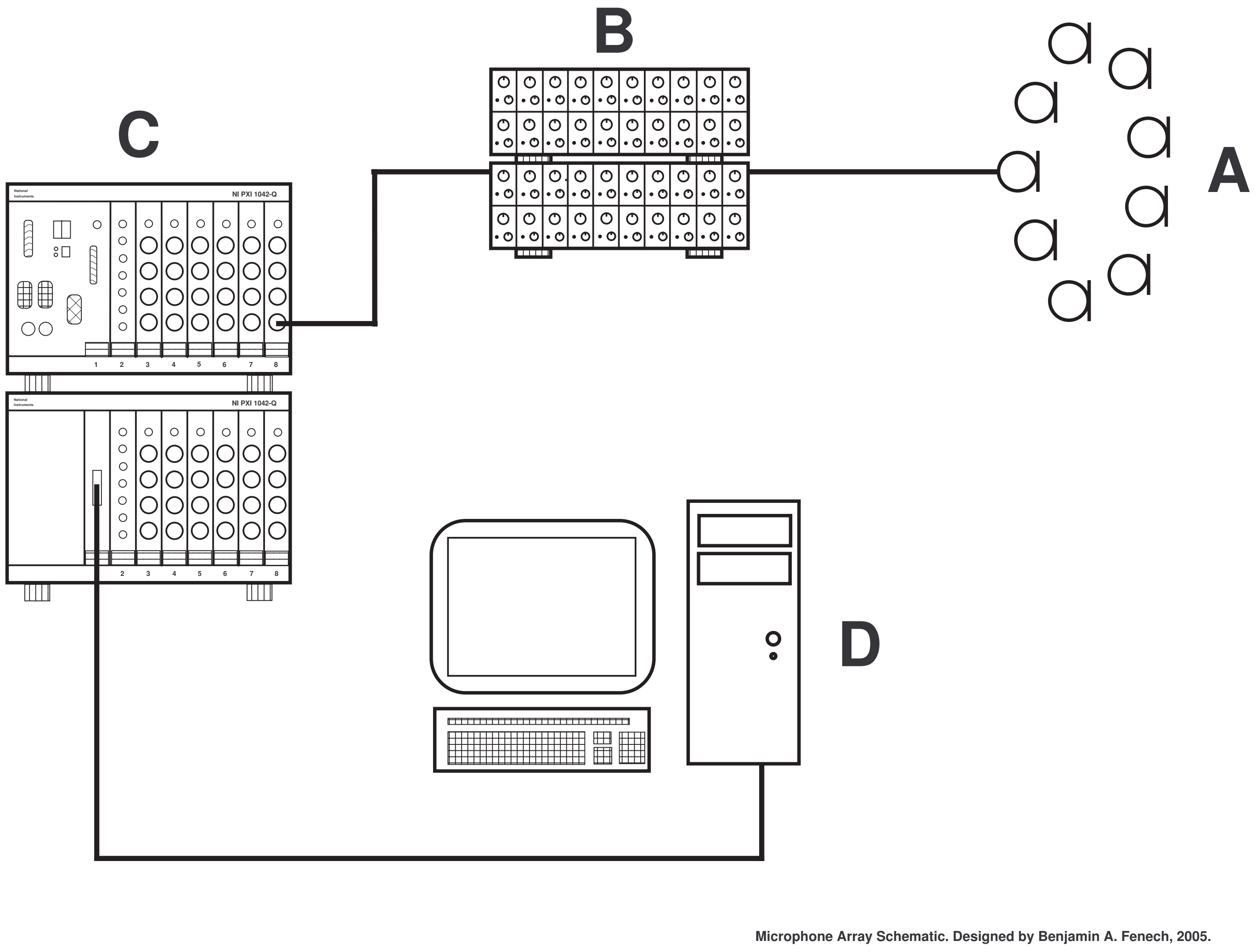

Figure 1: Microphone array – schematic of hardware

when testing very small scale models, although in Ref. [3] it has been shown that the cheaper electret microphones can potentially be used at frequencies up to 48 kHz.

Currently aeroacoustic measurements within SES are carried out using the Panasonic WM-61A electret microphones, which is also used extensively at the Institute of Sound and Vibration (ISVR). The microphone cartridges can either be fixed permanently in the array board, or mounted in plastic tubes to form a more rugged assembly. Each microphone is normally wired to an SMB female connector to facilitate replacements of either microphone or cables.

### 2.1.2 Array Design

The choice of the microphones' placement is often more important than the choice of the microphones themselves. For current beamforming techniques, the performance of the array is measured by the array resolution and average sidelobe levels. The resolution of an array is a measure of the finite size of a point source as it appears on a beamforming plot, and determines the closest distance to which two sources can be resolved. It is a function of aperture size, frequency and distance from the model. Array resolution at low frequencies is usually the determining factor for the array's aperture, i.e. the longest dimension encompassing all the sensors. In the high frequency range, spatial aliasing determines the inter-sensor spacing. If using a regularly-arranged array (e.g. square), the inter-sensor spacing has to be less than half the wavelength of the highest frequency of interest. In practice this is not practical, and aperiodic design arrays[4] are used. In such arrays microphone placement is chosen in such a way that the vector spacing between any two microphones is not repeated, and thus the adverse effects of spatial aliasing do not add up. These kind of arrays are still prone to random sidelobes (also known as phantom images), the levels of which increase with increasing frequency. High sidelobe levels, which can be comparable to the levels of the actual sources, make it difficult to interpret a beamforming plot.

In a closed-section wind tunnel, the microphone array has to be flush-mounted in a boundary section. This subjects the microphones to large hydrodynamic forces due to the turbulent boundary layer. At NASA, arrays are recessed behind a stretched Kevlar sheet; this setup drastically reduces the influence of the boundary layer noise at the lower end of the spectrum ($< 10$ kHz). However, complex resonances are introduced in the array's response.[5]

Currently there are a number of microphone arrays in use in the SES wind tunnels, all based on the principle of a multi-arm logarithmic spiral.

An odd-number of sensors is placed on a number of concentric circles. The location of the microphones is determined by a log-spiral intersecting these circles. The original array design, which can accommodate up to 63 sensors, has an aperture of 0.7 m, giving a resolution of approx. 1.6 $\lambda$ for a scan plane parallel to the array and at a perpendicular distance of 1 m from it. A porous cloth is attached to the surface of the array to smoothen the flow over the microphones.

Currently work is going on to design and build an array with a larger aperture and channel count. Furthermore, research is underway to explore the real benefits and implications of array recessing.

## 2.2 Microphone Cabling

In the existing arrays, each microphone is connected using an individual coaxial RG174 cable; this allows each signal to be individually screened. Cable labelling is of utmost importance, especially in higher channel count systems. Cables can also be grouped together to facilitate assembly/disassembly and troubleshooting.

## 2.3 Microphone Preamplifiers

Signals from a microphone capsule tend to be very weak, and the signal-to-noise ratio can be significantly compromised when using long cable runs. Instrumentation microphones normally have their own proprietary preamplifiers which are designed to attach physically to the capsule itself; these in turn have to be supplied with a fixed voltage or current. Electret microphones are much more flexible in terms of powering, and typically can operate with any voltage between 2 - 12 V. An external preamplifier is therefore required to supply this voltage and amplify the incoming signal. In most cases a variable gain preamplifier is the best choice because the signal level can be maximised to improve the signal-to-noise ratio. Another important feature is a high-pass filter, since the signals from flush-mounted microphones are dominated by low-frequency boundary layer noise, which is not useful for the measurements. An overload indicator also helps to avoid acquiring corrupted data.

A set of preamplifiers have recently been commissioned specifically for use in wind tunnel aeroacoustic applications. They have been designed for an ultra-wide frequency response (20 - 80,000 Hz), a large dynamic range (0 - 40 dB gain in 10 dB steps) and a user-selectable high-pass filter. The preamplifiers are assembled as two channels on one PCB, with one overload indicator per PCB. The variable gain and frequency cut-off controls are linked to groups of 16 channels. A total of 64 channels are housed in one 6U 19" rack case. In order to facilitate wiring up, input and output connectors are in the form of 32-pin EDAC block connectors, which means that each block connector accommodates 16 channels.

An investigation of the performance of these preamplifiers as been carried out, an it was shown that these preamps do have a very wide frequency response. Furthermore their response is repeatable across the different channels, which means that preamp channels can be swapped without influencing the results. Further details can be found in Ref. [3].

Another set of preamplifiers which are occasionally used for microphone array measurements are owned by ISVR. These come as 2U 19" rack cases housing 33 channels each. Each channel has its own input and output connectors, gain control and overload indicator. Input connectors are 6-pin LEMOs whilst output connectors are BNC (see Appendix A).

## 2.4 Data Acquisition cabling

Cabling from microphone preamps to data acquisition is very similar to microphone cabling, i.e. individual coaxial RG174 cable. Once again, the importance of proper cable labelling cannot be over-stressed, especially in higher channel count systems. Cables can be grouped together to facilitate assembly/disassembly and troubleshooting. The end connectors depend on the particular equipment they are attached to (see Appendix A).

## 2.5 Data Acquisition, Storage and Processing

### 2.5.1 Data Acquisition

The analog data from the preamplifiers has to be converted to digital prior to post-processing: this step is referred to as data acquisition. An important prerequisite for phased array measurements is that all channels have to be simultaneously sampled, since the technique relies on phase deviations in the microphone signals. This can be a very demanding requirement when you consider that typically microphone arrays consist of over hundred channels. The analog-to-digital converters must also have a large dynamic range to cope with the large differences between low frequency and high frequency noise, and 24-bit analog to digital converters (ADCs) are recommended. The combination of high-resolution ADCs, high channel count, high sampling rates (to capture information at the higher frequencies) and relatively long sample durations (to improve data accuracy through averaging) means that the system has to handle a large data throughput. Furthermore array requirements tend to change with time, so the system has to be modular for future upgradeability.

There are very few data acquisition systems that can satisfy such demanding applications, one of which is the Dynamic Signal Acquisition (DSA) family of products by National Instruments. The system is based on a chassis housing various input/output cards interfaced by the PXI bus. These cards can be controlled either by an embedded controller housed in the same PXI chassis, or by an external PC through a software-transparent link. There are two choices for multi-channel DSA cards: a four-channel PXI-4462 and an eight-channel PXI-4472. The 4462 has 4 BNC terminals, 24bit, 118dB dynamic range, sample rates up to 2048 ksamples/s, $\pm$ 316 mV to 42.4 V variable input range; the 4472 has 8 SMB (male) terminals, 24bit, 110dB dynamic range, sample rates up to 102.4kSamples/s (i.e. up to 45 kHz alias-free bandwidth), $\pm$ 10 V fixed input range.

To date, SES owns one 18-slot and two 8-slot PXI-chassis, and a mixture of PXI-4472 and PXI-4462 DSA cards. Due to the nature of the synchronisation signals issued by the hardware,[4] the maximum number of 8-channel cards which can be synchronised in one 18-slot chassis is fourteen (i.e. cards in slots 2 to 15), which gives a total of 112 channels. If a higher channel count system is desirable, two or more PXI chassis have to be synchronised together. This is done by placing a timing card in Slot 2 of each chassis, and wiring them in such a way that the oversample clock, SYNC and start triggers are routed from master to slave chassis. For further details how this is accomplished, please see Ref. [8].

National instruments hardware can be directly controlled using Labview software, a graphical based programming language. The latest versions of Labview and *DAQmx* (the device drivers) have simplified significantly the way how hardware is initialised and controlled. For example, to synchronise any number of channels in one PXI chassis, it is sufficient to specify the desired channels in one task; the drivers then take care of the routing of all timing and synchronisation pulses. Therefore, if there are seven PXI-4472 cards in one chassis, named[5] '*PXI1Slot2*', '*PXI1Slot3*', ... '*PXI1Slot8*', and we want to use all eight channels on each card, the *Physical Channels* entry would be

```
{PXI1Slot2/ai0:7,PXI1Slot3/ai0:7,
```

[4]PXI-4472 cards can only be synchronised by routing an *oversample* clock through the *PXI_STAR* line.[6] On the other hand PXI-4462 cards can also be synchronised through the *CLK_IO* line,[7] and any number of cards in one chassis can be synchronised.

[5]To check the name assigned to each card, just click browse and it will list all the channels available in that chassis.

```
PXI1Slot4/ai0:7,PXI1Slot5/ai0:7,
PXI1Slot6/ai0:7,PXI1Slot7/ai0:7,
PXI1Slot8/ai0:7}
```

where *ai0:7* stands for analog inputs 0 to 7.

The latest version of data acquisition software for microphone arrays is called Initialise and Acquire, version 2. This program allows the user to select the required input channels and specify the desired sample rate $f_s$, block size $K$ and total number of blocks $N$. When the program is run, all channels are synchronised and a block-mode[6] acquisition is performed. Each block is dumped to disk in binary format before the next block is read from the buffer. Acquiring data to RAM is not recommended for high channel count systems when using a 32-bit PC, due to the large amounts of data involved. Streaming to disk in binary format is a fairly efficient procedure, provided the file is kept open and a fast internal hard drive is used.

Binary data in Labview is written as double precision IEEE floating point with big-endian byte ordering and 64 bit long data type. The raw data is transposed [i.e. channel number in first dimension and sample number in second dimension]. A sample Matlab code to read the Labview binary file and prepare the data for the SotonArray software would be:

```
fid = fopen('filepath\filename.bin','r','s');
RawData = fread(fid,[M,K*N],'*double');
RawData = RawData';
fclose(fid);
```

A separate Labview program which allows two or more PXI chassis to be synchronised together is currently under development.

### 2.5.2 Data Storage

As explained in the previous sections, microphone array measurements generate significant amounts of data. Presently data is dumped to disk for post-processing at a later stage, although in the future the processing might be performed using data still in RAM (using distributed computing). In this case dumping to disk is still necessary for archival purposes.

[6]Please note that although the Labview program operates in a block-mode acquisition, the individual blocks are contiguous, as they are read from a buffer. In fact, if the dumping to disk stage is too slow, the buffer will fill up and an error occurs, prompting the user to increase the buffer size or decrease the number of blocks requested. What this means is that when performing an FFT analysis on the raw data, the user is not restricted to an FFT size equal to or smaller than the block size.

The best practice is to dump data to a fast internal PATA or SATA drive; having two drives in RAID 0 configuration is even better. This data can then be transferred to an external or network-attached storage device for archival purposes.

### 2.5.3 Post Processing

The most common microphone array processing technique is frequency-domain beamforming. The time data from $M$ microphones is Fourier transformed to the frequency domain and corrected for microphone sensitivities. The cross-spectral power matrix (CSM) $S_{\hat{\mathbf{p}}\hat{\mathbf{p}}}$ is computed using this corrected data. An imaginary scan plane is then defined (where a "noise map" is desired) and split into $N$ grid points. The vector of free-space Green's functions $\mathbf{g}$ between each grid point and each microphone is computed. The beamformer's output for the $n^{th}$ grid point is then

$$Q = \frac{\mathbf{g}^{\dagger} S_{\hat{\mathbf{p}}\hat{\mathbf{p}}} \mathbf{g}}{\left[\mathbf{g}^{\dagger}\mathbf{g}\right]^2}. \tag{1}$$

There are a number of optimisation techniques that can be used to improve the clarity of the plots, such as subtraction of a background noise CSM and removal of the leading diagonal of the CSM.

If quantitative values are required, an area integration can be performed by summing up the beamformer's output within an area enclosing the region of interest. The resulting quantities have to be corrected for the array's response function.

Data processing can be performed in real-time with the data acquisition, or as a separate process after the acquisition. The principle of beamforming is fairly simple (does not involve any inversions, regularisation or iterative algorithms), however it is a multi-dimensional technique, and calculations have to be made for thousands of grid points for each of a number of centre frequencies. When using a stand-alone computer, it is not feasible to post process the data concurrently with the acquisitions, as the CPU and RAM usage for both processes will conflict and slow down the system. The current practice is to choose just a few data runs to postprocess, this will somewhat slow down the experimental test schedule, however it ensures the integrity of the acquired data, and can also help in making informative changes to the test schedule. In the future this might change by integrating distributed computing in the process.

## 3 Software - SotonArray v.3.3

The array post-processing is written in Matlab. It is divided into a number of files: a main interface program and several functions files which are called by the main program. The main interface program is divided into three main sections:

- defining parameters
- computing the cross spectral matrix (CSM)
- carrying out the beamforming

The function files have different functions, e.g. fft, generate coordinates, etc. The software is written in such a way that the user can focus on the first section of the main interface program, and there is no need of any advanced knowledge of the phased array processing. The only exception is when the array is to be mounted in a non-standard position, since the coordinate system in a number of function files has to be changed.

This code is intended to be used with the array mounted in the vertical section, control-room side of the $7 \times 5$ wind tunnel, with the +ve $x$-axis pointing downstream. Wind flow can be from any direction, as it is input as a vector. The code allows a vertical scan plane of dimensions $2l_x \times 2l_y$ to be defined in the wind tunnel test section; the scan plane can rotate about a fixed pivot with a vertical axis of rotation (parallel to the array surface). If this is not the case, the function file *GenerateCoorGrid* has to be modified accordingly.

### 3.1 Input Parameters

In the first part of the program there is a brief description of what it does, version history, etc. Global variables are used to simplify the sharing of some variables between the main program and the function files.

```
% SotonArray -=CSM and BF complete version=- Copyright B. Fenech 2005-2007
% Main Beamforming script, where user defines all the input and output
% parameters. Script then calls the functions which perform the beamforming
% process, including CSM and weight vector generation

% Developed by Benjamin A. Fenech
% Original code created 18/10/05
% v.3.3 ironed out minor bugs in v.3.2
% v.3.2 (View overall microphone response, added possibility to omit data
% from potentially faulty microphones)
% Wind tunnel speed as a vector (relative to array coordinates)
% Renamed a selection of variables
% v.3.1 (added gain factor to faulty channels)
% Version 3.0 created 22/02/06 - 14/03/06
% Some major and minor improvements to the beamforming code, mostly:
% a) possibility to apply 2 types of sensitivities: pistonphone or
% frequency response function
% b) tidy up the fft part, and correct some bugs
% c) possibility to use previously generated CSM
% d) save fft data for individual spectra analysis (psd)

% Last modification on 20/11/06
clear all

time(1) = cputime;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Global Parameters                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global SampleFrequency BlockSize WindowType
global CSMBackground CSMOptimise CSMOptimiseType AveragingChoice N_oct
global OmitChoice OmitColumns
global ttl_param1 ttl_param2 ttl_param3
global dim_1 dim_2 dim_3 dim_4 resolution l_x l_y
global freq df
global X Y MicsTotal
global Mach c_0
global FileRootOut perform trial
```

The raw data can be imported either from ASCII or Matlab's proprietary "**.mat*" binary format. For microphone array data, ASCII is not recommended, as it is very inefficient in terms of reading/writing to disk speeds, and disk space cost. Unfortunately, Matlab's MAT format is not open source, and it is not practical to write data in this format directly from Labview. In section 2.5.1 a sample code is given how to read Labview's binary data efficiently in Matlab; this data can then be saved directly to MAT format with the *save* command. SotonArray assumes that the data exists under variable name *RawData*. In order to facilitate batch processing, it is recommended to save the MAT data with a filename having a fixed name and running number (for example *data n.txt*, where $n$ is a running integer. The loaded MAT data is assumed to have the following format: sample number in first dimension and channel number in second dimension.

In the next stage, the main file paths and file name patterns are defined. A good practice is to have a main root for the whole project, containing the data folders together with a *Misc. Files* folder which contains additional data such as sensor coordinates and calibration data (pistonphone or FRF). The microphone pistonphone sensitivities are obtained by calibrating each microphone individually using a calibrated pistonphone and the Labview VI *calibrate.vi*, which extracts the peak amplitude of the tone in the signal (corresponding to 1 Pa at 1 kHz). These values (Vpeak/Pa) are then input in an ASCII file as comma separated variables, according to the order that the microphones have been wired – i.e. the sensitivity of the microphone wired to DAQ card 1 channel 0 comes first. An alternative to pistonphone calibration is a full frequency response function (FRF) calibration, which compares the spectrum of each microphone to a reference microphone with a flat frequency response (such as an instrumentation microphone). The magnitude of this FRF represents the ratio of sensitivities of the $m$th microphone and the reference microphone. This type of calibration is a lengthy procedure, but is the most accurate. A separate Labview VI *RefCalibrate.vi* can be used for this type of calibration. The sensitivity of the reference microphone, in Vpeak/Pa, has to be measured to convert the sensitivity ratios to absolute sensitivities.

Other input files are the sensor coordinates (coordinates of the holes in the array), an ASCII file containing which holes are in use and in which order, and another ASCII file containing the physical microphone numbers in the order they are wired (used for FRF calibration only).

Two data conditioning processes can be defined at this stage. The first gives the option to omit data from any particular channels (due for example to dead channels which can degrade the beamforming data integrity); the second is to apply a predefined gain to any particular channels to correct, for example, for any incorrectly set preamplifiers.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Start of User Input                            %
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 Input directories and filenames                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Data is expected to be in *.mat format, since large ASCII data sets take
% a long time to load (and save in Labview). Here the expected filename is
% data xxx.mat. The data should already be assigned some variable name,
% depending on the Labview code default = RawData.
```

```
trial = '001';

MainRootPath = 'C:\Project Folder Path\';
DataFolder = 'Array Data Folder\';
DataFilename = ['data ',trial,'.mat'];

DataFilenameTXT = ['data ',trial,'.txt'];

% RAW DATA
% Uncalibrated pressure data from microphones, with no timestamps. Samples
% in first dim, microphones in second dimension
DataPath = [MainRootPath,DataFolder,DataFilename];
DataPathTXT = [MainRootPath,DataFolder,DataFilenameTXT];

% SENSITIVITIES - TWO POSSIBILITIES
% ASCII file containing microphone sensitivities (comma seperated file)
% IMPORTANT ORDER OF SENSITIVITIES MUST MATCH ORDER OF CHANNELS WIRED (i.e.
% columns in raw data file)
% NOTE values are Vpeak/Pa - as given by pistonphone calibration VI
SensitivityFile = [MainRootPath,'Misc. Files\sensitivities M.txt'];
% Root of directory containing FRF data in ASCII format, named according to
% microphone number
FRFPath = ['J:\ACHIEVA\Array Data\Koen III\Calibration\'];
CalibrationNameMAGBase = 'mag mic ';
CalibrationNamePHASEBase = 'phase mic ';
CalibrationNameExt = '.txt';
BKSensitivity = 1.36;                        % Vpeak/Pa

% If any channels have to be omitted for data integrity reasons, specify
% the COLUMN numbers to omit here. Data integrity can be checked by the
% microphone check included in this code. When omitting, 4 data arrays will
% be modified: the actual raw data, the microphone index file, the
% microphone sensitivity file and the microphone numbers file.
% Do NOT take into account column shifting - this will be done
% automatically later on
OmitChoice = 1;
OmitColumns = [21,32,51];

% If any channels have to be amplified or attenuated by a fixed factor, it
% can be done here
GainApply = 0;                          % apply gain corrections
% IMPORTANT: IF OMITTING COLUMNS IN THE PREVIOUS STEP, MAKE SURE THAT THE
% COLUMNS SPECIFIED HERE CORRESPOND TO THE (SHIFTED) COLUMNS AS A RESULT OF
% THE OMISSION PROCEDURE
GainColumns = [35,36];                  % columns to apply gain
GainFactor = [10,20];                   % extra gain for each channel (not dB)

% SENSOR COORDINATES
```

```
MicsCoorPath =...
    [MainRootPath,'Misc. Files\coor 7x9 0.35 unbrnk wcrcl v6.mat'];
% List of sensors used (optional)
MicsUsedPath = [MainRootPath,'Misc. Files\mic positions.txt'];
% List of microphone numbers - as wired in array (for FRF calibration)
MicsNumbersPath = [MainRootPath,'Misc. Files\mic numbers.txt'];
```

The next step is to define the processing parameters, together with other input variables. As explained in Section 2.5.1, it is not necessary to use the same block size as that specified in the acquisition, although the product of block size and number of averages cannot exceed the total number of available samples for obvious reasons. Usually it is better to have a smaller block size and more averages, unless a very fine resolution is explicitly required. Fast Fourier Transforms are typically based on a block size of $2^n$. The sample frequency has to match that used during the acquisition. A Hamming time window is usually applied for beamforming.

The temperature refers to the average temperature in the wind tunnel during the particular trial, and is used to correct the speed of sound. The flow velocity is used to correct for source convection effects, and it must have the same coordinate system as the sensor coordinates.

For phased array measurements in a closed-section wind tunnel, with the array flush-mounted on the wind tunnel walls, the data is contaminated with the turbulent boundary layer noise in direct contact with the microphones. It is good practice to try to physically separate this boundary layer from the microphones as much as possible, using a **smooth** porous cloth over the microphones, or better still, by recessing the array behind a stretched porous cloth. Significant improvements can also be achieved by performing some optimisations on the calculated Cross Spectral Matrix (CSM), namely a modification to the leading diagonal. The variable *CSMOptimise* specifies whether to perform this optimisation or not (recommended), and *CSMOptimiseType* allows a choice of removal and replacement (both work equally well). It is also possible to subtract a background CSM to improve the fidelity of the results (although the benefits are not very clear.) To do this, a data acquisition has to be performed with an empty wind tunnel running at the required speed, and the computed CSM saved as a background CSM. Finally, if the same run has been already processed and the CSM saved, putting *CSMReady* to unity avoids re-calculating the CSM, saving approx. 20% of the time.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                            Input variables                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Acquisition and processing parameters
SampleFrequency = 48000;        % Hz
BlockSize = 4096;                % equal to or less than acquisition N
FFTAverages = 100;
WindowType = 1;                 % 0 Hanning 1 Hamming

% Input Variables
temperature = 16.6;              % WT temperature (degrees C)
% Wind speed is a vector with the same coordinates as array
% eg. if wind is coming from -ve x direction and its only component is
% parallel to the x-axis, the vector is [u 0 0]
WTVelocity = [40,0,0];           % wind tunnel speed (m/s)

% CSM Optimisation
CSMBackground = 0;              % remove background noise CSM
CSMOptimise = 1;                % optimise CSM
CSMOptimiseType = 1;            % 0 remove diagonal 1 replace diagonal

CSMReady = 1;                   % compute CSM (0) or use saved one (1)
```

An important point when performing beamforming is ensuring that the scan plane coincides with the region of interest. Moreover, it is essential that the position of the model being tested is defined with respect to the array centre, so that potential source-generating locations on the model can be identified. It is more difficult than one might think to measure the exact position of the model with respect to the array centre, unless there is a physical connection between the array surface and the model. A rather crude way is to use measuring tapes and spirit levels; a better way would be to use laser instrumentation.

The present code is designed to create a scan plane centred around a reference point, which is also a pivot. The scan plane can rotate about a vertical axis (parallel to array surface) through the pivot, and an angle theta can be defined (clockwise positive, as viewed from above). To define the location of this reference point with respect to the array centre, three dimensions have to be input: *dim_1* in the $x-$direction (+ve downstream), *dim_2* in the $y-$direction (+ve upwards), and *dim_4* in the $z-$direction (+ve in front of the array, i.e. into the test section). Another dimension, *dim_3* specifies the perpendicular distance from the scan plane to the reference point (+ve towards the array).

The scan plane extends to a length of $2 \times l_x$ and a height of $2 \times l_y$, and is divided into a grid of $resolution \times resolution$ points.

```
% Fixing a reference point on the model with respect to array centre; if
% the model is rotating, the reference point must be equivalent to a
% centred pivot point. Important: with every new test session, the function
% files GenerateCoorGrid and SaveAndPlot have to be modified to make sure
% that the scan plane is in the right place. The present files are written
% specifically for the array mounted on the wall (control-room side) of 7x5
% WT. For further details refer to log book II p.45

% Coordinates of reference point on model wrt to array origin
% Wall array
dim_1 = -0.247;                       % x (m)
dim_2 = -0.076;                       % y (m)
dim_4 = 1.183;                        % z (m)
% distance of scan plane in front of reference point (in z direction)
dim_3 = 0.076;                        % z (m)
% angle of scan plane with array surface (clockwise +ve)
ScanAngle = 0;                       % degrees

% Scan Plane limits - Half the length and height of the scan plane,
% IMPORTANT LIMITS REFERENCED TO REFERENCE POINT ON SCAN PLANE - FUNCTION
% FILE GenerateCoorGrid CORRECTS COORDINATES TO BE RELATIVE TO ARRAY CENTRE
% (ORIGIN)
l_x = 0.5;                           % (m)
l_y = 0.5;                           % (m)
resolution = 120;                     % no. of grid points/side
```

Beamforming is performed for a set of frequencies; usually the centre frequencies for 1/3-octave bands from 2 kHz to 20 kHz are used. The resolution of the present array is too poor at low frequencies (below 2 kHz). It has been shown that frequency averaging is an effective way of reducing the influence of sidelobes in the resulting plots,[1] and averaging over 1/3-octave bands (*N_oct = 3*) is a good choice. For more tonal phenomena, 1/6-octave bands (*N_oct = 6*) can be used, and in very poor conditions, one might attempt to use full octave bands (*N_oct = 1*). If only a particular frequency is of interest, $FrequencyMultipleChoice$ can be set to 0; in this case, only $FrequencySingle$ will be calculated.

```
% Frequencies of interest
% Resolution is too poor below 2kHz
FrequencyInterest = [2000,2500,3150,4000,5000,6300,8000,10000,...
         12500,16000,20000];
```

```
% frequency averaging parameters
AveragingChoice = 1;               % 0 NO; 1 YES
N_oct = 6;                         % 1/N octave band (0 for NO averaging)
% Do beamforming for all frequencies of interest, or just for one
% frequency?
FrequencyMultipleChoice = 1;       % 0 NO; 1 YES
FrequencySingle = 4000;            % if FrequencyMultipleChoice = 0
```

The next section specifies several options which users are mostly interested in. These are:

- saving of beamformer plots – three formats are supported: jpeg (for quick thumbnail indexing viewing on Windows machines), eps (for lossless saving, and for inclusion in LaTeX documents) and ASCII (for importing in other plotting utilities, such as Tecplot®);
- calibration type, depending on the calibration procedure performed during the experiments (FRF calibration is most accurate)
- saving of Cross Spectral Matrices for future use (note that the same CSM can be used for different scan planes, but a new CSM has to be generated for different FFT parameters or calibration procedure) – CSMs are saved in Matlab native .mat format;
- saving power spectral densities of the individual microphones in the array – the saved files can be read by a separate standalone script file *ViewMultiChannelPSD_v11.m*;
- optimise code for speed – this option enables a significant increase in processing time (depending on the user's minimum number of frequencies input) without any noticeable degradation of results;
- processing time breakdown – this is intended for troubleshooting and performance optimisation purposes only.

The different options are well documented in the program itself.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                            User preferences                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% User choice:
```

```
% a) save beamformer plot
% options: 1. jpeg, 2. eps, 3. ascii, 4. jpeg+ascii 5. eps+ascii
% 6. jpeg+eps 7. jpeg+eps+ascii  0. do not save

% b) pistonphone/FRF calibration
% options: 0. no calibration 1. pistonphone calibration 2. FRF calibration

% c) save Cross Spectral Matrix (CSM)
% options: 0. do not save, 1. save background CSM, 2. save final CSM

% d) save one-sided psds
% options: 0. do not save, 1. save

% e) optimise for speed (frequency averaging part)
% options: 0. NO, integer = minimum number of frequencies to average

% f) show breakdown of processing time for the various processes
% options: 0 NO, 1 YES

perform = [0,2,0,0,12,1];
```

The next section defines the data acquisition hardware setup, in terms of the number of PXI cards and channels per card in use. This information is used to perform automatic data integrity checks.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                           Array Selection                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Cards setup in NI chassis (channels/card)
CardDescription = [8;8;8;8;8;8;7];
```

The final part of the first section defines the model output filename format. It assumes three folders have been created in the relevant data directory, named *CSM*, *FFT* and *Results*. If background CSM subtraction is going to be used, a subdirectory called *BKG* has to be present in the *CSM* directory.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    Output titles and filenames                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generate titles given the input variables
TitleParameters;
```

```
% Output filename path
FileRootOut = [MainRootPath,DataFolder,'Results\'];

% Input/Output CSM filenames
CSMPath = [MainRootPath,DataFolder,'CSM\CSM ',trial,...
    ' CAL ',num2str(perform(2)),' OPT ',num2str(CSMBackground),...
    num2str(CSMOptimise),...
    num2str(CSMOptimiseType),' fs',num2str(SampleFrequency/1000),' N',...
    num2str(BlockSize),' ',num2str(FFTAverages),' avrgs.mat'];
FFTPath = [MainRootPath,DataFolder,'FFT\FFT ',trial,...
    ' ',num2str(WTVelocity),'m_s CAL ',num2str(perform(2)),' fs',...
    num2str(SampleFrequency/1000),' N',...
    num2str(BlockSize),' ',num2str(FFTAverages),' avrgs.mat'];
CSMBackgroundPath = [MainRootPath,'Day 1\CSM\BKG\CSM background ',...
    num2str(WTVelocity),'m_s CAL ',num2str(perform(2)),...
    ' OPT 0',num2str(CSMOptimise),...
    num2str(CSMOptimiseType),' fs',num2str(SampleFrequency/1000),' N',...
    num2str(BlockSize),' ',num2str(FFTAverages),' avrgs.mat'];

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
%                            End of User Input                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

## 3.2 Generation of the Cross Spectral Matrix

User input is not required in the rest of the program, however is it is useful to have an idea what is going on in the following sections. In the first part, the raw time data is loaded, converted to Engineering units and transformed to the frequency domain. The CSM is calculated, and any optimisations applied.

First the speed of sound and Mach number are calculated, and a number of data integrity checks are performed.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                            Start of Program                           %
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                         Physical Constants                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Constants
c_0 = 331 + 0.6*temperature;          % speed of sound as a function  of
                                      % temperature
Mach = WTVelocity/c_0;                % Mach No.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              Load data and perform initial Data Tests               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MicsIndex = load(MicsUsedPath);
if OmitChoice == 1
    ab = 0;
    for aa = 1:length(OmitColumns)
        % The following step is done once to take into account the shifting
        % of columns to the left once preceding columns are omitted.
        OmitColumns(aa) = OmitColumns(aa) - ab;
        % Delete column, shift rest to left
        MicsIndex(OmitColumns(aa)) = [];

        ab = ab+1;
    end
end
clear aa

CardsTotal = size(CardDescription,1);
MicsTotal = sum(CardDescription);
if OmitChoice == 1
    MicsTotal = MicsTotal - length(OmitColumns);
end
```

Next the raw data is loaded, checked, and conditioned. Data omission and extra gain are applied at this stage.

```
if CSMReady ~= 1            % compute CSM

% Note: Raw data (units Volts) has to be in columns (corresponding to
% individual channels. If for example 512 samples are taken 100 times for
% 50 different channels, the raw data will consist of 50 columns each
% having 512*100 rows.
% Whenever the cards/channels are altered, the input parameters
% CardDescription, DataPath and SensitivityFile have to be altered

% Import Data (in MAT format, should appear in a variable RawData)

load(DataPath);
RawData = RawDataArray2;
clear RawDataArray2

if OmitChoice == 1
    for aa = 1:length(OmitColumns)
    RawData(:,OmitColumns(aa)) = [];
    end
end
clear aa
time(2) = cputime;
disp(['Finished loading data file.'])

if size(RawData,2) ~= MicsTotal
    warning('Microphone count mismatch: raw data with MicsTotal')
end
if size(RawData,2) ~= size(MicsIndex,2)
    warning('Microphone count mismatch: raw data with MicsUsed')
end

% Compare number of acquired blocks with number of averages requested
aa = size(RawData,1)/BlockSize;
aa = floor(aa);
if aa == 0
    aa = 1;
end
if aa < FFTAverages
    error('Number of fft averages requested exceeds available blocks')
else
    disp(['Using ',num2str(FFTAverages),' out of a total of ',...
        num2str(aa),' available blocks.'])
end
clear aa

% Apply extra gain to pre-defined channels
% Assumes that column shifting to the left has been taken into account when
% specifying GainColumns
if GainApply ~= 0
    MultiplyMatrix = ones(size(RawData));
    for aa = 1:length(GainColumns)
        MultiplyMatrix(:,GainColumns(aa)) = GainFactor(aa);
    end
    RawData = RawData.*MultiplyMatrix;
    clear MultiplyMatrix aa
end
```

Next the conditioned data is converted to Engineering units. Depending on the user's preference, this step is performed either at this stage as a simple division or after the FFT operation. In the latter case, the filenames for the microphone calibration data are generated at this stage by calling an external function *GenerateFRFFilenames*.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Convert to engineering Units                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Apply pistonphone calibration, if applicable
RawDataCalibrated = zeros(size(RawData));
if perform(2) == 1
    MicsSensitivity = load(SensitivityFile);
    if OmitChoice == 1
    for aa = 1:length(OmitColumns)
    MicsSensitivity(OmitColumns(aa)) = [];
    end
    clear aa
end
    % Correct from Vpeak/Pa to Vrms/Pa
    MicsSensitivity = MicsSensitivity./sqrt(2);
    if size(RawData,2) ~= length(MicsSensitivity)
        error('Pistonphone sensitivity data does not match raw data size')
    end
    for aa = 1:size(RawData,2)
        RawDataCalibrated(:,aa) = RawData(:,aa)./MicsSensitivity(aa);
    end
    display('Applied Pistonphone calibration')
else
    RawDataCalibrated = RawData;
end
clear RawData MicsSensitivity aa

if perform(2) == 2
    [FRFFilenamesMAG,FRFFilenamesPHASE] =...
    GenerateFRFFilenames(FRFPath,CalibrationNameMAGBase,CalibrationNamePHASEBase,...
    CalibrationNameExt,MicsNumbersPath);
    display('Prepared filenames for FRF calibration')
else
    FRFFilenamesMAG = [];
    FRFFilenamesPHASE = [];
end
time(3) = cputime;
```

The Fast Fourier Transforms are computed in function file *MultiChannelFFTandFRF*, where FRF calibration is also applied if applicable. the data is returned as one-sided spectrums.

At this stage, a quick data integrity check is performed. A cubic fit is applied to all the spectrums from the individual microphones, and any channels which differ from the median by a pre-set threshold are listed and plotted on a sensor coordinate plot. The user is prompted whether he/she wants to continue. All this is done within the function file *MicrophoneCheck*.

If this check does pick up faulty channels, it is recommended to break the operation and insert the list of faulty channels into the *OmitChoice* option in the beginning of the program.

Depending on the user preferences, another function file *MultiChannelPSDSave* generates and saves power spectrum densities for each channel.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Part I                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 FFT (and checking integrity of data)                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Sequence: full calibration (optional, perform FFT

% Perform FFT by calling external function
FrequencyDataOneSided =...
    MultiChannelFFTandFRF(RawDataCalibrated,FFTAverages,...
    FRFFilenamesMAG,FRFFilenamesPHASE,BKSensitivity);
clear RawDataCalibrated FRFFilenamesMAG FRFFilenamesPHASE
disp(['Finished FFT analysis.'])
time(4) = cputime;

% Check Data Integrity
MicrophoneCheck(FrequencyDataOneSided,MicsCoorPath,MicsIndex)
% Ask user whether to continue processing
ContinueChoice = 'y';
ContinueChoice = input('Do you wish to continue y/n [y]:','s')
if ContinueChoice == 'n'
    return
end

% Save FFT data (optional)
if perform(4) == 1
    MultiChannelPSDSave(FrequencyDataOneSided,FFTPath)
    disp(['Saved FFT data.'])
end
```

In the last stage, the CSM is generated within *BuildCSM* and optimised within *OptimiseCSM*. This array is saved according to user preferences. If the CSM had already been saved, all the previous steps are skipped and the pre-calculated CSM is loaded into memory.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Part II                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                  Generate optimise and save CSM                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Builds the averaged, optimised Cross Spectram Matrix from raw time data
% Sequence: build cross spectram matrix and average,
% remove background CSM (optional), optimise diagonal (optional), save
% matrix (optional)

% Build CSM by calling an external function
CrossSpectrum = BuildCSM(FrequencyDataOneSided);
time(5) = cputime;
disp(['Generated CSM.'])
clear FrequencyDataOneSided

% CSM Optimisation by calling external function
if CSMBackground ~= 1 & perform(3) ~= 1
    CSMBackgroundPath = [];
end
CSM = OptimiseCSM(CrossSpectrum,CSMBackgroundPath);
time(6) = cputime;
disp(['Optimised CSM.'])
clear CrossSpectrum

% Save cross spectral matrix to file
if perform(3) == 1
    save(CSMBackgroundPath,'CSM')
    disp(['Background CSM saved.'])
elseif perform(3) == 2
    save(CSMPath,'CSM','freq','df')
    disp(['Optimised CSM saved.'])
end
% End of CSM Generation

else        % Load pre-saved CSM
    load(CSMPath)
    disp(['CSM loaded.'])
    time(2:6) = cputime;
end
```

## 3.3 Frequency-Domain Beamforming

In this section, an imaginary scan plane is created and divided into an equally-distributed grid of points. Microphone weight vectors are calculated for vectors connecting each microphone to each grid point. The beamforming equation is than computed using the previously-calculated CSM. This is performed for a number of frequencies within an averaging band, and a median value is extracted for each grid point.

Once again, the main processes are carried out in external functions:

- *GenerateCoorMics* and *GenerateCoorGrid* generate the coordinates of each microphone and each grid point, in a well-defined order;
- *GenerateFreqBand* generates a list of frequencies depending on the user's averaging band and speed optimisation choices;
- *ChooseFreqNear* selects the frequency in the FFT analysis closest to the frequency of interest;
- *beamformer* generates the microphone weight vectors and computes the beamforming expression.

The data for each centre frequency is plotted as a spatial plot, by default as a 3-D coloured surface plot as viewed directly from above. Function file *PlotandSave* defines automatically details such as the viewing angle, colour scheme and plotting limits.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                Part III                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Beamforming                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Loads microphone coordinates, selects the one in use. Generates
% coordinates of scan plane. Calculates propagation vectors and microphone
% weight vectors. Uses pre-calculated CSM to calculate beamforming
% expression, prepares data for 3D plotting

% Edited weight vector calculation to avoid having to flip the plot
% matrices. In the calculation of the Propagation Vectors, the complex
% exponential is +ve, not -ve as indicated in book. This allows an
% off-centre grid plane to be defined directly. Note that this makes sense
% physically since during beamforming we are dealing with waves which are
% propagating towards the source (inwards). Using an e^(iwt) convention,
% waves propagating spherically inwards are denoted by a positive
% exponential e^(ikr)

% Optional Frequency averaging over 1/N octave band. Averaging in frequency
% band by mean OR median (specified in appropriate function file)
% Loading sensor coordinates and select microphones used
load(MicsCoorPath);
% Build vector of microphone coordinates by calling external function
x_m = GenerateCoorMics(coor,MicsIndex);
% Build vector of grid-point coordinates by calling external function
% IndexPrecision are half the inter-grid spacing in x and y coor, and are
% used to calculate the nearest grid reference index for integration
[x_b,IndexPrecisionX,IndexPrecisionY] =...
    GenerateCoorGrid(l_x,l_y,ScanAngle);

% FreqList1 are the distinct centre frequencies at which beamforming plots
% are to be generated
% FreqList2 are all the frequencies within the band centred at the
% frequency of interest, for use by frequency averaging algorithm
if FrequencyMultipleChoice == 1
    FreqList1 = FrequencyInterest;
else
    FreqList1 = FrequencySingle;
end

% Separate centre frequencies
for FreqIteration1 = 1:length(FreqList1)
    [FreqList2,FreqCentre] =...
        GenerateFreqBand(FreqList1,FreqIteration1);

    % Averaging within bands
    for FreqIteration2 = 1:size(FreqList2,2)
        [omega,FreqIndex] = ChooseFreqNear(FreqList2,FreqIteration2);

        BF = beamformer(omega,FreqIndex,x_m,x_b,CSM);

        BFIntermediate = reshape(BF,[resolution+1 resolution+1]);
        % BFSpatial contains the complex beamformer's output in a spatial
        % matrix, i.e. 1st dimension represents y coor and 2nd dim.
        % represents x coor.
        BFSpatial = BFIntermediate';
        % Only absolute values retained
        Zall(:,:,FreqIteration2) = abs(BFSpatial);
        clear BF*

    end     % End of frequency averaging

    Z = BFFreqAverage(Zall);

    Z = 10.*log10( Z./(4E-10) );    % Normalise with p_ref^2 (assuming mean
                                    % square pressures);
    Zclear = ClipPlot(Z);          % clip datapoints below predefined level

    PlotAndSave(Zclear,FreqCentre)
    time(6+FreqIteration1) = cputime;
end
```

The last part of the program gives the possibility of a breakdown of the processing time involved, which can be significant.

```
if perform(6) == 1
    for aa = 1:length(time)-1
        TimeDifference(aa) = time(aa+1) - time(aa);
    end
    PieNames = {'Load Data','Calibration','FFT','CSM','Optimise CSM'};
    for ab = 1:length(FreqList1)
        PieNames{1,5+ab} = [num2str(FreqList1(ab)/1000,'%2.1f'),' kHz'];
    end
    TimeTotal = sum(TimeDifference)
    figure()
    pie3(TimeDifference,PieNames),...
        title('Time taken by the different processes')
end

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
%                            End of Program                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

# 4 Limitations and Sources of Error

The main limitations of phased array measurements are:

- poor resolution makes it difficult to obtain useful results at low frequencies. For example, the 0.7 m diameter array yields a source region with a diameter of 0.55 m for a point source 1 m away from the array emitting noise at 1 kHz. An array with double the aperture has double the resolution, giving a source region of 0.27 m for the same conditions.

- the ability of the array to filter out background noise is finite. Therefore if the wind tunnel's operating background noise is significant, there is a large risk of not picking up significant noise sources on the model, especially if the model is a low-noise design.

The major sources of error in microphone array measurements are:

- electret microphones – the frequency response of these microphones somewhat varies from one capsule to another. To obtain best results, a frequency response calibration is necessary (calculating the transfer function between each microphone and a reference microphone when both are subjected to the same sound field). For qualitative results, a pistonphone calibration may be sufficient. It was shown that phase deviations up to $\pm$ 10 degrees between the array microphones do not contribute to inaccuracies in the results.[9]

- distance measurements – it is very important to know the exact whereabouts of the microphones and the test model with respect to some reference point, say the array's centre, especially at high frequencies. Whilst the microphone holes can be machined with reasonably good accuracy using a CNC, measuring the exact location of the test model is more problematic. One way of solving this problem is to put a point source at a known position on the model, and adjust the location of the model using the beamforming plots. Alternatively there are several accurate calibration algorithms which can be applied.[9, 10]

- image sources – beamformer plots will be contaminated by image sources. The levels of these sources (compared to the real sources) depends on the location of these image sources, and the array's response or point spread function (how good the array is at suppressing noise from that particular point).

- free-space Green's functions – clearly this is an inaccurate assumption, especially for small, hard-walled wind tunnel sections. Quantitative values are somewhat affected. It should be possible to correct for this by using measured Green's functions of the test space.[11]

- coherent sources – the beamforming theory assumes a point source at the focus point, in the absence of any other interfering sources. In practice there is usually a distribution of sources with some degree of coherency between them. This error will mostly influence quantitative values. The inverse method is not prone to this kind of error.[12]
- geometry of setup – it as been shown that the array's quantitative output is dependent on the array's aperture and the solid collection angle $\theta_{DA}$ (the included angle between the outermost sensors as seen by the source).[13] The apparent source size is also dependent on frequency (determined by the array's resolution).

## Acknowledgements

This work is funded by the UK Engineering and Physical Sciences Research Council under grant GR/S68446/01. The authors wish to thank Phil Nelson, Keith Holland and Malcolm Smith of the Institute of Sound and Vibration Research, and Alexander Carballo-Crespo of the School of Engineering Sciences for their valuable contributions.

# References

[1] Benjamin A. Fenech. *Aeroacoustic Measurements - Transfer Thesis.* PhD thesis, School of Engineering Sciences, 2006.

[2] William M. Humphreys Jr., Carl H. Gerhold, Allan J. Zuckerwar, Gregory C. Herring, and Scott M. Bartram. Performance analysis of a cost-effective electret condenser microphone directional array. *AIAA*, 2003-3195, 2003.

[3] Benjamin Fenech and Kenji Takeda. Frequency response analysis of microphone preamplifiers in the audible and ultrasonic regime. AFM 07/02, SES, University of Southampton, January 2007.

[4] Thomas J. Mueller, editor. *Aeroacoustic Measurements.* Springer, 2002.

[5] S.M. Jaeger, W.C. Horne, and C.S. Allen. Effect of surface treatment on array microphone self-noise. *AIAA*, 2000-1937, 2000.

[6] National Instruments. *NI 447X Specifications*, March 2006.

[7] National Instruments. *NI 446X Specifications*, March 2006.

[8] National Instruments. *GETTING STARTED WITH MULTICHASSIS SYNCHRONIZATION USING THE NI PXI-6653 AND THE NI PXI-4472*, December 2002.

[9] Marianne Mosher, Michael E. Watts, Srba Jovic, and Stephen M. Jaeger. Calibration of microphone arrays for phased array processing. *AIAA*, 97-1678-CP, 1997.

[10] Stanley T. Birchfield and Amarnag Subramanya. Microphone array position calibration by basis-point classical multidimensional scaling. *IEEE Transactions on Speech and Audio Processing*, 13(52):1025–1034, September 2005.

[11] Benjamin Fenech and Kenji Takeda. Towards more accurate beamforming levels in closed-section wind tunnels via de-reverberation. *AIAA*, 2007-3431, 2007.

[12] KR Holland and PA Nelson. Sound source characterisation: The focussed beamformer vs the inverse method. In *Tenth International Congress on Sound and Vibration*, July 2003.

[13] Thomas F. Brooks and William M. Humphreys Jr. Effect of directional array size on the measurement of airframe noise components. *AIAA*, 99-1958, 1999.

## A SotonArray – Detailed Schematic