AcousticVNA User's Guide

John Price
University of Colorado, Boulder
Version date: January 22, 2008

Table of Contents

1. Introduction

Welcome to the world of AcousticVNA, a flexible and accurate acoustic waveguide measurement system. This User's Guide will get you up and running with AcousticVNA no matter who you are: an engineer characterizing acoustic materials, a researcher modeling ventilation systems, an audio pro designing a new tweeter, a student learning about the physics of waves, or a teacher preparing classroom demonstrations. If you find that you need to know more about acoustics or acoustic waveguides, we suggest you have a look at the book Acoustic Waveguides (AW).[*] We will point you to specific sections from time to time, like this: For an overview of different kinds of acoustic waveguide measurements, see AW Section 1.1.

AcousticVNA is a collection of hardware and software components for acoustic waveguide measurements, suitable for both research and educational use. The hardware components are modular and can be assembled in many configurations. The software, AVNA Lab v1.0, is written in MATLAB, and is open source and freely distributed from our web site.

Figure 1.1(a) shows a set-up for studying radiation from a horn. On the left is a compression driver, a type of loudspeaker designed to feed cylindrical waveguide. The driver is connected to the rest of the system with a standard ARS-25 coupler, which consists of two flanges and an o-ring seal. If you are used to working with vacuum equipment, you will notice that our coupler design is similar to the ISO- KF seal; however, we have made some changes to provide a smooth and uninterrupted bore. Next comes the microphone section, a length of straight cylindrical waveguide with several miniature electret microphones mounted flush to the inner wall. The microphones are sealed by o-ring fittings so they may be removed easily for calibration. Finally there is another standard coupler, and the device under test, in this case a horn.

Not shown in the figure are the buffer amplifiers that adapt the electret microphone elements to drive standard audio microphone preamplifiers, the preamplifiers themselves, and a digitizer that captures the audio signals for analysis in a computer. The digitizer is one of the enabling technologies that makes AcousticVNA powerful and affordable. Thanks to the market for home audio recording equipment, units are now available with 8 channels of preamplification and 96 kHz simultaneous-sampling digitization for under $500. The simultaneous-sampling feature is essential; it allows measurement of the relative phases of the microphones.

[*] *Acoustic Waveguides*, by John C. Price, 2007, University of Colorado, Boulder.

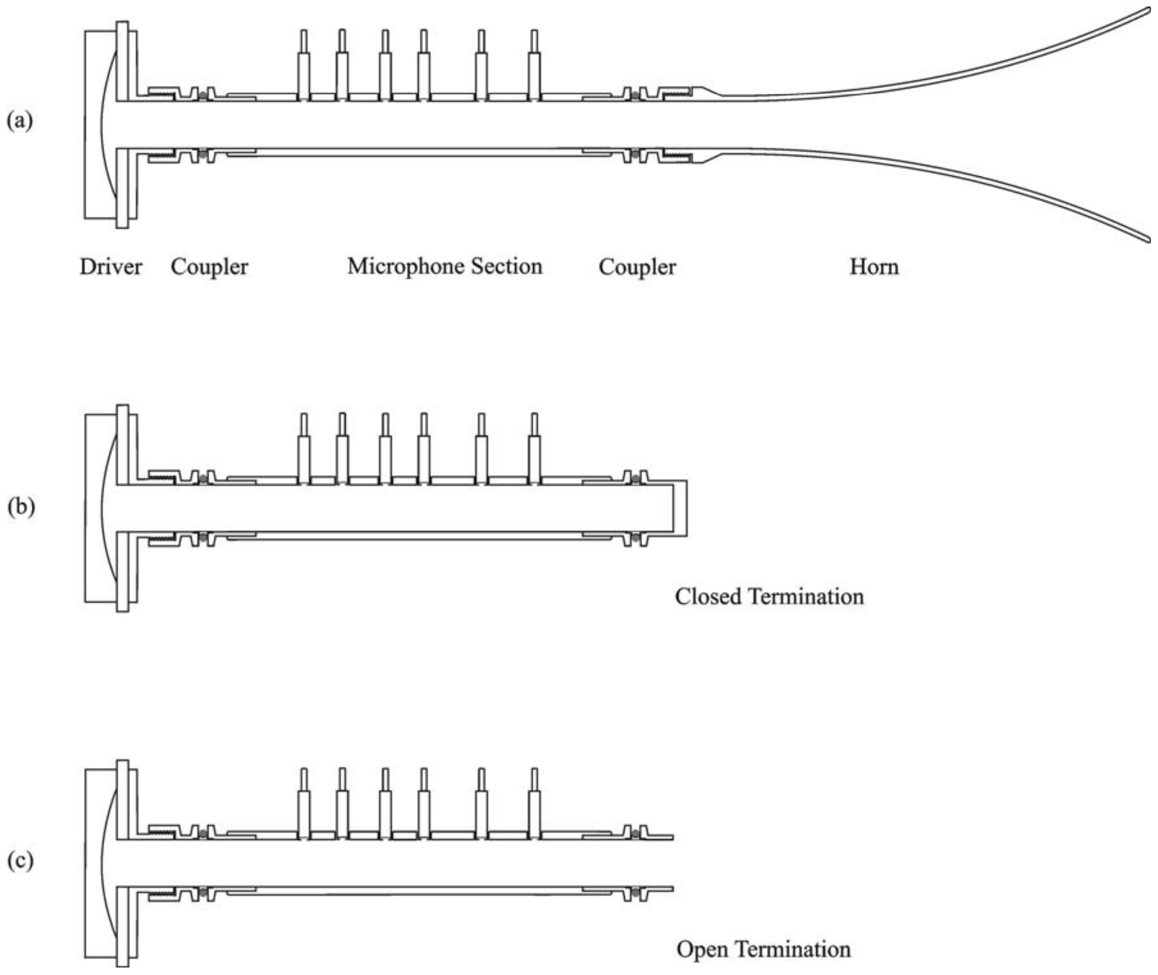


Figure 1.1. Reflectometer configurations for studying (a) radiation from a horn, (b) reflection from a closed end, (c) radiation from an open end and the ‘end correction.’

The signal that drives the compression driver is generated by the computer and may be a sine wave, a pulse, a random waveform, or any other waveform. In this manual we mostly consider the simplest case of sine-wave excitation. There is generally a small and inaccurately known time shift (the latency) between the recorded samples and the excitation waveform. Latency is not usually an important limitation, even though it means that the phase between the drive signal and microphone signals is unknown. In those cases where it is important to know the overall phase, one preamplifier channel can be used to record the drive signal.

Data from the microphones can be analyzed to yield the horn’s reflection coefficient, the ratio of the amplitude of the reflected left-going wave to the amplitude of the incident right-going wave. The reflection coefficient is a single complex number at each frequency that describes completely the linear behavior of the horn, as viewed from the waveguide that drives it. In this case, the magnitude of the reflection coefficient can

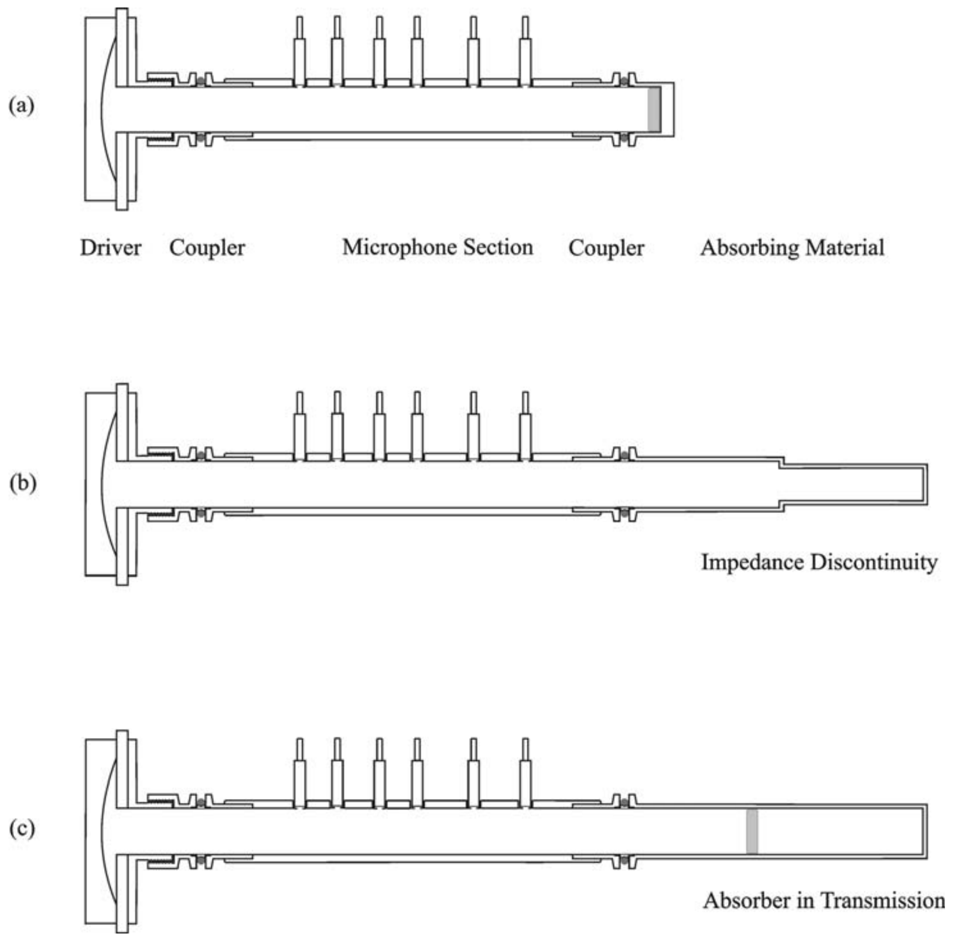


Figure 1.2. Reflectometer configurations for (a) reflection from an absorbing material, (b) impedance discontinuity, (c) transmission through an absorber.

tell us how much power is being radiated by the horn. Because of the primary importance of the reflection coefficient, an apparatus like this is often called a reflectometer; in this case a 'vector' reflectometer, because it measures both the magnitude and the phase of the reflection coefficient. The word vector is used because the reflection coefficient may be plotted as a vector on the complex plane.

Figures 1.1(b) and (c) show closed and open terminations in place of the horn. The closed termination is the simplest possible since it has a reflection coefficient of +1.0 at all frequencies, with no important corrections. The open termination has a reflection coefficient near -1.0, the change of sign being due to inversion of the wave upon reflection, or equivalently to the presence of a pressure node at the open end. However, in this case there are important and interesting corrections: the effective position of the reflection is not quite at the end of the tube but somewhat beyond it, and the magnitude of the reflection coefficient is less then unity because some power is lost to radiation, though much less than from a horn.

Figure 1.2(a) shows a closed end faced with an absorber. This arrangement is used for characterizing sound absorbing materials, and is similar to the apparatus specified for such measurements by ASTM standard (E 1050-98). In Figure 1.2(b) an extension has been added with a step in the diameter, allowing study of transmission and reflection by an abrupt change in impedance. Similarly, Figure 1.2(c) shows a set-up for study of transmission through an absorbing material. In these last two configurations analysis of the transmission property is complicated by the presence of a reflection at the far end. An alternative approach is to replace the closed end by an absorbing termination. However, it is difficult to make a compact absorbing termination that works well over a broad range of frequencies, so the more easily characterized closed end may be preferred.

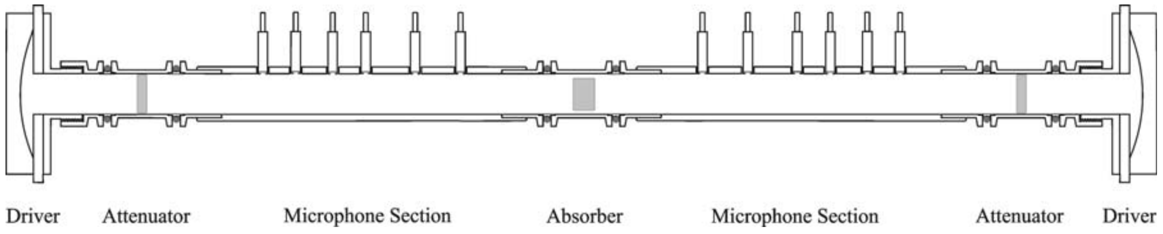


Figure 1.3. Acoustic vector network analyzer (VNA) for scattering matrix measurements.

A powerful arrangement that permits simultaneous transmission and reflection measurements with high accuracy is shown in Figure 1.3. Two drivers and two microphone sections are used. Only one driver operates at a time, but simultaneous coherent data from microphones on both sides is needed to measure the 2-by-2 scattering matrix of the object under test, in this case a sample of absorbing material. The scattering matrix, containing four complex numbers at each frequency, relates the two incident waves (one from each side) to the two out-going waves. The attenuators reduce reflections and simplify extraction of s-matrix elements. Alternatively, both drivers can be used, one acting as a sound source and the other as an active absorbing termination.

Electrical engineers who work with microwave and radio-frequency electronics use an closely analogous instrument called a vector network analyzer (VNA), where ‘network’ refers to the circuit under test. Since the late 1960s the VNA has been the most important measurement tool for high-frequency electronics, as essential as the oscilloscope is at lower frequencies. AcousticVNA brings the power and convenience of general-purpose vector network analysis to the world of acoustics.

2. Set-up and Verification

2.1 Software Set-up: Digitizer and MATLAB

These instructions assume that you are starting from scratch and need to set up a system that has never been used before. If you know that your digitizer drivers and the MATLAB system are already set up, skip to Section 2.2 on setting up AVNA Lab, the software used to operate the AcousticVNA hardware. All new users should read Sections 2.2, 2.3, and 2.4.

The AcousticVNA system requires a computer running Windows XP or Vista, with an IEEE-1394 port, and at least 512 MB ram. The CPU clock should be 1 GHz or faster, and dual processors are preferred.

If your computer does not already have an IEEE-1394 port, you can buy an inexpensive card to provide this feature. For laptops, we have used the Dynex ED-FC202 adapter, which is available for about $60. Be sure to read the instructions that come with your IEEE-1394 hardware, since you may have to install or update a driver.

We assume in this manual that your digitizer is a Presonus FP-10 or FirePod Firewire Recording Interface. The FirePod and FP-10 are almost identical units, but they have different drivers and it is important to install the correct one. Both units are combination 8-channel preamplifiers and digitizers that connect to your computer's IEEE-1394 port. It is likely that any preamplifier/digitizer combination supporting Steinberg's ASIO audio driver protocol for Windows could also be used. However, we have only tested the Presonus units and their drivers for compatibility with the MATLAB audio interface software.

To begin, download the latest driver for your FirePod or FP-10 from the Presonus web site at www.presonus.com. Be sure to get the correct driver for your operating system. If you don't already have it, download the latest version of the manual for your digitizer as well. Following the instructions in the manual, install the driver. Install all three parts of the driver software (ASIO, MIDI, and WDM) if you are given the choice, and do not connect and turn on the digitizer until you are told to do so. Set the sampling rate to 44.1 kHz, the clock source to internal, and the latency to 10 ms. You might want to change the sampling rate or latency later, but these are good starting points. Note that the latency setting here does not fix the actual latency to 10 ms. Rather, it is a request to the driver to try to achieve 10 ms latency. If your system can't keep up, you will experience ‘drop outs’ in the recorded data and should change to a longer latency setting. Finally, choose an optimization setting that matches your processor’s speed.

You can skip the final part of the Presonus instructions which tell you how to install the CUBASE recording software. CUBASE is not used by the AcousticVNA system. However, if you have problems later you might want to install it for debugging purposes. It allows you to test your digitizer without involving MATLAB or the AVNA Lab software.

If you want to set-up a vector network analyzer configuration with two or more microphone sections, you may need to run several FirePods, several FP-10s, or a combination of these at the same time. In some cases this requires driver updates or updates to firmware in the digitizers. See instructions on the Presonus downloads page.

Next, install MATLAB version 7.4 or later, following the MATLAB documentation. The procedure varies according to how your copy of MATLAB is licensed. If you are using a site license, you may have to contact your site administrator for help. MATLAB is available with many add-ons to the core program, and we recommend that you install everything that is covered by your license. For AcousticVNA you only need the MATLAB core, the Optimization Toolbox, and the Signal Processing Toolbox. The MATLAB Data Acquisition Toolbox does not currently provide ASIO support and is not used by AcousticVNA.

2.2 Software Set-up: AVNA Lab v1.0

AVNA Lab runs under on MATLAB. If your MATLAB skills are rusty, or if you have never used MATLAB before, take 10 minutes now to get oriented. Launch the application, and choose **MATLAB Help** under the **Help** menu. Under the *Contents* tab, expand the topics *MATLAB* and *Getting Started*. Read the section *Introduction*. Then, in the next section *Matrices and Arrays*, read and try out the examples in *Matrices and Magic Squares*, and *Expressions*. This will give you a quick introduction to the most important MATLAB syntax, and will introduce the MATLAB desktop. Later, you may want to go back and work through the rest of the *Getting Started* section.

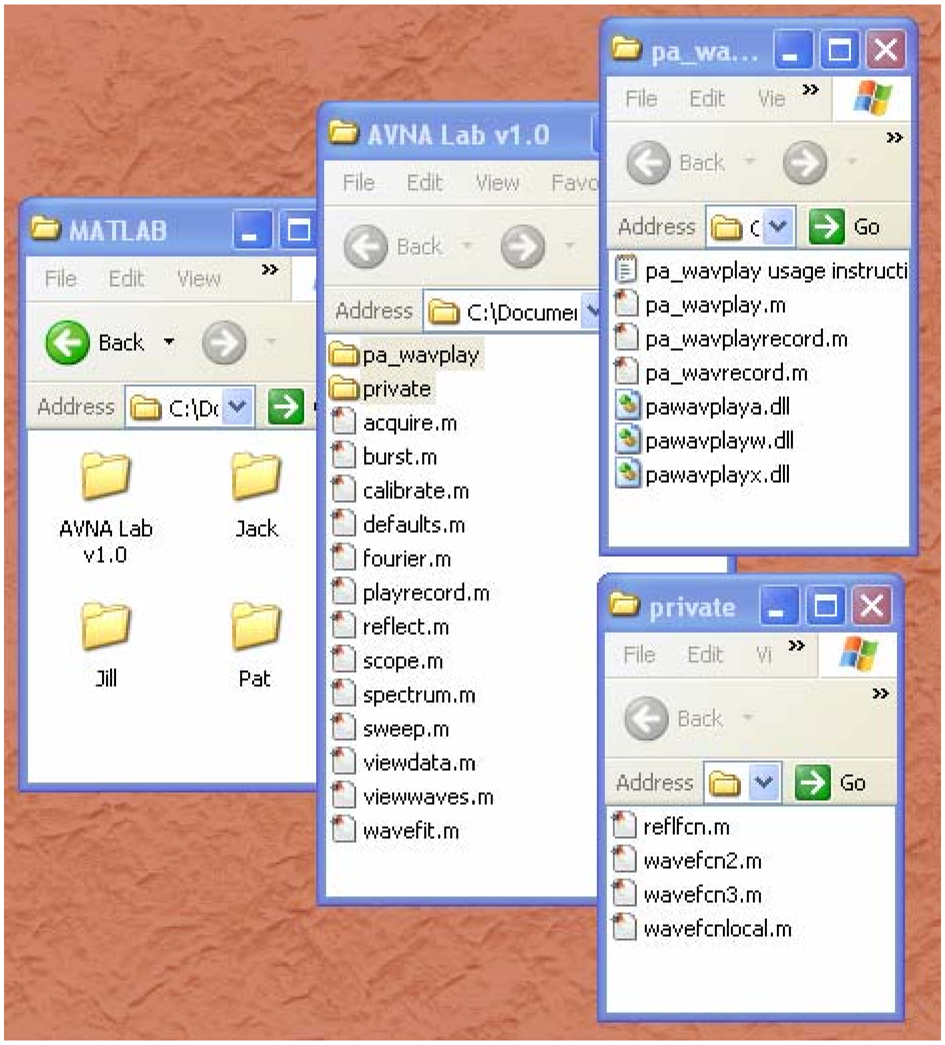


Figure 2.1. File organization for AVNA Lab and pa_wavplay. The system software is in *AVNA Lab v1.0*, while *Jack*, *Jill*, and *Pat* are user folders.

AVNA Lab is a set of MATLAB script and function files. The individual files are known as M-files in MATLAB jargon, after their file extension `.m`. They each contain a sequence of MATLAB commands just like those that are entered directly into the *Command* window. You can view (and edit) them using the *Editor* window. The difference between scripts and functions is that scripts use the base workspace that is visible in the *Workspace* window on the MATLAB desktop, while functions use a private workspace that is deleted when the function finishes executing. AVNA Lab requires a set of files called pa_wavplay that define MATLAB functions for communicating with your digitizer. The AVNA Lab files are available as a free download from our web site. The pa_wavplay files can be found by searching the MathWorks website at www.mathworks.com . They were developed by Matt Frear and are also available as a free download.

When MATLAB was installed, it created a folder with a path similar to:

```
C:\Documents and Settings\User Name\My Documents\MATLAB
```

By default, this path becomes the current directory when MATLAB is launched. The current directory is displayed in a toolbar widow near the top of the MATLAB desktop. We suggest that you arrange the AVNA Lab files and the pa_wavplay files in the …`\MATLAB` folder as shown in Figure 2.1. The folder *AVNA Lab v1.0* contains thirteen M-files that are the functions and scripts used to control AcousticVNA, a *private* folder, and the *pa_wavplay* folder. In MATLAB, a *private* folder contains functions that can only be called by routines in the folder containing the *private* folder. This folder cannot be renamed and must be immediately inside *AVNA Lab v1.0*. The *pa_wavplay* folder contains the six pa_wavplay routines (3 M-files and 3 .dll files) and a documentation file. If the files in the *AVNA Lab v1.0* folder are shared by several users, care should be taken not to alter them, except for documented updates.

The other folders in …`\MATLAB`, *Jack*, *Jill*, and *Pat*, are for user M-files and data.

To complete the set-up of AVNA Lab, open the MATLAB path browser by choosing **Set Path…** in the **File** menu. The window that appears lists all of the folders that MATLAB searches when it needs to find an M-file, in the order in which they are searched. You will see many paths like:

```
C:\Program Files\MATLAB\…
```

These contain the MATLAB system and add-on toolboxes, and should not be altered unless you are sure you know what you are doing. At the top of the list you should see the default current directory:

```
C:\Documents and Settings\User Name\My Documents\MATLAB
```

Click on **Add Folder…** and add the following two paths to the top of the list:

```
C:\Documents and Settings\User Name\My Documents
    \MATLAB\AVNA Lab v1.0

C:\Documents and Settings\User Name\My Documents
    \MATLAB\AVNA Lab v1.0\pa_wavplay
```

Do not add the …`\private` path (because private folders are included automatically with their enclosing paths) or the user paths …`\MATLAB\Jill`, …`\MATLAB\Jack`, or …`\MATLAB\Pat`.

When you get ready to write programs, you will begin by setting the current directory to your user folder, such as …`\MATLAB\Jill`. Your programs will have access to the AVNA Lab routines and the pa_wavplay routines, because their folders have been added to the search path. Your programs will also have access to whatever you place in your user folder because MATLAB always searches the current directory, and searches it first. You may want to create modified versions of the AVNA Lab M-files with the same names as the standard ones. If you put these in your user folder, your versions will 'overload' the standard ones for you only, and other users will not be disturbed. (If you were to add your user folder to the search path this might not be true, so don't do that unless you are certain that all your M-file names are unique to you.) All of the AVNA Lab routines search for data files only in the current directory, and only write to the current directly, so there is no risk that you will read or write someone else's data.

If every user of the system has a separate Windows account, the above precautions are not important. Just be sure that all the AVNA Lab routines and all the pa_wavplay routines are in folders on the search path, and that any routines you add are either in the current directory or on the search path. Also, make sure that the *private* folder is directly inside the folder containing the AVNA Lab routines, and don't forget that the AVNA Lab routines always read from and write to the current directly.

2.3 Hardware Set-up

To get started, you will need at least the components shown in Figure 2.2. If you can't tell from photo if you have the right parts, you can find more information about each component in Chapter 7 below. You may have other AcousticVNA components, such as a horn or flanged open. These will not be needed for the set-up procedures or for the Introductory Mesaurements discussed in Chapter 4 below. Some of the Applications described in Chapter 7 do require additional hardware.

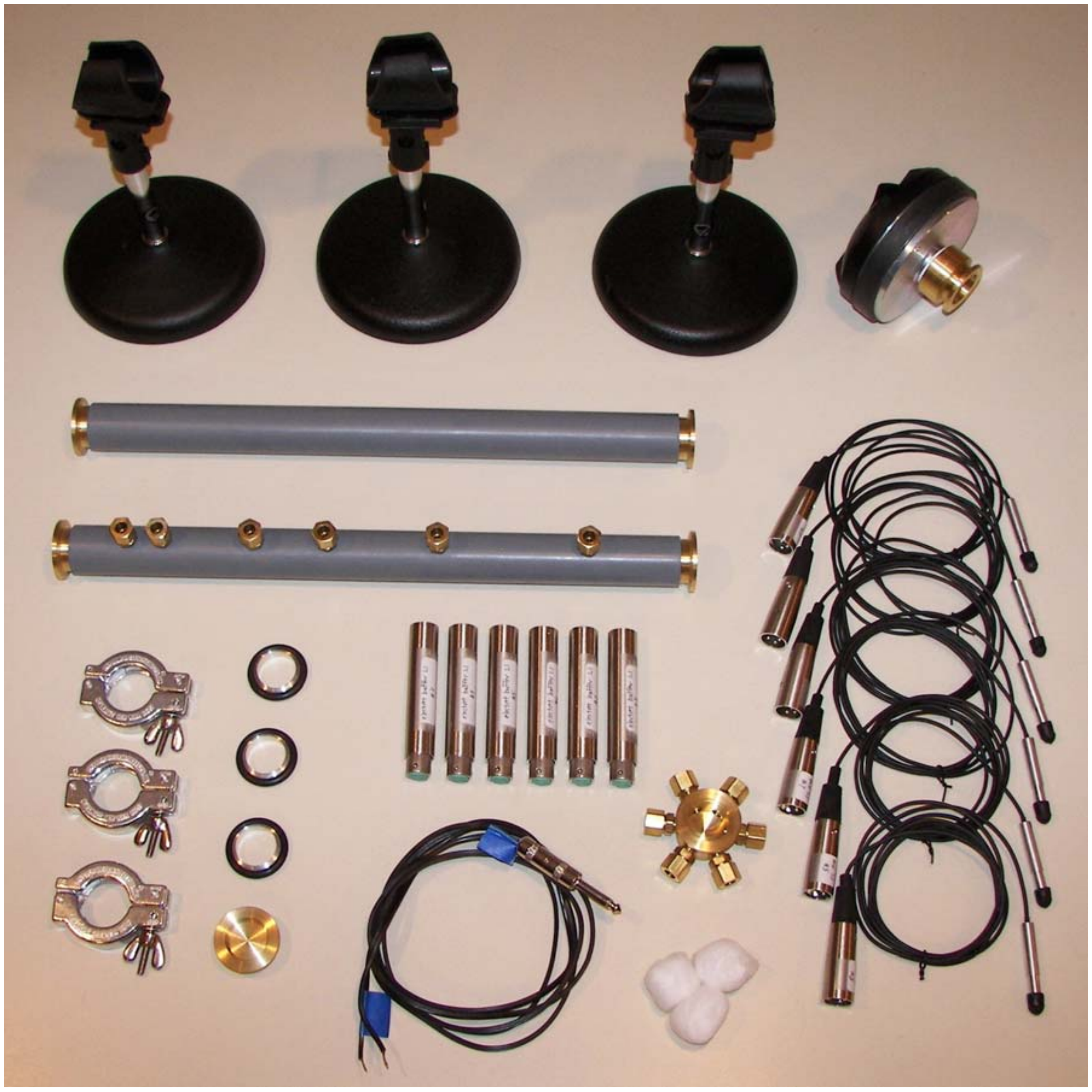


Figure 2.2. Minimum hardware for reflectometry. Starting at the top, the photo shows three waveguide stands, a compression driver, a straight section, and a six-station microphone section. Below the microphone section on the left are three coupler clamps, three seals with o-rings, and a closed end. At center are six buffer amplifiers, and below them is a cable for the compression driver. To the right is a six-station calibration cell, three cotton balls, and six microphones. Each microphone has a black rubber cap over the sensitive end for protection.

If you have not already done so, connect power to your computer and to your digitizer, turn both on, and connect an IEEE-1394 cable between the digitizer and the computer. The Presonus digitizers show gain drifts of several percent during warm-up, which takes about six hours. It is best to turn your digitizer on the day before you plan to use it and to leave it on until you are finished.

AcousticVNA waveguide sections are assembled using ARS-25 couplers, as shown in Figure 2.3. The two brass flanges are permanently attached to the waveguide sections. Be sure that the aluminum seal ring is properly aligned with both flanges before tightening the clamp, shown in Figure 2.4. When the clamp is tight, the inner edge of the flange will press against the aluminum seal ring, forming a smooth inner bore with no gap. The washer on the clamp should be between the wing-nut and the clamp body.

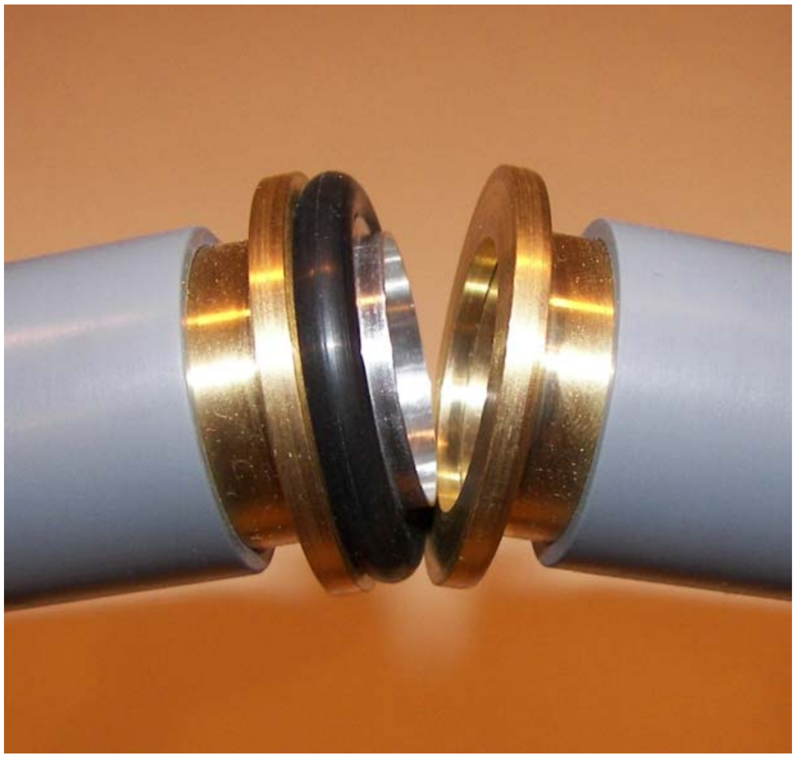


Figure 2.3. Assembling the ARS-25 coupler for 25 mm I.D. waveguide.

We will first assemble the system for microphone calibration. Connect the six buffer amplifiers to preamplifier inputs numbers three through eight of the digitizer, and connect the six microphones to the buffer amplifier inputs. Turn on the 48V phantom power for the buffers and mics by pushing in the two buttons at the left end of the digitizer's front panel. Once the microphones are calibrated it will be important to distinguish them. This will be easier to do if you connect your lowest numbered microphone to the lowest channel number, the next highest mic to the next channel, and so on. Do not remove the black protective caps on the microphones until you have to.

The sensitive ends of the microphones are fragile and should be kept covered whenever possible.

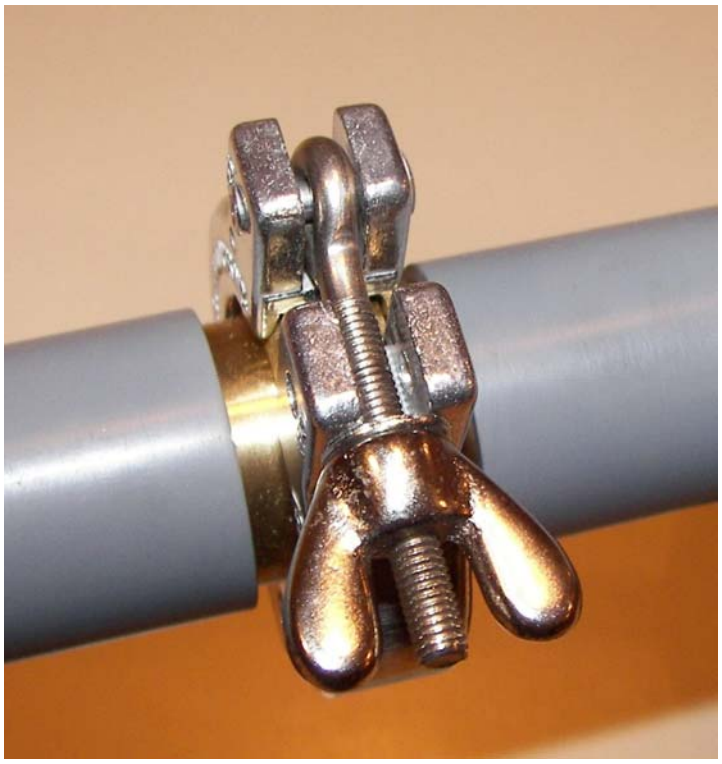


Figure 2.4. ARS-25 coupler with clamp.

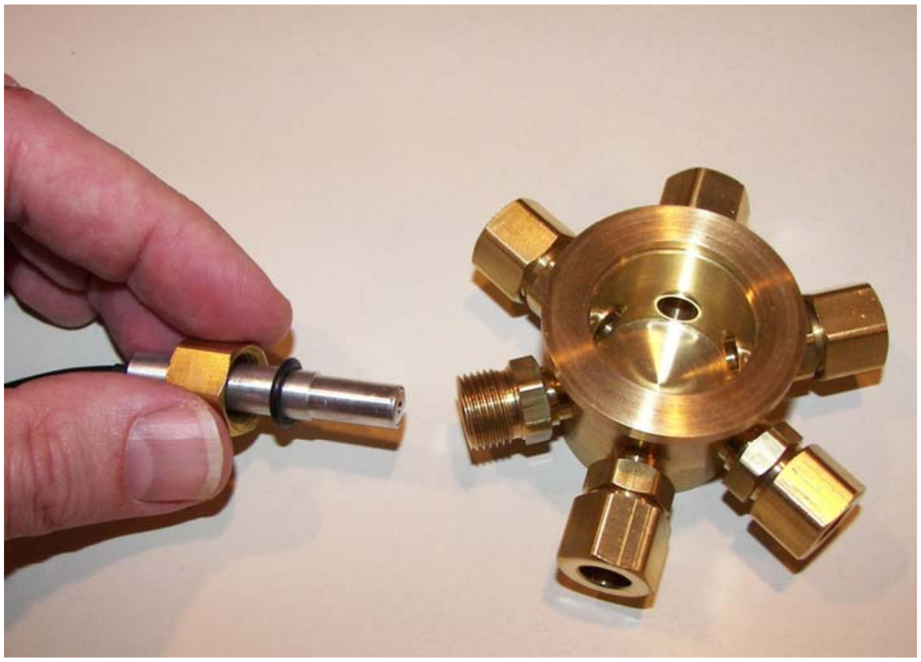


Figure 2.5. Assembling the calibration cell. Place the threaded cap and the o-ring on the microphone before inserting it into the calibration cell.

Referring to Figure 2.5, install the microphones in the calibration cell. The calibration cell has six o-ring compression seals for mounting the microphones. Before inserting a microphone, remove one of the threaded caps and an o-ring from the cell, remove the protective cap from the microphone, and then slide the threaded cap and the o-ring onto the microphone, being careful not to touch the exposed sensitive end of the microphone. Slide the microphone into the cell until it rests on the shoulder of the microphone body, and gently tighten the seal with your fingers only. You should be able to see that the end of the microphone is flush with the inner wall of the cell. Install all six microphones. Place the compression driver on its back with the coupler flange up, and attached the compression cell to the driver using a seal and a clamp. To avoid feedback, turn the Mixer control on the digitizer front panel fully clockwise, to the Playback (1-2) setting. Finally, connect the cable that goes from the headphone output of the digitizer to the compression driver. See Figure 2.6.

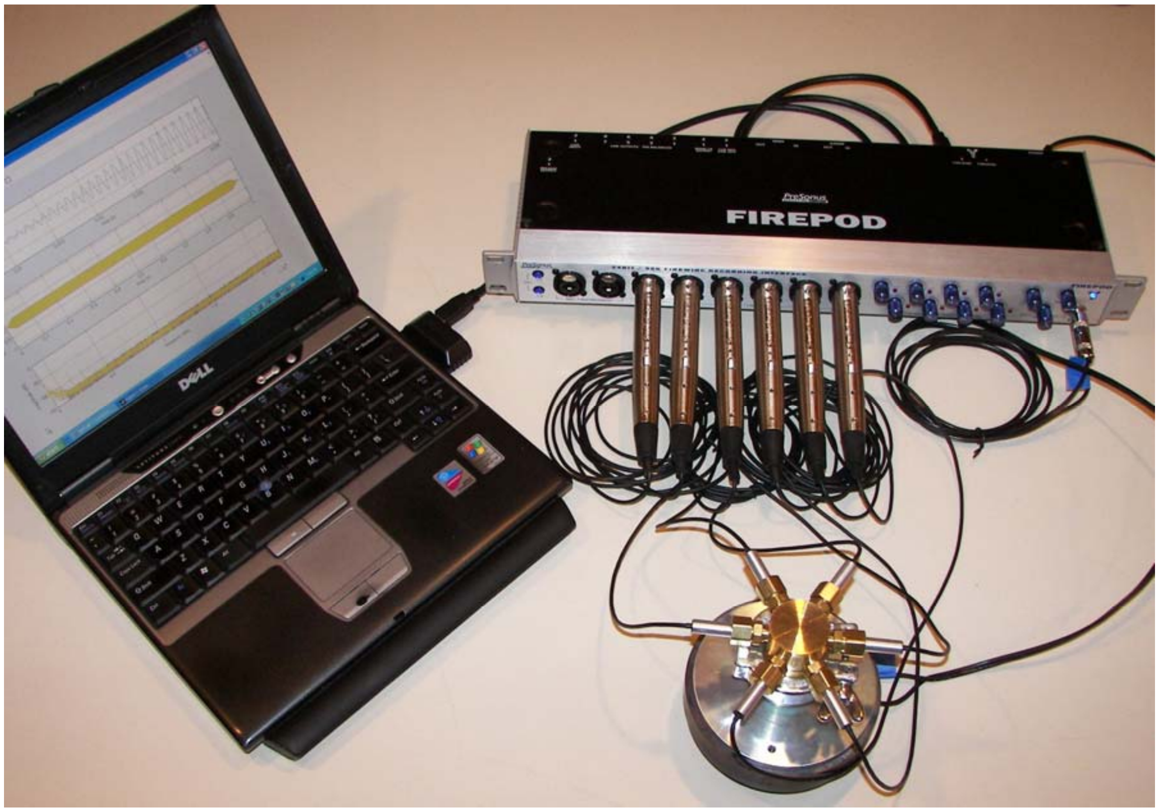


Figure 2.6. AcousticVNA set-up for microphone calibration.

2.4 System Test and Latency Adjustment

You are now ready to test the system for proper operation. Check that the digitizer is on and the 48V phantom power buttons are pushed in. The front-panel Main Level knob controls rear-panel outputs that will not be used. Turn it fully counter-clockwise. Check that the Mixer knob is set fully clockwise; this routes the play signals from the computer to the headphone output jack. Set the Phones level knob to about nine o'clock, and turn all eight Preamp Input Gain knobs counter-clockwise to their lowest settings.

Launch MATLAB, and wait until it initializes (see the status message at the bottom of the MATLAB desktop). Using the current directory window at the top of the MATLAB desktop, set the current directory to your user folder (see Figure 2.1) and enter in the *Command Window*:

```
>> scope
```

You should hear a beep, and MATLAB should respond with

```
Using ASIO driver
Recording on device 0
Playing on device 0
Converting result to doubles
Any command or 'q' to quit, 'r' to rescale:
```

You have invoked `scope.m`, one of the AVNA Lab script files. If MATLAB instead responds with

```
??? Undefined function or variable 'scope'.
```

or with

```
??? Undefined function or method 'pa_wavplayrecord' for
input arguments of type 'double'.
```

this means that it cannot find the `scope.m` script or the pa_wavplay files. Make sure your files are set up as in Figure 2.1, and that the …`\MATLAB\AVNA Lab v1.0` folder and the …`\MATLAB\AVNA Lab v1.0\pa_wavplay` folder are on the search path.

Assuming that `scope.m` is now running, you should find that it has created a *Figure* window containing three plots: the played time series, the recorded time series, and the spectrum of the recorded time series. (The *Figure* window could be hidden behind the MATLAB desktop.) The second two plots both have eight curves, one for each channel. Take a moment to play with the toolbar items at the top of the *Figure* window. If you select one of the *Zoom* tools and then right-click, you will see that it is possible to zoom to a selection, or to zoom only horizontally or only vertically. The *Pan* tool lets you pan data within the plot frame, and the *Data Cursor* can be used to read specific values. The *Edit Plot* tool (arrow icon) can be used to move, resize, or delete sub-plots. MATLAB has many other interactive plotting features that can be accessed through the *Figure* menus or through the *Show Plot Tools* icon.

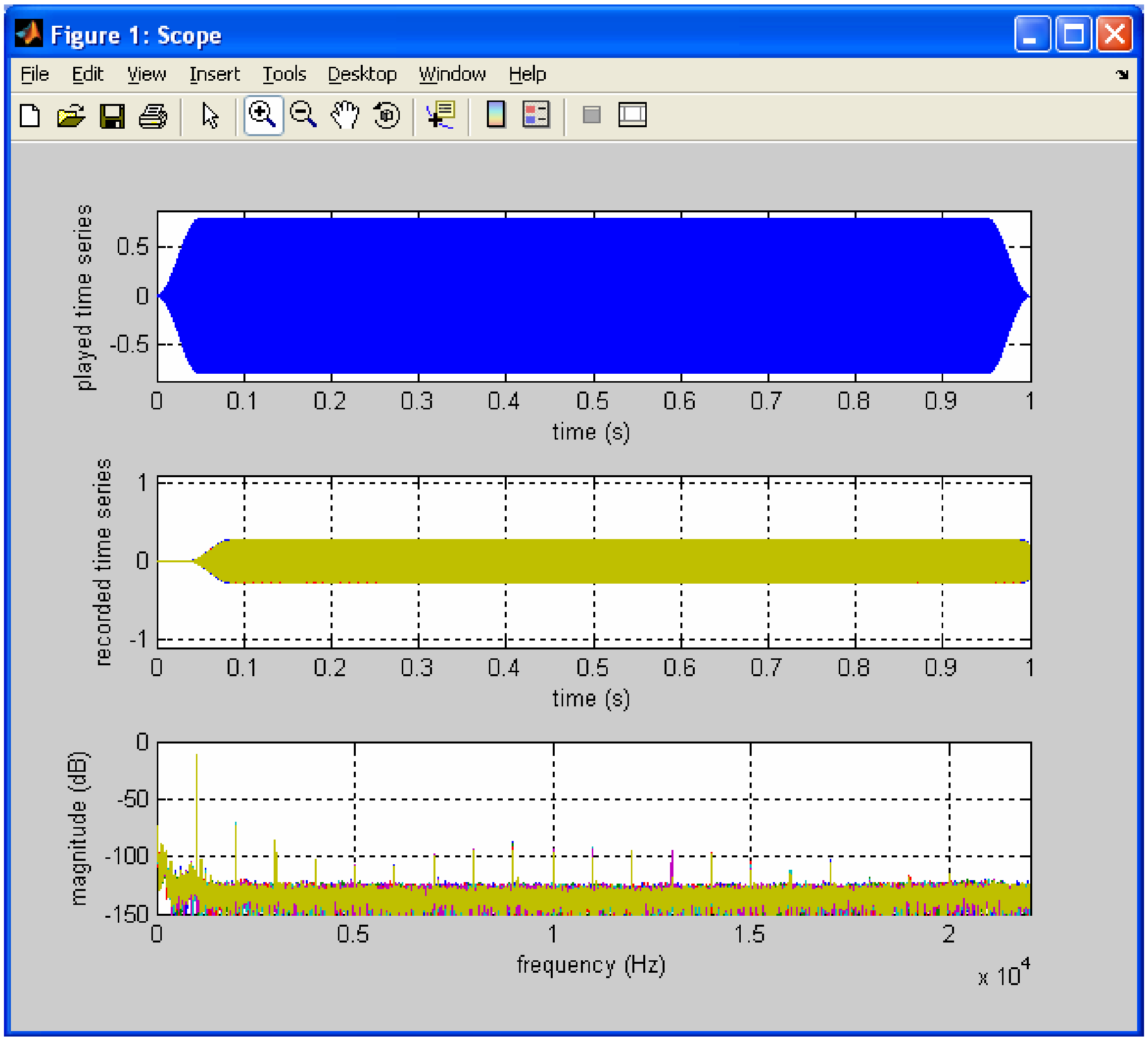


Figure 2.7. Plots created by `scope.m` script. The played sine-wave burst has 50 ms rise and fall envelopes to avoid creating clicks. The default value of `params.latency` should be adjusted so that the middle plot is aligned with the upper plot.

The `scope.m` script is an infinite loop. The beep will repeat and the plots will be updated each time you hit the Enter key. Zoom into the recorded time series plot and turn up the Preamp Input Gain knobs for channels 3 to 8 until the recorded amplitude is between 0.2 and 0.3 for each channel. The final gain settings should be around ten o'clock. Now enter `'r'` in the command window to rescale the plot axes, and you should see a plot similar to Figure 2.7. These gain settings are a useful starting point. If you explore more, you will discover that the input circuits of the digitizer begin to saturate at recorded amplitudes over 1.0.

The `scope.m` script will accept any valid MATLAB command from the keyboard. Try the effect of these commands: `level=0.5, freq=2000, dur=2.0`. The first of these changes the amplitude of the played sine-wave burst. `Freq` controls the frequency, and `dur` the duration. If you try higher levels you will find that the output circuits of the digitizer begin to saturate for played amplitudes over 1.0. You can view these variables and all others created by scope.m in the *Workspace* window. However, the workspace window does not show variable values while a script is running. Enter `q` at the command line to quit `scope.m` and see the changed values. Then enter `clear all` to clear the workspace variables.

There are three ways to get information about `scope` or any of the other AVNA Lab routines: Enter the command `help scope`, read about the routine in Chapter 6, or open the M-file in the *AVNA Lab v1.0* folder and read the comments.

The `scope.m` script relies on a set of hardware parameters that are contained in a data structure called `params`. This data structure is created with default values by the AVNA Lab function `defaults.m`. Enter in the *Command Window*

```
>> params=defaults
```

MATLAB responds with:

```
params =
    playdevice: 0
     recdevice: 0
    devicetype: 'asio'
         fsamp: 44100
         nchan: 6
     firstchan: 3
       latency: 0.0380
      micscale: 0.3000
       miclocs: [0 0.0782 0.2793 0.4396 0.6786 1]
     micoffset: 0.0500
```

The function `defaults` has returned a data structure named `params` with 10 fields, each assigned default values. (Your default values might be different.) To query one of the fields enter

```
>> params.nchan
ans =
     6
```

MATLAB responds with the value of the field. To change the value of a field enter

```
>> params.firstchan=1
params =
    playdevice: 0
     recdevice: 0
    devicetype: 'asio'
         fsamp: 44100
         nchan: 6
     firstchan: 1
       latency: 0.0380
      micscale: 0.3000
       miclocs: [0 0.0782 0.2793 0.4396 0.6786 1]
     micoffset: 0.0500
```

To reset the fields back to default values, invoke the function `defaults` again:

```
>> params=defaults;
```

The semicolon suppresses output to the *Command Window*. To check that the `params` fields have really been set back to defaults, double click on the `params` item in the *Workspace* window. The *Array Editor* will come up, showing the fields of the `params` data structure.

In Figure 2.7, the sine burst in the recorded time series starts about 50 ms late. (Your plot may show a delay or an advance, or neither.) In fact, the played time series always starts after the digitizer begins to sample its inputs by an amount called the latency time. The `params.latency` field (a time, in seconds) is used to correct for this effect. The AVNA Lab routines record for an extra time equal to `params.latency`, and then drop a time period at the beginning of the recorded time series equal to `params.latency`. Data will be wasted or invalid if `params.latency` is not set correctly.

To find the correct value for `params.latency`, invoke `scope` again and adjust `params.latency` until the second plot is aligned with the first to an accuracy of 10 ms. Use the command `params.latency=n.nnn` where `n.nnn` is the latency value you wish to set. Once you find the correct value, quit `scope` and set this value as the default for all users. Choose **Open…** from the **File** menu, navigate to the file `defaults.m` (in the *AVNA Lab v1.0* folder) and open it for editing. Edit the line `params.latency=n.nnn` to reflect the correct value, and save the file. Now, when you type `params=defaults` in the *Command Window*, you should see the updated `params.latency` default value, and when you invoke `scope` you should see the played and recorded time series aligned. Do not change any other fields of the shared copy of `defaults.m` that is located in the *AVNA Lab v1.0* folder. If you want to create a personal version of `defaults.m` with your own parameter values, just save it to your user folder, where it will 'overload' the shared version.

3. Relative Microphone Calibration

Reflection coefficients and S-parameters are ratios of amplitudes. Measuring them requires relative, but not absolute, calibration of the microphone channels. Both the relative sensitivity and phase shifts of the microphone channels must be known. In this section, we explain how to perform a relative calibration, and how to compare a new calibration with an older one.

If you have just finished the steps in the previous chapter, your system is already set up for calibration. Otherwise, check that your hardware is set up as described in Section 2.3 above, and run `scope` to check that `params.latency` is set correctly, as described in Section 2.4.

The principle of the calibration method is simple. The calibration cell has six-fold symmetry about the axis of the compression driver. If the pressure field in the cell also has six-fold symmetry, then each microphone is subject to exactly the same pressure. We measure the complex amplitude (or phasor) from each microphone at every frequency of interest, and use this data to normalize subsequent measurements. The phase reference is used is just the beginning of the time series. Because the latency varies somewhat, the overall phase has no meaning. However, the relative phases (and magnitudes) of the channels contain the important information for a relative calibration.

Check that the current directory is set to your user folder, and then enter this command in the *Command Window*:

```
>> calibrate(defaults,0.8,1.0,500,2000,50,'cal5c2k50e')
```

You should hear 50 tone bursts, each one second long, with frequencies ranging from 500 Hz to 2000 Hz. After `calibrate` finishes it will return a data structure to the temporary variable `ans`, and write a file in the current directory named `cal5c2k50e .mat`. (To help keep track of our files we like to include the frequency range in the file name. `5c2k50` means 500 Hz to 2 kHz, 50 steps.) As you can see, the `calibrate` function takes seven parameters. The first is the `params` data structure, which can be passed to `calibrate` using `defaults`, since this function evaluates to the `params` data structure. The next two parameters are the burst amplitude and duration. Then comes three parameters that specify the first frequency, the last frequency, and the number of frequencies at which to calibrate. The frequencies are distributed logarithmically over the specified interval. The last parameter is a string, the file name to which the calibration data will be written.

To check that the calibration file really exists, enter the command `load('cal5c2k50e')`. This will read the file and load its variables into the workspace. In this case, the file contains one data structure called `caldata`. You can explore the fields of `caldata` by double-clicking on it in the *Workspace* window. The `.phasors` field contains the measured phasors, the `.freqs` field is the frequency list, `.params` is the parameters data structure that was passed to `calibrate` when it was invoked, `.level` is the burst amplitude, `.duration` is the burst duration, and finally there is a time stamp field. The file has a `.mat` extension and can only be read by MATLAB. If you need to export data to other applications, MATLAB has several functions for writing text files.

MATLAB can readily plot the calibration data:

```
>> plot(caldata.freqs,abs(caldata.phasors))
```

(The `abs` function finds the magnitude of complex numbers.) However, AVNA Lab has functions for conveniently visualizing most of the data structures it creates.

Try this:

```
>> viewdata('cal5c2k50e')
```

You should see a plot similar to Figure 3.1. The top curve is the magnitude of the phasors plotted against frequency, with one curve for each channel. If you have set the system up as suggested in Chapter 2, all of the channels should have amplitudes close to 0.25 at 1 kHz. Several resonances are visible; these are coupled acoustic-mechanical resonances that involve both the air in the cell and the compression driver diaphragm. The resonances cause the pressure amplitude to vary over a wide range; however, there is little evidence of these resonances in the second plot, where each phasor has been normalized to the first-channel phasor. The last sub-plot shows the phase relative to the first channel.

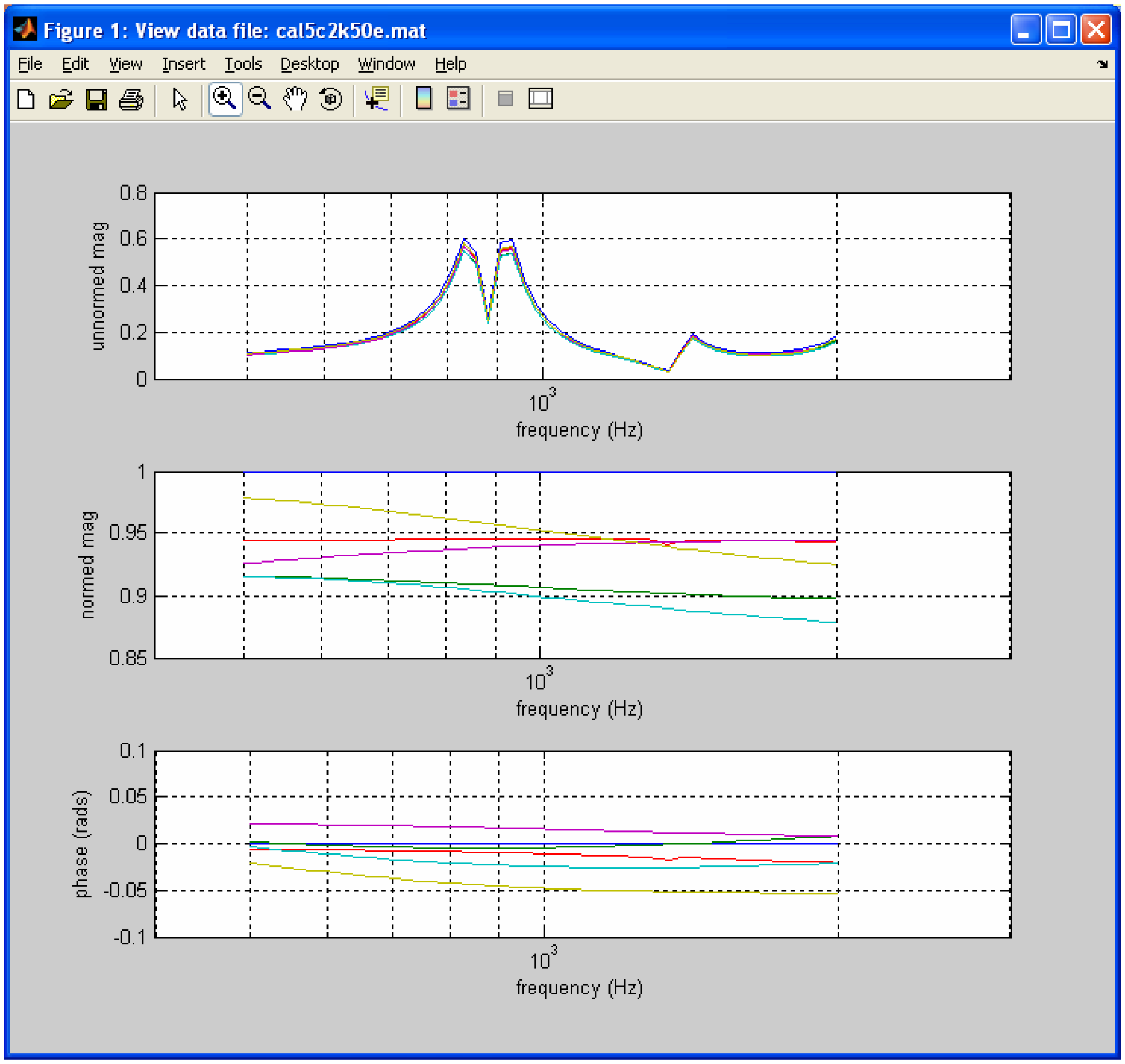


Figure 3.1. The calibration file `cal5c2k50e.mat` as displayed by `viewdata`.

Create a second calibration file by entering:

```
>> calibrate(defaults,0.8,1.0,500,2000,50,'cal5c2k50f')
```

Note that you do not have to type this in. You can use the up-arrow to scroll through previous commands, and edit only the characters that need to be changed. When `calibrate` finishes, you can compare the two calibration files with the command:

```
>> viewdata('cal5c2k50e','cal5c2k50f')
```

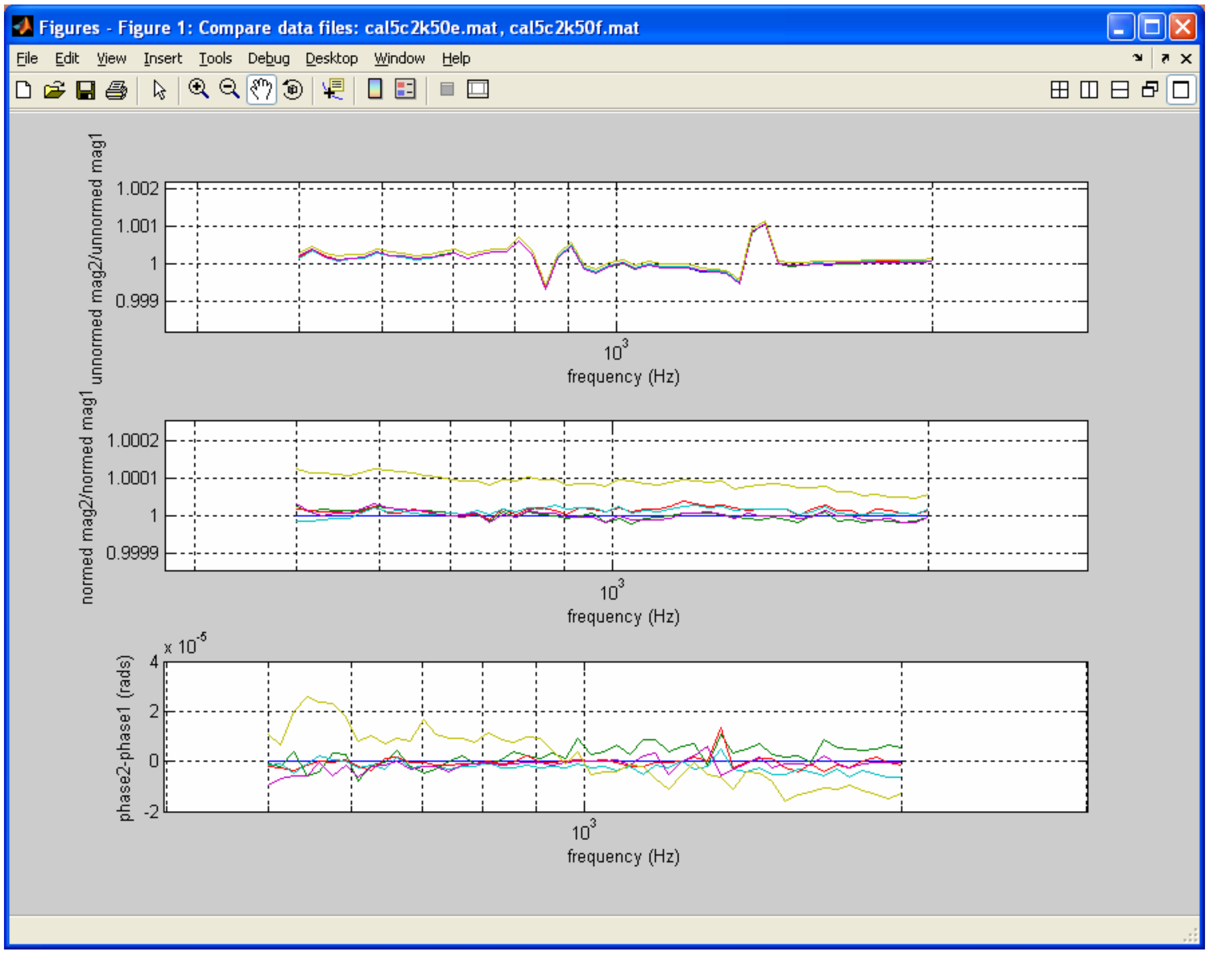


Figure 3.2. Comparing the calibration files `cal5c2k50e.mat` and `cal5c2k50f.mat`.

The first sub-plot is the ratio of the phasor magnitudes in the two files, without any normalization to the first channel. We see that all channels move together, with about 0.1 % variations of the magnitude ratio. If we are primarily interested in the reflection coefficient, magnitude variations that are common to all channels are not important. They can be removed by normalizing to the first-channel phasor in both files, as is done in the second sub-plot. Now we see that the magnitude ratios between channels have not changed by more than about a part in 10,000 between the two calibration files. The final plot shows that the phases relative to the first channel differ for the two calibration files by only a few times 10^{-5} radians.

The comparison shown in Figure 3.2 is between two calibration files recorded about 30 minutes apart in a laboratory environment, without removing the microphones from the calibration cell or changing anything else about the apparatus. This is an ideal situation. If the time period between calibrations is longer, or there are larger temperature drifts, or the microphones become contaminated, the calibration stability is bound to suffer.

Stability of the calibration depends upon the microphones, the buffer amplifier gains, the preamps in the digitizer, and the converter in the digitizer. If any of the Preamp Input Gain knobs are adjusted, or the microphones or buffer amplifiers are switched between channels, new calibration files will be needed. Note that the AVNA Lab software only allows a calibration file to be used with data taken at the same set of frequencies. Because of this, it is helpful to have calibration files for several different frequency resolutions.

4. Introductory Measurements

In this chapter we describe four simple measurements: reflection from a closed end, propagation through a straight section, absorption by cotton ball, and reflection from an open end. We recommend that new users try each of these measurements to become acquainted with AcousticVNA. If you are unfamiliar with the concepts of reflection coefficient and/or reference plane, see AW Sections 3.3 and 3.5.

4.1 Reflection from a Closed End

Set up the apparatus as shown in Figure 4.1. The compression driver is mounted at the left end of the microphone section, and the closed end is mounted at the right end. Assemble the two ARS-25 couplers, referring to Figures 2.3 and 2.4 above. The six microphones are mounted in the compression fittings on the microphone section. These fittings as the same as those on the calibration cell, shown in Figure 2.5.

Notice in Figure 4.1 that the microphones are irregularly spaced along the waveguide. This provides better frequency coverage than any regular arrangement. Assemble the system with the two closely-spaced microphones at the driven end, as shown in the photo. You could install the microphone section the other way around, but default parameters in the software would have to be changed. Similarly, the default software parameters assume that the first microphone is connected to channel 3 of the digitizer and is in the location closest to the driver, the next microphone is connected to channel 4 and in the next location, *etc.*. To avoid confusion, it is best to move the microphones directly from the calibration fixture to the microphone section without disconnecting them from the buffer amps or digitizer. Remember that the microphones are fragile, and you should avoid touching the sensitive ends.

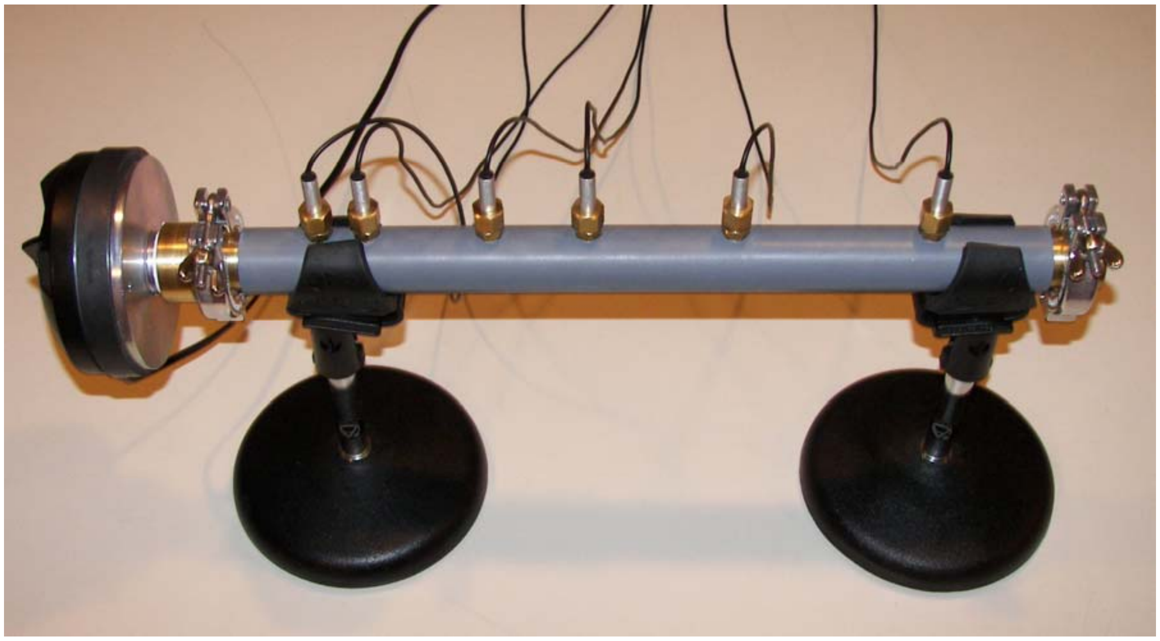


Figure 4.1. AcousticVNA set-up for studying reflections from a closed end. The two closely-spaced microphones are at the same end as the driver.

To collect data, launch MATLAB, set the current directory to your user folder, and enter this line in the *Command Window*:

```
>> acquire(defaults,0.8,1.0,'cal5c2k50e','dat5c2k50closed')
```

The `acquire` function is very similar to the `calibrate` function, and it produces a similar data structure. The first three arguments are the `params` data structure (supplied by the function `defaults`), the burst amplitude and the burst duration. Next is the name of the calibration file that will be used. The last parameter is the output file name for the data. There are no parameters specifying the frequency list, because it is taken from the calibration file.

After `acquire` returns, enter the command `load(‘dat5c2k50closed’)` and inspect the data structure `phasordata` that appears in the `Workspace` window. The `.phasors` field contains the measured complex phasors for each microphone at each frequency, the `.freqs` field is the frequency list, `.params` contains the parameters data structure that was passed to `acquire`, `.level` is the burst amplitude, and `.duration` is the burst duration. The next field gives the name of the calibration file that was used, and finally there is a time stamp. The `phasordata` structure contains an ‘audit trail’; it includes the conditions under which the data was taken and points to the calibration file where the calibration conditions can be found. All of the files created by AVNA Lab have this feature.

If you want to add notes to a data structure, just enter

```
>> phasordata.notes='This file is for the Users Guide.'
>> save('dat5c2k50closed')
```

MATLAB is very weakly typed, so new variables and fields can be created simply by assigning something to them. Data structure fields are referred to by name in AVNA Lab, so if you add a field with a new name it will not cause problems.

The action of `acquire` is very similar to the action of `calibrate`. It plays a sine-wave burst, measures the phasor for each microphone, and then repeats at the next frequency. However, once the data has been collected, `acquire` calibrates the data using phasors in the calibration file. The calibration phasors are first divided by the first-channel calibration phasor at each frequency. The first-channel phasor thus becomes equal to one, and the other calibration phasors have magnitudes and phases that are due to differences in gain or phase-shift between the channels. Then, the data phasors are divided by their corresponding calibration phasors, removing these differences from the data.

The next step is to do a fit to the measured phasors to find the right-going and left-going wave amplitudes. (We always use right-going to mean away from the driver, regardless of the true orientation of the system.) The most general one-dimensional pressure wave is

$$p = A_R \exp(-ikx)\exp(-\alpha x) + A_L \exp(+ikx)\exp(+\alpha x), \tag{4.1}$$

where A_R and A_L are the right- and left-going wave amplitudes, and k is the wave number, related to the phase velocity v by

$$k = \frac{\omega}{v}. \tag{4.2}$$

The attenuation constant α models damping of the wave due to dissipation at boundaries. Equation 4.1 gives the pressure phasor at each position x. To find the time-dependent pressure, multiply by $\exp(+i\omega t)$ and take the real part.

The AVNA Lab function `wavefit` finds the wave amplitudes. Try this command:

```
>> wavefit(defaults,
              '3param','dat5c2k50closed','wav5c2k50closed3')
```

Several fitting methods are implemented. The one specified here, `'3param'`, describes the phase velocity v by a constant plus a small term proportional to $\omega^{-1/2}$, and the attenuation α by a single term proportional to $\omega^{1/2}$. These forms are motivated by the Kirchhoff-Helmholtz theory for the dissipation effects (see AW Section 3.5).

Load the output file by entering

```
>> load('wav5c2k50closed3')
```

and examine the data structure `wave` in the *Workspace* window. Most of the fields are self explanatory: `.Ramp` and `.Lamp` are the fit right- and left-going wave amplitudes, `.vphase` and `.atten` are the phase velocity and attenuation constant at each frequency, `.phasors` are the phasor data that were fit, `.freqs` is the frequency list, `.params` is the parameters structure from the from the data file, and the last two fields give the data file name and the name of the fitting method used. The .phasors field is included so that the fit residuals can be computed.

There is one other field called `.global`. Double-click on it to examine its contents. This is a cell array, a MATLAB structure that can contain mixed data types. It gives the names and values of the three fit parameters that describe the phase velocity and the attenuation. They are referred to as global parameters because they apply to all frequencies, while the wave amplitudes are fit separately at each frequency. The values in the `.vphase` and `.atten` fields are computed from the global parameters. (In other fitting methods, `.vphase` and `.atten` are fit locally at each frequency.) There is an AVNA Lab function called `viewwaves` for examining files containing the `wave` data structure. However, let's move on to the reflection coefficient.

The reflection coefficient $S(x)$ is the ratio of the left-going pressure amplitude to the right-going pressure amplitude:

$$S(x) = \frac{A_L \exp(+ikx)\exp(+\alpha x)}{A_R \exp(-ikx)\exp(-\alpha x)}, \tag{4.3}$$

and is a function of position. The location x at which the reflection coefficient is evaluated is known as the reference plane. Enter the command:

```
>> reflect('wav5c2k50closed3')
```

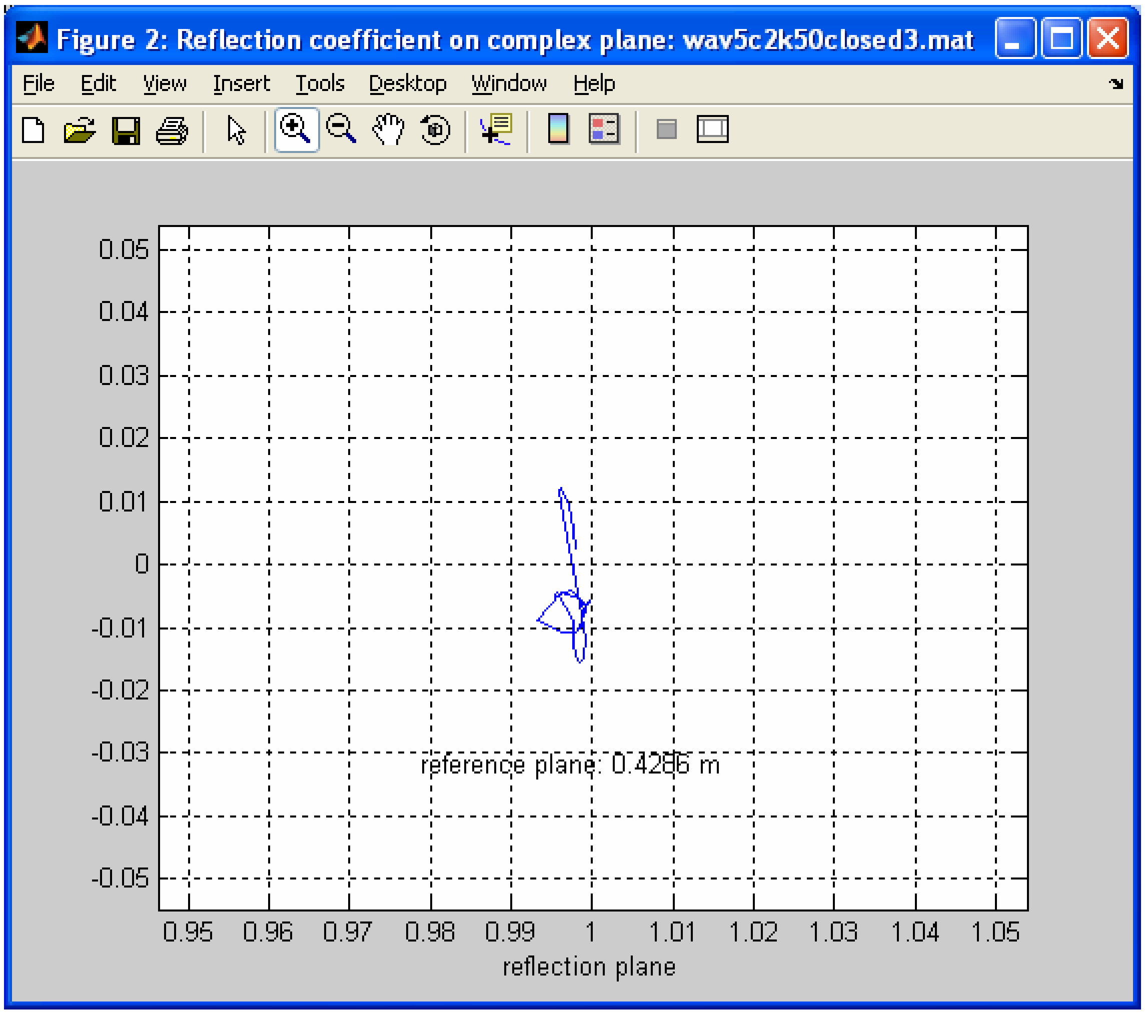


Figure 2: Reflection coefficient on complex plane: wav5c2k50closed3.mat
File Edit View Insert Tools Desktop Window Help
0.05
0.04
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
reference plane: 0.4286 m
0.95 0.96 0.97 0.98 0.99 1 1.01 1.02 1.03 1.04 1.05
reflection plane

Figure 4.2. Reflection coefficient for a closed end, with the reference plane at the location of the closed end.

The reflection coefficient will be plotted on the complex plane, and as magnitude and phase versus frequency. If the default reference plane location is coincident with the closed end, your result should be within a few percent of +1 on the complex plane, as shown in Figure 4.2. A reflection coefficient of +1 means that the pressure wave is reflected off the closed end without any change in amplitude or phase.

If your result is more spread-out in phase than shown in Figure 4.2, you can try to improve it by adjusting the location of the reference plane. While `reflect` is running you can enter a new value for the reference plane location and the plots will update. The reference plane is measured (positive to the right) from the face of the flange at the left end of the microphone section. Changes of just a tenth of a millimeter make a visible difference in the reflection plot. Since the closed end is the simplest acoustic termination, we will use it as a reference for other measurements. Find the reference plane location where the reflection data are closest to the point +1 on the complex plane and record it. Later, we will compare other reflection plane locations to this value. If you wish, you can edit `reflect.m` to change the default reference plane location. Store the edited version in your user folder. (Be sure to quit `reflect` by typing `q` before you do this. Most featured of the MATLAB desktop do not work when a program is running.)

Try moving the reference plane 1 cm to the right. You will see an arc at positive angles, and a linear variation of the phase with frequency. The fit phase velocity can viewed with `viewwaves` or by plotting the `.vphase` field of the wave structure. It is slightly frequency dependent. To a first approximation you can use an average value.

If you want to preserve your reflection data, give `reflect` an output file name as a second input parameter. Once you have adjusted the reference plane to the value you want, type `s` to save the reflection data to the output file. As before, you can `load` the output file and examine the data structure it contains.

4.2 Straight Section and Wave Attenuation

Assemble AcousticVNA as shown in Figure 4.3, with a straight section after the microphone section, and the closed end following the straight section. For this example, we will automate the data collection and analysis. Open the *Editor* window in the MATLAB desktop, and click on the New M-File icon to create a blank document. Enter these commands in the file, and save it to your user folder with the name `quick.m`:

```
% quick.m script acquires phasor data, fits wave amplitudes, and
% plots the reflection coefficient.

acquire(defaults,0.8,1.0,'cal5c2k50e','dat5c2k50extend')
wavefit(defaults,'3param','dat5c2k50extend','wav5c2k50extend3')
reflect('wav5c2k50extend3')
```

Check that the current directory is set to your user folder, and then try the script by entering `quick` in the *Command Window*. It works, but there is a problem. If you run it a second time, you will get errors because the output files will already exist.

Here is a better way:

```
function quicker(base)
% quicker.m is a function that acquires phasor data, fits wave
% amplitudes, and plots the reflection coefficient.  It takes the
% base name for the output files as input. Base must be a string.

datafile=['dat' base];
wavefile=['wav' base];

acquire(defaults,0.8,1.0,'cal5c2k50e',datafile)
wavefit(defaults,'3param',datafile,wavefile)
reflect(wavefile)
```

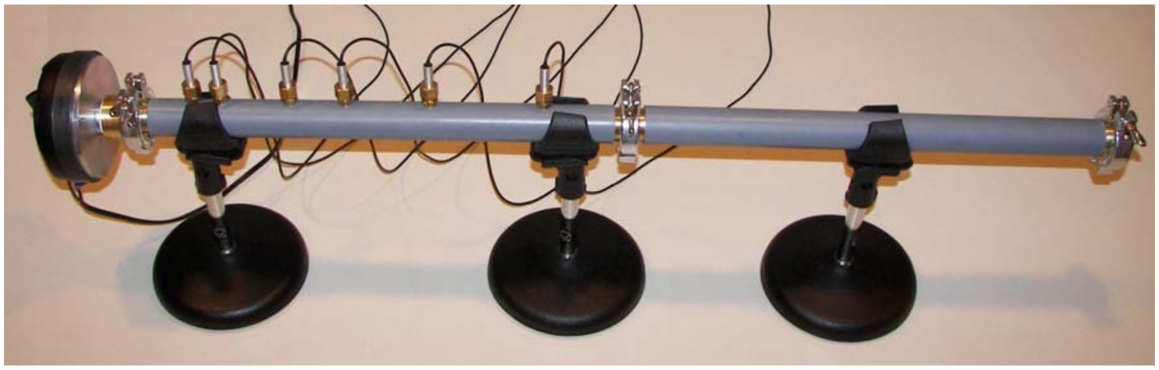


Figure 4.3. AcousticVNA set-up for studying propagation through a straight section with a closed end.

Now we pass the string `base` and construct full file names using the MATLAB concatenation operator `[]`. With this function, you can collect data and make plots of the reflection coefficient simply by entering `quicker('firsttry')`, and then the next time, `quicker('nexttry')`.

Typical reflection data are shown in Figure 4.4. Note that the phase plot always starts in the interval $(+\pi, -\pi)$, regardless of the phase delay actually present between the reflected and incident wave at the starting frequency. Because of this, one should be cautious about using the absolute value of the phase delay to infer the line length.

However, we can relate the phase slope to the line length. The general formula for translating a reflection coefficient is (see AW, Equation 3.14, 3.30)

$$S(x) = S(l)\exp(2i\frac{\omega}{v}(x-l))\exp(2\alpha(x-l)). \tag{4.4}$$

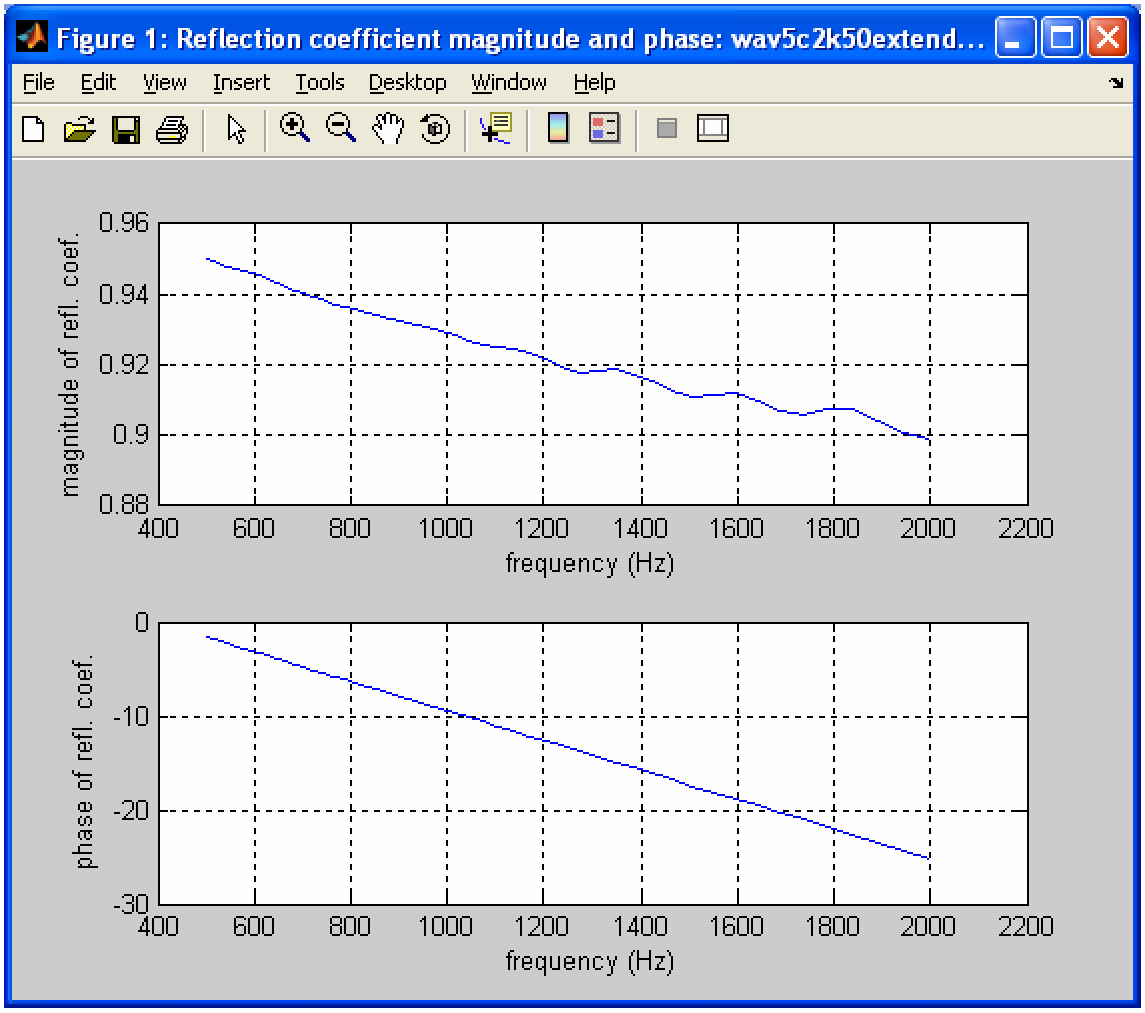


Figure 4.4. Magnitude and phase of the reflection coefficient for a straight section with a closed end. The reference plane is at 428.6 mm, the location of the closed end without the straight section. The straight section moves the closed end 429.0 mm to the right.

Suppose l is the location of the closed end, so $S(l) = +1$, and x is the reference plane. Then the phase slope is

$$\frac{d}{df}\arg(S(x)) = 4\pi\frac{x-l}{v}. \tag{4.5}$$

Using the data cursor to read from the graph gives a measured phase slope of $-0.01570\,\text{rad/Hz}$. With the fit phase velocity v of 341.2 m/s, this corresponds to $l - x = 426.3\,\text{mm}$. Inserting the straight section moves the closed end a distance equal to the flange-face to flange-face length of the straight section, plus 4 mm for the additional seal. The measured distance was $429 \pm 1\,\mathrm{mm}$, close to the value inferred from the phase slope.

The data can also be used to measure the attenuation constant α in the straight section. Again using (4.4) and $S(l) = +1$, we find

$$\left|S(x)\right| = \exp(2\alpha(x-l)). \tag{4.6}$$

From the graph we have $\left|S\right| = 0.899$ at 2 kHz, which implies $\alpha = 0.124\ \mathrm{m}^{-1}$ in the straight section. The fit value in the microphone section at 2 kHz is $\alpha = 0.120\ \mathrm{m}^{-1}$. These can both be compared to the Kirchhoff-Helmholtz prediction, which is $\alpha = 0.1087\ \mathrm{m}^{-1}$ for a 25 mm I.D. tube at 2 kHz and standard conditions (see AW, Section 3.5). The slightly higher dissipation values seen in the microphone and straight sections may be due to surface roughness. The Kirchhoff-Helmholtz theory assumes a perfectly smooth surface.

4.3 Absorption by Cotton Ball

Next we return to the configuration shown in Figure 4.1, but with a small cotton ball (about half of a standard cosmetic cotton ball) just in front of the closed end. The reflection data are shown in Figure 4.5. The termination is now much more absorbing than it was without the cotton. Notice that the phase is negative, corresponding to an increase in the effective length of the line by about 3 mm.

It may seem surprising that adding material can increase the effective length. The cotton ball is near a pressure anti-node and a velocity node. It influences the reflection primarily by changing the effective compressibility of the medium, rather than by changing the inertia. The heat capacity of the cotton tends to make the gas more nearly isothermal, which softens the equation of state and increases the compressibility. This has an effect similar to increasing the volume near the end of the line, or the line length. The theory of sound propagation in porous media is rather involved and it can be difficult to make convincing comparison with experiments, but measurements like this have practical importance for characterizing acoustic materials used for sound damping (see references in AW, Section 1.1).

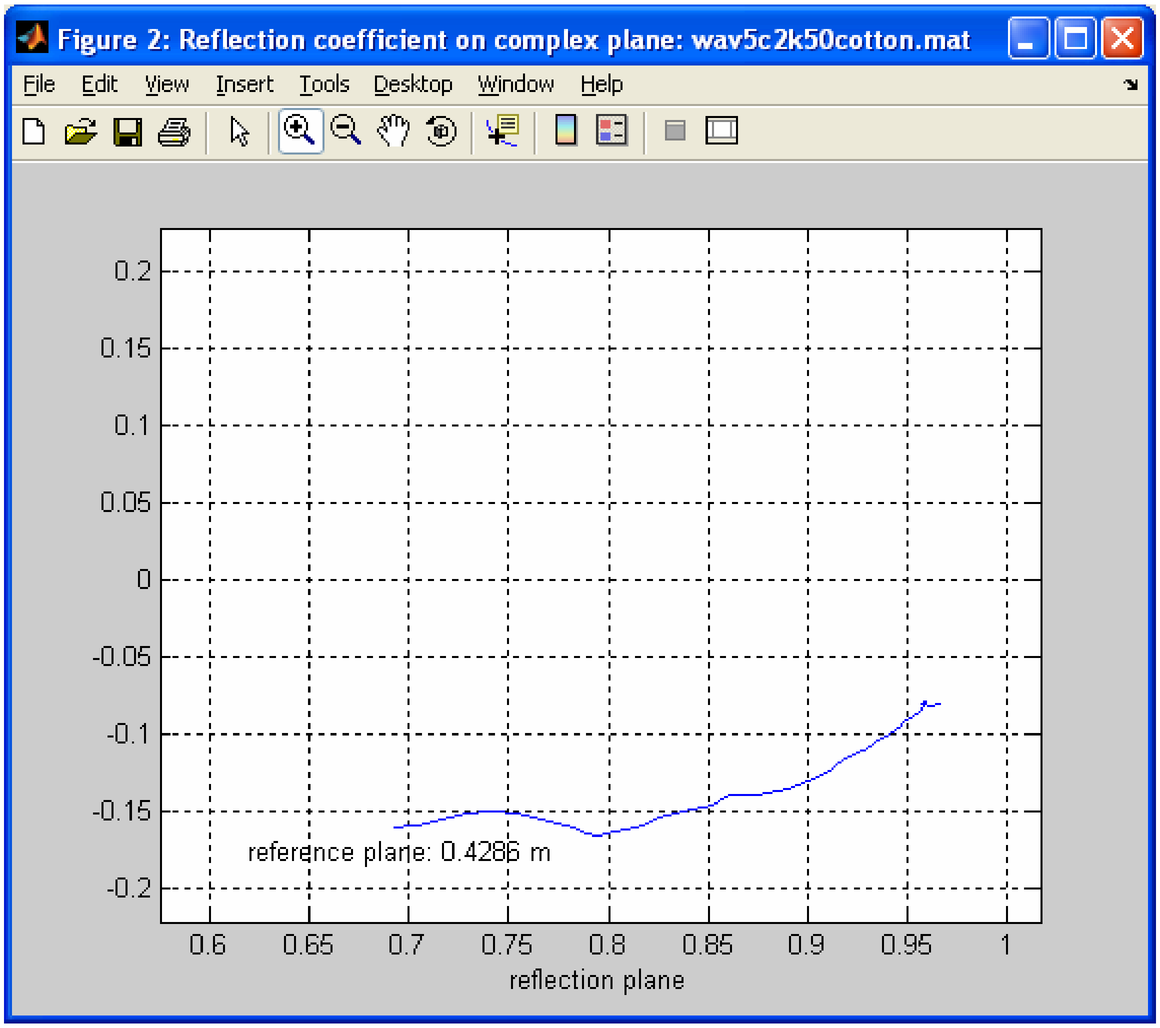


Figure 4.5. Reflection coefficient for a closed end with a cotton ball. The reference plane is at the closed end. The trajectory starts at 500 Hz on the right and ends at 2 kHz at the left.

4.4 End Correction and Radiation Resistance of an Open End

Closed ends are nearly ideal in the sense that the reflection coefficient at the plane of the closed end is almost exactly +1. (There are very small corrections due to the thermal boundary layer.) However, the plane of an open end is only approximately a location with a reflection coefficient of −1 (see AW Section 3.2 for discussion and references).

Figure 4.5 shows the reflection coefficient, with the apparatus set up as in Figure 4.1 except with the closed end (and its seal) removed. The reference plane has been adjusted to place the data as close to the real axis as possible. The resulting location, 428.0 mm, is 8.4 mm beyond the plane of the open end, or the face of the flange. (The cavity in the closed end is 5 mm deep, and the seal is 4.0 mm thick. Thus the open end is at 428.6−9.0=419.6 mm.) At low frequencies, the open end does have an almost ideal reflectance of −1, but with an end correction of 8.4 mm, or 0.67*a*, where *a* is the inside radius of the waveguide. Detailed calculations show that the end correction for a tube with an infinite flange is 0.85*a*, and for a tube without a flange is 0.61*a*. Our result is in between these values, as expected for a small flange.

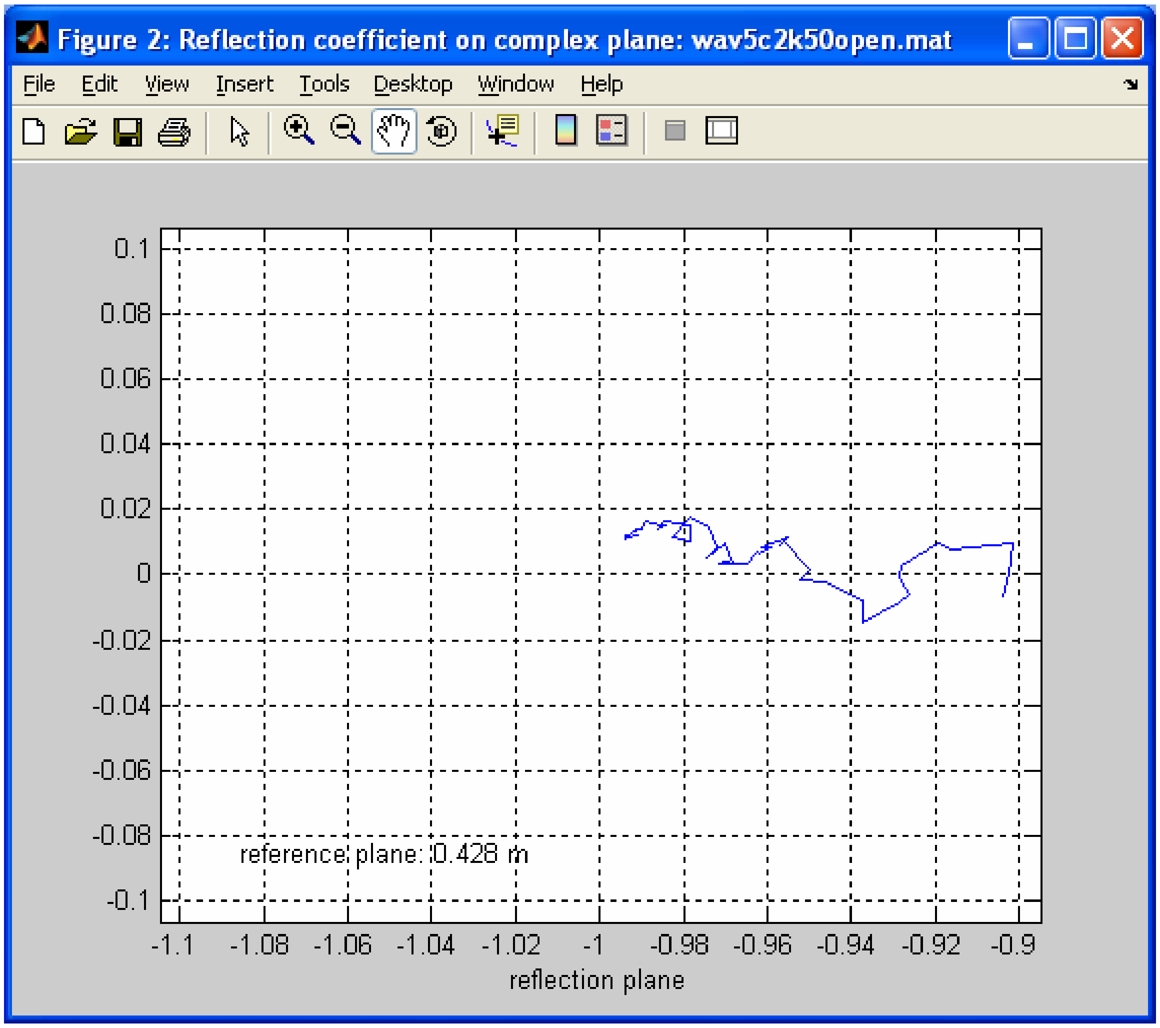


Figure 4.6. Reflection coefficient on the complex plane for an open end, with the reference plane at 428.0 mm, or 8.4 mm beyond the face of the flange. The trajectory starts very close to the point −1 at 500 Hz and then moves to the right, ending at 2 kHz..

When the frequency is increased, the magnitude of the reflection coefficient begins to decrease. This is due to the power lost to radiation, or equivalently, to the radiation resistance R. Theory predicts

$$R = bZ_0(ka)^2 + \ldots \tag{4.7}$$

to lowest order in *ka*. Here, Z_0 is the line impedance and *b* is a numerical constant, equal to 1/2 for an infinite flange and 1/4 for a tube without a flange.

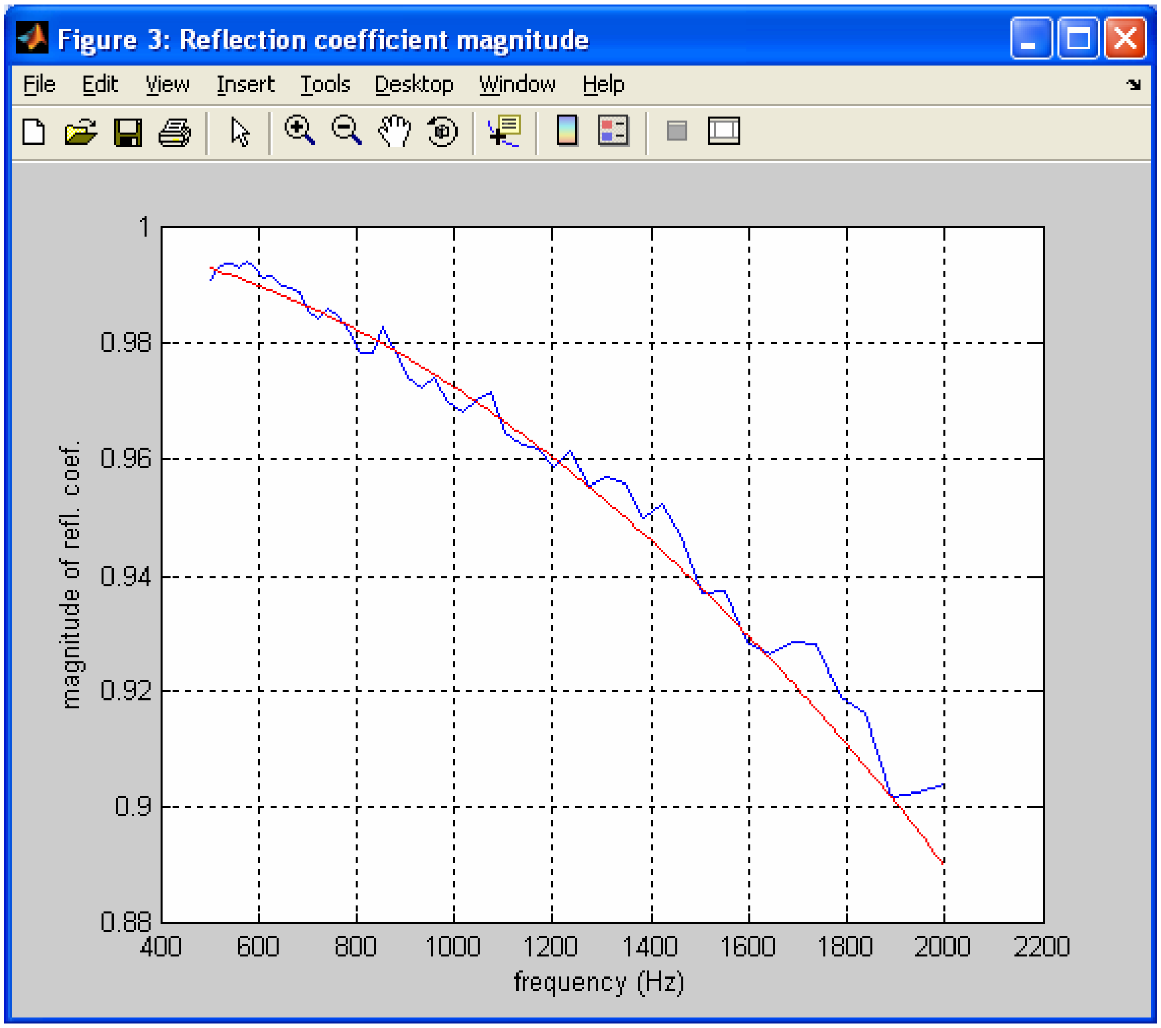


Figure 4.7. Magnitude of the reflection coefficient for the open end (blue curve) with the reference plane at 428.0 mm. Plotted in red is the predicted magnitude from (4.8) with $v = 341.6$ m/s and $b = 0.26$.

The corresponding reflection coefficient is

$$S = \frac{R - Z_0}{R + Z_0} \approx -1 + 2\frac{R}{Z_0} = -1 + 2b\left(\frac{\omega a}{v}\right)^2 . \tag{4.8}$$

As shown in Figure 4.7, this fits the data well with $b = 0.26$, a value close to the value given by the lowest order theory for a tube without a flange.

5. Applications

In this chapter we briefly describe some possible applications of the AcousticVNA system.

Impedance Discontinuity

There are two ways to make an impedance discontinuity. The first, by changing the waveguide diameter, is illustrated in Figure 1.2(b). In this case the impedance change is not abrupt because the acoustic flow must spread continuously at the discontinuity. A more abrupt discontinuity can be created by placing a thin membrane across a plane perpendicular to the axis of the waveguide, and then filling each side of the membrane with a different gas. Measurements can be made in reflectometer or VNA modes.

Woodwind Tone Holes

Woodwinds are cylindrical acoustic waveguides with one of several kinds of acoustic amplifiers at the blowing end (reed, lip-reed, or air-jet). The tone holes (finger holes) vary the effective acoustic length of the tube. They can be studied with AcousticVNA by attaching a straight tube with holes to a microphone section, and observing the reflection for various fingerings. Alternately, the complete S-matrix of a tone hole can be measured. Effects of forked fingerings and of interactions between closely-spaced tone holes can also be observed.

Ocarina

The ocarina is a globular or vessel flute. There are several holes which can be covered or uncovered to change the pitch, but, quite unlike the situation with an ordinary woodwind, the pitch is largely independent of the location of the holes. The vessel and holes together function as a Helmholtz resonator. The holes act together like a single variable inertance, which resonates with the compliance of the vessel. The inertance of holes of various geometries can be studied with AcousticVNA by making a closed end with a hole on the axis. Both the diameter of the hole and its length are important. Smaller holes have more inertance, but their losses are also greater, lowering the Q of the resonator.

Horns

The theory of acoustic horns is old and well developed. Horns for loudspeakers attempt to match the waveguide to the radiation field with small reflections over some bandwidth. Musical instrument horns are designed with larger reflections so that the instrument will have high-Q resonances and definite pitch. Both the effective length and the radiation resistance can readily be measured by reflectometry. The acoustic environment outside the horn can be important and difficult to control, especially for large, low-reflection horns.

Periodic Media and Acoustic Bandgaps

Waves propagating in media with periodic variations of impedance exhibit band-gaps, or frequency intervals where there are no propagating modes. This effect should be observable in a tube with periodically varying diameter. If an absorber is placed after the periodic section, the tube will be highly reflective inside the band-gap and absorptive outside.

Optical Modes

In solid state physics, and optical mode is a propagating lattice distortion with finite frequency in the limit where the wavelength goes to infinity. In uniform acoustic waveguides, all propagating modes above the lowest one are optical. Propagation of such modes can be observed by coupling the microphone section to a tube of larger diameter. A 25 mm I.D. cylindrical microphone section will be single-mode up to 8 kHz, but a tube with larger inside diameter *d* will begin to propagate a mode with one diametrical node at a frequency that is lower by a factor of (25 mm)/*d*. To excite the first optical mode, the transition between the microphone section and the larger tube must break cylindrical symmetry. All optical modes are dispersive, and any transmission or reflection measurement that includes contributions from higher modes will also be sensitive to the dispersion relation.

Resonators

The theory of acoustic cavities is analogous to that for electromagnetic cavities, and many of the same kinds of experiments can be done. A cavity of any shape (cylindrical, rectangular, spherical, *etc*.) can be coupled to the reflectometer through an aperture, the size of which controls both the amount of coupling and the degree to which the reflectometer disturbs the modes that would be present without the coupling. Lumped-element models are very helpful for describing cavities and cavity coupling. At very weak coupling (small aperture), the cavity Q will be limited by the thermal and shear boundary layers, or in exceptional cases, by bulk viscosity. At stronger coupling the Q may be controlled by radiation into the reflectometer. When the Q has equal contributions from these two sources, the coupling is said to be critical. Acoustic cavities have been used to make very precise measurements of the thermodynamic properties of gasses and liquids.

Gas Physics

The phase velocity in a uniform waveguide is very close to the free-space sound velocity:

$$v_S = \sqrt{\frac{\partial p}{\partial \rho}},$$

where the derivative of pressure with respect to density is taken under adiabatic conditions. For an ideal gas, v_S is independent of pressure at constant temperature and proportional to the square root of absolute temperature at constant pressure. The small pressure dependence of the sound velocity at constant temperature in real gasses can be used to measure the second virial coefficient, which parametrizes deviations from ideality in the equation of state. The sound velocity also varies inversely as the square root of the molecular weight, so it depends dramatically on gas composition at constant temperature and pressure.

The attenuation in a waveguide provides access to the shear viscosity and the thermal diffusivity, through the shear and thermal boundary layer thicknesses. However, these two effects are of similar magnitude and are difficult to disentangle. In a cavity, Q values of the different modes can depend differently on the two boundary layer thicknesses. In particular, the breathing mode of a spherical cavity has no tangential acoustic velocity at the boundary, and thus no losses due to the shear viscosity.

Wall Attenuation

The Kirchhoff-Helmholtz attenuation theory applied to perfectly smooth walls. What happens if the walls are very rough, say lined with sand paper? Does it have the same frequency dependence as it has with smooth walls?

Absorbing Materials

Many kinds of sound absorbing materials are used in architectural acoustics and in other engineering areas, such as vehicle design. These include, plastic foams, fabrics, porous ceramics, and various kinds of composite structures. Some commercial sound absorbing materials are characterized by reflectometry. The parameter that is usually reported is given the symbol α and called the 'normal incidence sound absorption coefficient.' In our notation it is given by

$$\alpha = 1 - |S|^2$$

Terminators and Attenuators

In electronics, resistive terminators and attenuators with impedances and connectors matched to standard transmission lines are commonplace. They can have excellent properties and work over many decades of frequency. For example, co-axial terminators are available with return loss (magnitude squared of the reflection coefficient) less than -50 dB from dc to 5 GHz. Nothing like this exists for acoustic waveguides; even rather crude terminations working over several octaves are difficult to make and bulky. In the experimental literature one typically finds long line sections stuffed with absorbers such as cotton or steel wool. Is there a better way to make these components?

Acoustic Amplifiers

Woodwind musical instruments are acoustic oscillators. Like all oscillators, they have a gain element and a resonator. The resonator is a cylindrical waveguide, in some cases straight and in some tapered. The amplifier is either an air-jet (flutes, recorders, whistles, *etc*.) a reed (clarinet, oboe, saxophone) or a lip reed (trumpet, trombone). To a first approximation, the resonator is open at the amplifier end for an air-jet instruments and closed for reeds and lip reeds. The blowing end of a woodwind can be attached to a reflectometer. When pressure is applied to the mouthpiece, a reflection coefficient magnitude greater than one indicates that the amplifier has gain. Measurements can be made as a function of frequency, blowing pressure, and incident amplitude. The reed and lip reed amplifiers are very non-linear, so strong harmonics will be present in the reflected wave.

Vector Network Analyzer and S-parameters

The AcousticVNA hardware can be used with any number of ports, and the Presonus Firepod and FP-10 can be daisy-chained up to three units, for up to 24 microphone channels. However, the AVNA Lab v1.0 software only supports a single port.

Pulse Propagation

The hardware allows for arbitrary waveforms to be played, including pulses. However, you should not expect the measured pressure pulse to duplicate the played waveform, because of frequency dependence in the compression driver. It may be possible to write software that adjusts the played waveform adaptively to achieve a desired pulse shape at the first microphone. Alternately one could try to characterize the transfer function from the played waveform to the acoustics pressure, and use this information to compute waveforms that will generate desired acoustic pulse shapes. To avoid problems from the latency, it would be necessary to record the voltage applied to the compression driver with an unused microphone channel.

Stochastic Excitation

In so far as the system of interest is perfectly linear, there is no need to study just one frequency at a time, as is done with the AVNA Lab v1.0 software. The opposite approach is to use a very broadband waveform, such as random or pseudo-random noise. The most commonly specified standard method for characterizing acoustics materials uses a stochastic waveform (ASTM Standard E 1050-98), and a two-channel FFT analyzer to extract correlations between microphones. This method, or a version of it extended to more microphones, could be implemented with the AcousticVNA system.

Flow

In a flowing medium, the sound velocity relative an external stationary reference frame is offset by the vector flow velocity. One approach to studying such situations would be to inject gas between the compression driver and the microphone section, let it flow through the microphone section, and then out through an open end. This allows one to study the effect of changes in flow without any change in geometry, and avoids the possibility of damaging pressure build-up.

Coupled Mechanical-Acoustic Systems

The strong resonance present in the calibration cell is due to a coupled system involving the loudspeaker diaphragm, the compliance of air in the cell, and also the external electrical circuit. To study these effects in a controlled way, the apparatus can be set up as a reflectometer and a second compression driver can be used at the object under test. The reflection coefficient of the second driver will depend on the electrical impedance connected to its terminals.

Finite-Element Calculations

Finite element calculations can be used to solve the Helmholtz equation and to predict reflection coefficients of both simple and complex geometries. Calculations in 2-d are simpler and more precise than those in 3-d. For cylindrically symmetric situations, the Helmholtz equation can be solved on a constant-theta plane in 3-d cylindrical coordinates. The PDE Toolbox in MATLAB has enough flexibility to solve such problems. Problems involving radiation into free space require some kind of absorbing boundary condition, so they are more difficult than closed problems. Attenuation can be handled most simply by adding a dissipative bulk term to the equations, but dissipative boundary conditions can also be implemented.

6. AVNA Lab v1.0 Software Reference

In this chapter we describe the MATLAB scripts and functions that comprise AVNA Lab v1.0. The routines are primarily aimed at measuring the reflection coefficient as a function of frequency by playing and recording sine-wave bursts. One burst is played and recorded at each frequency in a list. The frequency list is chosen at the time of calibration, and then inherited by the other routines from the calibration file. One can create any number of calibration files, each with a different frequency list.

This preliminary version of the code is limited to one-port measurements.

6.1 Overview

The main data acquisition and analysis work-stream depends on four functions which are used in the sequence: `calibrate` → `acquire` → `wavefit` → `reflect`. Each of these routines gets an input file from the previous routine in the stream, and provides an output file for the next routine, except that `calibrate` needs no input file. To identify the different kinds of files, we use the name of the variable that contains the file name string, which is always the same for a given kind of file. For example, the variable containing the name string of a calibration file is named `calfile`, and we refer to this kind of file as a `calfile`. In cases where we need to refer to more than one file of this kind, we use `calfile1`, `calfile2`, *etc.*. Of course, when the routines are actually used, variables in the argument lists that contain file names can be given any name, as can the files themselves.

The input and output files for each of the main work-stream routines are given in Table 6.1. Each kind of file contains a single data structure. To help identify the different kinds of structures, we give each one a fixed name. For example, the data structure contained in a `calfile` is called `caldata`. As for files, we append an integer if we need more than one structure of a particular kind; for example `caldata1` and `caldata2`.

The process starts with `calibrate`, which performs a relative calibration of microphones in a calibration cell. `Calibrate` creates a `calfile`, which contains the data structure `caldata`. Next, `acquire` plays sine-bursts and measures the resulting phasor (complex amplitude) of each microphone at each frequency, using the relative calibration to correct the data. The results of `acquire` are stored in a `datafile`, which contains the data structure `phasordata`. The routine `wavefit` uses the results of `acquire` to find left- and right-going complex wave amplitudes, the phase velocity, and the attenuation constant at each frequency. These results are saved in a `wavefile` containing the structure wave. Finally, the results of wavefit are used by reflect to compute the reflection coefficient, which is saved in a reflfile containing the structure refl.

Function	Input File	Input Data Structure	Output File	Output Data Structure
calibrate	--	--	calfile	caldata
acquire	calfile	caldata	datafile	phasordata
wavefit	datafile	phasordata	wavefile	wave
reflect	wavefile	wave	reflfile	refl

Table 6.1. Main routines with their files and data structures.

In addition to the main work-stream routines, there are four other top-level routines. The script scope is used to interactively play, record, and view single sine-bursts of any frequency, amplitude, or duration. It is used for hardware set-up and debug. The function defaults creates a data structure called params whose fields describe the current hardware configuration. This data structure is needed as input to several of the other routines. Files of the types calfile and datafile may be viewed or compared using viewdata, and files of the type wavefile may be viewed using viewwaves.

There are five other functions which are used by the top-level routines, and which can also be called directly by the user to perform lower-level functions. These are burst, playrecord, fourier, sweep, and spectrum.

Finally, there are four functions in a private folder. In MATLAB, functions in private folders are only for use by routines in the folder directly enclosing the private folder. Three of the private functions (wavefcn2, wavefcn3, wavefcnlocal) are called by wavefit, and the other (reflfcn) is called by reflect.

All of the AVNA Lab routines read and write files in MATLAB's own .mat data format. MATLAB includes mechanisms for accessing .mat files from other software systems, and for writing files in other formats, should the need arise. AVNA Lab reads and writes files only to the current directory. Name strings for the files are always given without the .mat extension.

In AVNA Lab, phasors (complex amplitudes) are converted to sinusoids by multiplying by $\exp(+i\omega t)$ and taking the real part.

6.2 Data Structures

<u>`params`</u>

Contains parameters describing the hardware configuration.

Created by: `defaults`

Fields:

`params.playdevice`

integer, the play device number used by `pa_wavplay` drivers.

`params.recdevice`

integer, the record device number used by `pa_wavplay` drivers

`params.devicetype`

device type used by pa_wavplay drivers, see help file for `pa_waveplay`

`params.fsamp`

integer, sampling rate for play and record, allowed values determined by hardware

`params.nchan`

integer, number of record channels

`params.firstchan`

integer, channel number of first record channel, other channels are consecutive

`params.latency`

The latency correction parameter, which should be set to the actual time in seconds that the start of playing is delayed relative to the start of recording, when simultaneous play and record are desired.

`params.micscale`

distance from first mic to last mic in meters

`params.miclocs`

A row vector of dimensionless mic locations. Must start with 0.0, and end with 1.0, and must have `params.nchan` elements

`params.micoffset`

The distance in meters from the fiducial plane to first microphone. The fiducial plane is the face of the left flange (end closest to driver) of the microphone section.

<u>`caldata`</u>

Calibration data structure, contained in a `calfile`.

Created by: `calibrate`

Fields:

`caldata.phasors`

complex amplitudes for each frequency and channel
number of rows = number of frequencies

number of columns = number of channels

`caldata.freqs`

column vector of frequencies

`caldata.params`

the parameters data structure passed to `calibrate`

`caldata.level`

burst amplitude

`caldata.duration`

burst duration in seconds

`caldata.timestamp`

time of creation of the file containing this structure

<u>`phasordata`</u>

Data structure for measured microphone phasors, contained in a `datafile`.

Created by: `acquire`

Fields:

`phasordata.phasors`

complex amplitudes for each frequency and channel
number of rows = number of frequencies
number of columns = number of channels

`phasordata.freqs`

column vector of frequencies, taken from the calibration file

`phasordata.params`

the parameters data structure passed to `acquire`

`phasordata.level`

burst amplitude

`phasordata.duration`

burst duration in seconds

`phasordata.calfile`

string, name of the `calfile` passed to `acquire`

`phasordata.timestamp`

time of creation of the file containing this structure

<u>`wave`</u>

Data structure for the fit wave amplitudes, contained in a `wavefile`.

Created by: `wavefit`

Fields:

`wave.Ramp`

Complex wave amplitude for the right-going wave, a column vector with one entry for each frequency.

`wave.Lamp`

Complex wave amplitude for the left-going wave, a column vector with one entry for each frequency.

`wave.vphase`

Column vector containing the phase velocity at each frequency. In some methods it is fit independently at each frequency; in others it is fixed or calculated from a number of fit quantities.

`wave.atten`

Column vector containing the attenuation constant at each frequency. In some methods it is fit independently at each frequency, in others it is calculated from a number of fit quantities.

`wave.phasors`

The fit phasors. The fitting routine adjusts the wave amplitudes and propagation parameters so that the fit phasors are as close as possible to the data phasors in the input structure `phasordata`. The number of rows is equal to the number of frequencies and there is one column for each channel. This is the same format as in `phasordata.phasors`.

`wave.global`

A cell array containing any quantities that are fit along with the wave amplitudes. These quantities are global in the sense that they apply to all frequencies. The format is: {'name1',value1,'name2',value2,...} There may be any number of name, value pairs.

`wave.freqs`

column vector of frequencies, same as input frequencies in `phasordata.freqs`

`wave.params`

the parameters data structure input to `wavefit`

`wave.datafile`

a string, the name of the `datafile` passed to `wavefit`

`wave.method`

fitting method, the string passed to `wavefit` named `method`

<u>`refl`</u>

Data structure for the reflection coefficient, contained in a `reflfile`.

Created by: `reflect`

Fields:

`refl.reflcoef`

complex reflection coefficient, a column vector with one entry for each frequency

`refl.freqs`

column vector of frequencies, inherited from input structure `wave`

`refl.refplane`

location of the reference plane, measured in meters from the fiducial plane.

`refl.wavefile`

name of the input `wavefile`

6.3 Acquisition and Analysis Functions

`calibrate(params,level,duration,firstf,lastf,nf,calfile)`

Performs relative calibration of microphones placed at a common reference plane.

Routine type: M-file function
Returns: `caldata` data structure
Reads: nothing
Writes: a `calfile` containing `caldata` data structure
Uses: `sweep`
Argument list:

`params`
: hardware configuration parameters data structure

`level`
: real number, amplitude of played sine-wave burst

`duration`
: real number, duration in seconds of played sine-wave burst

`firstf`
: real number, first frequency

`lastf`
: real number, last frequency

`nf`
: integer, number of frequencies

`calfile`
: string, name of output `calfile` without `.mat` extension

The routine `calibrate` plays a sine-wave burst at the frequency `firstf` while recording each microphone channel, and finds the phasor (complex amplitude) for each microphone. It then repeats these steps at the next frequency, continuing for a total of `nf` frequencies ending with `lastf`. The frequencies are uniformly distributed on a logarithmic scale. (See comments in code for linear scale.) Once all phasors have been measured, `calibrate` writes an output file in the current directory with the name contained in the variable `calfile.` The output file contains the `caldata` data structure. If the output file already exists in the current directory, `calibrate` reports an error and returns without doing anything. Normally `calibrate` is used when the microphones are at a common reference plane in a calibration cell. However, it may be used any time to play bursts and measure the resulting phasors without normalizing to a pre-existing calibration.

`acquire(params,level,duration,calfile,datafile)`

Plays sine-bursts at a series of frequencies, records microphones, measures the microphone phasors at each frequency, and calibrates the phasors.

Routine type: M-file function
Returns: `phasordata` data structure
Reads: a `calfile` containing the data structure `caldata`
Writes: a `datafile` containing the `phasordata` data structure
Uses: `sweep`
Argument list:

`params`
    hardware configuration parameters data structure

`level`
    real number, amplitude of played sine-wave burst

`duration`
    real number, duration in seconds of played sine-wave burst

`calfile`
    string, name of input `calfile` without `.mat` extension

`datafile`
    string, name of output `datafile` without `.mat` extension

The routine `acquire` plays a sine-wave burst at a list of frequencies while recording each microphone channel, and finds the phasor (complex amplitude) for each microphone and frequency. The frequencies are obtained from the frequency list in the input calibration file. The phasors in the input calibration file are normalized by dividing them by the first-channel calibration phasor at each frequency. The phasors measured by `acquire` are calibrated by dividing them by the corresponding normalized calibration phasor. Finally, `acquire` writes an output file in the current directory with the name contained in the variable `datafile.` The output file contains the `phasordata` data structure. If the output file already exists in the current directory, `acquire` reports an error and returns without doing anything. The name of the input calibration file is contained in the variable `calfile`. An error is also reported if the input calibration file does not exist, or if the number of channels in the input calibration file does not agree with the number of channels specified in the `params` data structure that is passed to `acquire`.

`wavefit(params,method,datafile,wavefile)`

Finds the right- and left-going wave amplitudes that best model the phasors contained in the input file. Also finds the best-fit phase velocity and attenuation.

Routine type: M-file function
Returns: `wave` data structure
Reads: a `datafile` containing the data structure `phasordata`
Writes: a `wavefile` containing the `wave` data structure
Uses: private functions `wavfcn3, wavfcn2, wavfcnlocal`
Argument list:

`params`
: hardware configuration parameters data structure

`method`
: String specifying the fitting method. Implemented methods are `'3param'`, `'2param'` and `'local'`.

`datafile`
: string, name of input `datafile` without `.mat` extension

`wavefile`
: string, name of output `wavefile` without `.mat` extension

The routine `wavefit` finds the right- and left-going complex wave amplitudes, the phase velocity and the attenuation constant that best fit the phasor data in the input file (see Equations (4.1) and (4.2) for the form of a general wave). Here and elsewhere in AVNA Lab, right-going always refers to propagation away from the compression driver, regardless of the actual orientation of the microphone section. The name of the input data file is passed in `datafile`. The output data structure `wave` is written to the file whose name is passed in `wavefile`. Several fitting methods are implemented and these are selected by the input string variable named `method`. In all methods, no constraint is placed on the values of the right- or left-going wave amplitudes and no relation between them is assumed. The three implemented fitting methods are:

`'3param'` fits the wave amplitudes at each frequency, and fits three global parameters to model the phase velocity v and the attenuation α :

$$\alpha = \beta\sqrt{\omega}, \quad v = v_S(1 - v_S\delta/\sqrt{\omega}),$$

The three fit parameters are the free-space sound velocity v_S, and the coefficients β and δ. According to the Helmholtz-Kirchhoff theory, β and δ are equal, but in this method they are treated as independent parameters. (In air under standard conditions, for a 25 mm I.D. tube, Helmholtz-Kirchhoff theory predicts $\alpha = \beta \cong 9.5 \cdot 10^{-4}$ m^{-1} s$^{1/2}$.)

`'2param'` fits the wave amplitudes at each frequency, and fits two global parameters used to model the propagation constant at all frequencies. The two fit parameters are the phase velocity, assumed to be independent of frequency, and the β coefficient of the attenuation, defined above.

`'local'` fits the wave amplitudes, the phase velocity and the attenuation constant independently at each frequency.

In all methods, the values of the fit attenuation and phase velocity at each frequency are reported in the `wave` data structure fields `wave.atten` and `wave.vphase`.

An output file is written to the current directory using the name contained in the variable `wavefile.` The output file contains the `wave` data structure. If the output file already exists in the current directory, `wavefit` reports an error and returns without doing anything. The name of the input data file is contained in the variable `datafile`. An error is also reported if the input data file does not exist in the current directory, or if the number of channels in the input data file does not agree with the number of channels specified in the `params` data structure that is passed to `wavefit`.

`reflect(wavefile,reflfile)`

Calculates the reflection coefficient and plots it, both on the complex plane and as magnitude and phase versus frequency.

Routine type: M-file function
Returns: `refl` data structure
Reads: a `wavefile` containing the data structure `wave`
Writes: a `reflfile` containing the `refl` data structure
Uses: private function `reflfcn`
Argument list:
`wavefile`
    string, name of input `wavefile` without `.mat` extension
`reflfile` (optional)
    string, name of output `reflfile` without `.mat` extension

The routine `reflect` is used to calculate, plot, and store the reflection coefficient. The input `wavefile` must contain a `wave` data structure. If the input file is not present in the current directory, an error will be reported. When the reflection coefficient plots have been displayed, the user is prompted to type a new value for the reference plane location (in meters). After it is entered the plots are updated. This continues until the user types `'q'` to quit or `'s'` to save the `refl` data structure to a `reflfile`. The file name variable `reflfile` is an optional input parameter. If it is specified, the file it names must not already be present in the current directory, or else an error will be generated. See the code comments to change the default starting location of the reflection plane. The reflection plane location is measured from the fiducial plane, which is at the face of the flange at the driven end of the microphone section. See the section on the `params` data structure for additional geometry information.

6.4 Auxiliary Routines

`scope`

Plays a sine-wave burst and records all channels, plots played and recorded waveforms, plots recorded spectrum, prompts user to change a parameter, and repeats.

Routine type: M-file script
Reads: nothing
Writes: nothing
Uses: `defaults`, `burst`, `playrecord`, `spectrum`

The script `scope` is used to interactively play and record sine-wave bursts while adjusting parameters of the burst, the hardware levels, and the correction for latency. The script takes no parameters and is invoked simply by entering `scope` at the command line. First, `scope` calls `defaults` to bring the default `params` data structure into the workspace. Then, a sine-wave burst is played and all channels are recorded, using the hardware configuration specified in `params`. Default values for the burst amplitude, duration and frequency are coded in the M-file. The first 5% and last 5% of the burst duration are smooth transitions with a sine-squared envelope. Plots are created of the played time-series, the recorded times-series and a flat-top windowed spectrum of the recorded time-series. Finally, `scope` prompts the used to enter a command. After a command is entered, `scope` plays and records another burst and updates the plots.

Recognized commands are `'r'` to rescale the plots, `'q'` to quit the script, or any valid MATLAB command. Some useful MATLAB commands are:

`level=0.5`	change the burst amplitude to 0.5
`dur=2.0`	change the burst duration to 2.0 seconds
`freq=1345`	change the burst frequency to 1345 seconds
`params.latency=0.015`	change the latency correction to 15 ms

These commands only change the values of variables currently in the workspace. To change the default values that will be used when `scope` is next invoked, edit the `scope.m` file for `level`, `dur`, and `freq`, or the `defaults.m` file for any of the `params` fields.

Both the play time-series and the record time-series are pure numbers to the AVNA Lab software. The connection between the pure numbers and the actual sound pressure levels are determined by the digitizer hardware, the input and output gain settings, the compression driver, and the microphones. Generally, clipping or saturation on both record and play occur when the record and play time-series values are near 1.0. Clipping and saturation can be checked by examining the spectrum plot.

The spectrum plot is also useful for measuring fundamental and harmonic amplitudes. No special relationship between the burst frequency and the sampling rate is required for accurate amplitude measurements because a flat-top window is used with very small 'picket-fence' errors.

The best value for the latency correction parameter `params.latency` can be found by examining the two time-series plots. When a sine-burst is played, the played samples always start slightly after the start of the recorded samples. To correct for this, recording is continued for the play duration plus an additional time interval given by `params.latency`. Then, the first `params.latency` time interval of the recorded time-series is deleted. The plots generated by `scope` show the played time-series and the time-shifted recorded time-series. The value of `params.latency` should be adjusted until the two plots are aligned to within 10 ms. Then, the `defaults.m` file should be edited to set the correct default value of `params.latency`.

<u>`defaults`</u>

Creates a `params` data structure with default parameter field values that specify the hardware configuration.

Routine type: M-file function
Returns: `params` data structure

Reads: nothing
Writes: nothing
Uses: nothing
Argument list: none

The function `defaults` returns the `params` data structure defined in Section 6.2. The function name can be passed directly to other AVNA Lab routines that require the `params` data structure as input. For example

```
>> acquire(defaults,0.5,2.0,'cal21','dat21')
```

passes the `params` structure to `acquire`. The default values of the `params` fields can be changed by editing `defaults.m`. Several versions of `defaults` (for example `defaults4ch`, `defaults6ch`,…) can be created if several hardware configurations are in use. Note, however, that `scope` calls defaults directly and will always use the version named `defaults`. No other routines call `defaults` directly.

`viewdata(datafile1,datafile2)`

Creates graphs of the phasors in a `datafile` or a `calfile`, or creates graphs comparing two files of either type.

Routine type: M-file function
Returns: nothing
Reads: one or two files of type `datafile` or `wavefile`
Writes: nothing
Uses: nothing
Argument list:

`datafile1`
    String, name of input `datafile` without `.mat` extension. May also be a `calfile`.

`datafile2` (optional)
    String, name of input `datafile` without `.mat` extension. May also be a `calfile`.

The routine `viewdata` displays graphs of the phasor data contained in files of type `datafile` or `calfile`. If two file names are given, the data in the two files are compared. Only the `.phasors` and the `.freqs` fields of the data structures in the files are used, so any files containing a single data structure with these fields can be used as input. Errors are generated if a specified file does not exist, or if two files are specified but they cannot be compared. For files to be compared, the `.phasors` fields must be of the same size (same number of channels and frequencies), and corresponding frequency values in the `.freqs` fields must agree to within 0.01 Hz.

If one file is specified, `viewdata` displays three graphs: 1) phasor magnitudes, 2) phasor magnitudes normalized to the first channel, and 3) phase relative to the first channel. Each quantity is plotted versus frequency with separate colored-coded curves for each channel. Colors are assigned to channel numbers sequentially in the standard MATLAB order. If $\varphi_n(f)$ refers to the *n*th channel phasor at frequency *f*, the quantities plotted are:

$$1)\ |\varphi_n(f)| \quad 2)\ |\varphi_n(f)| / |\varphi_1(f)| \quad 3)\ \arg(\varphi_n(f)) - \arg(\varphi_1(f))$$

If two files are specified, `viewdata` displays three comparison graphs: 1) the ratio of the phasor magnitudes in the second file to the phasor magnitudes in the first file, 2) the ratio of the normalized magnitudes in the second file to the normalized magnitudes in the first file, 3) the phase relative to channel one in the second file minus the phase relative to channel one in the first file. If $\varphi_{n1}(f)$ and $\varphi_{n2}(f)$ refer to the *n*th channel phasor at frequency *f* in the first and second files, the quantities plotted are:

$$1)\ \frac{|\varphi_{n2}(f)|}{|\varphi_{n1}(f)|} \quad 2)\ \frac{|\varphi_{n2}(f)| / |\varphi_{12}(f)|}{|\varphi_{n1}(f)| / |\varphi_{11}(f)|}$$

$$3)\ \left[\arg(\varphi_{n2}(f)) - \arg(\varphi_{12}(f))\right] - \left[\arg(\varphi_{n1}(f)) - \arg(\varphi_{11}(f))\right]$$

`viewwaves(wavefile)`

Creates graphs of the data in a `wavefile`.

Routine type: M-file function
Returns: nothing
Reads: one file of type `wavefile`
Writes: nothing
Uses: private function `wavefcnlocal`
Argument list:
`wavefile`
String, name of input `wavefile` without `.mat` extension.

The routine `viewwaves` creates graphs for inspecting files containing the `wave` data structure generated by `wavefit`. The input `wavefile` must be present in the current directory or an error will be generated. To compare the fit phasors in the `wave` data structure with the experimentally measured phasors that were used to generate the fit, `viewwaves` also opens the `datafile` containing the `phasordata` structure. This file must also be present in the current directory. The name of the file containing the `phasordata` structure is taken from the field `wave.datafile`.

The first two graphs show the magnitudes of the right- and left-going wave amplitudes versus frequency (see Equations (4.1) and (4.2) for the form of a general wave):

1) $|A_R(f)|$ 2) $|A_L(f)|$

Graphs 3) and 4) show the phase velocity v and attenuation α versus frequency.

The normalized chi-squared for each channel is shown in graph 5). Its is defined as

$$5)\ \chi_n^2(f) = \frac{|\tilde{\varphi}_n(f) - \varphi_n(f)|^2}{|A_R(f)|^2},$$

where $\varphi_n(f)$ are the measured phasors from the `datafile`, and $\tilde{\varphi}_n(f)$ are the fit phasors from the `wavefile` (the index n runs over the channels).

Finally, graph 6) is a polar plot of the measured and fit phasors for each channel, at the first frequency in the list `wave.freqs`. The measured phasors are plotted with a black X, and the fit phasors are plotted with a red O. Also, fit phasors are evaluated at 100 locations starting at the first microphone and ending at the last one. These are plotted as a red curve on the complex plane.

After all plots are displayed the user is prompted to enter `'+'` or `'-'` to scan up or down through the frequency list. Each time the frequency is changed the polar plot 6) is updated. If enter key is held down, the frequency scans rapidly in the direction of the most recent change. Entering `'q'` quits the routine.

6.5 Lower-Level Functions

`burst(params,level,duration,frequency)`

Returns a sine-wave burst time-series.

Routine type: M-file function
Returns: data structure `timeseries` with fields

`.samples`
column vector of samples

`.times`
column vector of times of the samples, starting at zero

Reads: nothing
Writes: nothing
Uses: nothing
Argument list:

`params`
hardware configuration parameters data structure

`level`
real, amplitude of the sine time-series

`duration`
real, total duration in seconds of the time-series, including transitions

`frequency`
real, frequency of the sine-burst

The function burst creates a sine-wave burst time-series with sine-squared envelope transitions at the beginning and end to avoid clicks. The transitions are each 5% of total duration.

`playrecord(params,playseries)`

Plays an arbitrary time-series and records multiple channels simultaneously, with time-shifting to correct for the latency delay.

Routine type: M-file function
Returns: data structure `recordseries` with fields

`.samples`
Recorded samples corrected for latency with one column for each channel. The number of samples per channel is the same as number of samples in

the input `playseries`. The number of channels is determined by `params.nchan`.

`.times`

column vector of times of the samples, starting at zero

Reads: nothing
Writes: nothing
Uses: nothing
Argument list:

`params`

hardware configuration parameters data structure

`playseries` data structure with the fields:

`.samples`

time-series column vector to play

`.times`

times of the samples, starting at $t = 0$. (This field may be present but it is not used.)

Plays and records multiple channels simultaneously. To correct for latency, recording is continued for a time `params.latency` seconds longer than the duration of the input `playseries`, and the first `params.latency` seconds of the recorded time series is dropped. The play and record sampling rate is determined by `params.fsamp`, not by `playseries.times`.

`fourier(params,timeseries,frequency)`

Computes the fourier amplitude, or phasor, of a multi-channel time series at a specified frequency.

Routine type: M-file function
Returns: `phasors`, a row vector containing one phasor for each channel in `timeseries`.
Reads: nothing
Writes: nothing
Uses: nothing
Argument list:

`params`

hardware configuration parameters data structure

`timeseries` data structure with the fields:

`.samples`

time-series column vector

`.times`

Column vector of times of the samples, starting at $t = 0$. (This field may be present but it is not used.)

`frequency`

frequency at which the phasors are to be determined

To find the phasors, `fourier` first deletes the first and last 10% of the time-series, in an effort to eliminate transients. Next, it trims the time-series so it contains, as nearly as possible, an integer number of cycles of the specified frequency. The time-series sampling rate is assumed to be `params.fsamp`. Then the time-series is integrated times a complex exponential at the specified frequency to find the phasor. This algorithm requires no special relationship between `frequency` and the sampling frequency, or the time-series duration. It works well when there are higher harmonics in the time series.

`sweep(params,level,duration,freqs)`

Plays a sine-wave burst at each frequency, records multiple channels simultaneously, and returns a complex phasor for each frequency and channel.

.

Routine type: M-file function

Returns: data structure `phasorseries` with fields

`.phasors`

The phasors (complex amplitudes) for each frequency and channel. The number of rows is equal to the size of the input frequency list `freqs`. There is one column for each channel and the number of channels is equal to `params.nchan`.

`.freqs`

column vector of frequencies, same as the input frequency list `freqs`.

Reads: nothing

Writes: nothing

Uses: `burst`, `playrecord`, `fourier`

Argument list:

`params`

hardware configuration parameters data structure

`level`

amplitude of each sine-wave burst

`duration`

duration of each sine-wave burst

`freqs`

column vector of frequencies

The function `sweep` measures phasors for multiple channels using sine-wave bursts. The input frequency list `freqs` need not be ordered and can contain repeated frequencies. Accurate phasors are measured for arbitrary frequencies below the Nyquist limit. There does not need to be any special relationship between the sine burst frequency, the sampling frequency, and the burst duration.

.

`spectrum(params,timeseries)`

Computes the magnitude spectrum of a time-series using the FFT and a flat-top window.

Routine type: M-file function
Returns: `spectseries` data structure, containing the fields:

`.mags`

An array of measured magnitudes with one column for each channel in `timeseries` and number of rows equal to the size of `.freqs` field.

`.freqs`

Column vector of frequencies corresponding to the entries in `.mags`. The first frequency is zero and the last frequency is exactly the sampling frequency divided by 2.

Reads: nothing
Writes: nothing
Uses: nothing
Argument list:

`params`

hardware configuration parameters data structure

`timeseries` data structure with the fields:

`.samples`

column vector of samples

`.times`

Column vector of times of the samples, starting at $t = 0$. (This field may be present but it is not used.)

The function `spectrum` uses a Fast Fourier Transform with a flat-top window to compute the magnitude spectrum of `timeseries`. No phase information is provided and the frequency resolution is relatively poor, but amplitudes can be measured accurately with very little ‘picket fence’ error. Before computing the spectrum, `fourier` removes the first and last 10% of the time series to allow for transients. Then one sample is dropped if necessary to leave an even number of samples, which centers the last frequency bin at exactly one-half the sampling rate. Finally, the time-series is windowed, transformed, and the magnitude is computed. The normalization factor used adjusts the spectrum so that it gives the amplitude of a sinusoidal time-series, not the rms value. The time-series sampling rate is assumed to be `params.fsamp`. The sample times in `timeseries.times` are not used.

7. Hardware Reference

7.1 Mechanical Drawings

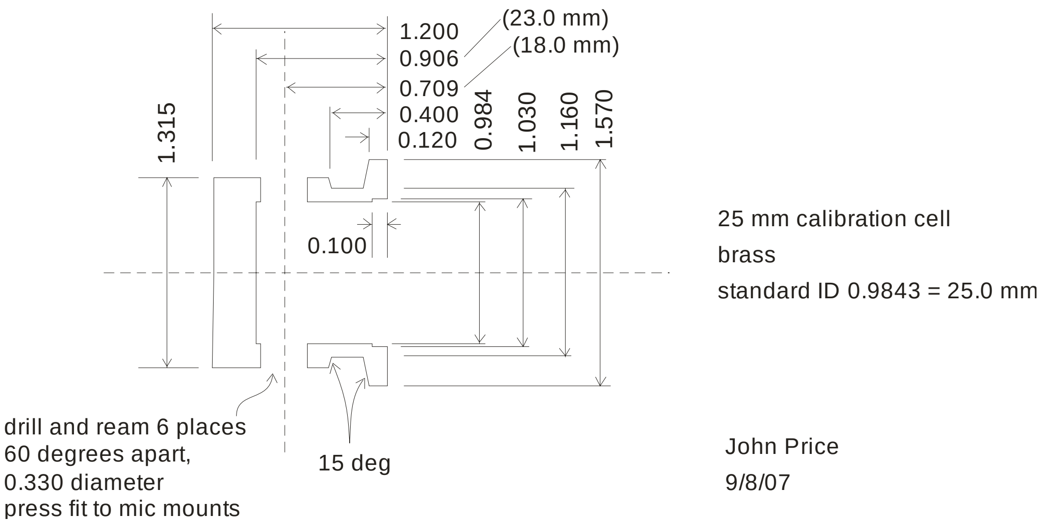
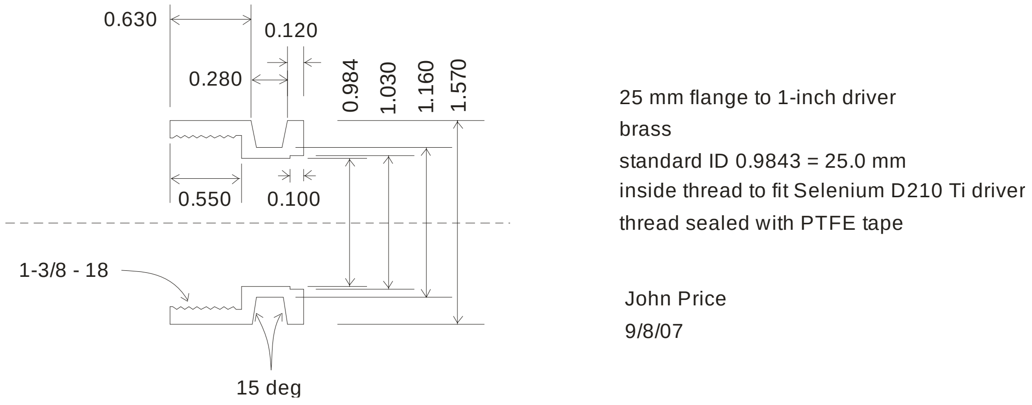
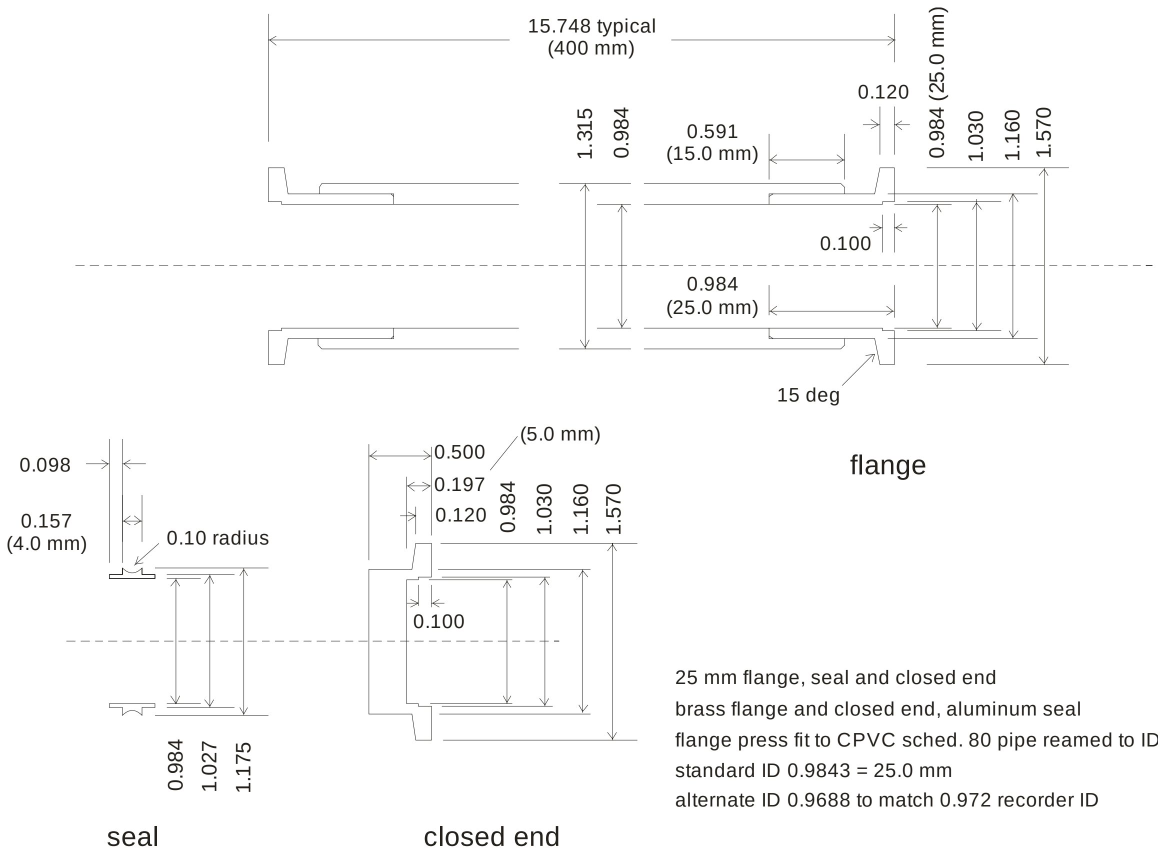


Figure 7.1. Calibration Cell



0.630
0.120
0.280
0.984
1.030
1.160
1.570
0.550
0.100
1-3/8 - 18
15 deg
25 mm flange to 1-inch driver
brass
standard ID 0.9843 = 25.0 mm
inside thread to fit Selenium D210 Ti driver
thread sealed with PTFE tape
John Price
9/8/07

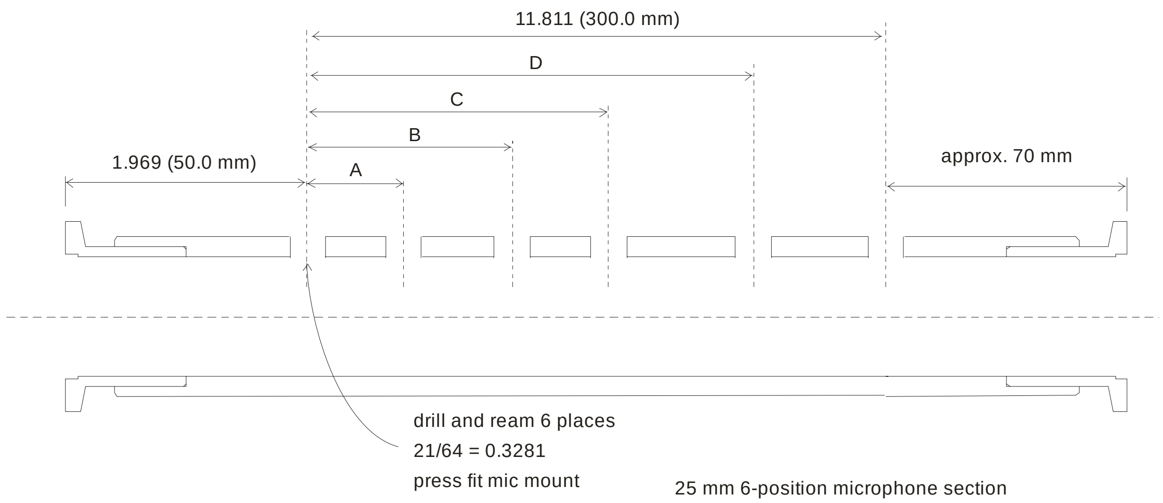
Figure 7.2. Compression Driver Adaptor



25 mm flange, seal and closed end
brass flange and closed end, aluminum seal
flange press fit to CPVC sched. 80 pipe reamed to ID
standard ID 0.9843 = 25.0 mm
alternate ID 0.9688 to match 0.972 recorder ID

John Price
9/8/07
revised 12/21/08

Figure 7.3. 25 mm Flange, Seal, Closed End



A =0.9237 (inches)
B =3.2990
C =5.1920
D =8.0151

25 mm 6-position microphone section
brass flange
flange press-fit to CPVC sched. 80 pipe reamed to ID
standard ID 0.9843 = 25.0 mm
alternate ID 0.9688 to match 0.972 recorder ID

John Price
9/8/07

Figure 7.4. Microphone Section