OPTIMAL CONFIGURATION OF ADJUSTABLE NOISE SUPPRESSORS

A Thesis
Presented to
The Academic Faculty

by

Elliott Ross Gruber

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Mechanical Engineering

Georgia Institute of Technology

May, 2013

OPTIMAL CONFIGURATION OF ADJUSTABLE NOISE SUPPRESSORS

Approved by:

Dr. Kenneth A. Cunefare, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Wayne J. Book
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Mardi C. Hastings
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: [March 27, 2013]

ACKNOWLEDGEMENTS

I must acknowledge several people who helped me complete this work. First, I must thank both my mother, Amy Gruber, and my father, Larry Gruber, for expertly raising me and giving me the guidance that allowed me to reach this point in my life. Next, I would like to thank my advisor, Dr. Kenneth Cunefare, for providing invaluable insight and guidance to help me complete this project as well as continue my growth as an engineer and researcher. I would also like to thank Dr. Wayne Book and Dr. Mardi Hastings for agreeing to participate on the reading committee, with extra gratitude to Dr. Book for allowing me to use his laboratory facilities. Thanks must also be directed to my officemates and friends for all of their help, both for expanding my knowledge of engineering theory and application as well as brightening my day when inevitable setbacks occurred. In particular, I need to thank the now graduated Dr. Nicholas Earnhart and Dr. Benjamin Beck for sharing the lessons they learned through the years with me.

I must direct special thanks to Eaton Hydraulics for providing the funding necessary to complete this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Symbol	
A	Cross Sectional Area [m^2]
A,B,C,D,E,F,G,H,	Complex Wave Amplitudes
ABN	Airborne Noise
c	Speed of Sound [m/s]
CPR	Charge Pressure Ratio
D	Diameter [m]
D	Time Weighting Factor
f	Frequency [Hz]
$\mathscr{F}$	Objective Function
FBN	Fluid-borne Noise
FWF	Frequency Weighting Factor
G_{xx}	Autospectral Density
G_{xy}	Cross-Spectral Density Between *x* and *y*
H_{ij}	Transfer Function Relating i and j
k	Spring Constant [N/m], wavenumber [rad/m]
L	Length [m]
L'	Effective Length
m	Mass [kg]
M	Mach Number
O	Selected set
p	Acoustic Pressure [Mpa]
P	Pressure [Mpa]
Q	Quality Factor, Volume Flowrate [L/min]
Q	Acoustic Velocity [m/s]
r	Radius [m]
R_s	Specific Resistance
S	Cross Sectional Area [m^2], Separation Distance [m]
SBN	Structure-borne Noise
TL	Transmission Loss [dB]
t_w	Wall Thickness
TWF	Time Weighting Factor

u	Acoustic Particle displacement [m]
U	Set of Pressures
V	Volume [m^3], Flow Velocity [m/s]
W	Acoustic Energy [W]
W	Frequency Weighting Factor
Z	Impedance [Pa*s/m]
α	End Correction Coefficient
λ	Wavelength [m], Lame's first parameter
ρ	Density [kg/m^3]
σ	Sheet Density
φ	Porosity
ω	Frequency [rad/s]
Ω	Frequency Bandwidth

Subscript	
annulus	Device Annulus
B	Bladder
c	Charge
Chamber	Device Chamber
compressed	Compression of the bladder under pressure
D	Downstream
d	Dynamic
f	fluid
h	Hole
i	Incident
n	modes
p	Perforate Layer
Pipe	Inlet/Outlet Pipe
port	Inlet/Outlet Port
R	Region
r	radial
s	System
shell	Outer Shell of Device
t	Transmitted
T	Total
U	Upstream
x	axial

SUMMARY

Noise generated by fluid power applications can be treated using bladder-style suppressors, and an optimal operating condition for these devices is sought in this thesis. Bladder-style suppressors employ a compliant nitrogen-charged bladder to create an impedance change within the system, reflecting the noise back to the source and preventing it from propagating downstream. The noise in a hydraulic system is created by a pump, the flow source in a hydraulic system, and can be separated into three categories: fluid-borne noise, structure-borne noise and airborne noise. Fluid-borne noise places addition stress on sealing surfaces, potentially causing leaks. Airborne noise can be uncomfortable, even hazardous depending on the level. Bladder-style suppressors primarily treat fluid-borne noise; however, it is seen in the literature that fluid-borne noise is the cause of structure-borne and airborne noise.

This thesis presents an optimization method for finding the optimal charge pressure for implementation with a given system operating over a broad range of system pressures. The optimization weights suppressor performance by the spectral content of the fluid-borne noise as well as the duty cycle of the system. A single charge pressure works well over a small range of system pressures, though many fluid power applications operate over a larger range of system pressure than the usable range of a suppressor. For systems operating over an extremely broad pressure range, two suppressors charged to different pressures are used to treat the noise in the entire system pressure range.

To determine suppressor performance experimental measurements were performed, and models developed, of the transmission loss of this type of device. A multi-microphone method using transfer function relationships between six sensors determines the transmission loss of the suppressor under test. An equivalent fluid model modeling the wave behavior both upstream and downstream, as well as within the suppressor, was created to predict suppressor transmission loss.

Optimal configurations are found for a set of system pressures, charge pressures and duty cycles. Analysis of the results shows the time weighting has a more significant impact on the optimum charge pressure than the frequency weighting, as shown by duty cycles considered in this thesis. In addition, all charge pressures selected as optimal for either single suppressor optimizations or double suppressor optimizations, exhibit the highest transmission loss for a single system pressure in the pressure duty cycle for a simulated machine.

CHAPTER 1

INTRODUCTION

A common method to treat the noise produced by fluid power applications is implementing bladder-style suppressors into a system; however, little is currently known about the optimal configuration of bladder-style suppressors. Bladder-style suppressors use a complaint bladder to create an impedance change, reflecting noise and preventing it from propagating downstream. Internal features within the suppressor may also introduce damping. The compliance of the bladder is controlled by charging the bladder with nitrogen to a given pressure, known as the charge pressure. Nitrogen is selected for use because of it is inert and noncombustible. Bladder-style suppressors , as seen in Figure 1, are constructed from two end caps, an outer shell, a perforate layer, the bladder, and an annulus as seen in Figure 1 [1-3]. The suppressor must be charged prior to use, and the perforate layer and annulus are used to ensure the pressurized bladder does not extrude into the flow path before the system is pressurized, as well as adding damping to the device [1]. The goals of the work presented in this thesis are to find the optimal charge pressure condition for a single bladder-style suppressor operating in a single-system pressure fluid power system, and as well as find the optimal charge pressure condition for either a single bladder-style suppressor or a pair of bladder-style suppressors operating over a broad range of system pressures.

Performance of all noise control devices, including bladder-style suppressors, is measured by transmission loss (*TL*), the ratio of incident power to transmitted power. Higher *TL* is indicative of better performance. *TL* varies by frequency similar to any noise source, including a pump in a hydraulic system. The spectral content of pump noise is broadband, and a favorable characteristic of bladder style suppressors, compared to other common noise control devices, is their broadband *TL*. Outside of a paper published by Marek, et al. [4] no other work has been found on the performance of bladder-style suppressors.

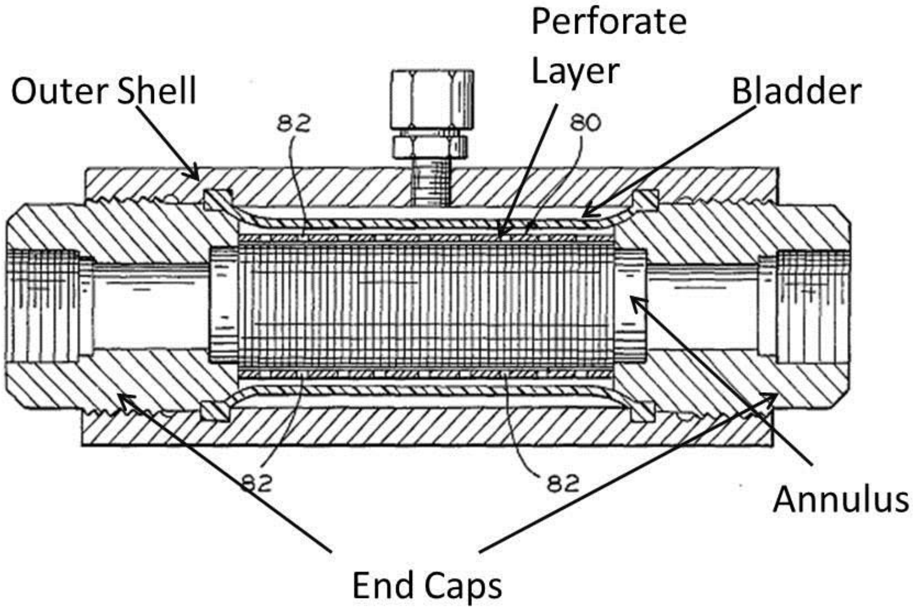

Figure 1: Bladder style suppressor [1]

The noise produced by fluid power systems can be damaging to equipment, as well as uncomfortable, even hazardous, to work around. The main source of noise energy in a hydraulic system is the pump, which is required for producing flow through the system. The flow rate will fluctuate slightly during operation, because of non-constant flow from the pump and fluid compressibility, causing a flow ripple. The flow ripple couples with the hydraulic fluid and system components to create a dynamic pressure ripple, which is superimposed onto the mean system, or static, pressure. The dynamic pressure ripple is the damaging element to system elements, potentially exposing sealing surfaces to strong pressure pulses causing leaks. In addition, system components are subject to stress cycles causing material fatigue and potential failure.

Noise generated from the pressure ripple can be separated into three general categories: fluid-borne noise (FBN) i.e. the pressure waves within the fluid, structure-borne noise (SBN) i.e. the vibrations of pipes and other system components, and air-borne noise (ABN) i.e. breakout noise. The exact frequency-domain transfer-function relationships between the three noise types are complicated and difficult to model, but Johnson and Edge [5] have shown that FBN is the cause of both SBN and ABN. A reduction of FBN causes a reduction of both SBN and ABN, lessening the stress on sealing surfaces as well as noise within a worksite.

The noise control solution needs to be robust for a system operating over a broad range of system pressures and adequately treat broadband FBN to control ABN and SBN. This goal was brought forward by an industry collaborator, Eaton Hydraulics, who is developing a new valving technology for use in a hydraulic system operating over a broad system pressure range. Observations of this technology showed significant broadband FBN, a potential for extreme amounts of ABN and a noise treatment solution was needed.

While bladder-style suppressors have not been extensively studied in the literature, a similar device used in airborne noise control, mufflers, have been studied. A basic type of airborne muffler is an expansion chamber, a section of inline rigid pipe with a larger radius than the pipes connecting immediately upstream and downstream of the device, seen in Figure 2. The change of cross sectional area creates a change in specific acoustic impedance causing some acoustic energy to be reflected, reducing the amount of transmitted energy [6]. An expansion chamber works as a periodic band stop filter in the frequency domain, analogous to a capacitor connected in series to an electrical circuit [6]. Some expansion chambers have inlet/outlet extensions, Figure 2b, to change the behavior of the chamber. The extensions have been studied by Selamet and Li [7], and the work notes the extensions help the *TL* become more broadband than a device without extensions, a desirable characteristic for use with hydraulic systems. The *TL* of empty expansion chambers operating in an air system can be improved in several ways; multiple chambers targeting specific frequencies, a perforate layer separating the flow path from the expansion area and using a fibrous lining to dissipate acoustic energy [8, 9].

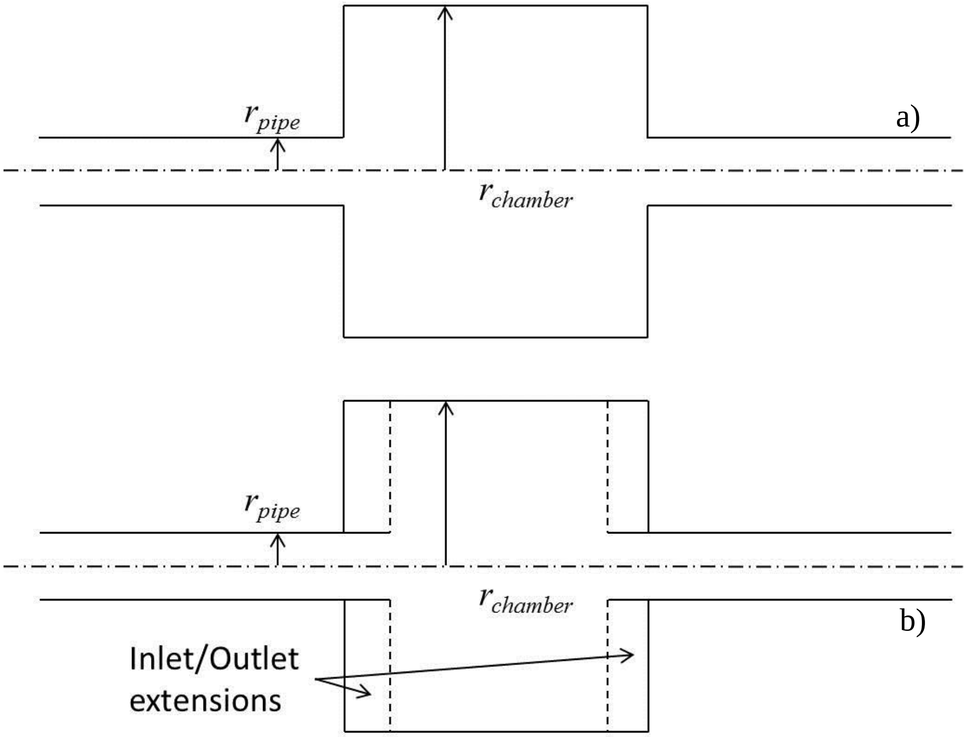

Figure 2: Expansion chamber, a) Without inlet/outlet extensions b) With inlet/outlet extensions

A major difference between airborne mufflers and bladder-style suppressor relate to the operational fluids, air and hydraulic oil, respectively. The most notable difference is the sound speeds, 343 m/s in air and 1400 m/s in hydraulic fluid. In all media, particle displacement is inversely proportional to sound speed; particle displacement is greater for air than hydraulic fluid. Particle velocity is the first temporal derivative of acoustic particle displacement, and is proportional to particle displacement. Damping is a function of particle velocity. Therefore, damping will be more effective for airborne systems due to the lower sound speed and alternative noise suppression techniques will be needed for hydraulic systems. One noise suppression technique, employed by reactive silencers for airborne systems, is reflecting noise back to the source using a specific impedance change.

In bladder-style suppressors, the compliance of the bladder changes the impedance of the inlet port, causing some acoustic energy to be reflected to the source and decreasing the amount of transmitted energy. The impedance of the suppressor is dependent on the ratio of the charge pressure in the bladder to the system pressure, known as the charge pressure ratio (CPR). Wilkes and McLean, a bladder-style suppressor manufacturer, suggests charging the suppressors to a CPR of 0.5, while industry contacts assert that a higher CPR causes bladder-suppressors to perform better [1-3, 10]. Example *TL* curves are presented in Figure 3 and demonstrate the effect of changing CPR has on *TL* over the entire frequency range of interest. As described above a partial focus of this thesis is to find a CPR which exhibits the highest TL for a given system pressure.

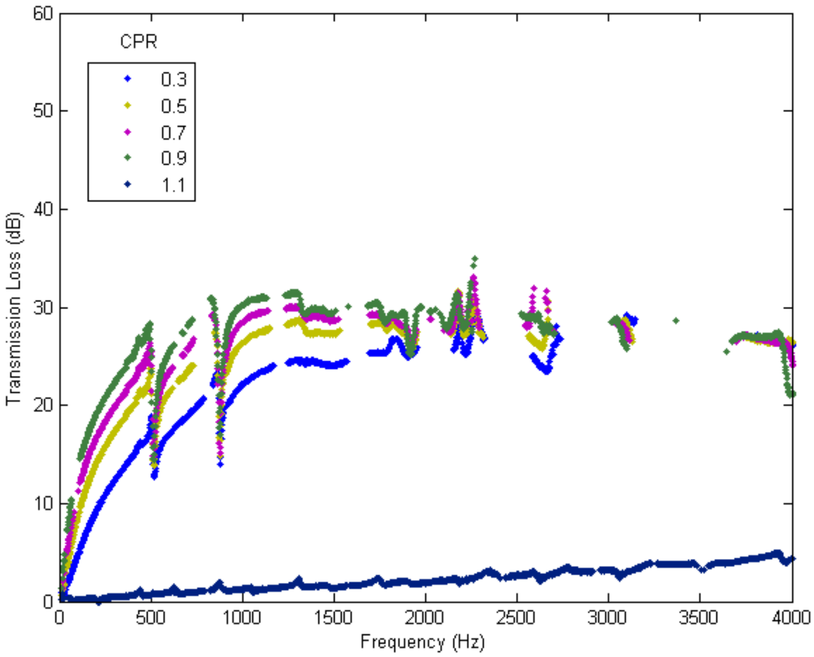

Figure 3: Example transmission loss curves for a WM-5081 bladder-style suppressor operating at a system pressure of 10.3 MPa and varying CPR

A major drawback to bladder-style suppressors is that the devices are designed to target a small range of system pressures for a given charge pressure; however, many fluid power applications operate over a large range of system pressures, as seen in Figure 4 which shows the time fraction spent at each system pressure of the boom actuator on a hydraulic excavator, and charge pressure cannot be adjusted during operation. Selection of an optimal charge pressure for use with a given duty cycle is important for the best performance of the hydraulic system, both in terms of stress on components and ABN. Outside of work done by Marek, et al. [4], no other work has been found on characterization bladder-style suppressors. Marek uses linear acoustics as well as continuity of pressure and volume velocity to develop a model for predicting the behavior of a bladder-style suppressor. The effect of individual suppressor components is studied, and the effect of the CPR is also presented. Besides work by Gruber, et al. [11] no other literature was found about the optimization of bladder style suppressors operating over a broad range of system pressures.

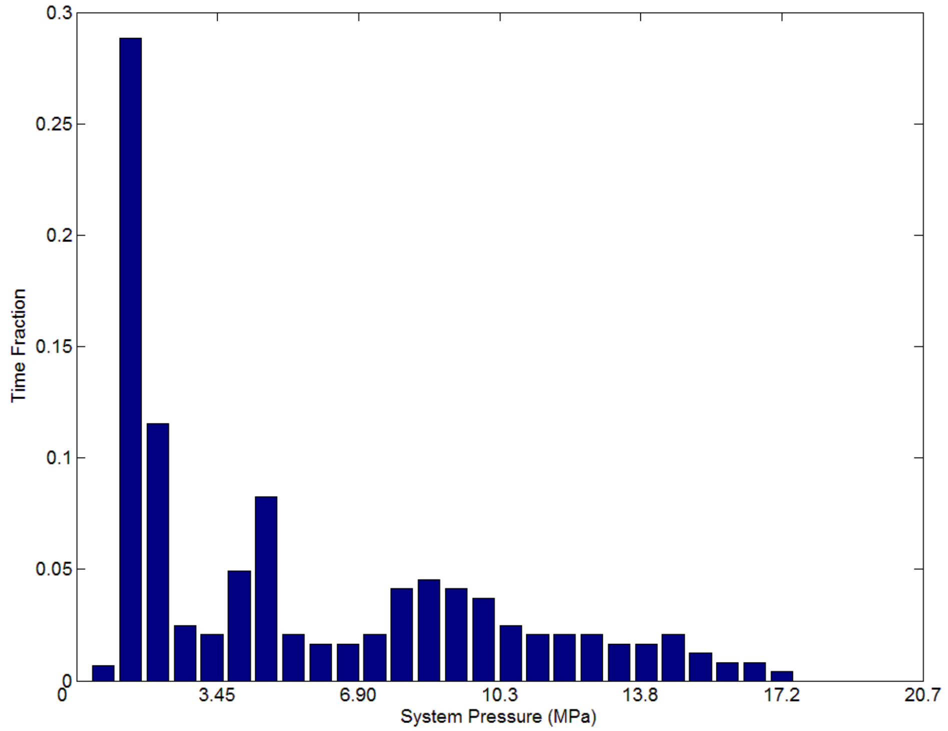

Figure 4: Example system pressure usage for the boom actuator on a hydraulic excavator

This thesis describes the development of an approach to find an optimal charge pressure condition using a model of a bladder-style suppressor and information pertaining to system pressure time history and the spectral content of pressure ripples at those system pressures. The optimal charge pressure is found through optimization of an objective function. The objective function weights *TL* by two weighting factors; a frequency-weighting factor (FWF) and a time weighting factor (TWF). The FWF accounts for the spectral content of the FBN, weighting the objective function towards frequencies with higher energy. The TWF accounts for the fraction of time a system spends at each different system pressures weighting the objective function towards the most-used system pressures. The optimal charge pressure condition may be found for both single and double suppressor configurations. Two dissimilarly charged suppressors are employed for systems operating over a broad range of system pressures, allowing each suppressor to effectively target different system pressure ranges.

The remainder of this work will discuss how *TL* is measured on actual devices and modeled for bladder-style suppressors, the creation of the objective function to find the optimal charge pressure condition, and example results of the optimization of the objective function. The measurement of *TL*, based on work by Earnhart, et al. [12] further discussed in Chapter 2, expands on current methods using multiple pressure transducers to resolve pressure amplitudes upstream and downstream of the device, allowing *TL* to be calculated. The modeling of *TL* for a bladder style suppressor, based on work by Marek, et al. [4] further discussed in Chapter 3, is predicted by an equivalent fluid model which simulates the acoustic pressure and particle behavior within the device. In Chapter 4, the details of a two suppressor system, used for noise treatment in systems operating over extremely large pressure ranges, are presented. The optimization objective function is discussed in Chapter 5. The optimization of the objective function is used to determine the optimal charge pressure for a given usage, a goal of this thesis.

CHAPTER 2

TRANSMISSION LOSS MEASUREMENT, CALCULATION AND EXAMPLE FOR A SINGLE SUPPRESSOR

Transmission loss (*TL*) is the ratio of transmitted acoustic energy to incident energy,

$$TL = 10\log_{10}\frac{W_i}{W_t} \tag{2.1}$$

where W_i is the incident acoustic energy, W_t is the transmitted acoustic energy and higher transmission loss is indicative of better performance [6]. The techniques used for *TL* measurement in this work are discussed below. *TL* is not system dependent allowing for easy comparison between devices tested in different systems. Example *TL* data for a bladder style suppressor will be presented.

2.1 Transmission Loss Measurement

A method to measure *TL* of hydraulic elements is based on several existing techniques [5, 13-22] and implemented by Earnhart, et al. [12] to characterize the behavior of noise control devices, and this technique is applied specifically to bladder-style suppressors for this work. An acoustic element, modeled as a two-port four-pole system, can be described by a transfer matrix relating acoustic pressure and velocity at the upstream and downstream ports. A two-microphone method to measure the properties of an acoustic element was developed by Seybert and Ross [13] and improved by Chung and Blaser [14, 15]. To and Doige [16, 17] further developed the two-microphone method using reference pipes both upstream and downstream to determine the transfer matrix of the test article. Their work is further developed for systems with mean flow, such as hydraulic systems, with use of time averaging by Lung and Doige [18]. A drawback to the two-microphone method is indeterminacy occurring at frequencies corresponding to an integer multiple of half wavelengths between the microphones. In order to resolve the half-wavelength indeterminacy Kojima and Edge [19] as well as others [20-22] use a three-microphone method, with the microphones spaced at unequal intervals, to solve for the transfer matrix elements. A least-squares fit is applied to the transfer functions between the sensors to robustly estimate the pressure waves.

The International Standard pertaining to acoustic measurements in a hydraulic system is ISO-15086 [23-25]. ISO-15086-1 [23] states the mean velocity of the flow must be less than 1% of the speed of sound, which is a Mach number of 0.01, for accurate measurements. The Mach number for a given system is calculated by

$$M = \frac{V}{c}, \tag{2.2}$$

where c is the speed of sound, nominally 1400 m/s in hydraulic fluid. Flow velocity, V, is calculated by

$$V = \frac{Q}{A}, \tag{2.3}$$

where Q is the volume flow-rate and A is the cross sectional area of the pipe. The flow velocity and Mach number associated with the flow in pipes of any diameter and any flow rate can be calculated. This thesis focuses on pipe diameters of 0.019 m and 0.038 m with a volume flow-rate of 37.85 liters/minute. The flow velocity and Mach numbers associated with flow in these pipes are shown in Table 1. The Mach number values are both at least an order of magnitude below the 0.01 of Mach threshold, therefore, mean flow can be assumed negligible for measuring *TL* in systems using pipes of this flow and pipe size. ISO-15086-2 [24] employs a three microphone method to avoid half-wavelength indeterminacy, with the microphone positions seen in Figure 5. The minimum distances from the pipe inlet to the first transducer, x_1, as well as the distance from the last transducer, x_2, to the end of the pipe are dependent on the pipe internal diameter, as shown in the legend for Figure 5.

Table 1: Flow velocity and Mach Number for pipe diameters used in this thesis

Inlet Pipe ID (m)	Flow velocity (m/s)	Mach number
0.019	2.2135	0.0016
0.038	0.5534	0.0004

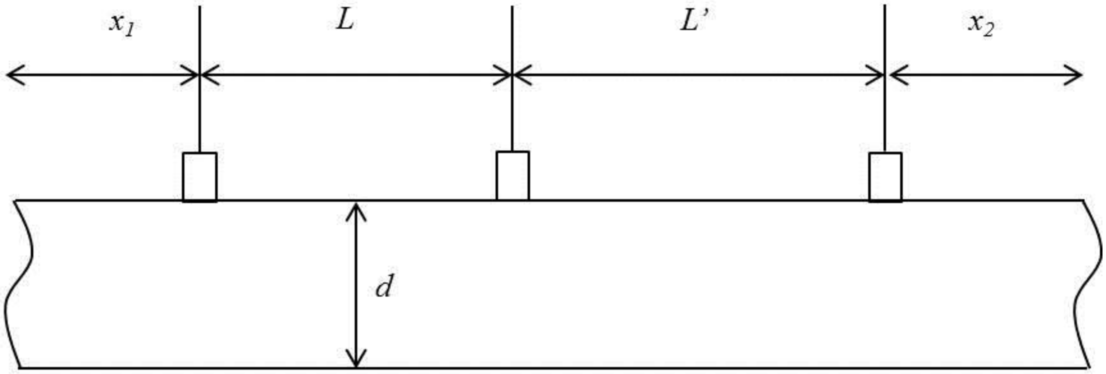

Figure 5: ISO 15086-2 dimensions, x_1≥10d, x_2≥10d, L=330±2mm, L'=470±2mm

2.1.1 Experimental Transmission Loss Method

The testing method described above was implemented to measure two different sized bladder style suppressors. Two test rigs, each with matching internal diameters to the test article for which it was designed, were built in accordance with ISO-15086-2 [24].

A schematic of a representative test rig can be seen in Figure 6. Flow is provided to the system at 37.85 liters per minute from a Sauer Danfoss H1 bidirectional 9-piston pump driven by a Siemens 60 HP variable-speed AC motor operating at 1500 rpm. The motor receives power from a Siemens Simovert Masterdrive variable-frequency drive.

The frequency of the drive and pump displacement are set and controlled using xPC-Target over a CAN-bus interface. Upstream of the test section, a partially closed needle valve provides broadband noise across the frequency range of interest, 0 to 4000 Hz, to the test section. The test section of both rigs includes two rigid pipe sections; a smaller rig has an internal diameter of 0.019 m for its test section and a larger rig has an internal diameter of 0.038 m in its test sections with the test device between the pipes. The system has six piezoelectric pressure sensors, PCB model 101A06, labeled in Figure 6 as x_0 to x_5, and their placement varies with internal diameter in accordance with the ISO-15086-2 standard. Each piezoelectric sensor is mounted flush with the inside surface of the test section. The data from each sensor are collected by a 24-bit, 8-channel National Instruments data acquisition board, model 4472, mounted inside of a PC. Data is captured at 10800 samples/second and each sample record is 5120 samples long. Every test run is a vector average of 100 sample records. The transfer functions H_{ij}, between sensor i to sensor j, are the same transfer functions in the work of Johnston, et al. [22]. The reference sensor for upstream and across test component transfer functions is sensor x_1. The upstream transfer functions used for measurement are H_{01}, and H_{21}, the across test component transfer functions are H_{31}, H_{41} and H_{51}, and the downstream transfer functions are H_{34} and H_{54}.

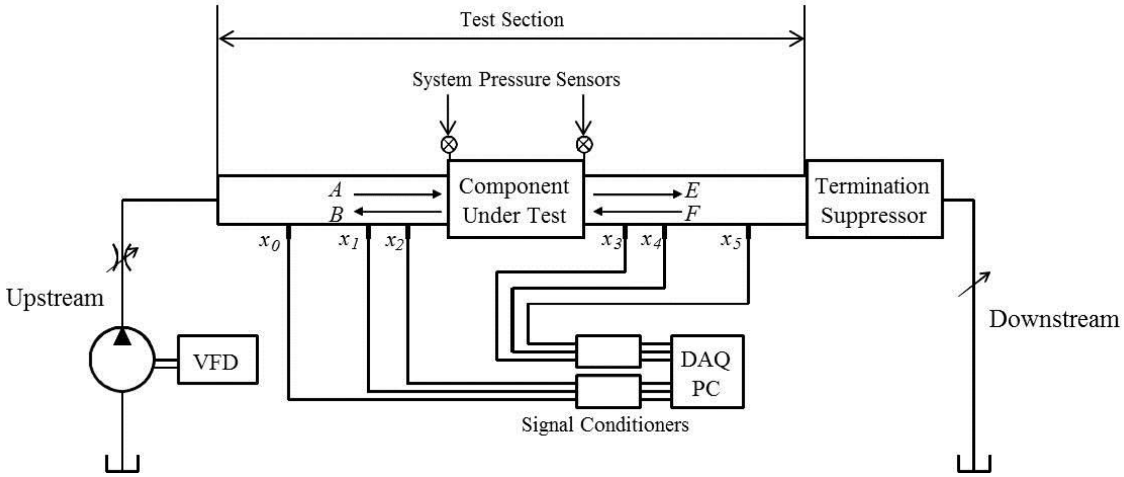

Figure 6: Schematic of test setup for measurement of fluid acoustic properties of a suppressor under test.

The transfer functions are used to calculate the wave amplitudes in the upstream and downstream test section. The pressure in the upstream section is

$$p_{upstream} = \left(Ae^{-\gamma x} + Be^{\gamma x} \right) e^{j\omega t} \tag{2.4}$$

and the particle velocity is

$$Q_{upstream} = \frac{Ae^{-\gamma x} - Be^{\gamma x}}{Z_0} e^{j\omega t} \tag{2.5}$$

where *A* and *B* are the complex amplitudes, γ is the complex wavenumber and Z_0 is the specific impedance. The upstream wave amplitudes are calculated by placing the measured acoustic pressures at each transducer into an over-determined matrix,

$$F = \begin{bmatrix} e^{-\gamma x_0} & e^{\gamma x_0} \\ e^{-\gamma x_1} & e^{\gamma x_1} \\ e^{-\gamma x_2} & e^{\gamma x_2} \end{bmatrix}. \tag{2.6}$$

The wave amplitudes are solved for by using a pseudoinverse to compute the least-squares average of

$$X = Fb, \tag{2.7}$$

where

$$X = \begin{pmatrix} A/p_1 \\ B/p_1 \end{pmatrix}, \tag{2.8}$$

where p_1 is the acoustic pressure at sensor 1, and

$$b = \begin{pmatrix} H_{01} \\ 1 \\ H_{21} \end{pmatrix}, \tag{2.9}$$

where H_{ij} is the transfer function between sensor i and j. The downstream pressure and particle velocity are given by

$$p_{downstream} = \left(Ee^{-\gamma x} + Fe^{\gamma x}\right)e^{j\omega t} \tag{2.10}$$

and

$$Q_{downstream} = \frac{Ee^{-\gamma x} - Fe^{\gamma x}}{Z_0} e^{j\omega t}\,. \tag{2.11}$$

Similarly to the upstream section, the waves amplitudes in the downstream section are calculated using

$$Y = Gc \tag{2.12}$$

where

$$Y = \begin{pmatrix} C/p_1 \\ D/p_1 \end{pmatrix}, \tag{2.13}$$

$$G = \begin{bmatrix} e^{-\gamma x_3} & e^{\gamma x_3} \\ e^{-\gamma x_4} & e^{\gamma x_4} \\ e^{-\gamma x_5} & e^{\gamma x_5} \end{bmatrix}, \tag{2.14}$$

and

$$c = \begin{pmatrix} H_{31} \\ H_{41} \\ H_{51} \end{pmatrix}. \tag{2.15}$$

The acoustic pressure, p_1, and volume velocity, Q_1, at the upstream port are related to p_2 and Q_2 at the downstream ports by a transfer matrix with elements t_{ij},

$$\begin{pmatrix} p_1 \\ Q_1 \end{pmatrix} = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \begin{pmatrix} p_2 \\ Q_2 \end{pmatrix}. \tag{2.16}$$

Pressure and velocity can be calculated from the wave amplitudes using the relations in Equations (2.4), (2.5), (2.10) and (2.11) and the wave amplitudes calculated above. The elements of the transfer matrix, Equation (2.16), can be used to calculate *TL,* using

$$\mathrm{TL} = 20\log_{10}\frac{1}{2}\left| t_{11} + \frac{t_{12}}{Z_0} + Z_0 t_{21} + t_{22} \right|. \tag{2.17}$$

Equation (2.17) can be simplified by assuming the test suppressor is geometrically symmetric end to end, and the system is assumed to be reciprocal, as seen in Pierce [8], resulting in

$$t_{11} = t_{22} \tag{2.18}$$

and

$$t_{21} = \frac{1 + t_{11}^2}{t_{12}}. \tag{2.19}$$

It can be shown from Equation (2.16), (2.18) and (2.19) that the elements of the transfer matrix can be solved for using

$$\begin{array}{ll} t_{11} = \dfrac{p_1 Q_1 + p_2 Q_2}{p_2 Q_2 + p_1 Q_1} & t_{12} = \dfrac{p_2^2 - p_1^2}{p_2 Q_2 + p_1 Q_1} \\ t_{21} = \dfrac{Q_2^2 - Q_1^2}{p_2 Q_2 + p_1 Q_1} & t_{22} = \dfrac{p_1 Q_1 + p_2 Q_2}{p_2 Q_2 + p_1 Q_1} \end{array}. \quad (2.20)$$

Using Equations (2.4), (2.5), (2.10) and (2.11) the transfer matrix elements can be solved for in terms of wave amplitudes

$$\begin{array}{ll} t_{11} = 1 & t_{12} = -Z_0 \dfrac{A^2 + 2AB + B^2 - F^2 - 2EF - E^2}{A^2 - B^2 - F^2 + E^2} \\ t_{21} = -\dfrac{A^2 - 2AB + B^2 - F^2 + 2EF - E^2}{Z_0 \left(A^2 - B^2 - F^2 + E^2 \right)} & t_{22} = 1 \end{array} .(2.21)$$

Substituting the values found in Equation (2.21) into Equation (2.17), *TL* is then

$$TL = 20 \log_{10} \left| \frac{A^2 - F^2}{AE - BF} \right|. \quad (2.22)$$

Assuming an anechoic termination, *F=0*, the equation takes the familiar form

$$TL = 20 \log_{10} \left| \frac{A}{E} \right|. \quad (2.23)$$

It can be shown that only the plane wave modes of waves *A*, *B*, *E* and *F* propagate in the upstream and downstream pipes, respectively. The cut-on angular frequency, the lowest frequency for a given mode to propagate, for a given mode is defined by

$$\omega_{lm} = c k_{lm}, \quad (2.24)$$

where, *c* is the speed of sound in meters per second, and

$$k_{lm} = \frac{j'_{lm}}{a}, \quad (2.25)$$

where j'_{ml} are the input values causing extrema of $J_m(z)$, the *m*th order Bessel function of argument *z*, and *a* is the circular cross section radius of the pipe [6]. The plane wave mode has indices of l=0 and m=0, and its cut-on frequency is 0 Hz; the lowest non-plane wave mode is l=0 and m=1. By applying equation (2.25) through (2.24), the frequency range of interest, 0 to 4000 Hz, is well below the cut-on frequencies in both inlet pipes, and the calculated values of the cut-on frequencies are shown in Table 2.

Table 2: Cut-on frequencies of first non-planar mode in a cylindrical pipe

Inlet Pipe ID (m)	Cut-on frequency of first non-plane mode (kHz)
0.019	43.0
0.038	21.5

Figure 6 also shows two static pressure sensors, used to measure mean system pressure, are mounted in the system, one immediately upstream of the test suppressor and one immediately downstream of the test suppressor. The measured difference between the sensors yields the pressure loss across the device, which was never measured to be greater than the minimum sensor resolution of 70 kPa for any test.

The system pressure is controlled by a second needle valve located downstream of the test section. To isolate the noise generate by the loading valve a termination suppressor is connected downstream of the test section, and isolates the test section from downstream noise, improving coherence in the transfer functions between the dynamic pressure sensors. A thermocouple is used to measure the temperature of the hydraulic fluid for each test; the temperature of the compressed gas in the test suppressor is estimated to be equal to the temperature of the hydraulic fluid.

2.1.2 Sensor Calibration

The sensors are calibrated pursuant to ISO 15086-2 [24]. The sensors are all located on the same axial position of a calibration block, Figure 7, which is perpendicular to the direction of the pressure wave. As testing measurements occur on a frequency-by-frequency basis, calibration must also be done on a frequency-by-frequency basis. The inner diameter of the calibration block is narrower than either rig, and it can be shown that plane wave behavior can also be assumed within the block. Plane wave behavior ensures the pressure sensors will be exposed to the same pressure wave amplitude at each frequency with no phase difference. The sensor outputs are then compared using the same transfer functions as used during testing. Ideally, the magnitude of all transfer functions will be 1 and the phase will be 0°. Slight manufacturing differences in each sensor will cause deviations from the ideal value. ISO 15086-2 [24] sets the maximum allowable deviation for amplitude larger than 1% needs to be corrected, while a phase deviation larger 0.5° need to be corrected, though in practice all differences are corrected.

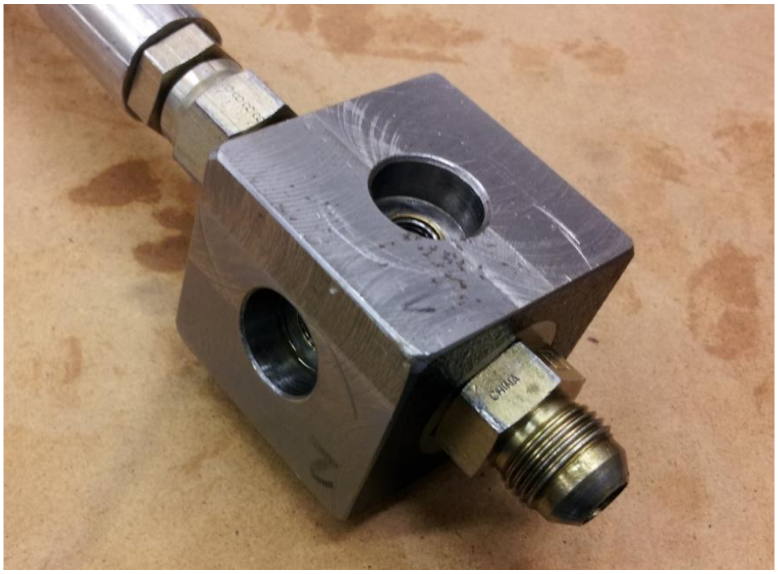

Figure 7: Calibration block without sensors

2.1.3 Coherence

The coherence between any sensor pair is used to ensure the transfer function is correctly relating data taken at those two sensors. Coherence values range from 0 to 1, with higher values meaning higher correlation and linear relationship between the sensors. The coherence is calculated by

$$C_{xy} = \frac{\left|G_{xy}\right|^2}{G_{xx}G_{yy}} \quad (2.26)$$
,

where G_{xy} is the cross-spectral density between x and y, and G_{xx} and G_{yy} are the respective autospectral densities. When the coherence value decreases it indicates noise has entered the measurement, as acoustic waves propagate linearly in hydraulic oil. Frequencies with a coherence value of less 0.9 are considered invalid and ignored in computation of *TL*. The termination suppressor ensures noise does not enter from downstream, improving the coherence.

2.2 Measured Transmission Loss Performance of Single Suppressors

In order to investigate the effect of charge pressure ratio (CPR) on suppressor performance the method developed in Section 2.1 was applied to both a WM-5081 suppressor and a WM-5138 suppressor. The resulting transmission loss curves are seen in Figure 8 - Figure 11. Figure 8 and Figure 9 show the effect of the changing the charge pressure while holding system pressure constant at 10.3 MPa. In the frequency ranges where no *TL* data is shown, the coherence for any of the transfer functions used dropped below 0.9 and was deleted from the data set. The *TL* rises across the entire frequency range of interest as the CPR approaches 1. In Figure 8, an improvement of seven to ten dB is seen across the range of frequencies by raising the CPR from 0.3, corresponding to a charge pressure of 3.10 MPa, to a CPR of 0.9, which corresponds to a charge pressure of 9.31 MPa. The improvement in *TL* is significant, and shows why optimizing the suppressor(s) to the correct charge pressure(s) is necessary in practice. With a CPR of over 1 the *TL* drastically drops to less than a maximum of five dB, approximately twenty-five dB less than the *TL* obtained with a CPR of 0.9. It can also be seen that a suppressor with a CPR of 0.3 out-performs a suppressor with a CPR over 1 for the entire frequency range. The downward spikes in *TL* at frequencies of 600 and 1000 Hz are attributed to numerical artifacts, and the cause of the artifacts is not known precisely, but may be due to standing-waves in the upstream and downstream test sections contaminating the measurements. Figure 9 shows a similar trend of *TL* in relation to CPR, the only difference is suppressor size as well as line size. Conclusions are difficult to draw from Figure 9 due to significant numerical artifacts in frequencies above 500 Hz. In frequencies below 500 Hz, Figure 9 shows that increasing the CPR improves *TL*. It will be shown below, in Section 5.1.1, that a majority of the spectral content of the pressure ripple is in frequencies below 500 Hz, allowing this data to inform this thesis.

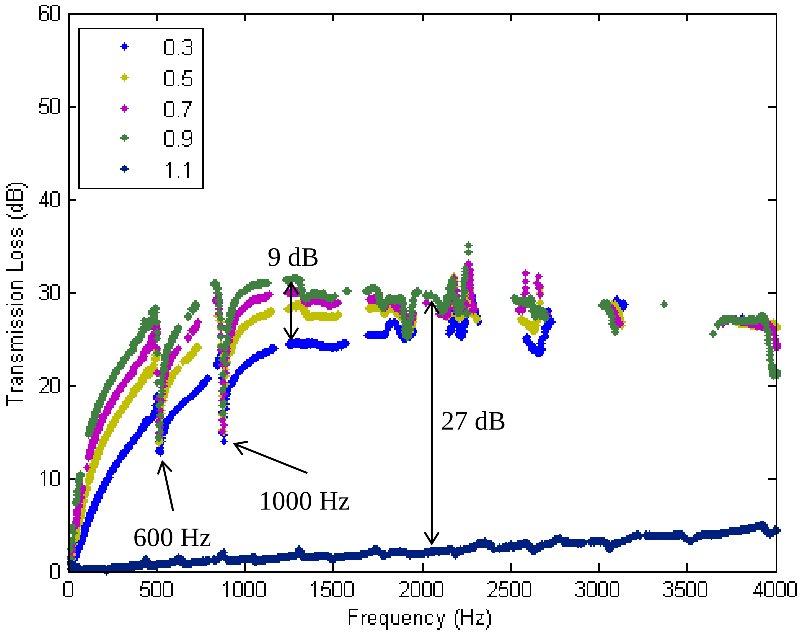

Figure 8: Transmission loss for WM-5081 Suppressor at 10.3 MPA system pressure with varying charge pressures

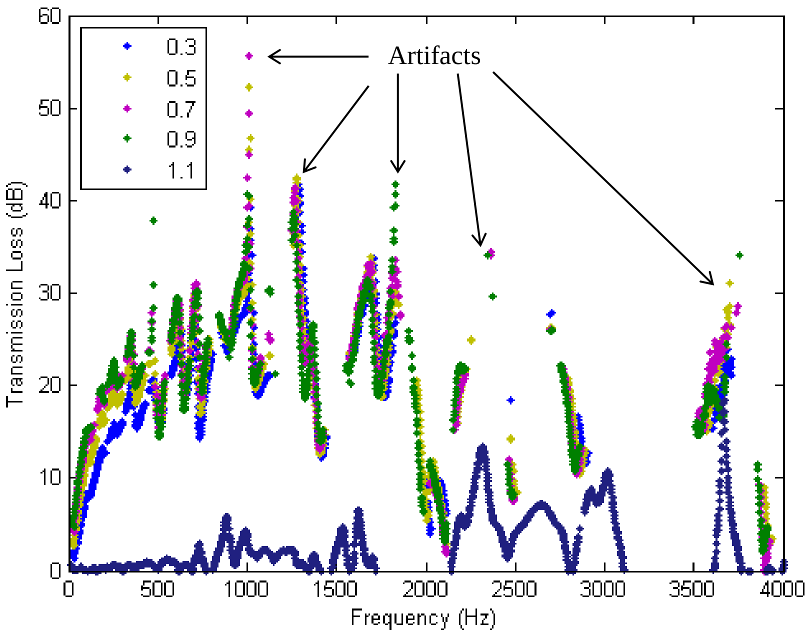

Figure 9: Transmission loss for WM-5138 Suppressor at 10.3 MPA system pressure with varying charge pressures

Figure 10 and Figure 11 show the effect of holding the CPR constant at 0.5 while varying system pressure, which requires that the charge pressure be increased for each increase in system pressure. In Figure 10, the lower system pressures exhibit higher *TL*, especially in the frequency range from 0 to 1000 Hz. In the frequency range from 1500 Hz to 2000 Hz the *TL* values converge, but the lower system pressure still exhibit the highest *TL*. The largest difference between *TL* of two system pressures for similar frequencies is approximately 7 dB, which shows that for the same CPR, suppressors operating at a lower system pressure exhibit higher *TL*. Similar to Figure 9, data taken from the rig designed for the WM-5138 shows significant numerical artifacts in frequencies above 500 Hz; however the data follows similar trends as the data in Figure 10. The *TL* decreases to close to 0 dB at a frequency of 2000 Hz for all measured TL curves.

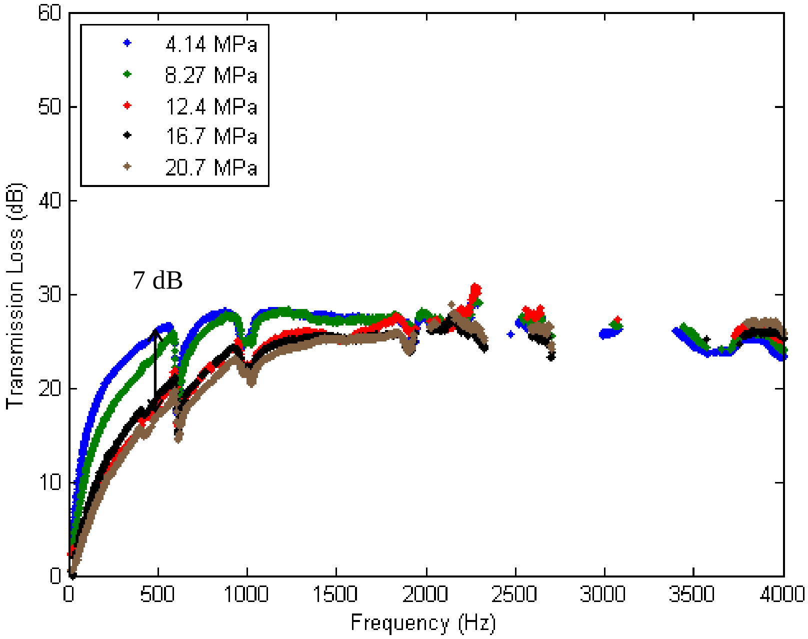

Figure 10: Transmission loss for WM-5081 Suppressor at 50% CPR for several system pressures

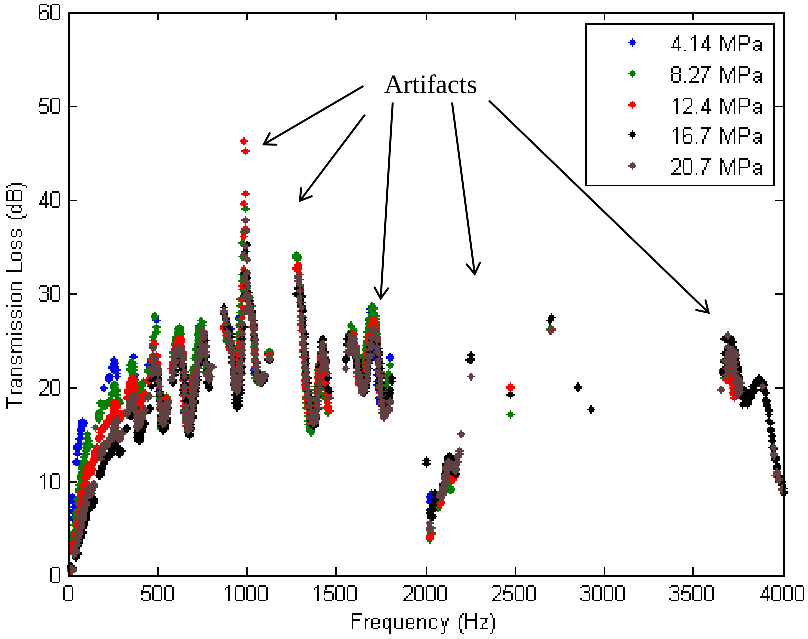

Figure 11: Transmission loss for WM-5138 Suppressor at 50% CPR for several system pressures

CHAPTER 3

MODELING OF SINGLE SUPPRESSOR TRANSMISSION LOSS

An equivalent-fluid model is used to predict the transmission loss performance of a suppressor. Its full development is seen in Marek, et al. [4]. Several authors have published work detailing the modeling of mufflers in airborne applications. Airborne mufflers primarily use fibrous linings as a damper to absorb acoustic energy, while fluid-borne suppressors primarily use nitrogen contained in a bladder to add compliance to the device. However, many similarities do exists between the geometry of fluid-borne suppressors and airborne suppressors, allowing the current literature pertaining to airborne muffler models to inform the fluid-borne bladder-style model. Work from Selamet [26] greatly influenced the model developed below, as the work focuses on the perforate layer as well as inlet/outlet extensions, both geometries found in the Wilkes and McLean suppressors being studied in this thesis [1]. Studies by Selamet and Li [7] and Denia, et al. [27] were informative as to the acoustic behavior in the inlet/outlet extensions in expansion chambers and mufflers, both studies note the extensions improve the broadband nature of *TL*. Both studies use the Helmholtz equation,

$$\nabla^2 P + k^2 P = 0, \tag{3.1}$$

to model the pressure in the entire system. Other studies from Denia, et al. [27], Lee, et al. [28] and Selamet, et al. [29] helped further describe the acoustic behavior, i.e. the pressure and particle displacement, at the perforate layer. The aforementioned studies calculate the specific impedance of a single hole in the perforate layer,

$$Z_h = \frac{p_1 - p_2}{u_h} = R_S + i\omega\rho_0 l_{eff} \,, \tag{3.2}$$

where p_1 and p_2 are the acoustic pressures directly on each side of the perforate layer, u_h is the particle velocity at the hole, R_s is the specific resistance, ω is the angular frequency, ρ_0 is the density of the fluid and l_{eff} is

$$l_{eff} = t_w + \alpha d_h \,, \tag{3.3}$$

where t_w is the wall thickness of the layer, d_h is the hole diameter and α is the end correction coefficient. The impedance of the perforate layer can be calculated with

$$Z_p = \frac{Z_h}{\phi} \,, \tag{3.4}$$

where ϕ is the porosity of the perforate layer.

Even with their similarities, several other important differences remain between airborne muffler models and fluid-borne suppressor models. First, most airborne mufflers are designed to work at or near atmosphere pressure, while fluid-borne suppressors operate up to pressures of 34.5 MPa, causing different behavior, such as sound speed, in the nitrogen-charged bladder [6]. Second, mean flow can affect acoustic measurements and must be taken into account when measuring *TL*. The hydraulic fluid model, as shown in Marek, et al. [4], assumes linear acoustic behavior in the entirety of the suppressor. The model also assumes the mean flow velocity is minimal compared to the speed of sound in the fluid.

3.1 Suppressor Modeling

The suppressor is modeled by Marek, et al. [4] using three coaxial regions shown in Figure 12 along with waves *A*, *B*, *E, F, G* and *H*. The regions are separated based on similarity of boundary conditions. The first region represents the upstream and downstream pipes, therefore representing waves A, B, E and F, which can be assumed to be plane waves as shown Chapter 2, and only hydraulic fluid is present in this region. The second region of the device begins at the most upstream inlet port of the annulus, shown as $x=0$ in Figure 12, and ends at the most downstream port of the annulus, shown as $x=L$. Because of the non-rigid behavior at the bladder, plane wave behavior may not be assumed in this region, therefore $G_{R,n}$ will represent all forward traveling modes and $H_{R,n}$ will represent all reverse traveling modes. The subscript R represents the region the wave is propagating in, as waves G and H will propagate in both Region 2 and 3; the subscript n represents the index of a given wave mode, with 0 indicating the plane-wave mode. Region 3 represents the area where the fluid and nitrogen may contact the bladder, from opposite radial directions, upstream of $x=0$ or downstream of $x=L$. The difference between Regions 2 and 3 is Region 2 includes the perforate layer and annulus while Region 3 does not. The positive direction of travel is to the right in Figure 12.

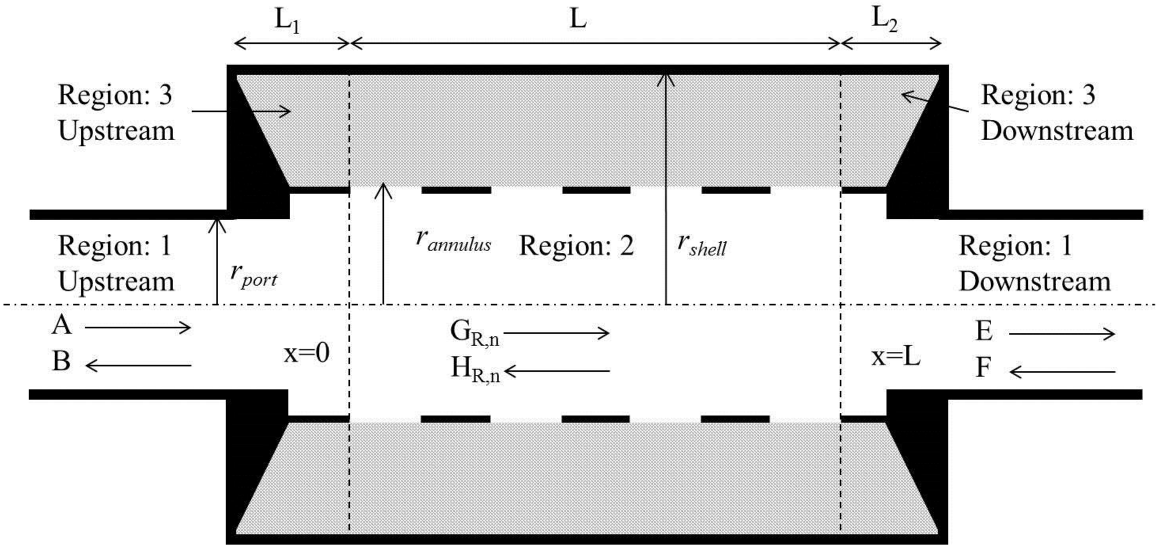

Figure 12: Suppressor model and acoustic waves

In order to accurately model suppressor behavior, the geometry of the suppressor must be known. The geometry can be calculated from dimensions shown in Figure 12. First the charged volume of gas is calculated using geometry. The initial volume of the bladder, before the system is pressurized, is

$$V_0 = \pi\left(L_1 + L + L_2\right)\left(r_{shell}^2 - r_{annulus}^2\right), \tag{3.5}$$

where *L* is the length of Region 2, *L₁* and *L₂* represent the upstream and downstream lengths of Region 3 and the radii are shown in Figure 12. The radius of the bladder when it is charged with nitrogen and the system is pressurized, $r_{compressed}$, can be calculated with

$$r_{compressed} = \sqrt{r_{shell}^2 - \frac{P_c V_0}{P_s\left(L_1 + L + L_2\right)\pi}}\ , \tag{3.6}$$

where P_c is the charge pressure, and P_s is the system pressure. The model assumes the bladder is limp mass, i.e. the bladder is assumed to have mass and oscillate but it is non-elastic.

The model uses Lamé parameters, λ_f and λ_B, to represent the elasticity of the hydraulic fluid and the bladder, respectively. The parameter λ_B represents both the bladder and the pressurized nitrogen. Lamé's second parameter, μ, represents the shear modulus, zero for both fluids, making λ_f and λ_B directly equivalent to the bulk moduli for both media. In addition, a shear modulus of zero means only longitudinal waves will propagate in the suppressor.

Sound speeds are defined as

$$c_f = \sqrt{\frac{\lambda_f}{\rho_f}}\ , \tag{3.7}$$

$$c_B = \sqrt{\frac{\lambda_B}{\rho_B}}\,, \tag{3.8}$$

and for angular frequency ω in rad/s, wavenumbers k are defined as

$$k_f = \frac{\omega}{c_f}, \quad k_B = \frac{\omega}{c_B}, \tag{3.9}$$

where the subscripts f and B designate fluid and bladder, respectively.

The wave number for a given propagating mode, n, in a given region, R, can be decomposed into radial and axial components, with subscripts r and x representing radial and axial mode components, respectively:

$$k_f^2 = k_{Rx,n}^2 + k_{Rrf,n}^2, \tag{3.10}$$

$$k_B^2 = k_{Rx,n}^2 + k_{RrL,n}^2. \tag{3.11}$$

Within the suppressor the wavenumber in the axial direction is the same for the hydraulic fluid and the nitrogen in the bladder; however, the radial wavenumber can differ between the propagation media. Acoustic displacement is represented by $u_{Rr,n}$ and $u_{Rx,n}$, where the subscript R represents the region, the subscript r or x represents radial or axial modes, and the subscript n represents the mode number. The acoustic particle displacements are given for all three regions by:

$$u_{1Ur,n} = -k_{1rf,n} J_1\left(k_{1rf,n} r\right) A \mathrm{e}^{-\mathrm{i}k_{1x,n}x}\, \mathrm{e}^{\mathrm{i}\omega t}, \tag{3.12}$$

$$u_{1Dr,n} = -k_{1rf,n} J_1\left(k_{1rf,n} r\right) E\, \mathrm{e}^{-\mathrm{i}k_{1x,n}x'}\, \mathrm{e}^{\mathrm{i}\omega t}, \tag{3.13}$$

$$u_{2r,n} = \begin{cases} -k_{2rf,n} y_{1,n} J_1\left(k_{2rf,n} r\right) G_{2,n}\, \mathrm{e}^{-\mathrm{i}k_{2x,n}x}\, \mathrm{e}^{\mathrm{i}\omega t}, \; r < r_{annulus} \\ -k_{2rf,n}\left(y_{2,n} J_1\left(k_{2rf,n} r\right) + y_{3,n} Y_1\left(k_{2rf,n} r\right)\right) G_{2,n}\, \mathrm{e}^{-\mathrm{i}k_{2x,n}x}\, \mathrm{e}^{\mathrm{i}\omega t}, \; r_{annulus} \le r < r_{compressed} \\ -k_{2rL,n}\left(y_{4,n} J_1\left(k_{2rL,n} r\right) + y_{5,n} Y_1\left(k_{2rL,n} r\right)\right) G_{2,n}\, \mathrm{e}^{-\mathrm{i}k_{2x,n}x}\, \mathrm{e}^{\mathrm{i}\omega t}, \; r \ge r_{compressed} \end{cases}, \tag{3.14}$$

$$u_{3Ur,n}=\begin{cases}-k_{3rf,n}\left(y_{6,n}J_1\left(k_{3rf,n}r\right)+y_{7,n}Y_1\left(k_{3rf,n}r\right)\right)G_{3U,n}\,\mathrm{e}^{-\mathrm{i}k_{3x,n}x}\,\mathrm{e}^{\mathrm{i}\omega t},\ r<r_{compressed}\\-k_{3rL,n}\left(y_{8,n}J_1\left(k_{3rL,n}r\right)+y_{9,n}Y_1\left(k_{3rL,n}r\right)\right)G_{3U,n}\,\mathrm{e}^{-\mathrm{i}k_{3x,n}x}\,\mathrm{e}^{\mathrm{i}\omega t},\ r\geq r_{compressed}\end{cases},\quad(3.15)$$

$$u_{3Dr,n}=\begin{cases}-k_{3rf,n}\left(y_{6,n}J_1\left(k_{3rf,n}r\right)+y_{7,n}Y_1\left(k_{3rf,n}r\right)\right)G_{3D,n}\,\mathrm{e}^{-\mathrm{i}k_{3x,n}x'}\,\mathrm{e}^{\mathrm{i}\omega t},\ r<r_{compressed}\\-k_{3rL,n}\left(y_{8,n}J_1\left(k_{3rL,n}r\right)+y_{9,n}Y_1\left(k_{3rL,n}r\right)\right)G_{3D,n}\,\mathrm{e}^{-\mathrm{i}k_{3x,n}x'}\,\mathrm{e}^{\mathrm{i}\omega t},\ r\geq r_{compressed}\end{cases},\quad(3.16)$$

$$u_{1Ux,n}=-\mathrm{i}k_{1x,n}J_0\left(k_{1rf,n}r\right)A\mathrm{e}^{-\mathrm{i}k_{1x,n}x}\,\mathrm{e}^{\mathrm{i}\omega t},\quad(3.17)$$

$$u_{1Dx,n}=-\mathrm{i}k_{1x,n}J_0\left(k_{1rf,n}r\right)A\mathrm{e}^{-\mathrm{i}k_{1x,n}x'}\,\mathrm{e}^{\mathrm{i}\omega t},\quad(3.18)$$

$$u_{2x,n}=\begin{cases}-\mathrm{i}k_{2x,n}y_{1,n}J_0\left(k_{2rf,n}r\right)G_{2,n}\,\mathrm{e}^{-\mathrm{i}k_{2x,n}x}\,\mathrm{e}^{\mathrm{i}\omega t},\ r<r_{annulus}\\-\mathrm{i}k_{2x,n}\left(y_{2,n}J_0\left(k_{2rf,n}r\right)+y_{3,n}Y_0\left(k_{2rf,n}r\right)\right)G_{2,n}\,\mathrm{e}^{-\mathrm{i}k_{2x,n}x}\,\mathrm{e}^{\mathrm{i}\omega t},\ r_{annulus}\leq r<r_{compressed}\\-\mathrm{i}k_{2x,n}\left(y_{4,n}J_0\left(k_{2rL,n}r\right)+y_{5,n}Y_0\left(k_{2rL,n}r\right)\right)G_{2,n}\,\mathrm{e}^{-\mathrm{i}k_{2x,n}x}\,\mathrm{e}^{\mathrm{i}\omega t},\ r\geq r_{compressed}\end{cases},\quad(3.19)$$

$$u_{3Ux,n}=\begin{cases}-\mathrm{i}k_{3x,n}\left(y_{6,n}J_0\left(k_{3rf,n}r\right)+y_{7,n}Y_0\left(k_{3rf,n}r\right)\right)G_{3U,n}\,\mathrm{e}^{-\mathrm{i}k_{3x,n}x}\,\mathrm{e}^{\mathrm{i}\omega t},\ r<r_{compressed}\\-\mathrm{i}k_{3x,n}\left(y_{8,n}J_0\left(k_{3rL,n}r\right)+y_{9,n}Y_0\left(k_{3rL,n}r\right)\right)G_{3U,n}\,\mathrm{e}^{-\mathrm{i}k_{3x,n}x}\,\mathrm{e}^{\mathrm{i}\omega t},\ r\geq r_{compressed}\end{cases},\quad(3.20)$$

$$u_{3Dx,n}=\begin{cases}-\mathrm{i}k_{3x,n}\left(y_{6,n}J_0\left(k_{3rf,n}r\right)+y_{7,n}Y_0\left(k_{3rf,n}r\right)\right)G_{3D,n}\,\mathrm{e}^{-\mathrm{i}k_{3x,n}x'}\,\mathrm{e}^{\mathrm{i}\omega t},\ r<r_{compressed}\\-\mathrm{i}k_{3x,n}\left(y_{8,n}J_0\left(k_{3rL,n}r\right)+y_{9,n}Y_0\left(k_{3rL,n}r\right)\right)G_{3D,n}\,\mathrm{e}^{-\mathrm{i}k_{3x,n}x'}\,\mathrm{e}^{\mathrm{i}\omega t},\ r\geq r_{compressed}\end{cases},\quad(3.21)$$

where J_m and Y_m are m^{th} order Bessel functions of the first and second kind, relative complex amplitudes of coefficients $y_{1,n}$ to $y_{5,n}$ and $y_{6,n}$ to $y_{9,n}$ are unique for each mode n in Regions 2 and 3, and $x'=x-L$. Acoustic pressure for R region and n mode is represented by:

$$p_{1U,n}=k_f^2J_0\left(k_{1rf,n}r\right)A\mathrm{e}^{-\mathrm{i}k_{1x,n}x}\,\mathrm{e}^{\mathrm{i}\omega t},\quad(3.22)$$

$$p_{1D,n}=k_f^2J_0\left(k_{1rf,n}r\right)A\mathrm{e}^{-\mathrm{i}k_{1x,n}x'}\,\mathrm{e}^{\mathrm{i}\omega t},\quad(3.23)$$

$$p_{2,n} = \begin{cases} k_f^2 y_{1,n} J_0\left(k_{2rf,n} r\right) G_{2,n}\, \mathrm{e}^{-\mathrm{i}k_{2x,n}x}\, \mathrm{e}^{\mathrm{i}\omega t},\, r < r_{annulus} \\ k_f^2\left(y_{2,n} J_0\left(k_{2rf,n} r\right) + y_{3,n} Y_0\left(k_{2rf,n} r\right)\right) G_{2,n}\, \mathrm{e}^{-\mathrm{i}k_{2x,n}x}\, \mathrm{e}^{\mathrm{i}\omega t},\, r_{annulus} \le r < r_{compressed} \\ k_L^2\left(y_{4,n} J_0\left(k_{2rL,n} r\right) + y_{5,n} Y_0\left(k_{2rL,n} r\right)\right) G_{2,n}\, \mathrm{e}^{-\mathrm{i}k_{2x,n}x}\, \mathrm{e}^{\mathrm{i}\omega t},\, r \ge r_{compressed} \end{cases} ,\tag{3.24}$$

$$p_{3U,n} = \begin{cases} k_f^2\left(y_{6,n} J_0\left(k_{3rf,n} r\right) + y_{7,n} Y_0\left(k_{3rf,n} r\right)\right) G_{3U,n}\, \mathrm{e}^{-\mathrm{i}k_{3x,n}x}\, \mathrm{e}^{\mathrm{i}\omega t},\, r < r_{compressed} \\ k_L^2\left(y_{8,n} J_0\left(k_{3rL,n} r\right) + y_{9,n} Y_0\left(k_{3rL,n} r\right)\right) G_{3U,n}\, \mathrm{e}^{-\mathrm{i}k_{3x,n}x}\, \mathrm{e}^{\mathrm{i}\omega t},\, r \ge r_{compressed} \end{cases} , \tag{3.25}$$

$$p_{3D,n} = \begin{cases} k_f^2\left(y_{6,n} J_0\left(k_{3rf,n} r\right) + y_{7,n} Y_0\left(k_{3rf,n} r\right)\right) G_{3D,n}\, \mathrm{e}^{-\mathrm{i}k_{3x,n}x'}\, \mathrm{e}^{\mathrm{i}\omega t},\, r < r_{compressed} \\ k_L^2\left(y_{8,n} J_0\left(k_{3rL,n} r\right) + y_{9,n} Y_0\left(k_{3rL,n} r\right)\right) G_{3D,n}\, \mathrm{e}^{-\mathrm{i}k_{3x,n}x'}\, \mathrm{e}^{\mathrm{i}\omega t},\, r \ge r_{compressed} \end{cases} . \tag{3.26}$$

Assuming mean flow can be neglected allows an assumption that the values for the reverse traveling wavemodes in Equations (3.12) to (3.25) can be found by replacing the forward mode with its matching reverse traveling mode; and by replacing all instances of $k_{Rx,n}$ with $-k_{Rx,n}$. The negligible flow assumption can be validated by calculation of the Mach number. As shown in Chapter 2, the Mach number for flow in pipes with diameters of 0.019 m and 0.038 m are below 0.001 and allow mean flow to be neglected.

As discussed above, a given mode, *n*, in a given region *R* is described by a unique axial wavenumber, $k_{Rx,n}$. In order to solve for the wave number, an eigenequation must be solved for in each region. Since the mean flow velocity is negligible the solutions of the eigenequation are $\pm k_{Rx,n}$, meaning either forward or reverse travelling modes need to be solved for, in this case the positive travelling modes are obtained. Solving the eigenequation requires boundary conditions which accurately reflect the physical system being modeled. In Region 1, where waves *A* and *B* propagate upstream of the suppressor and waves *E* and *F* propagate downstream of the suppressor, the pipe boundary is assumed to be rigid; therefore a zero radial displacement condition must be met at the outer wall,

$$\left[u_{1r,n} \right]_{r=r_{port}} = 0\,. \tag{3.27}$$

Equations (3.12), (3.13), (3.22) and (3.23) are solved for use the boundary condition seen in equation (3.27) to solve for the pressure and particle displacement in Region 1.

Region 2, where waves *G* and *H* propagate, includes the hydraulic fluid, the bladder, the annulus and the perforate layer. Equations (3.28) to (3.33) as well as Equations (3.14), (3.19) and (3.24) are solved simultaneously with boundary condition to find the wavenumber, $k_{2x,n}$ as well as the relative amplitudes of $y_{1,n}$ through $y_{5,n}$. The first boundary condition follows from assuming that the outer shell of the suppressor is rigid, and there is zero displacement at the outer wall,

$$\left[u_{2r,n} \right]_{r=r_{shell}} = 0\,. \tag{3.28}$$

The displacement of the nitrogen in the bladder must match the displacement of the hydraulic fluid,

$$\left[u_{2r,n} \right]_{r=r_{compressed}-} = \left[u_{2r,n} \right]_{r=r_{compress}+}, \tag{3.29}$$

where $r_{\text{compressed}}$- and $r_{\text{compressed}}$+ represent the limit as r approaches $r_{\text{compressed}}$ from either side of the bladder. In addition the forces must also be balanced across the bladder,

$$\left[p_2 \right]_{r=r_{compressed}-} = \left[p_2 + \sigma_s \ddot{u}_{2r} \right]_{r=r_{compressed}+} = \left[p_2 - \omega^2 \sigma_s u_{2r} \right]_{r=r_{compressed}+}, \tag{3.30}$$

where $\ddot{u}_{2r}$ is the second temporal derivative of acoustic displacement u_{2r} and σ_s is the area density of the bladder calculated from

$$\sigma_s = \frac{m_b}{2\pi r_{compressed} L_T}, \tag{3.31}$$

where m_b is the mass of the bladder and L_T is the length of the bladder. In addition, the acoustic displacement and impedance condition across the perforate layer are given by

$$\left[u_{2r,n}\right]_{r=r_{annulus}-} = \left[u_{2r,n}\right]_{r=r_{annulus}+}, \tag{3.32}$$

and

$$\left[p_{2,n}\right]_{r=r_{annulus}+} - \left[p_{2,n}\right]_{r=r_{annulus}-} = Z_p\left[u_{2r,n}\right]_{r=r_{annulus}}, \tag{3.33}$$

where Z_p can be calculated from (3.4). Equations (3.14), (3.19) and (3.24) are solved with the boundary conditions in Equations (3.28) through (3.33), and the resulting eigenvalues are the wavenumbers, $k_{2x,n}$, which allow the acoustic pressure and particle displacement for each mode in Region 2 to be calculated. The boundary conditions in Region 3 are similar to the conditions in Region 2, omitting the perforate layer:

$$\left[u_{3r,n}\right]_{r=r_{shell}} = 0, \tag{3.34}$$

$$\left[u_{3r,n}\right]_{r=r_{compressed}-} = \left[u_{3r,n}\right]_{r=r_{compressed}+}, \tag{3.35}$$

$$\left[p_3\right]_{r=r_{compressed}-} = \left[p_3 - \omega^2\sigma_s u_{3r}\right]_{r=r_{compressed}+}. \tag{3.36}$$

Solving Equations (3.20), (3.21), (3.25) and (3.26) with the boundary conditions shown in Equations (3.34) through (3.36), the resulting eigenvalues are they wavenumbers, $k_{3x,n}$, which allow the acoustic pressure and particle displacement for each mode in Region 3 to be calculated.

A finite number of radial modes, N, are then considered to solve for the modal amplitudes for all waves by using continuity of pressure and displacement at the boundaries between regions. An anechoic termination is assumed for this model, meaning

$$F = 0\,. \tag{3.37}$$

The forward traveling plane wave *A*, is assumed to be unity for the entire frequency range of interest and the other wave amplitudes are in reference to *A* for a given frequency. Other simplifying assumptions are rigid boundaries at $x = -L_1$ and $x = L + L_2$, allowing a simple relationship between the forward and reverse traveling waves in Region 3,

$$G_{3U,n} = H_{3U,n}\mathrm{e}^{-2\mathrm{i}k_{3x,n}L_1}\,, \tag{3.38}$$

$$H_{3D,n} = G_{3D,n}\mathrm{e}^{-2\mathrm{i}k_{3x,n}L_2}\,. \tag{3.39}$$

The remaining axial modes may be solved in the form of are integrals,

$$\int_0^{r_a}\sum_{n=0}^{N-1}\left(\left[p^+_{1U,n}\right]_{x=0} + \left[p^-_{1U,n}\right]_{x=0}\right)rdr = \int_0^{r_a}\sum_{n=0}^{N-1}\left(\left[p^+_{2,n}\right]_{x=0} + \left[p^-_{2,n}\right]_{x=0}\right)rdr\,, \tag{3.40}$$

$$\int_0^{r_a}\sum_{n=0}^{N-1}\left(\left[p^+_{1D,n}\right]_{x=L} + \left[p^-_{1D,n}\right]_{x=L}\right)rdr = \int_0^{r_a}\sum_{n=0}^{N-1}\left(\left[p^+_{2,n}\right]_{x=L} + \left[p^-_{2,n}\right]_{x=L}\right)rdr\,, \tag{3.41}$$

$$\int_{r_{annulus}}^{r_c}\sum_{n=0}^{N-1}\left(\left[p^+_{3U,n}\right]_{x=0} + \left[p^-_{3U,n}\right]_{x=0}\right)rdr = \int_{r_{annulus}}^{r_c}\sum_{n=0}^{N-1}\left(\left[p^+_{2,n}\right]_{x=0} + \left[p^-_{2,n}\right]_{x=0}\right)rdr\,, \tag{3.42}$$

$$\int_{r_{annulus}}^{r_c}\sum_{n=0}^{N-1}\left(\left[p^+_{3D,n}\right]_{x=L} + \left[p^-_{3D,n}\right]_{x=L}\right)rdr = \int_{r_{annulus}}^{r_c}\sum_{n=0}^{N-1}\left(\left[p^+_{2,n}\right]_{x=L} + \left[p^-_{2,n}\right]_{x=L}\right)rdr\,, \tag{3.43}$$

$$\int_0^{r_b}\sum_{n=0}^{N-1}\left(\left[u^+_{2x,n}\right]_{x=0} + \left[u^-_{2x,n}\right]_{x=0}\right)rdr = \begin{cases} \displaystyle\int_0^{r_b}\sum_{n=0}^{N-1}\left(\left[u^+_{1Ux,n}\right]_{x=0} + \left[u^-_{1Ux,n}\right]_{x=0}\right)rdr,\ r_b < r_{port} \\ U_U\,,\ r_{port} \le r_b < r_{annulus} \\ \displaystyle U_U + \int_{r_1}^{r_b}\sum_{n=0}^{N-1}\left(\left[u^+_{3Ux,n}\right]_{x=0} + \left[u^-_{3Ux,n}\right]_{x=0}\right)rdr,\ r_b \ge r_{annulus} \end{cases} ,(3.44)$$

$$\int_0^{r_b}\sum_{n=0}^{N-1}\left(\left[u^+_{2x,n}\right]_{x=L}+\left[u^-_{2x,n}\right]_{x=L}\right)rdr=\begin{cases}\int_0^{r_b}\sum_{n=0}^{N-1}\left(\left[u^+_{1Dx,n}\right]_{x=0}+\left[u^-_{1Dx,n}\right]_{x=0}\right)rdr,\, r_b<r_{port}\\ U_D,\, r_{port}\le r_b<r_{annulus}\\ U_D+\int_{r_1}^{r_b}\sum_{n=0}^{N-1}\left(\left[u^+_{3Dx,n}\right]_{x=L}+\left[u^-_{3Dx,n}\right]_{x=L}\right)rdr,\, r_b\ge r_{annulus}\end{cases}\qquad ,(3.45)$$

$$U_U=\int_0^{r_{port}}\sum_{n=0}^{N-1}\left(\left[u^+_{1Ux,n}\right]_{x=0}+\left[u^-_{1Ux,n}\right]_{x=0}\right)rdr\,, \qquad (3.46)$$

$$U_D=\int_0^{r_{port}}\sum_{n=0}^{N-1}\left(\left[u^+_{1Dx,n}\right]_{x=L}+\left[u^-_{1Dx,n}\right]_{x=L}\right)rdr$$

$$r_a=\frac{m+1}{M}r_{port}\,,\quad r_b=\frac{m+1}{M}r_{shell}\,,\quad r_c=r_{annulus}+\frac{m+1}{M}\left(r_{shell}-r_{annulus}\right), \qquad (3.47)$$

where m = 0 to M-1, and M = N.

The *TL* for a single suppressor can then be found using

$$TL=20\log_{10}\left(\frac{A}{E}\right) \qquad (3.48)$$

where A and E are the wave amplitudes.

For a system operating over a broad system pressure range, there is a possibility the suppressor operates with a CPR over 1. This condition violates the linearity assumptions made by the model as the bladder remains in contact with the annulus. In order to model this condition, the boundary conditions in Regions 2 and 3 are changed to rigid behavior at $r_{annulus}$, emulating an expansion chamber.

3.2 Predicted Transmission Loss Curves for a Single Suppressor

For a bladder-style suppressor, *TL* is a function of its geometric dimensions, shown in Figure 13, and CPR. Critical dimensions include the inner radius of the inlet and outlet ports, $r_{port,}$ the outer radius of the annulus, $r_{annulus}$, the inner radius of the shell,

r_{shell}, and the effective internal length of the device, L. In addition to the geometric properties, *TL* is also dependent on the ratio of nitrogen charge pressure to system pressure. Two models of commercially available suppressors are considered in this thesis; the models are the Wilkes & McLean WM-5081 and the Wilkes & McLean WM-5381. The dimensions of the suppressors are shown in Table 3. Each suppressor model requires inlet and outlet pipes of proper size to ensure constant mean flow through the system. The diameters of the inlet and outlet pipes for the two suppressor models are 0.019 m and 0.038 m, respectively.

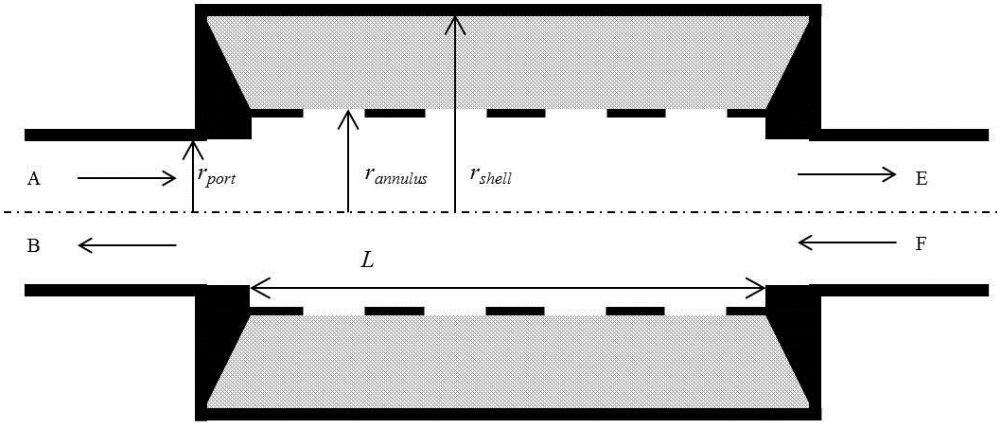

Figure 13: Single suppressor dimensions and acoustic waves

Table 3: Dimensions of bladder-style suppressor used in this study

Parameter	WM-5081	WM-5138
r_{port}	0.00953 m	0.0176 m
$r_{annulus}$	0.0102 m	0.0252 m
r_{shell}	0.0241 m	0.0417 m
L	0.0447 m	0.0682 m

The model was used to simulate a variety of charge pressures and system pressures, as depicted in Figure 14, Figure 15, Figure 16 and Figure 17. *TL* is presented in the frequency domain, and a higher magnitude signifies a greater amount of noise has been suppressed at that frequency. Figure 14 and Figure 15 show the predicted *TL* values increase as the (CPR) increases for a WM-5081 suppressor and a WM-5138 suppressor with highest *TL* exhibited at a CPR of 0.9, though the *TL* drastically decreases once the CPR is over 1. This prediction matches observed behavior for these devices. In Figure 14, the largest difference in *TL* of 13 dB between CPRs less than one occurs at approximately 3000 Hz, emphasizing the importance of a properly charged suppressor. The 35 dB *TL* drop-off with an overcharged suppressor further emphasizes the need to avoid using overcharged suppressors in practice. However, this is problematic if system pressure varies widely. Figure 15 shows predicted *TL* curves the same charge pressures of a WM-5138 suppressor. Predicted *TL* drops to almost zero for each charge pressure ratio in the frequency range between 2700 Hz and 3100 Hz, due the length of the device corresponding to the half-wavelength associated with this frequency. This behavior is also seen in the real device. The biggest difference in *TL* between suppressors with a CPR less than 1 is approximately 10 dB at 1500 Hz. The increase in predicted *TL* accentuates the effect of CPR, and the CPR of 0.9 exhibits the highest *TL*. The decrease in *TL* to a suppressor with a CPR higher than 1 is even more drastic, emphasizing the negative effect of charging suppressors to higher than system pressure. Note that the high predicted *TL* presented here may exceed the measurement capabilities discussed in Chapter 2 because the transmitted signal may be close to or below the ambient noise level.

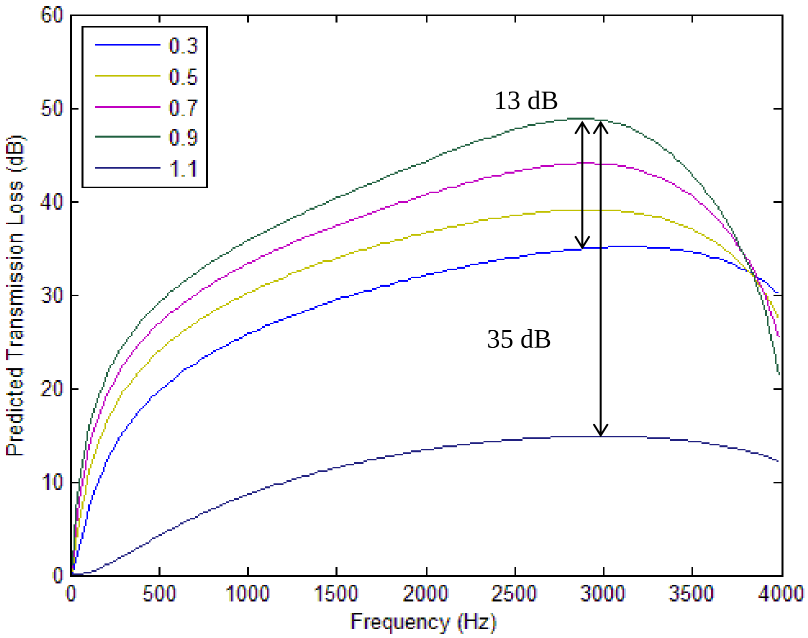

Figure 14: Predicted transmission loss for WM-5081 Suppressor at 10.3 MPA system pressure as a function of CPR

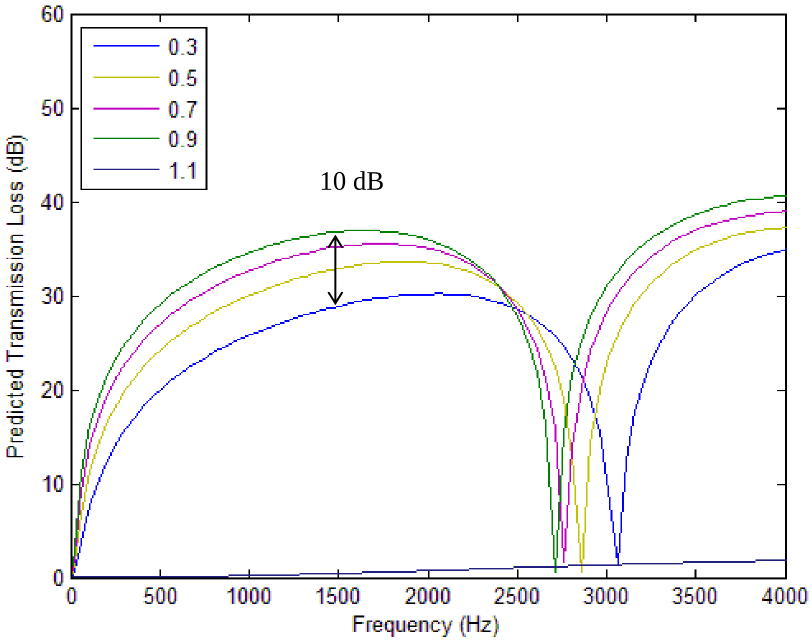

Figure 15: Predicted transmission loss for WM-5138 Suppressor at 10.3 MPA system pressure with varying CPR

Figure 16 and Figure 17 show a comparison of predicted *TL* for a suppressor at a constant CPR of 0.5 but a discrete set of system pressures, for a WM-5081 suppressor and WM-5138 suppressor, respectively. In the modeling of both suppressors, as system pressure increases the *TL* decreases over the full frequency range. In Figure 16, there is a large increase near 3000 Hz in the predicted *TL* performance for the 4.14 MPa system pressure. While the predicted *TL* for the other system pressures also increase near 3000 Hz, none have a significantly large rise. At 3000 Hz, the difference in *TL* between the 4.14 MPa curve and 8.27 MPa curve is nearly 20 dB. The difference between the 8.27 MPa curve and the 20.7 MPa curve at the same frequency is 10 dB. In Figure 17, the largest *TL* difference, ignoring the nulls in *TL*, is almost 25 dB at approximately 2000 Hz. In Figure 17, the predicted 0 dB *TL* point shifts by 700 Hz as the simulated system pressure increases. Both suppressor sizes show differences in *TL* for the same CPR at different system pressures. The differences in the predicted *TL* curves are interesting to note; however, the system will not operate at a fixed charge ratio in practice but instead operate with a fixed charge pressure with varying system pressure, leading to a varying CPR.

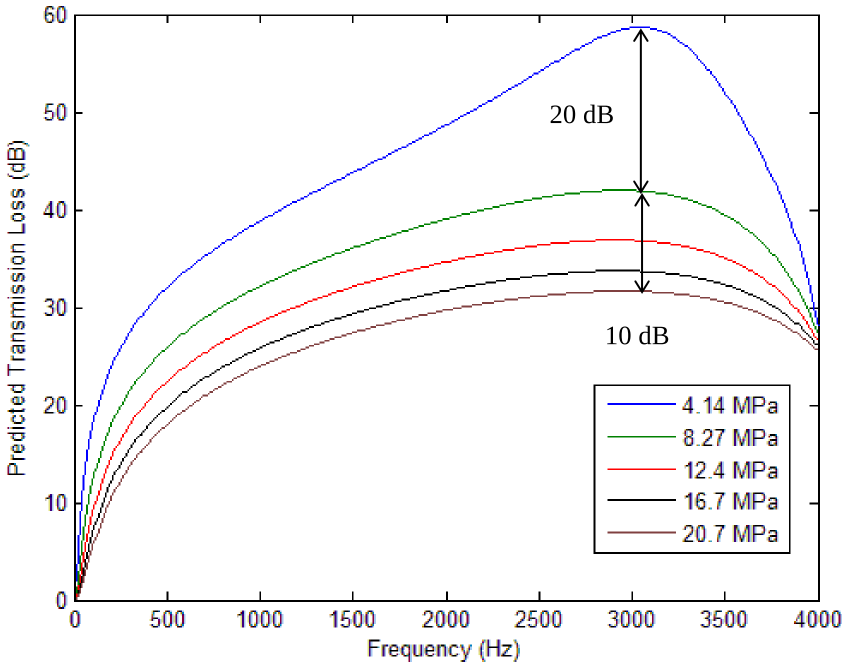

Figure 16: Predicted transmission loss for WM-5081 Suppressor, with 50% CPR of varying system pressures

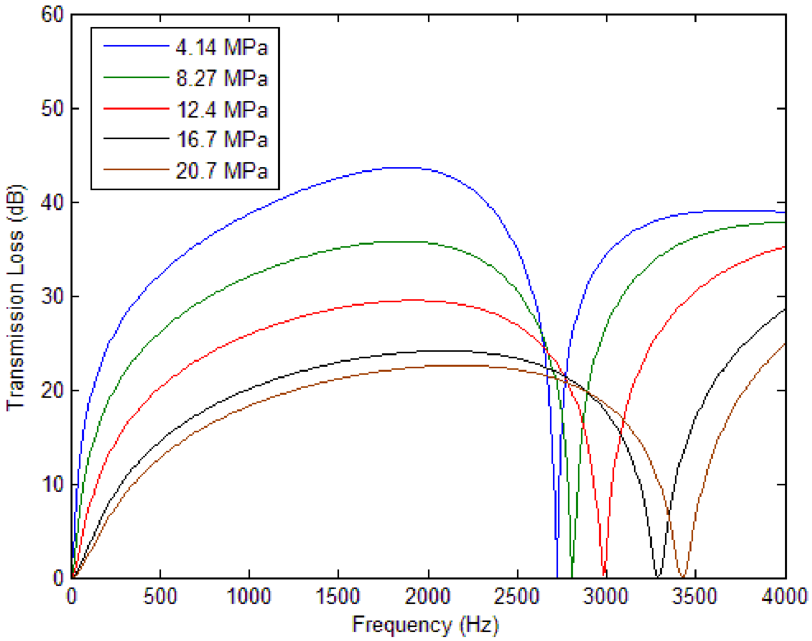

Figure 17: Predicted transmission loss for WM-5138 Suppressor, with 50% CPR of varying system pressures

3.3 Comparison of Measured Transmission Loss & Predicted Transmission Loss

The predicted *TL* performance was compared to measured *TL* performance at the same conditions in order to validate the model. The predicted *TL* for a single WM-5081 suppressor is compared to measured *TL* in Figure 18. The predicted *TL* and measured *TL* show good agreement in the frequencies below 500 Hz. Numerical artifacts in measured *TL* at frequencies of 600 and 900 Hz make comparison difficult. The predicted *TL* diverges from the measured *TL* in the frequency range of 1000 Hz to 4000 Hz. The model only simulates the noise transmission path through the fluid, while the measurement technique is effected by all noise transmission paths, such as through the pipes and suppressor shell. The non-fluid transmission paths may impose a limit on the maximum *TL* in practice, explaining the difference in the predicted and measured *TL* curves. The data presented in Figure 18 suggests the limit may be near 30 dB. To determine what effect, if any, this will have on optimization results, a maximum *TL* constraint of 30 dB will be applied to an optimization case.

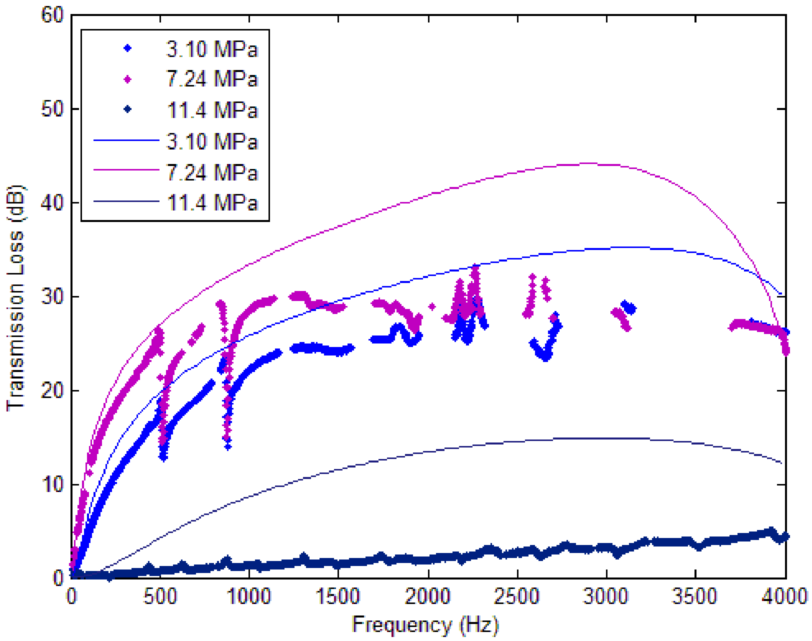

Figure 18: Comparison of transmission loss for a single WM-5081 Suppressor at a system pressure of 10.3 MPa for a variety of CPR

The predicted *TL* and measured *TL* for a single WM-5138 are shown in Figure 19. The comparison is similar to that of the WM-5081, where the predicted *TL* and measured *TL* have good agreement below 500 Hz, while numeric artifacts make it difficult to draw conclusions above this frequency. However, the predicted *TL* for a WM-5138 suppressor does follow a similar path to the mean value of the artifacts.

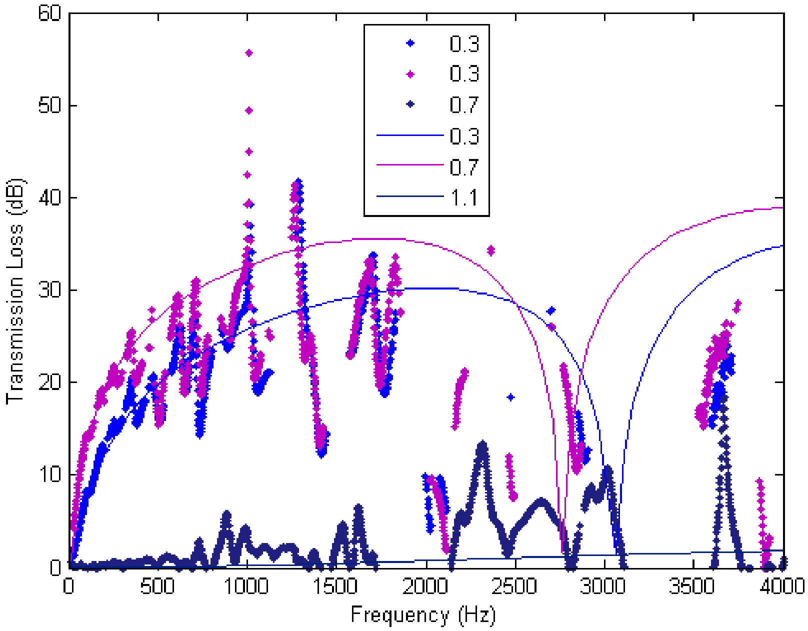

Figure 19: Comparison of transmission loss for a single WM-5138 Suppressor at a system pressure of 10.3 MPa for a variety of CPR

CHAPTER 4

MEASUREMENT AND MODELING OF A TWO-SUPPRESSOR SYSTEM

Many hydraulic systems in the field operate over a broad pressure range; this range is outside the operational range for a single suppressor and requires two suppressors for optimal noise control. Two suppressors allow a broader range of system pressures to be effectively targeted. This section discusses selecting the optimal architecture for a two-suppressor system, the measurement of the *TL* exhibited by a two-suppressor system and a modeling the *TL* exhibited by a two-suppressor as well as a comparison between the measured and modeled *TL*.

4.1 Modeling of Two-Suppressor System Architecture

First, an architecture for the two suppressors must be selected. There are two possible general architectures for combining two suppressors; in series or in parallel, as seen in Figure 20 and Figure 21. The merits of the two architectures can be compared through use of transfer matrices. The effective transfer matrix for each architecture may be used as the basis for comparison. A transfer matrix relates the acoustic pressure and acoustic volume velocity at two points,

$$\begin{bmatrix} P_D \\ Q_D \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} P_U \\ Q_U \end{bmatrix}, \tag{4.1}$$

where *a*, *b*, *c* and *d* represent the frequency-dependent elements of the transfer matrix, *P* and *Q* are the Fourier coefficients of acoustic pressure and velocity, respectively and the subscripts represent the upstream and downstream ports. As seen in Figure 20 and Figure

21 the systems are each comprised of three major elements: pipes, suppressor 1 and suppressor 2. Each of these elements can be represented by individual transfer matrices:

$$\begin{bmatrix} P_D \\ Q_D \end{bmatrix} = \begin{bmatrix} a_p & b_p \\ c_p & d_p \end{bmatrix} \begin{bmatrix} P_U \\ Q_U \end{bmatrix}, \tag{4.2}$$

$$\begin{bmatrix} P_D \\ Q_D \end{bmatrix} = \begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} P_U \\ Q_U \end{bmatrix}, \tag{4.3}$$

$$\begin{bmatrix} P_D \\ Q_D \end{bmatrix} = \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} \begin{bmatrix} P_U \\ Q_U \end{bmatrix}, \tag{4.4}$$

where Equation (4.2) represents the transfer matrix of the pipe, Equation (4.3) represents the transfer matrix of suppressor 1 and Equation (4.4) represents the transfer matrix of suppressor 2.

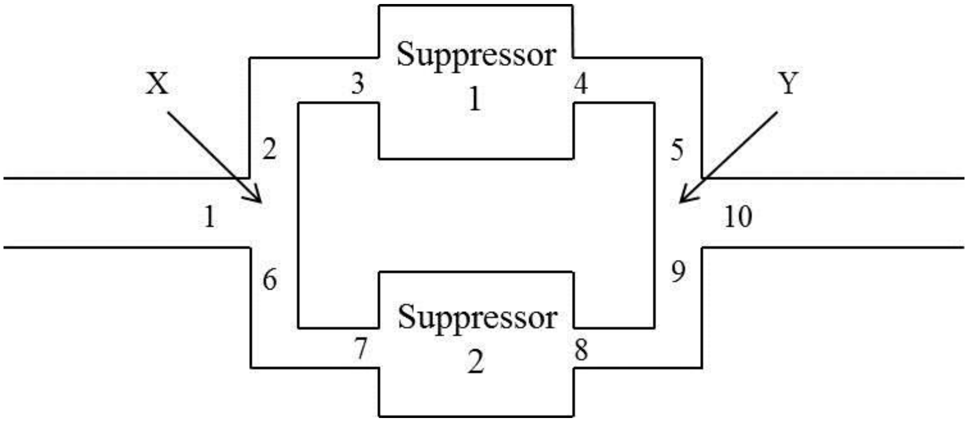

Figure 20: Parallel suppressor architecture

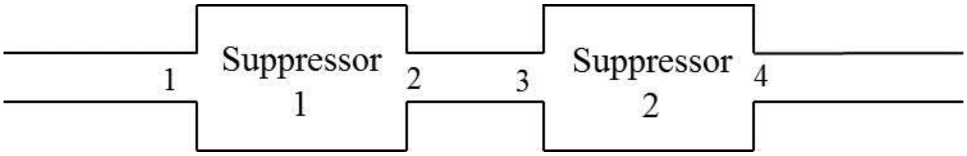

Figure 21: Series suppressor architecture

The effective transfer matrix for the parallel architecture is developed first. The parallel architecture requires 10 separate ports shown in Figure 20 to assemble the effective transfer matrix. In addition, boundary conditions were needed to solve for the effective transfer matrix: at point X continuity of pressure requires P_1=P_2=P_6 and continuity of volume velocity requires Q_1=Q_2+Q_6, and at point Y continuity of pressure requires P_5=P_9=P_{10} and continuity of volume velocity requires Q_{10}=Q_5+Q_9. The effective transfer matrix of the parallel architecture is then:

$$\begin{bmatrix} P_{10} \\ Q_{10} \end{bmatrix} = \begin{bmatrix} A_{parallel} & B_{parallel} \\ C_{parallel} & D_{parallel} \end{bmatrix} \begin{bmatrix} P_1 \\ Q_1 \end{bmatrix}, \tag{4.5}$$

$$\begin{aligned} A_{parallel} = {} & b_2 c_p{}^2 + a_2 a_p c_p + a_p c_2 d_p + c_p d_2 d_p \\ & + \frac{\left(d_2 d_p{}^2 \left(a_1 a_p{}^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{\left(b_p{}^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \\ & + \frac{\left(a_2 b_p c_p \left(a_1 a_p{}^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{\left(b_p{}^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \\ & + \frac{\left(b_2 c_p d_p \left(a_1 a_p{}^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{\left(b_p{}^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \\ & + \frac{\left(b_p c_2 d_p \left(a_1 a_p{}^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1)\right)\right)}{\left(b_p{}^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \end{aligned} \quad , \tag{4.6}$$

$$B_{parallel} = d_2 d_p^2 + a_2 b_p c_p + b_2 c_p d_p + b_p c_2 d_p\,, \tag{4.7}$$

$$\begin{aligned} C_{parallel} = c_p & \left(\begin{aligned} & a_1 \left(a_p - \frac{\left(b_p\left(a_1 a_p^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{\left(b_p^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \right) \\ & + b_1 \left(c_p - \frac{\left(d_p\left(a_1 a_p^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{(b_p^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p)} \right) \end{aligned} \right) \\ + c_p & \left(\begin{aligned} & a_2 \left(a_p + \frac{\left(b_p\left(a_1 a_p^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{(b_p^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p)} \right) \\ & + b_2 \left(c_p + \frac{\left(d_p\left(a_1 a_p^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{\left(b_p^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \right) \end{aligned} \right) \\ + d_p & \left(\begin{aligned} & c_1 \left(a_p - \frac{\left(b_p\left(a_1 a_p^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{\left(b_p^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \right) \\ & + d_1 \left(c_p - \frac{\left(d_p\left(a_1 a_p^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{\left(b_p^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \right) \end{aligned} \right) \\ + d_p & \left(\begin{aligned} & c_2 \left(a_p + \frac{\left(b_p\left(a_1 a_p^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{\left(b_p^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \right) \\ & + d_2 \left(c_p + \frac{\left(d_p\left(a_1 a_p^2 + a_p b_1 c_p + a_p b_p c_1 + b_p c_p d_1\right)\right)}{\left(b_p^2 c_1 + a_1 a_p b_p + a_p b_1 d_p + b_p d_1 d_p\right)} \right) \end{aligned} \right), \end{aligned} \tag{4.8}$$

$$D_{parallel} = c_p(a_2 b_p + b_2 d_p) + d_p\left(b_p c_2 + d_2 d_p\right). \tag{4.9}$$

Next the effective transfer matrix of the series architecture is developed. The series configuration is shown with its nodes in Figure 21. This leads to an effective transfer matrix of:

$$\begin{bmatrix} P_4 \\ Q_4 \end{bmatrix} = \begin{bmatrix} A_{series} & B_{series} \\ C_{series} & D_{series} \end{bmatrix} \begin{bmatrix} P_1 \\ Q_1 \end{bmatrix}, \tag{4.10}$$

$$A_{series} = a_2\left(a_1 a_p + b_p c_1\right) + b_2\left(a_1 c_p + c_1 d_p\right), \tag{4.11}$$

$$B_{series} = a_2\left(a_p b_1 + b_p d_1\right) + b_2\left(b_1 c_p + d_1 d_p\right), \tag{4.12}$$

$$C_{series} = c_2\left(a_1 a_p + b_p c_1\right) + d_2\left(a_1 c_p + c_1 d_p\right), \quad (4.13)$$

$$D_{series} = c_2\left(a_p b_1 + b_p d_1\right) + d_2\left(b_1 c_p + d_1 d_p\right). \quad (4.14)$$

The effective transfer matrices are then compared to determine which configuration is preferred by maximizing reduction of downstream transmitted energy. The lowest transmitted energy is desired; the downstream pressure of the parallel architecture is P_{10} and the downstream pressure of the series architecture is P_4. An anechoic termination is assumed for both systems. A simplifying assumption of lossless pipe was made by setting the transfer matrix of the pipe to the identity matrix. The architectures were compared in a situation where one of the suppressors was assumed to be lossless, (TL=0 for the entire frequency range), a worst case scenario. The downstream pressure in the parallel configuration, P_{10}, is then

$$P_{10} = P_1, \quad (4.15)$$

indicating that the parallel suppressor architecture has become acoustically transparent and transmits all acoustic pressure downstream. The downstream pressure, P_4, in the series configuration for the same condition is

$$P_4 = a_2 P_1 + b_2 Q_1 \quad (4.16)$$

which is identical to a single suppressor architecture. This extreme case demonstrates that the downstream pressure of a parallel configuration is shown to be dependent on the suppressor exhibiting the worse performance, that is the “weak link” dominates the system. In contrast, the downstream acoustic pressure of a series configuration is dependent on an addition of suppressor performance, which does not allow for a “weak link” to dominate the system. Therefore, series configuration of multiple suppressors will be examined in this thesis using the equivalent fluid model presented in Chapter 3 to predict wave amplitudes. The relationship between wave amplitude and transfer matrix elements developed in Chapter 2 can be used to calculate the *TL* for two-suppressors in series.

4.2 Measured Transmission Loss of a Two-Suppressor System

The measurement technique presented in Chapter 2 was also applied for two suppressors in line with a separation distance of 10 cm. The measured *TL* curves are shown below in Figure 22 and Figure 23. The charge pressure pair exhibiting the highest TL in Figure 22 occurs when both suppressors have CPRs that exhibited the highest TL for a single suppressor configuration. In the frequency range from 1000 Hz to approximately 3500 Hz there are very few data points because *TL* is very high and the noise of the system exceeds the signal causing the coherence of these frequencies to decrease. As stated in Section 2.1.3, data points with coherence value less than 0.9 are neglected, and the points in this range do not cross this threshold. A reason the data points have coherence values less than 0.9 is the suppressor configuration is performing beyond the ability of the current set-up to measure by reducing the transmitted signal to levels at or below the level of system noise.

In Figure 22 the charge pressure pair of [5.17, 11.4] MPa, CPR of [0.5, 1.1], shows the effect of a one suppressor having a CPR higher than 1. The *TL* exhibited by the charge pressure pair of [5.17, 11.4] MPa drops by approximately 20 dB from a charge pressure pair of [9.31, 9.31] MPa, CPR of [0.9, 0.9]. The *TL* exhibited by the charge pressure pair of [5.17, 11.4] MPa is on the order of a single suppressor with a CPR less than 1 for the entire frequency range. Similar effects are exhibited by two WM-5138 suppressors, seen in Figure 23. Again, combining charge pressures exhibiting high *TL* for a single suppressor creates the charge pressure pair with the highest *TL*. The difference is best seen in the low frequency range below 500 Hz where there are few numerical artifacts and little low coherence dropout, similar to the results presented for the WM-5081 suppressor model. Also similar to the results of a WM-5081, the charge pressure pair of [5.17, 11.4] MPa for a WM-5138 shows the decrease in *TL* when a suppressor has a CPR above 1.

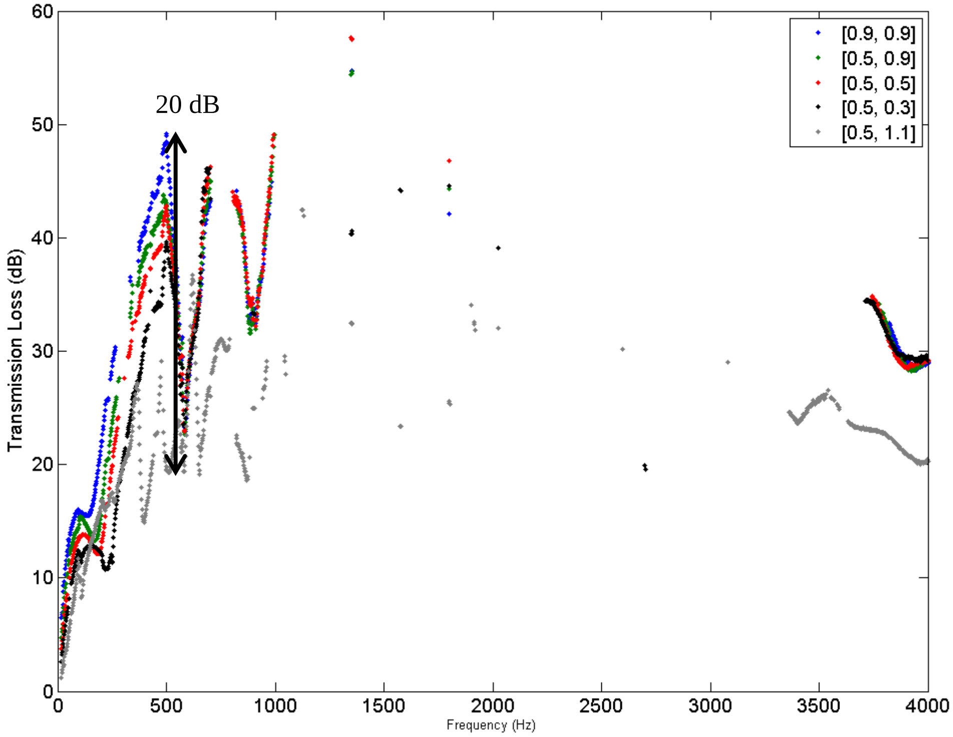

Figure 22: Transmission loss for two WM-5081 Suppressors at 10.3 MPA system pressure with varying charge pressures

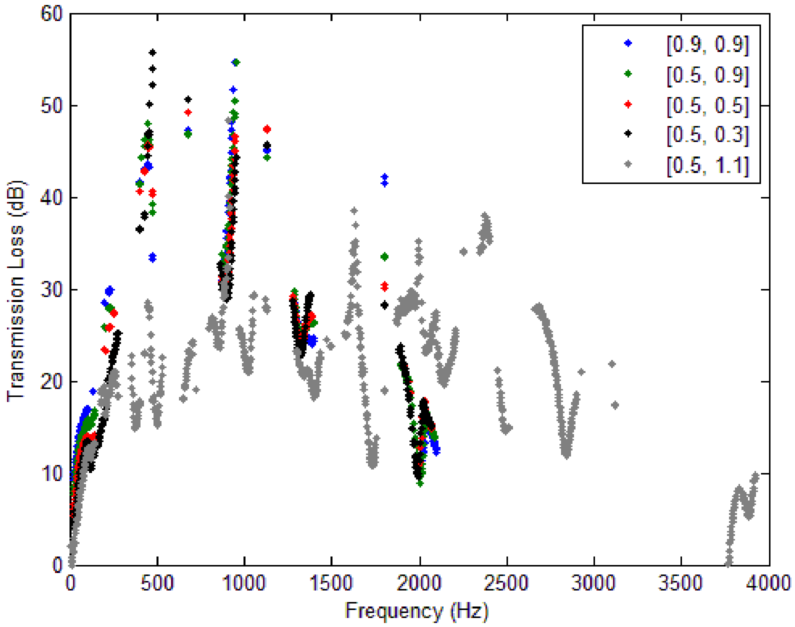

Figure 23: Transmission loss for two WM-5138 Suppressors at 10.3 MPA system pressure with varying charge pressures

To demonstrate changing CPR order in a suppressor pair does not affect *TL* two sets of data at the same system pressure were measured with charge pressure order reversed, and the results are seen in Figure 24 and Figure 25. The data for both suppressors shows negligible difference when the CPRs are reversed, confirming the predictions of the model. For the remainder of the thesis, any result presented for a given charge pressure pair [X, Y] MPa will be assumed valid for the charge pressure pair of [Y, X] MPa.

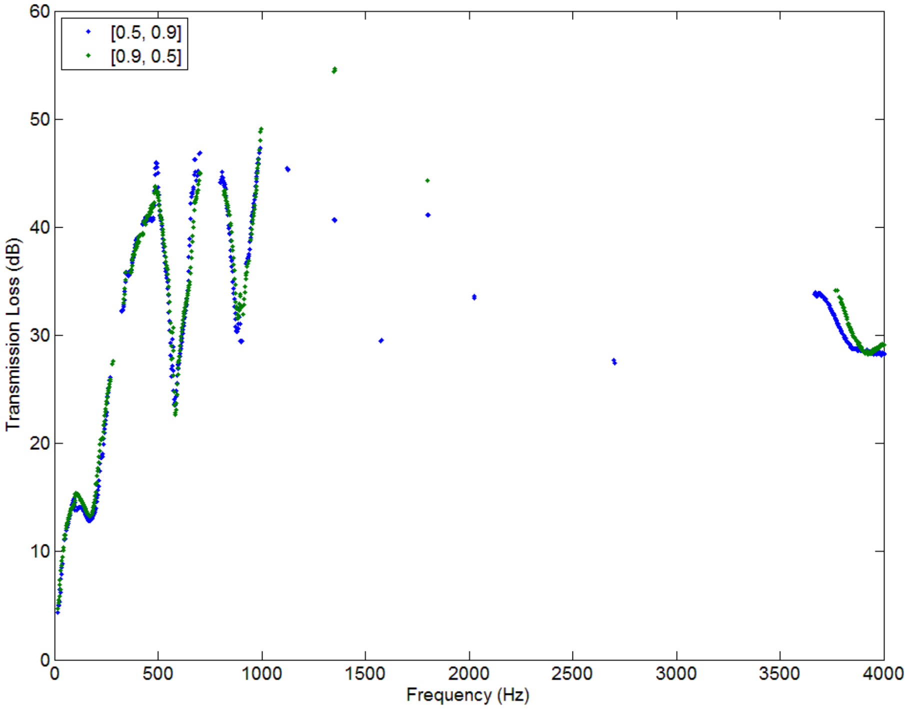

Figure 24: Transmission loss for two WM-5081 Suppressors at 10.3 MPA system pressure changing CPR order

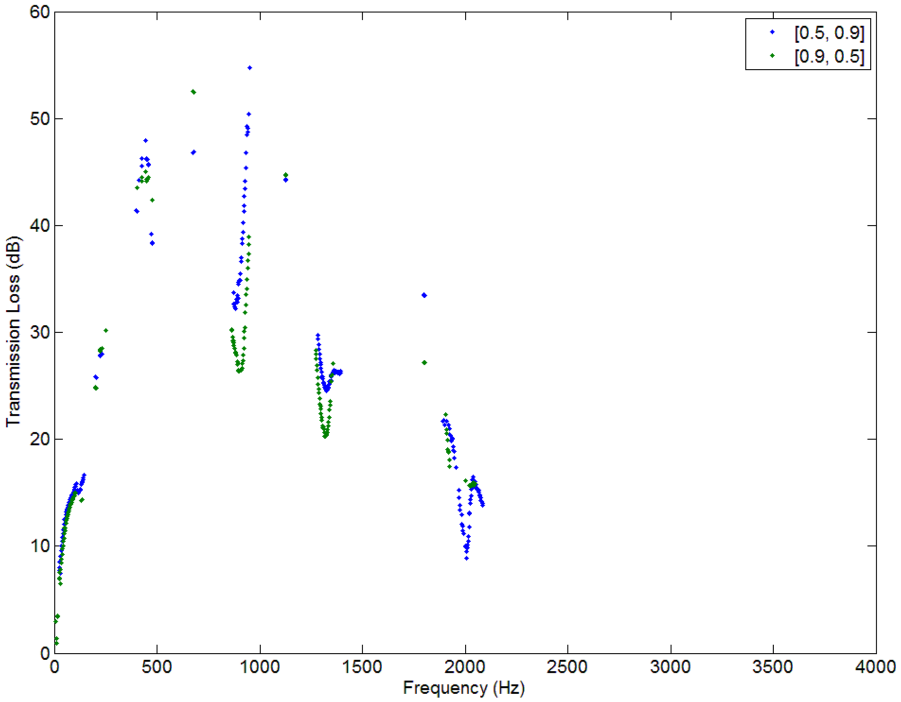

Figure 25: Transmission loss for two WM-5138 Suppressors at 10.3 MPA system pressure changing CPR order

4.3 Modeled Transmission Loss of a Two-Suppressor System

The model developed in Chapter 3 can be modified to predict transmission loss for a two suppressor setup. First, the wavefield of the downstream suppressor is simulated with an assumption of an anechoic termination. The upstream suppressor is then simulated, but the assumption of an anechoic termination is no longer valid, however; the upstream waves of the downstream suppressor, waves C and D in Figure 26, can be used to calculate the impedance at the downstream port of the upstream suppressor ensuring the upstream suppressor has an output matching the input to the downstream suppressor. Modeling the upstream suppressor with this condition and renormalizing all waves to *A* in Figure 26 allows the *TL* of a two suppressor set up to be calculated, again with

$$TL = 20\log_{10}\left|\frac{A^2 - F^2}{AE - BF}\right|. \tag{4.17}$$

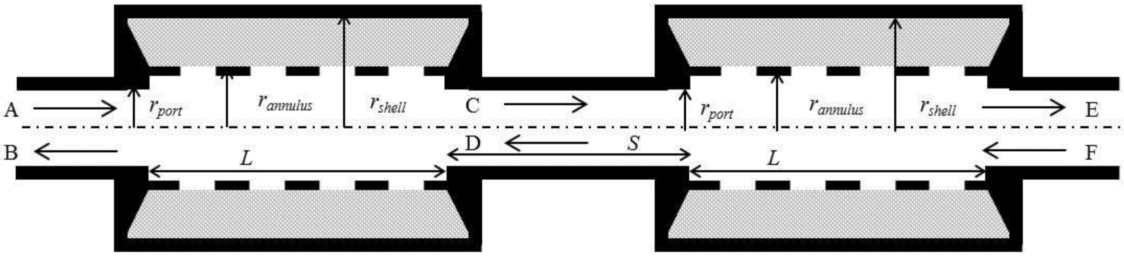

Figure 26: Two suppressor configuration and acoustic waves

The predicted *TL* for a number of two suppressor simulations can be seen in Figure 27 and Figure 28, for two WM-5081 suppressors and two WM-5138 suppressors, respectively. Both Figures show predicted *TL* increases as the CPR of both suppressors approach 1 from below. When one suppressor of the charge pressure pair has a CPR above 1 the *TL* decreases to a similar magnitude of one suppressor. For a WM-5081 suppressor, *TL* shown in Figure 27, the decrease from a charge pressure pair of [5.17, 9.31] MPa, CPRs of [0.5, 0.9], to a charge pressure pair of [5.17, 11.4] MPa, CPRs of [0.5, 1.1], is approximately 60 dB at a frequency of 3000 Hz. For each charge pressure case with both CPRs less than 1 the *TL* approaches 0 dB at approximately 250 Hz. Similar effects are exhibited by WM-5138 Suppressors, shown in Figure 28. The decrease from a charge pressure pair of [5.17, 9.31] MPa, CPRs of [0.5, 0.9], to a charge pressure pair of [5.17, 11.4] MPa, CPRs of [0.5, 1.1], is approximately 40 dB at 2000 Hz, showing the importance of suppressors with a CRP less than 1 with respect to performance. As with two WM-5081 suppressors, two WM-5138 suppressors exhibit a *TL* drop-out in frequencies near 250 Hz. Two WM-5138 suppressors exhibit another *TL* drop-out near 2750 Hz, this is an effect of suppressor geometry and is seen in the predicted *TL* for a single WM-5138 in Figure 15. Note that the predicted *TL*s probably exceed the measurement capabilities of the current experimental test set-up.

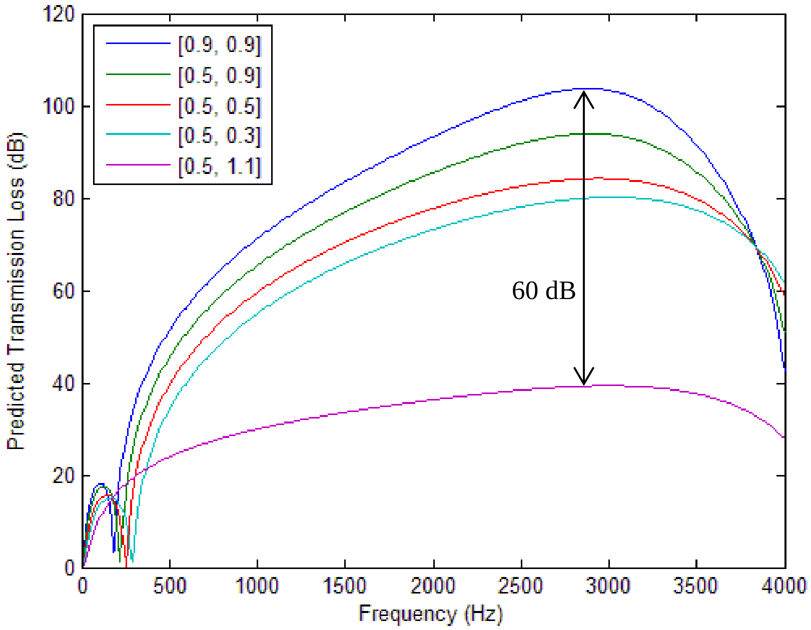

Figure 27: Predicted transmission loss for two WM-5081 Suppressors at 10.3 MPA system pressure with varying CPR

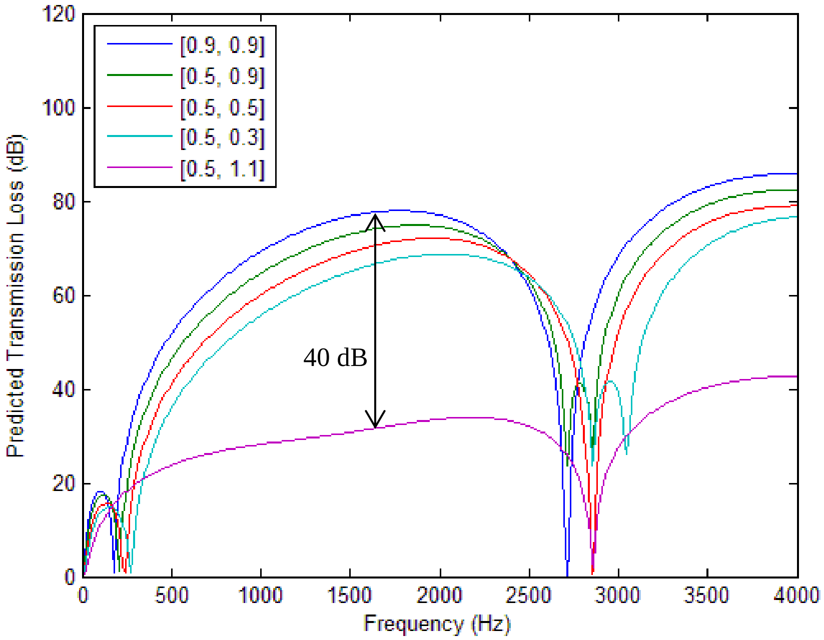

Figure 28: Predicted transmission loss for two WM-5138 Suppressors at 10.3 MPA system pressure with varying CPR

Since the model is inherently for linear systems, *TL* for charge pressures [X,Y] is the same as for charge pressures [Y,X]. Physically, it means that for a suppressor system, which suppressor is charged to pressure X and which pressure Y does not matter.

4.4 Comparison between Measured and Modeled Transmission Loss

The predicted *TL* was compared to measured *TL* for two suppressor configurations. The predicted *TL* and measured *TL* for two WM-5081 suppressors are show in Figure 29. Figure 30 shows frequencies from 0 to 500 Hz, and the measured *TL* shows extremely good agreement with the model in this frequency range. Above 500 Hz, numeric artifacts and the data dropout due to the exhibited *TL* being higher than the limit of the test rig make comparison difficult. An important feature to note is the low frequency drop out near 250 Hz; for the predicted *TL*, the *TL* decreases close to zero, while the measured *TL* shows a decrease at a similar frequency but the *TL* does not decrease all the way to zero.

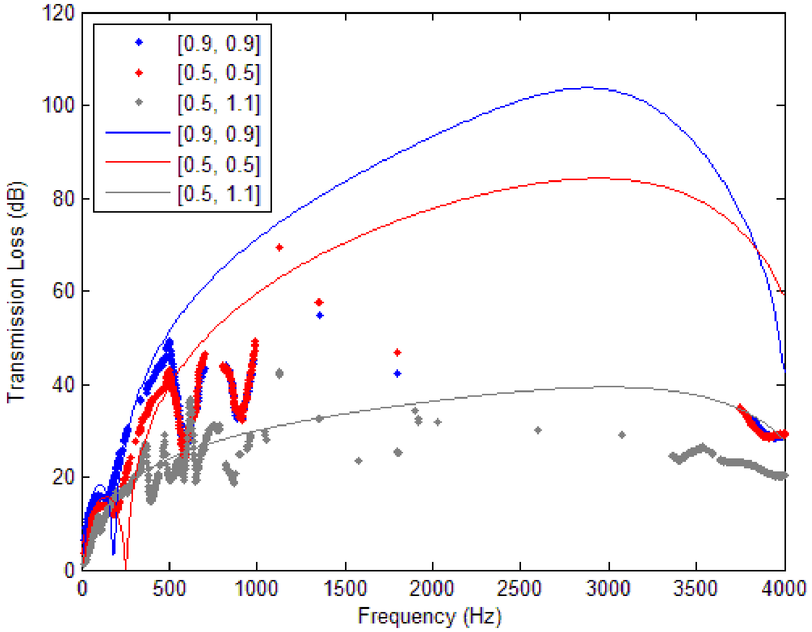

Figure 29: Comparison of transmission loss for two WM-5081 Suppressors at a system pressure of 10.3 MPa for a variety of charge pressure pairs. Frequency range: 0 to 4000 Hz

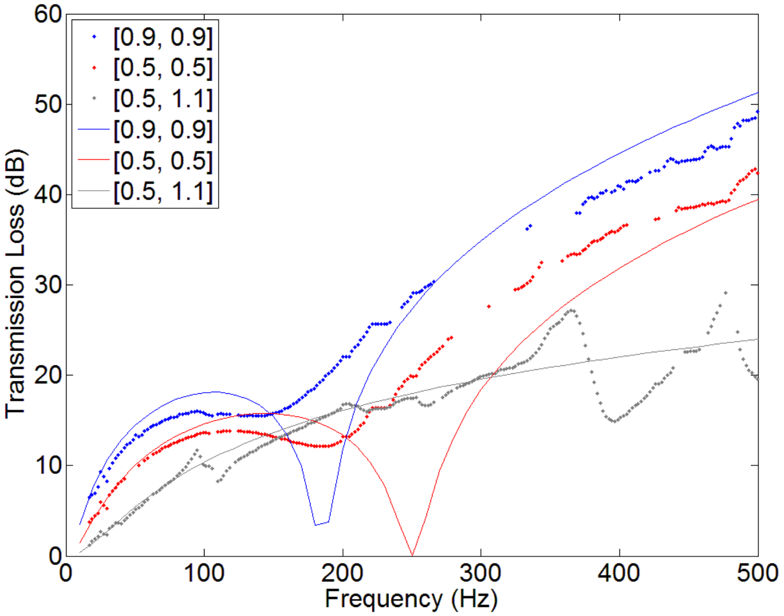

Figure 30: Comparison of transmission loss for two WM-5081 Suppressor at a system pressure of 10.3 MPa for a variety of charge pressure pairs. Frequency range: 0 to 500 Hz

The predicted *TL* and measured *TL* for two WM-5138 suppressors are shown in Figure 31. The measured *TL* exhibited low coherence across the frequency range of interest likely because the TL was reducing the signal below the threshold of system noise. For both suppressor models the CPR pair of [0.5, 1.1] loses fewer data points to deletion by the coherence thresholds, as lower *TL* is expected so the signal to noise ratio is higher in the downstream section which keeps coherence high. In addition, the low frequency range still shows good agreement for charge pressure pairs with CPRs less than 1. For the charge pressure pair of [5.17, 11.4] MPa, CPR of [0.5, 1.1], there is very good agreement between the model in the frequency range of 2700 HZ to 3000 Hz where *TL* approaches 0 dB. The results presented shows the model can be used to predict *TL*.

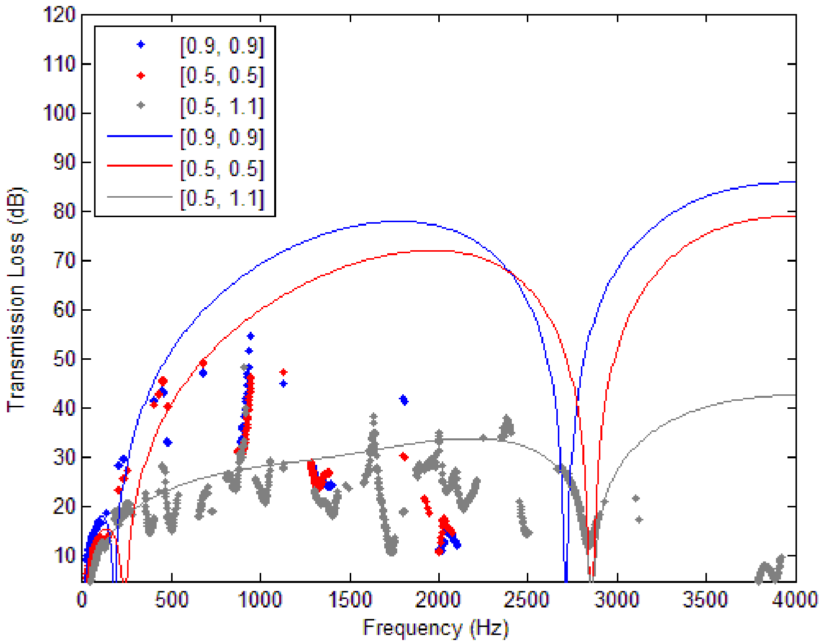

Figure 31: Comparison of transmission loss for two WM-5138 Suppressor at a system pressure of 10.3 MPa for a variety of charge pressure pairs

The WM-5081 suppressor model exhibits similar *TL* behavior as the WM-5138 suppressor model with respect to CPR, for both single and double suppressor configurations. The rest of the thesis will only consider WM-5138 suppressor configurations.

CHAPTER 5

OPTIMIZATION OF SUPPRESSOR CHARGE PRESSURE

An optimization routine is developed which uses a direct-search method to determine the optimal charge pressure configuration for bladder-style suppressors in a hydraulic system through maximization of an objective function. The objective function for the optimization weights predicted *TL* for a either a single suppressor or pair of suppressors in series. The predicted *TL* for single and double suppressor configurations is obtained by using an equivalent fluid model developed by Marek [4] as discussed previously. An optimal condition must target the spectral content of the pressure ripple and the most used system pressures. In order to accomplish this two weighting factors are used: a frequency weighting factor (FWF) and a time weighting factor (TWF). The FWF weights the objective function towards the frequencies of the pressure ripple with the highest acoustic energy. It is very difficult to predict the exact frequency content of the pressure ripple as it fluctuates with each system component and pressure, as such the FWF should be based on measurements on an identical physical system corresponding to the model of the one being optimized. The TWF weights the system to the most used operating pressures. Overall system behavior, including suppressor performance, is dependent on system pressure; for accurate optimizations the TWF needs to represent the intended usage of the hydraulic system. The development of the objective function is described below, as well as the individual effects of the FWF and TWF.

5.1 Objective Function

The objective function considered here is applicable to any device exhibiting *TL*, as long as correct optimization variables are chosen. For example, a Helmholtz Resonator can be optimized if an applicable model is used with the objective function and correct optimization variables, such as neck radius, neck length and cavity volume. Bladder-style suppressors are the focus of this thesis, and finding the optimal charge pressure condition will be the focus of the optimization. The optimal charge pressure condition is found by maximizing the objective function

$$\mathfrak{F}\left(P_{c,1},P_{c,2}\right)=\frac{1}{|U|}\sum_{(i,j,k)\in U}^{N}|D_i|\,\frac{1}{|\Omega|}\sum_{f\in\Omega}^{M}\left|W_i\left(f\right)\right|\bullet\left|TL\left(f,P_{s,i},P_{c,j},P_{c,k}\right)\right|, \tag{5.1}$$

where the optimal charge pressure is defined by

$$\left(P_{c,1},P_{c,2}\right)^{*}=\underset{P_{c,1},P_{c,2}}{\arg\max}\,\mathfrak{F}\left(P_{c,1},P_{c,2}\right). \tag{5.2}$$

In Equation (5.1), *TL* is the predicted transmission loss for the suppressor from the model described previously, *f* is the frequency in Hertz, **Ω** is the frequency bandwidth of interest, $p_{s,i}$ is the system or load pressure and $p_{c,j}$ and $p_{c,k}$ are the charge pressures for a two suppressor configuration. If the optimization is being used for a single suppressor, then only $p_{c,j}$ is used in Equation (5.1). The system pressure and both charge pressures belong to the set *U*; the pressure range of interest. The pressure range is dependent on the anticipated system pressures, as the charge pressure will range from the lowest usable charge pressure to the highest system pressure used as suppressors with a CPR higher than 1 exhibit relatively poor *TL*, and any suppressor charged higher than the highest system pressure will always operate in an overcharged condition. Weighting factors *W* and *D*, described in further detail in Sections 5.1.1 and 5.1.2, respectively, capture the spectral content of the pressure ripple and time-dependent aspect of the system pressure. Both weighting factors are normalized: the FWF, represented by *W,* has a maximum value of 1 while each TWF, represented by *D*, has a total value of 1.

The spectral content of a given pressure ripple will depend upon how the flow is generated and how the flow ripple couples with elements in the system to produce a pressure ripple. For example, the pressure ripple due to positive displacement pumps will be comprised of frequency components dominated by the pumping element’s fundamental frequency and its harmonics; the magnitude of the pressure ripple and its spectral content may depend on the load pressure. It is desirable to ensure exhibited *TL* targets the dominant spectral components to reduce maximum possible energy. This is accomplished through the use of a frequency weighting factor (FWF), *W,* in the objective function, defined as

$$W_i(f) = \frac{\left|P_{d,i}(f)\right|}{\max\limits_{\omega,i}\left|P_{d,i}(f)\right|}, \tag{5.3}$$

where $|P_{d,i}(f)|$ is the magnitude of the dynamic pressure ripple at the i^{th} system pressure as each system pressure may have a unique pressure ripple. *W* is normalized to the highest pressure ripple frequency component of all system pressures being considered. $|P_{d,i}(f)|$ can be measured in-situ on the fluid borne noise of a physical system or modeled; for this work the FWF was measured to ensure correspondence to the system being optimized.

A fluid power system may spend different amounts of time at different load pressures depending on its usage cycle. The *TL* of a suppressor is dependent on the CPR, so a suppressor charged to a given pressure will exhibit different *TL* for changes in system pressure. To account for the time dependency of the system and weight the objective function to the most used system pressures, a time weighting factor *D* is incorporated into the objective function. *D* is defined by

$$D_i = \frac{t_i}{t_{total}}, \tag{5.4}$$

where t_i is the amount of time the system spends at the i^{th} system pressure, and t_{total} is the total time in a complete duty cycle, therefore summing all D_i in a duty cycle is equal to one, $\Sigma Di=1$.

TL exhibited by a suppressor is dependent on several factors: the system pressure, p_s, suppressor geometry, and the charge pressures $p_{c,j}$ and $p_{c,k}$. The system pressure is dictated by the system's usage, and cannot be changed by the objective function. The suppressor geometry is also fixed for a given optimization and cannot be adjusted during operation. The only independent variables are the charge pressures, and the objective function searches through all charge pressure in the set *U* to determine which gives the maximum value of the objective function.

The remainder of the chapter examines the individual components of the objective function, *TL*, FWF and TWF, and their impact on the optimal results. Chapter 2 and Chapter 3 show the results of measured and predicted *TL*. A FWF for four system pressures measured on a test rig constructed at Eaton is shown in Figure 32. The Eaton test rig is identical to the Georgia Tech test rig, except for the valve creating noise and the Eaton valve better represents the noise source seen in the field. Four TWFs are also presented: two are representative of anticipated field usages, a third represents a system operating mostly at a single system pressure and a fourth represents a mixed duty cycle of the two anticipated usages.

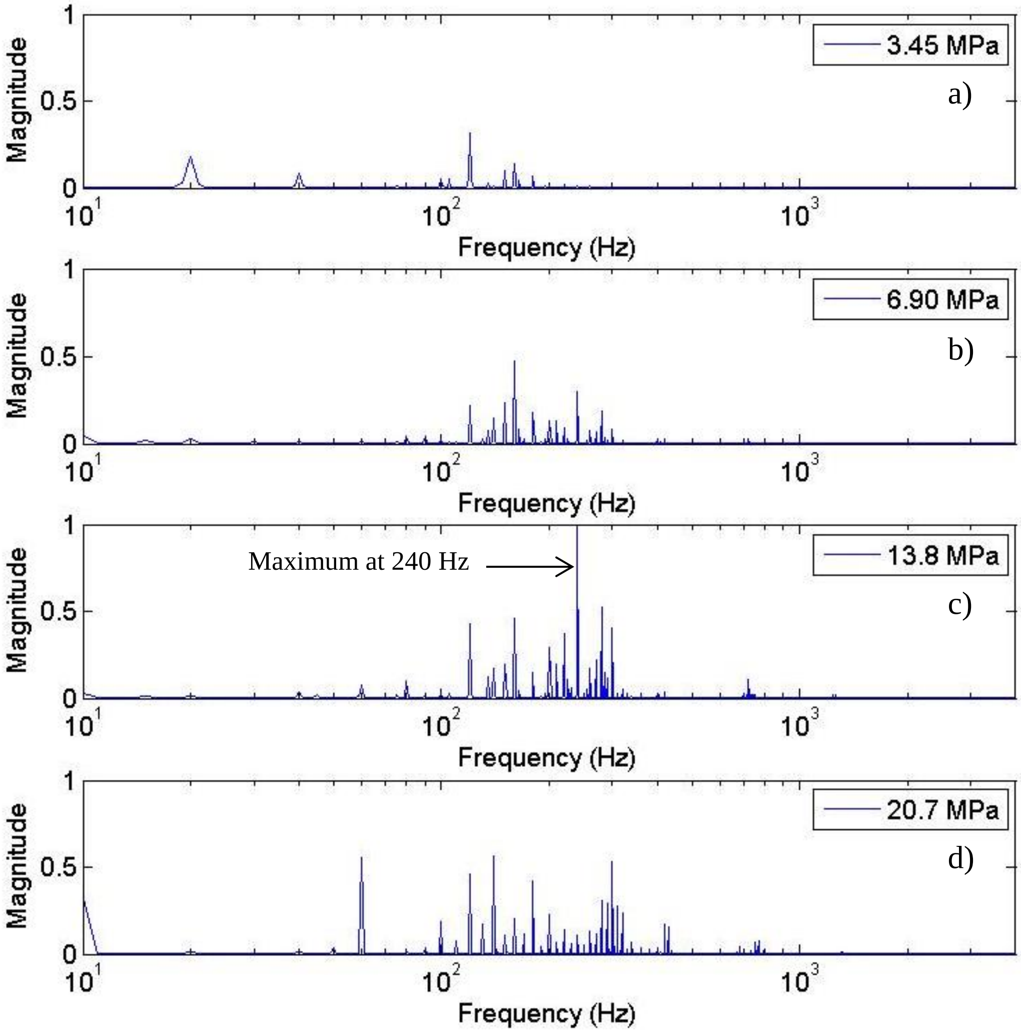

Figure 32: Frequency weighting factor (FWF) for system pressures of: a) 3.45 MPa, b) 6.90 MPa c) 13.8 MPa d) 20.7 MPa. 0-4000 Hz frequency range

5.1.1 Frequency Weighting Factor

The objective function is frequency-weighted using the FWF, Equation (5.3), to account for variation in energy density over the frequency band of interest. The spectral content of the pressure ripple in a given hydraulic system is due to a variety of factors, including the choice of pump, valves, flow path diameter changes, and line lengths in the system. To weight different pressure ripples consistently, the ripples are measured at same position, and the frequency content of the pressure ripple assumed incident on the suppressor is normalized to the maximum pressure ripple amplitude at all load pressures under consideration. This yields a maximum FWF of 1 at the frequency of maximum pressure ripple among all ripples at each load pressure. The FWF at all other frequencies and load pressures will have a value between 0 and 1, depending upon the spectral content of the pressure ripple. Using the FWF ensures frequencies with little acoustic energy are ignored while frequencies with high energy will contribute significantly to the objective function value. In order to get an accurate FWF for a given system, a measurement of pressure ripple should be taken in-situ during anticipated usage at a series of system pressures to capture a representative sampling of the system components.

An example set of FWFs for four load pressures, shown in Figure 32, was generated from data measured on a test rig at Eaton Hydraulics. Eaton employed a different noise source upstream of the test section than Georgia Tech, which is more representative of noise sources seen in the field, thus these FWFs will be used with the optimization. With the exception of the 240 Hz component at the 13.8 MPa load pressure, Figure 32c, generally, higher system pressures have a higher magnitude of FWF reflecting increased magnitude of pressure ripple with increasing load pressure. The mean value of the FWF taken over all frequencies for each pressure can be seen in Table 4, and the higher system pressures have a higher FWF, indicating more energy in those system pressures. For the lowest three system pressures the mean FWF value increases proportionally to the system pressure, plateauing between 13.8 MPa and 20.7 MPa.

Table 4: Mean frequency weighting values of data taken on Eaton test rig

System Pressure (MPa)	Mean FWF value
3.45	0.000367
6.90	0.000745
13.8	0.00144
20.7	0.00146

5.1.2 Time Weighting Factor

The objective function is weighted to the most-used load pressures using the TWF, Eq. (5.4). In practice, hydraulic system duty cycles typically encompass a broad range of load pressures, often idling near 0.690 MPa and reaching pressures up to 35 MPa, with unequal time spent at each load pressure. As noted earlier, the load pressure affects both *TL*, by changing CPR, and the FWF, as seen in Section 5.1.1, and the amount of time the system spends at each pressure needs to be accounted for in the objective function. The TWF is a time fraction of each load pressure relative to some user-defined complete work cycle. Longer usage at a given load pressure will bias the objective function towards those load pressures.

Example TWFs are shown in Figure 33 to Figure 38; two TWFs are representative of system in the field and one is an arbitrary usage of a system predominantly operating at a single pressure. The TWF in Figure 33 represents pressure in the hydraulic circuit for an excavator’s boom and Figure 34 represents boom pressure of an excavator performing back-filling. Both of these TWFs are based on field-measured data provided by Eaton Hydraulics. The example TWFs represent more system pressures than are represented in the FWF, presented in Section 5.1.1. In order to represent the same number of system pressures in the TWF as individual system pressures represented in the FWF, the system pressures are combined together using the bins shown in Figure 33 and Figure 34. The pressure ranges are based on proximity to a pressure represented in the FWF. The lower system pressures are grouped into bin 1, and these pressures are assumed to have no significant pressure ripple and are neglected. The TWF is renormalized using only non-zero system pressure ranges, as seen in Figure 35 and Figure 36. An arbitrary TWF, TWF Case 3, for a system working heavily in the 13.8 MPa range is shown in Figure 37. TWF Case 4, shown in Figure 38, represents a system working equally between TWF Case 1 and TWF Case 2.

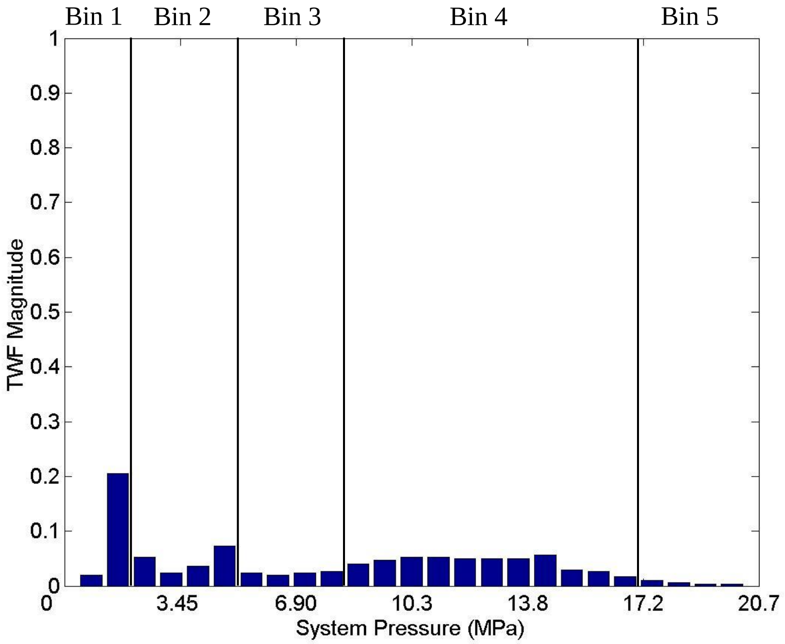

Figure 33: TWF Case 1: boom pressure, trenching run

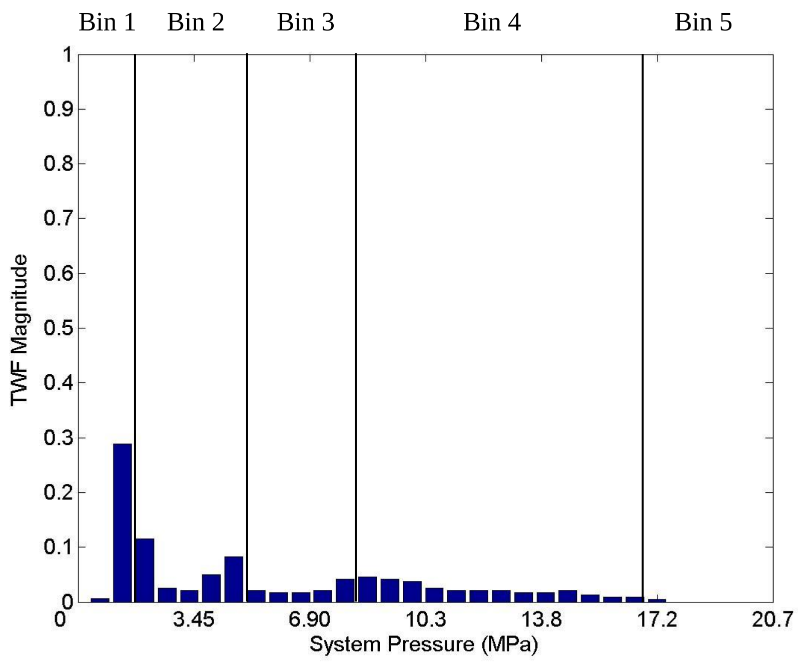

Figure 34: TWF Case 2: boom pressure, back filling

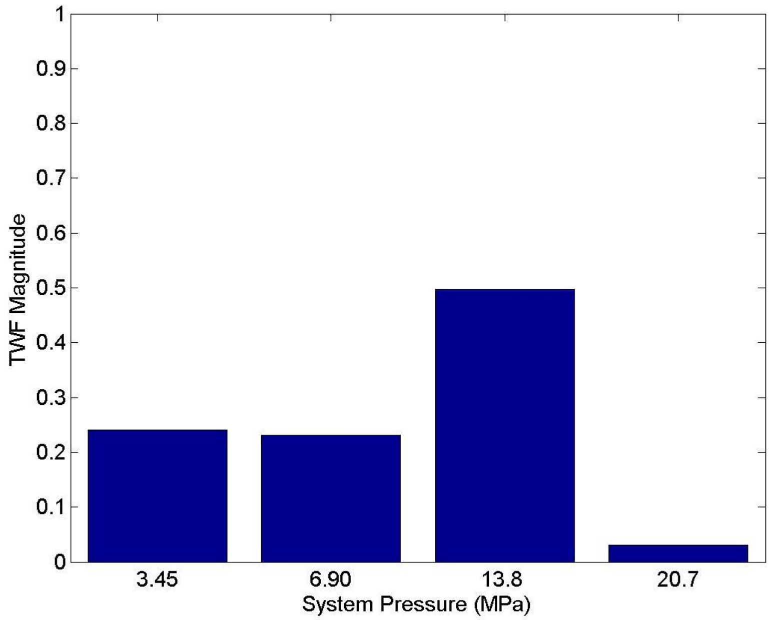

Figure 35: TWF Case 1: boom pressure, trenching run. Aggregated pressures derived from bins shown in Figure 33

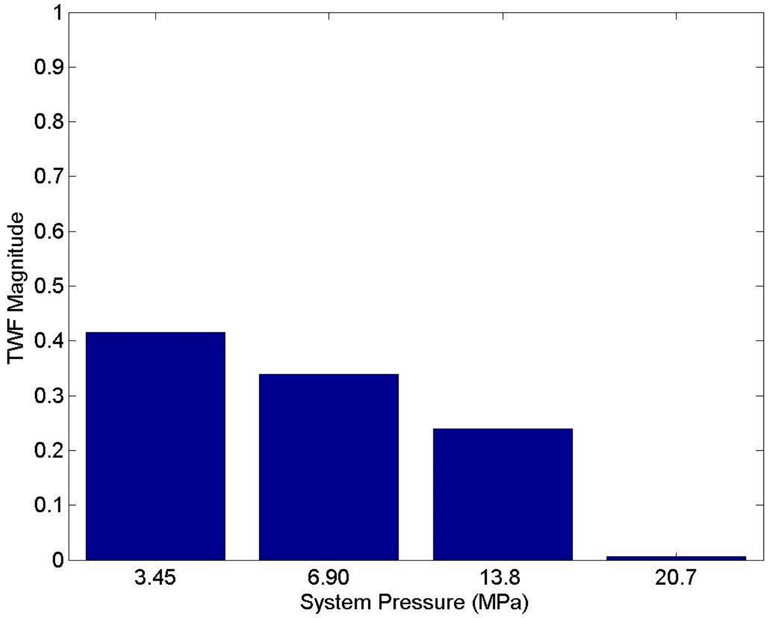

Figure 36: TWF Case 2: boom pressure, back filling. Aggregated pressures derived from bins shown in Figure 34

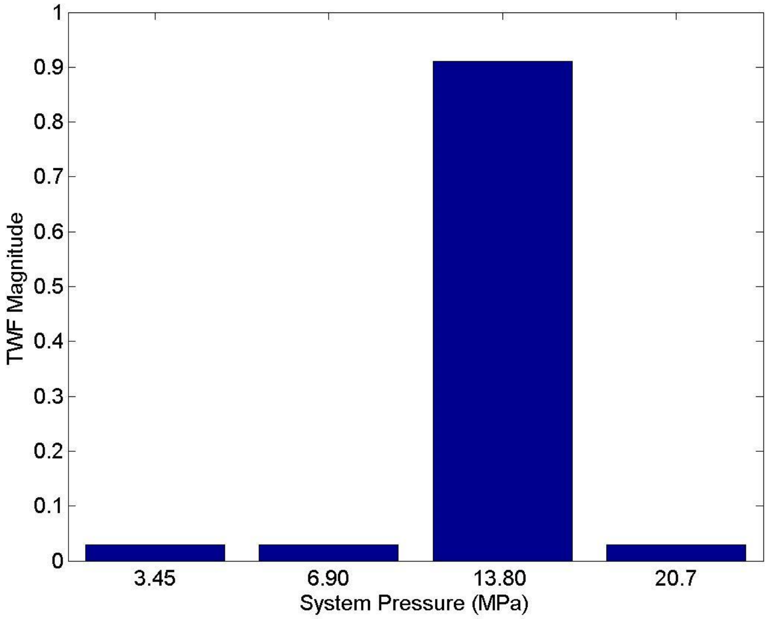

Figure 37: TWF Case 3: arbitrary usage of a system operating heavily at 13.8 MPa

A hydraulic system may be used with different duty cycles during a work period. The work period is defined by the length of time between recharging the suppressors to an optimal condition. For the above TWFs the work period is one duty cycle. In practice it may not be feasible to recharge the suppressors after each duty cycle and a work period may last several duty cycles altering the optimal condition. To account for this, the TWF can be adjusted to represent more than one duty cycle, but instead represent the total anticipated usage for the work period. An arbitrary TWF for a work period of equal time spent trenching and back filling is presented in Figure 38.

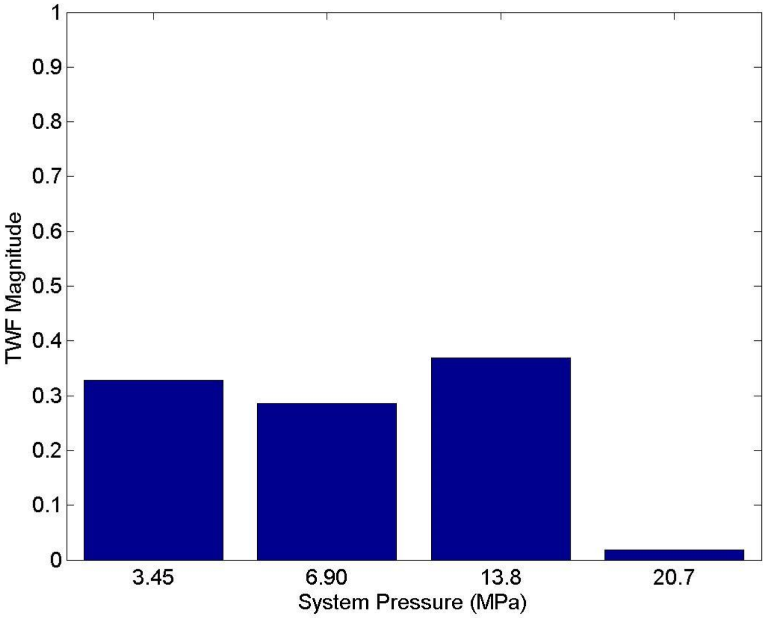

Figure 38: TWF Case 4: arbitrary work period usage derived from a system operating in a duty cycle of 50% back filling and 50% trenching

5.2 Example Optimizations

The objective function was calculated using the FWF and TWFs from Section 5.1.1 and 5.1.2, respectively. The FWF, found in Section 5.1.1, was selected because it is representative of a noise source seen in practice. TWF Cases 1 and 2, seen in Figure 35 and Figure 36, were selected because they are representative of anticipated system pressures based on measurements taken from an existing system. The results from these two TWF cases will be representative of anticipated optimal charge pressure configurations. TWF Case 3, seen in Figure 37, was selected because it represents a system operating at a single system pressure, producing a different optimal configuration and allowing for effects of the FWF to be assessed. TWF Case 4, seen in Figure 38, is a combination of TWF Cases 1 and 2, and it was selected to represent a mixed system usage, as changing the charge pressure between every system task may be infeasible. In Section 2.2, a potential maximum achievable *TL* was observed, as suppressors only control FBN, and the suppressors do not treat the other noise transmission paths. TWF Case 1 is analyzed with a hypothetical *TL* ceiling of 30 dB, as discussed in Section 3.3.

The optimizations presented below are normalized to their own maximum value. The optimal point is defined with a normalized objective function value of 1; however other charge pressure configurations may have a normalized objective function value of over 0.95, judged to be insignificant difference and valid choice for use in practice. All local optima with a normalized objective function value above 0.95 are said to be in the selected set, *O*.

For a system operating over a broad pressure range, there is a strong possibility that the selected set includes more than one charge pressure configuration. In order to select a charge pressure configuration from the selected set, factors not directly captured in the objective function are analyzed, which include gradient of the objective function and ABN. During use, the mass of nitrogen contained in the bladder will decrease because of imperfect sealing of the bladder and diffusion through the bladder. The rate at which nitrogen escapes is not known, but assumed to be similar for all suppressors regardless of CPR. The loss of nitrogen lowers the charge pressure of the bladder, decreasing a suppressor's effectiveness. The decrease in nitrogen, and the associated decrease in suppressor performance, allows the charge pressure configurations in the selected set to be compared. The gradient of the objective function for all charge pressures in the selected set is calculated

$$\left|\nabla\mathscr{F}\right|\Big|_{\left(P_{c,1},P_{c,2}\right)\in O}, \tag{5.5}$$

where $\mathscr{F}$ is the objective function, $p_{c,1}$ and $p_{c,2}$ are the charge pressure configurations in the selected set, O, and the lowest gradient magnitude is most desirable. The charge pressure configuration with the smallest gradient magnitude will be selected as its performance suffers the least with decreasing nitrogen pressure. If the first factor is not large enough to differentiate between the configurations in O, a second factor of which charge pressure configuration produces less ABN in practice will be considered. The breakout noise is different for each system and should be measured in-situ for the configurations in the selected set, O.

For a system with a single suppressor operating at a given system pressure, there is a unique charge pressure which produces the highest *TL* over the entire frequency range of interest. This charge pressure is the optimal charge pressure for the system usage, and will be labeled as the single pressure optimum. A two suppressor system operating at the same system pressure as a single suppressor has the optimal condition of both suppressors charged to the single pressure optimum. For a system with a single suppressor operating over a range of system pressures, the selected set will be shown to be comprised of the single pressure optima of the system pressures represented in the

TWF. For a system with two suppressors operating over a range of system pressure, the selected set will be shown to be comprised of charge pressure pairs made up of single pressure optima of system represented in the TWF; however, the optimal pairing may be comprised of dissimilar single pressure optima.

All optimizations are found using the procedure shown in Figure 39. First the TL is calculated for all charge pressure and system pressure cases by TL_calc.m, found in Appendix A. Next, the TL is weighted by the appropriate FWF and TWF by WeightingOpt.m, found in Appendix B. A direct-search method finds the normalized objective function values above 0.95 and places these values into the selected set. The values in the optimal are compared with the factors not directly captured by the objective function are used to select the optimal charge pressure configuration.

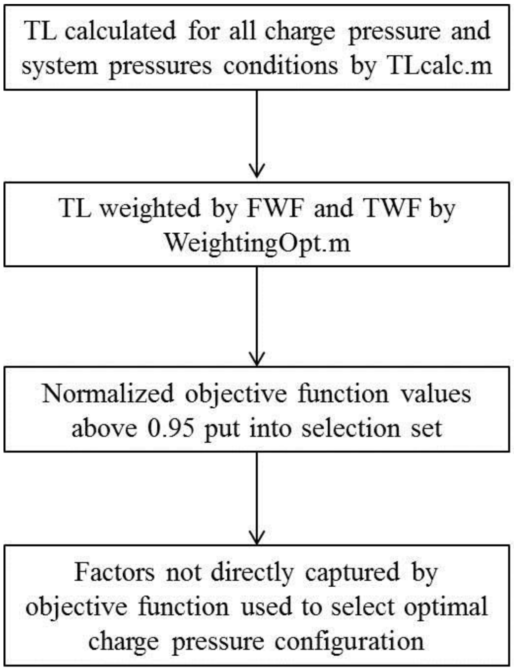
TL calculated for all charge pressure and system pressures conditions by TLcalc.m
TL weighted by FWF and TWF by WeightingOpt.m
Normalized objective function values above 0.95 put into selection set
Factors not directly captured by objective function used to select optimal charge pressure configuration

Figure 39: Optimization procedure

5.2.1 TWF Case 1: Trenching Run

The objective function was calculated using TL exhibited by a single WM-5138 suppressor, TWF Case 1, seen in Figure 35, and the FWF seen in Figure 32. The normalized objective function values are presented in Figure 40. The highest value of the objective function occurs at a charge pressure of 13.1 MPa. There are three more local optima at charge pressures of 6.21 MPa, 2.76 MPa and 20.0 MPa, with normalized objective function values of 0.98, 0.76 and 0.05, respectively. All the local optimal pressures are in the set of single pressure optima. The local optimum at 6.21 MPa has a normalized objective function value over 0.95, placing it in the selected set, *O*. The normalized objective function values at 2.76 MPa and 20.0 MPa charge pressures do not cross the threshold for the selected set. The effect of suppressors with a CPR above 1 can be seen in Figure 40, in the region outlined in a box. The charge pressures in this region, 13.7 MPa to 20.7 MPa, have a CPR above 1 for all system pressures other than 20.7 MPa, and assumed to perform as an expansion chamber. The high CPR significantly lowers the *TL* of the suppressor and decreases the normalized objective function value to less than 0.03 for the range in the box.

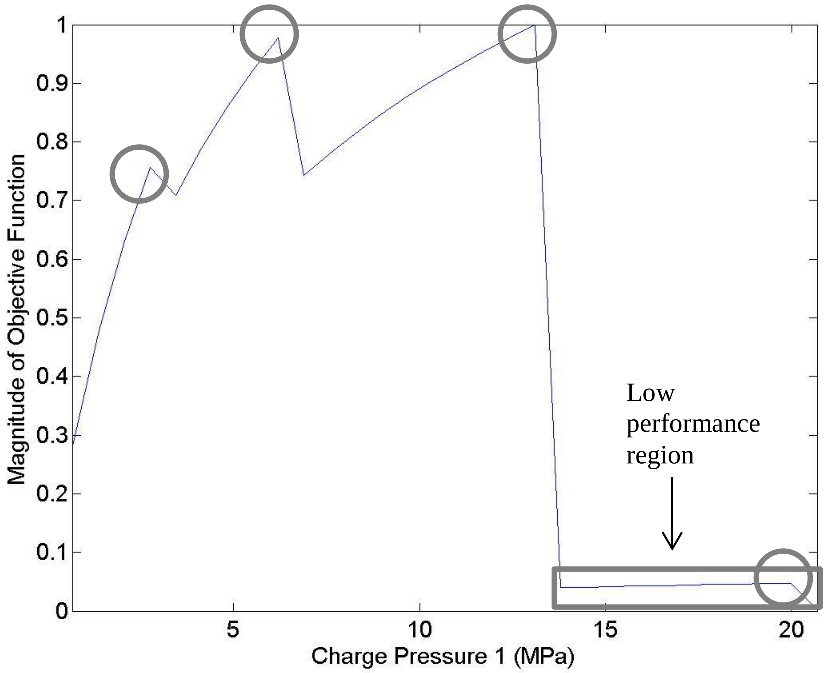

Figure 40: Objective function values: TWF Case 1, one WM-5138 Suppressor. Circles indicate local optima at charge pressures of 2.76, 6.27, 13.1 and 20.0 MPa, respectively. Box indicates low performance region

The selected set for this optimization is comprised of a charge pressure of 6.27 MPa and a charge pressure of 13.1 MPa, one of which will be selected for use. When the factors not directly captured by the objective function are considered, a charge pressure of 13.1 MPa has a smaller gradient than a charge pressure of 6.21 MPa, making 13.1 MPa the selected charge pressure for a single suppressor in a system operating in TWF Case 1. A system operating in TWF Case 1 spends over 50% of its duty cycle at a system pressure of 13.8 MPa, and the select charge pressure, 13.1 MPa, is the single pressure optimum for that system pressure. The system spends approximately a quarter of its duty cycle at system pressures of 3.45 MPa and 6.90 MPa, respectively. However, the single pressure optima associated with this system pressure do not have similar objective function values. While the single pressure optima of 3.45 MPa will never have a CPR of over 1, its CPR with a system pressure of 13.8 MPa is 0.22, a CPR which exhibit low *TL* and low objective function value. The single pressure optima of 6.90 MPA has a CPR of 0.45, meaning this pressure performs better at a system pressure 13.8 MPa, where the system spends over half its time.

The sharp drop offs in normalized objective function value with increasing charge pressure evident in Figure 40 occur at charge pressures matching system pressures in the TWF. The effect is explained by analyzing a charge pressure of 6.90 MPa. This charge pressure has a CPR of 1 for a system pressure of 6.90 MPa and 0.5 for a system pressure of 13.8 MPa. Charge pressures slightly higher than 6.90 MPa also have a CPR of over 1 for a system pressure of 6.90 MPa, and exhibit poor performance because of the assumption they behave as expansion chambers. However, their CPRs are higher for a system pressure of 13.8 MPa, while being less than 1, and their *TL* improves as well as their objective function values. For charge pressures slightly lower than 6.90 MPa the CPR for system pressures of 6.90 and 13.8 MPa are less than 1, meaning the suppressor does not perform as an expansion chamber for either system pressure, which means this charge pressure range exhibit the highest overall *TL* for both system pressures discussed.

The objective function was calculated using TL exhibited by two WM-5138 suppressors, TWF Case 1, seen in Figure 35, and the FWF seen in Figure 32. The normalized objective function values are shown in Figure 41. The values of the objective function are symmetric about charge pressure 1 equal to charge pressure 2 because *TL* is not dependent on charge pressure order. For a single system pressure the optimal charge pressure condition would be charging both suppressors to the single pressure optimum for this system pressure. The downside to a configuration with two identically charged suppressors is for lower system pressures the suppressors are both overcharged. A setup of dissimilar charge pressures will not exhibit as high of *TL* for larger system pressures but reduces the chance of both suppressors having a CPR higher than 1. The optimal point has a charge pressure pair of [2.76, 13.1] MPa, both single pressure optima for the system pressures of 3.45 MPa and 13.8 MPa, respectively. In the analysis of a single suppressor for the same TWF Case, seen above, it was shown that a charge pressure of 6.90 MPa has a higher objective function value than a charge pressure of 2.76 MPa. Charge pressure pairs of [2.76, 13.1] MPa and [6.21, 13.1] MPa generate similar *TL* curves for a system pressure of 13.8 MPa but there is a drastic difference for a system pressure of 3.45 MPa, as seen in Figure 42. The difference in *TL* is the cause of the difference in objective function value. The box in Figure 41 shows the region where the charge pressures of both suppressors are above 13.8 MPa, and the charge pressure pair exhibits extremely low objective function values. The system spends very little time at a system pressure of 20.7 MPa which is the only system pressure where the suppressor pairings with these charge pressures would have a CPR less than one.

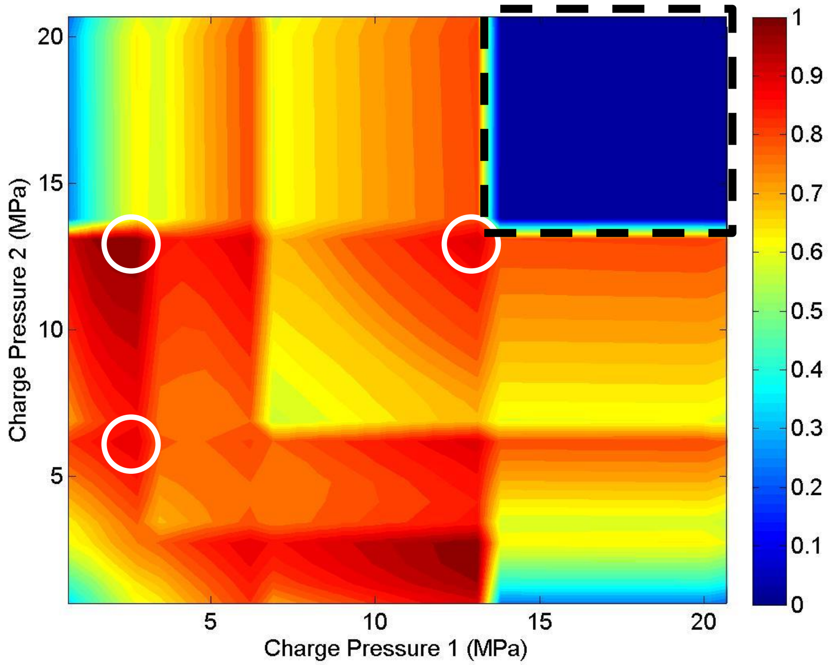

Figure 41: Objective function values: TWF Case 1, two WM-5138 Suppressors. Circles indicate local optima at charge pressure pairs of [2.76, 13.1] MPa, [2.76, 6.21] MPa and [13.1, 13.1] MPa, respectively. Box indicates low performance area.

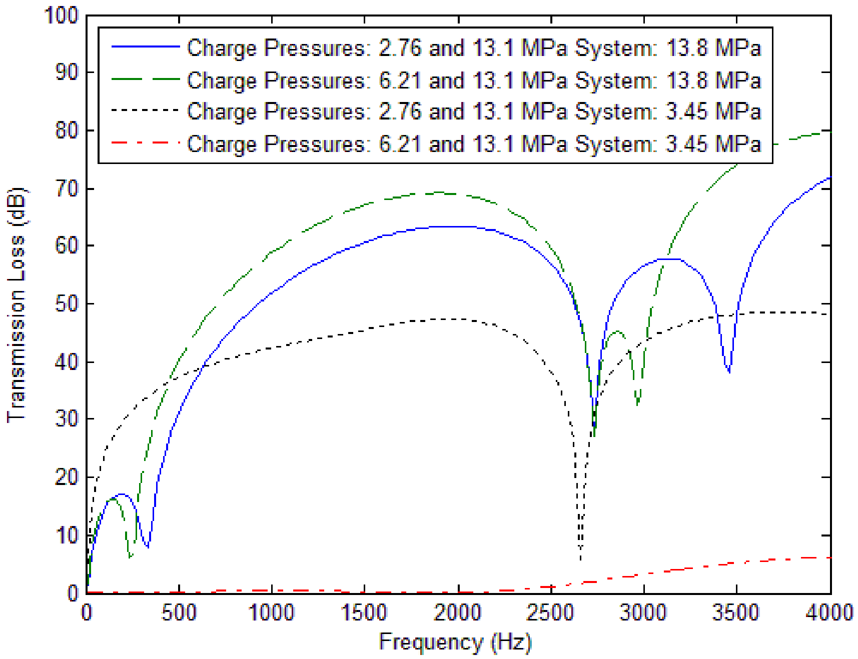

Figure 42: *TL* Peformance for fwo WM-5138 Suppressors with charge pressures matching optimal states in Figure 41

Other local optima in Figure 41 are found with charge pressure pairs of: [13.1, 13.1] MPa with a normalized objective function value of 0.92, [6.21, 13.1] MPa with a normalized objective function value of 0.917 and a charge pressure pair of [2.76, 6.21] MPa with a normalized objective function value of 0.903. None of these values cross the threshold of 0.95 to be considered in the selected set.

5.2.2 TWF Case 2: Back Filling

The objective function was calculated using TL exhibited by a single WM-5138 suppressor, TWF Case 2, seen in Figure 36, and the FWF seen in Figure 32. The normalized objective function values are presented in Figure 43. The system being optimized spends more time in the lower pressures than the system for TWF Case 1. With the TWF shifted to lower pressure so does the optimal charge pressure; a charge pressure of 6.21 MPa for this TWF Case. This charge pressure is the single pressure optimum for

6.90 MPa, which is the second largest system pressure value in the second TWF case. Charge pressures of 2.76 MPa and 6.27 MPa define the selected set for this TWF case, and are both in the set of single pressure optima. This result shows the single pressure optimum for the largest system pressure value in the TWF is not always the optimal charge pressure. The next highest local optimum is at a charge pressure of 2.76 MPa, with objective function value of 0.997. The factors not captured by the objective function were analyzed and the magnitude of the gradient near a charge pressure of 6.21 MPa is smaller than the gradient near a charge pressure of 2.76 MPa, therefore the charge pressure of 6.21 MPa would be selected for use in practice.

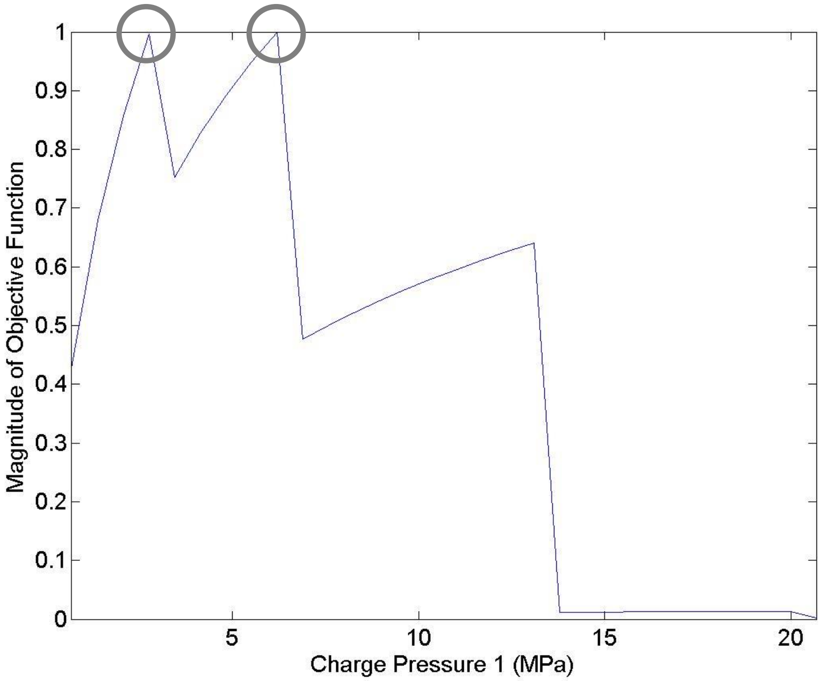

Figure 43: Objective function values: TWF Case 2, one WM-5138 Suppressor. Circles indicate local optima charge pressures of 2.76 and 6.27MPa, respectively

The objective function was calculated using TL exhibited two WM-5138 suppressors, TWF Case 2, seen in Figure 36, and the FWF seen in Figure 32. The normalized objective function values are shown in Figure 44. The optimal charge condition is at a charge pressure pairing of [2.76, 6.21] MPa. For this TWF Case, shown in Figure 36, the system spends the most time at a system pressure of 3.45 MPa followed by a system pressure of 6.90 MPa, the single pressure optimum of these system pressures are the charge pressures found in the optimal charge pressure pair. In addition, a charge pressure pair of [2.76, 13.1] MPa has an objective function value of 0.994, placing it in the selected set. The factors not captured by the objective function were used to determine which charge pressure pair to use. For TWF Case 2, a charge pressure pair of [2.76, 13.1] MPa will be selected. The box on Figure 44 indicates an area of relatively low objective function value. The low objective function values are cause by the charge pressure pairs in this region have a CPR less than one for a system pressure of 13.7 MPa and 20.9 MPa, where this system spends 25% of its duty cycle. This emphasizes the need for properly charged suppressors.

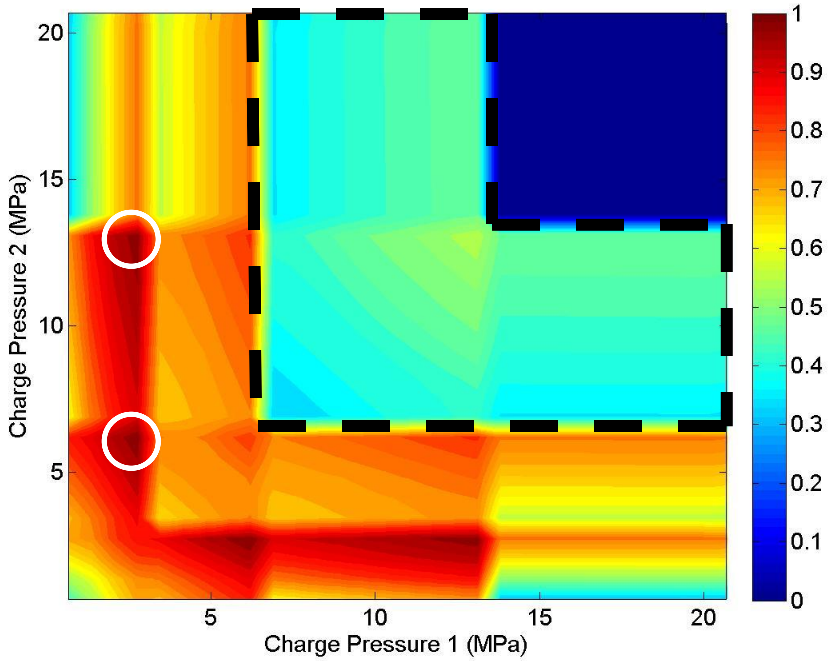

Figure 44: Objective function values: TWF Case 2, two WM-5138 Suppressors. Circles indicate local optima at charge pressure pairs of [2.76, 13.1] MPa and [2.76, 6.21] MPa, respectively

Despite the system spending 75% of its TWF at pressures where a charge pressure pair of [2.76, 13.1] MPa exhibits *TL* on the order of a single suppressor, since one of the suppressors has a CPR greater than 1, the charge pressure pair has an overall normalized objective function value of 0.99. The reason for the charge pressure pair exhibiting low *TL* yet having a high normalized objective function value is found from analysis of the FWF. The difference in *TL* for charge pressure pairs in the selected set at both system pressures under consideration, 6.90 and 13.8 MPa, was weighted by their respective FWF. The results of the frequency weighting are shown in Figure 45, where positive values mean a charge pressure pair of [2.67, 13.1] MPa performs better and negative values mean a charge pressure pair of [2.67, 6.21] MPa performs better. In the frequency range of 160 Hz to 210 Hz the charge pressure pair of [2.76, 13.1] MPa outperforms the charge pressure pair of [2.76, 6.27] MPa at both system pressures. The frequency weighted *TL* was also averaged; for a system pressure of 13.8 MPa the mean frequency weighted *TL* is 0.0052, this means the charge pressure pair of [2.76, 13.1] MPa preforms better at this system pressure. For a system pressure of 6.90 MPa the mean frequency weighted *TL* is -0.0041, meaning the charge pressure pair of [2.76, 6.21] MPa performs better at this system pressure. Comparing the magnitudes of frequency weighted *TL*, the charge pressure pair of [2.76, 13.1] MPa outperforms the charge pressure pair of [2.76, 6.27] MPa for system pressures of 6.90 and 13.8 MPa before being time weighted. After time weighting and normalization, the normalized objective functions values are found to be 1 for a charge pressure pair of [2.67, 6.21] MPa, and 0.99 for a charge pressure pair of [2.67, 13.1] MPa.

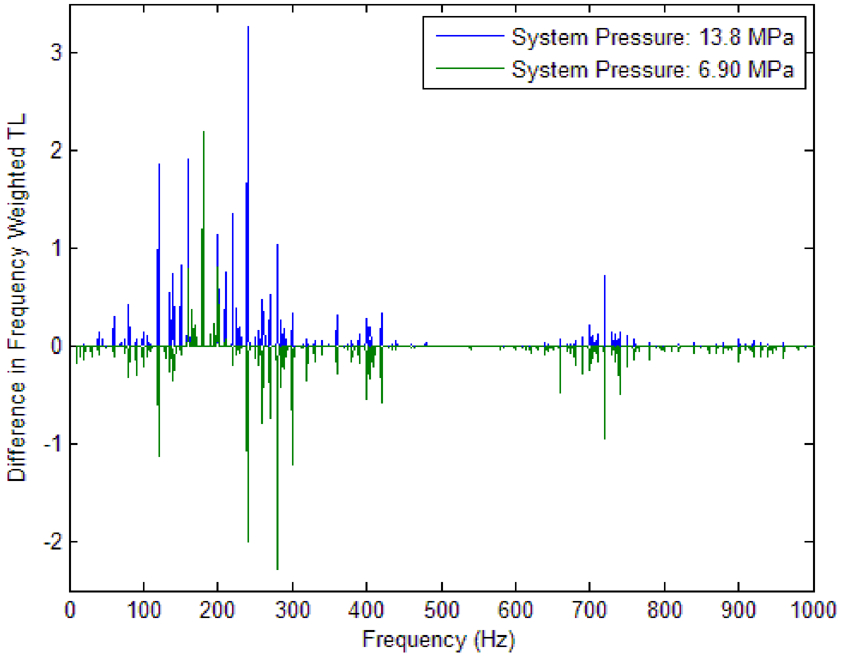

Figure 45: Difference in frequency weighted *TL* for charge pressure pairs of [2.76, 6.21] MPa and [2.76, 13.1] MPa at listed system pressures

5.2.3 TWF Case 3: Arbitrary Usage

The objective function was calculated using TL exhibited by a single WM-5138 suppressor, TWF Case 3, seen in Figure 37, and the FWF seen in Figure 32. The normalized objective function values are shown in Figure 46. The optimal value is at a charge pressure of 13.1 MPa, and no other charge pressures are in the selected set. The predominant system pressure of TWF Case 3 is 13.7 MPa, which has a single pressure optimum of a charge pressure of 13.1 MPa, which is the only point in the selected set and selected for use with this TWF.

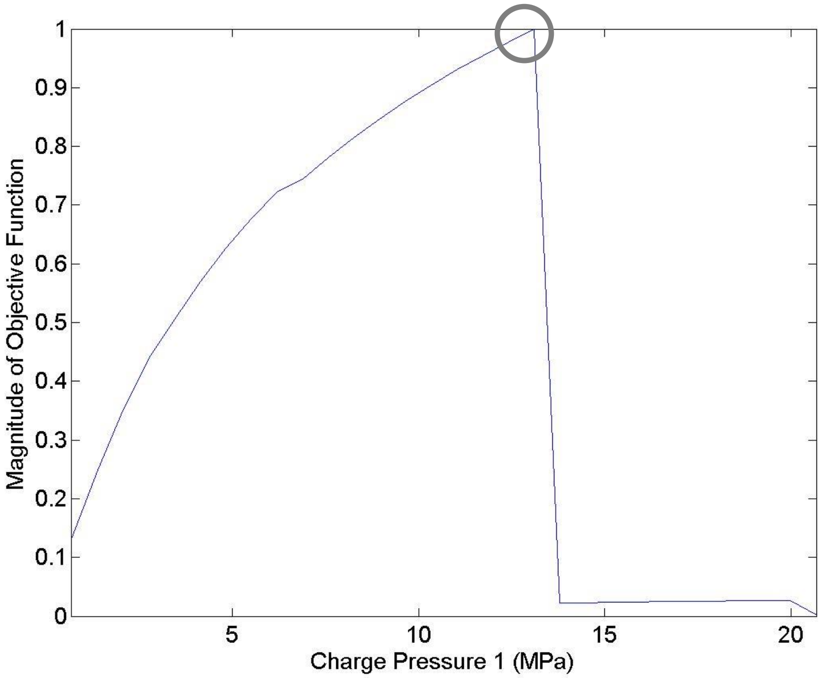

Figure 46: Objective function values: TWF Case 3; one WM-5138 Suppressor. Circle indicates a local optimum at a charge pressure of 13.1 MPa

The objective function was calculated using TL exhibited by two WM-5138 suppressors, TWF Case 3, seen in Figure 37, and the FWF seen in Figure 32. The normalized values of the objective function are seen in Figure 47. The optimal charge pressure pair is [13.1, 13.1] MPa, where 13.1 MPa is the single pressure optimum for 13.8 MPa system pressure. The optimal charge pressure configuration of a two-suppressor system operating with predominately one system pressure is a pairing of single pressure optima. An effect of the FWF is seen in the region outlined in Figure 47. For a system operating at a given pressure it is seen that increasing CPR improves *TL*, until the CPR becomes larger than one. Generally the highest *TL* is desirable for the optimal condition. However, in the box on Figure 47, charge pressure pairs exhibiting lower *TL* have higher normalized objective function values. For example; a charge pressure pair of [4.14, 13.1] MPa exhibits higher *TL* than a charge pressure pair of [0.69, 13.1] MPa; however, a charge pressure pair of [0.69, 13.1] MPa has a higher objective function value than a charge pressure pair of [4.14, 13.1] MPa. A comparison of the two *TL* curves to the FWF, shown in Figure 48, for a system pressure of 13.7 MPa explains this behavior. The *TL* curve for a charge pressure pair of [4.14, 13.1] MPa has low *TL* at the highest values of the FWF, in the region of 240 Hz outlined in the box, while a charge pressure pair of [1.38, 13.1] MPa has a *TL* near 20 dB for the same frequencies, and this range is outlined in black. Above 300 Hz, the *TL* for the charge pressure pair of [4.14, 13.1] MPa is higher than the charge pressure pair of [0.69, 13.1] MPa, however the values of FWF are below 0.001 in this range. The charge pressure pair of [0.69, 13.1] MPa has a higher objective function value than a charge pressure pair of [4.14, 13.1] MPa because of the weighting of the FWF. Also shown in Figure 48 is the *TL* curve of a charge pressure pair of [13.1, 13.1] MPa, the optimal charge pressure. The *TL* exhibited by a charge pressure pair of [13.1, 13.1] MPa approaches 0 dB at a frequency of 200 Hz; however, the *TL* increases to a similar value of a charge pressure pair of [0.69, 13.1] MPa at 240 Hz, and exceeds it above 240 Hz leading to a higher normalized objective function value and selection as the optimal charge pressure configuration.

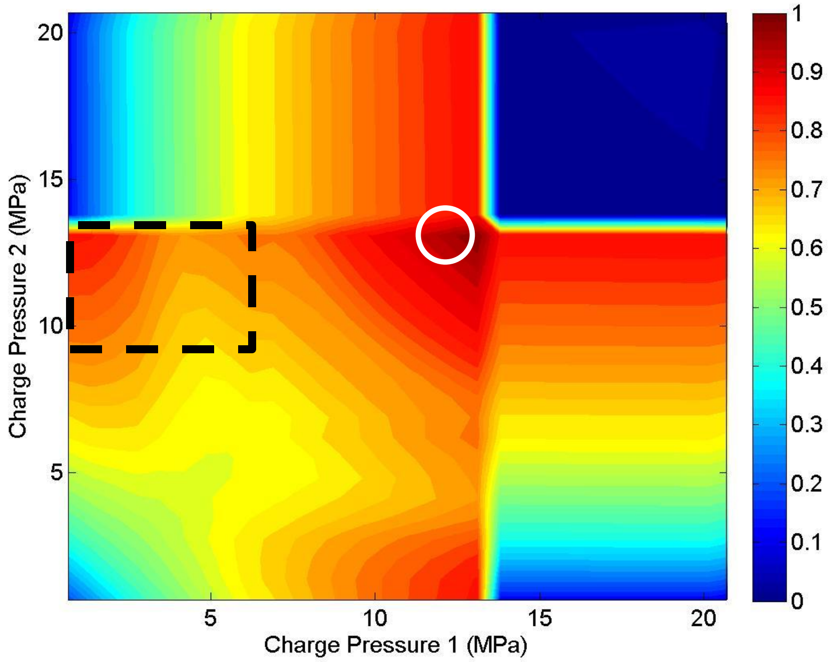

Figure 47: Objective function values; TWF Case 3, two WM-5138 Suppressors. Circle indicates a local optimum at a charge pressure pair of [13.1, 13.1] MPa. Box indicates region of low overall *TL* and high normalized objective function value.

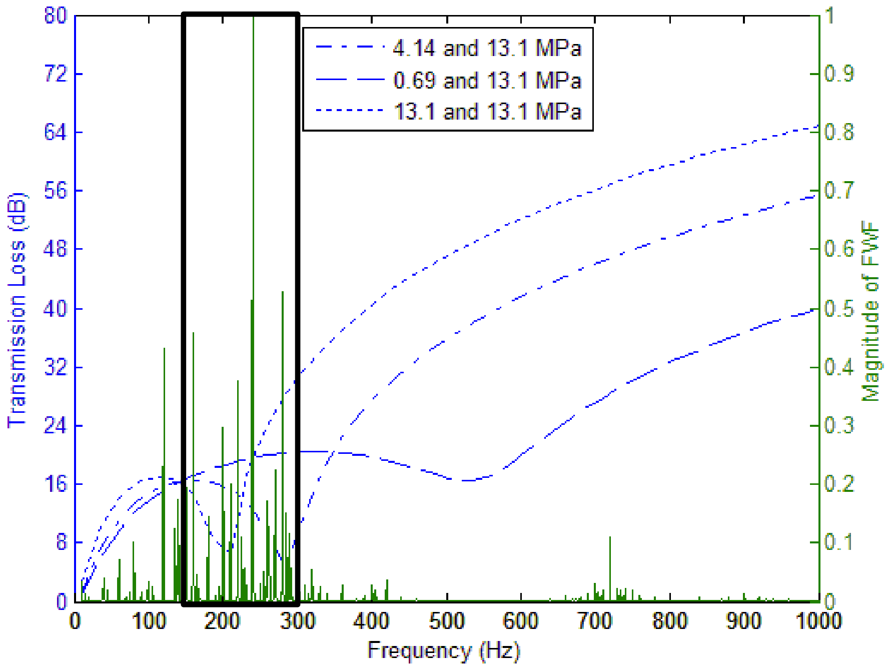

Figure 48: Transmission loss curves for two charge pressure pairs and FWF at 13.7 MPA. Box indicates region of high FWF value

5.2.4 TWF Case 4: Mixed Usage

The objective function was calculated using TL exhibited by a single WM-5138 suppressor, TWF Case 4, seen in Figure 38, and the FWF seen in Figure 32. The normalized objective function values are presented in Figure 49. TWF Cases 1 and 2 have been combined to form TWF Case 4, and it follows the objective function values for this TWF case should have similarity to the objective function values of TWF Cases 1 and 2. The selected set for TWF Case 1 is 6.21 MPa and 13.1 MPa, and the selected set for TWF Case 2 is 2.76 MPa and 6.21 MPa. The overlapping point of the selected set, 6.21 MPa, becomes the global optimum for this TWF case. The other points from the selected sets of TWF Case 1 and TWF Case 2 fall below the selected set threshold of 0.95.

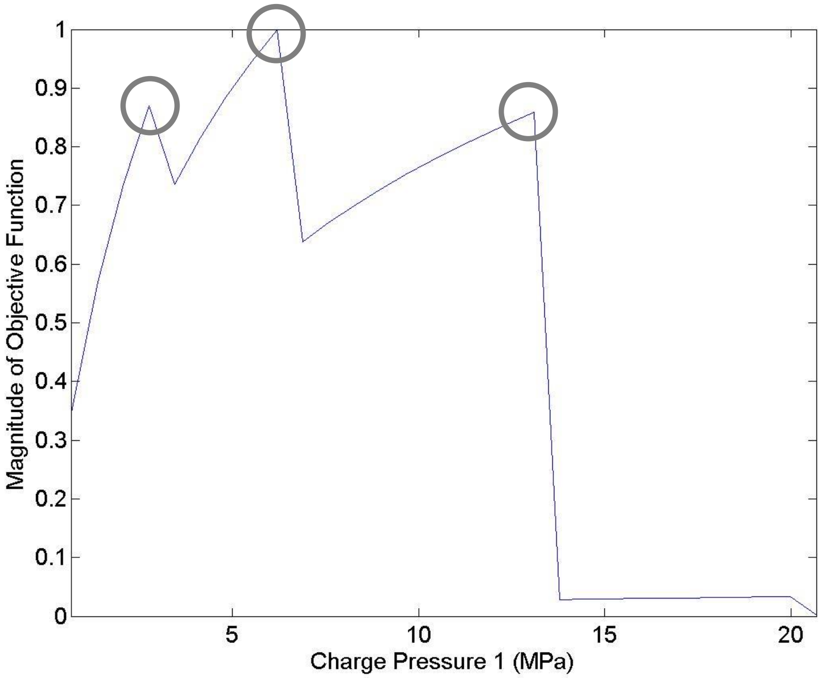

Figure 49: Objective function values: TWF Case 4; one WM--5138 Suppressor. Circles indicate local optima at charge pressures of 2.76, 6.21 and 13.1 MPa, respectively.

The objective function was calculated using TL exhibited by a single WM-5138 suppressor, TWF Case 4, seen in Figure 38, and the FWF seen in Figure 32. The normalized objective function values are presented in Figure 50. The optimal charge pressure pair is [2.76, 13.1] MPa. This follows from the objective function values of TWF Case 1, seen in Figure 41, and objective function values of TWF Case 2, seen in Figure 44, as a charge pressure pair of [2.76, 13.1] MPa has a value of 1 and 0.99 for TWF Cases 1 and 2, respectively. A local optimum with an objective function value of 0.95 occurs at a charge pressure pair of [2.76, 6.21] MPa. This charge pressure pair is also found to be in the selected set of TWF Case 2 and has a normalized objective function value above 0.9 for TWF Case 1. The factors not directly captured by the objective function must be considered to differentiate between the local optima in the selected set. The gradient in the region near the charge pressure pair of [2.76, 13.1] MPa is smaller than the gradient in the region near the charge pressure pair of [2.76, 13.1] MPa, thus the charge pressure pair of [2.76, 13.1] MPa would be selected for use with this TWF.

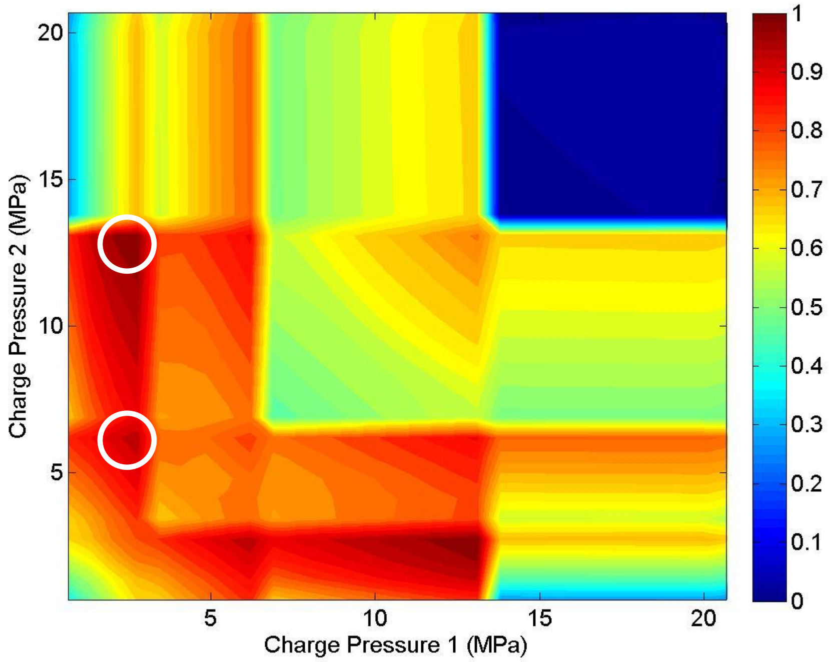

Figure 50: Objective function values; TWF Case 4, two WM-5138 Suppressors. Circles indicate local optima at charge pressure pairs of [2.76, 13.1] MPa and [2.76, 6.21] MPa, respectively.

5.2.5 *Results with a 30 dB Constraint on Transmission Loss*

As seen in Section 3.3, there may be an upper limit to *TL* in a given system. A maximum *TL* ceiling of 30 dB was applied to the results from TWF Case 1 even if the model predicted *TL* greater than 30 dB. The objective function values for TWF Case 1, seen in Figure 35, with a single suppressor are calculated with this constraint and the normalized values are shown in Figure 51. The objective function has a selected set comprising of 6.27 MPa and 13.1 MPa, the same selected set as the unconstrained *TL* case, also presented in Figure 51. The factors not captured by the objective function are used to select the optimal condition from the selected set. For the capped *TL* case of a system operating at TWF Case 1, the objective function for a charge pressure of 13.1 MPa has a smaller gradient than that of a charge pressure of 6.21 MPa. The charge pressure of 13.1 MPa would be selected for use in the system, the same pressure selected for TWF Case 1 with unconstrained *TL*. Since the difference between the objective function values for unconstrained *TL* and constrained *TL* is difficult to determine in Figure 51, the difference between the objective function values are shown in Figure 52. The largest difference occurs at a charge pressure of 13.1 MPa, the optimal condition. The difference of the two magnitudes is 0.0065, considered to be insignificant, indicating that imposing a maximum *TL* constraint of 30 dB will not affect single suppressor optimization.

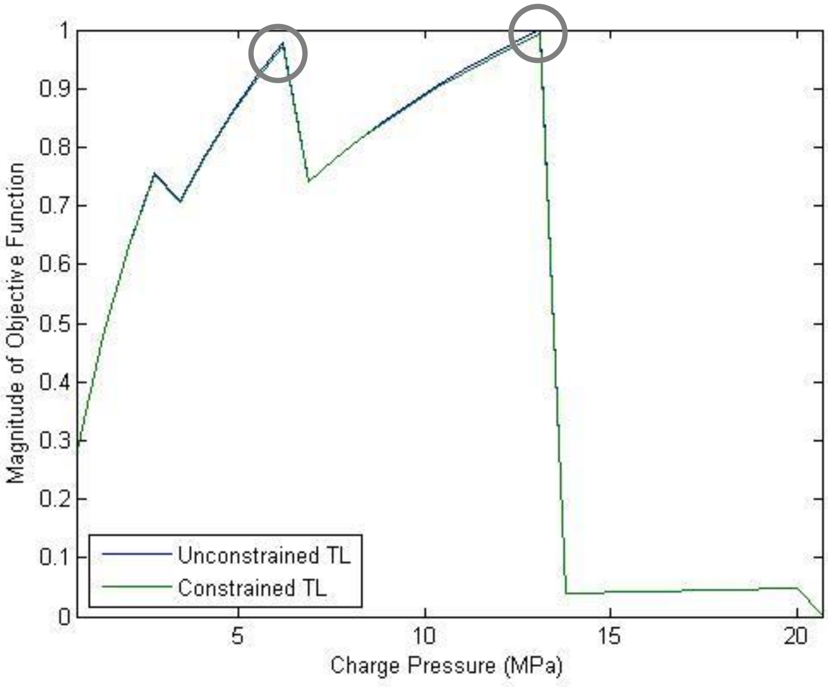

Figure 51: Objective function values: TWF Case 1, one WM-5138 Suppressor, unconstrained *TL* and constrained *TL*. Circles represent Charge Pressures of 13.1 and 6.27 MPa, respectively.

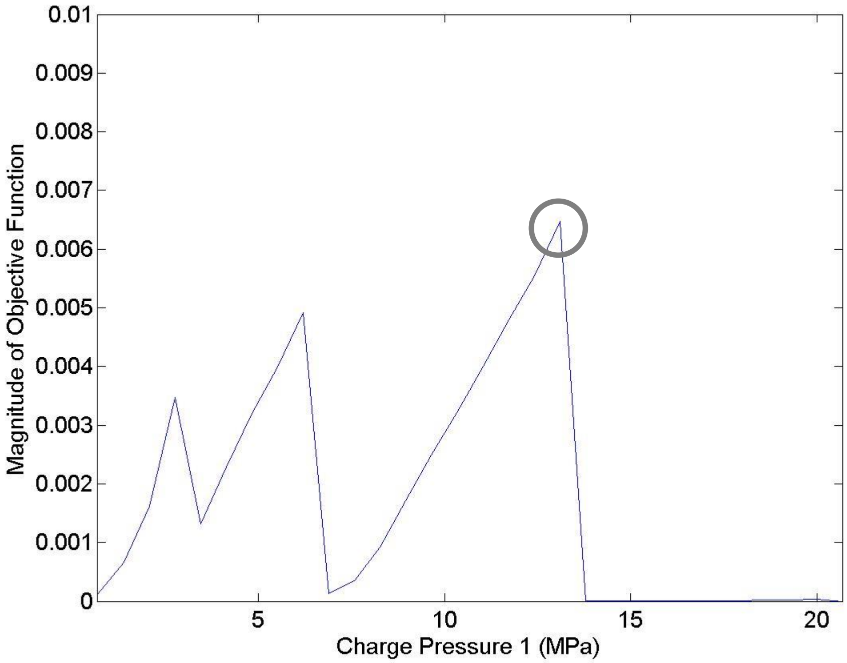

Figure 52: Difference in objective function values for constrained and unconstrained *TL*. Circle indicates charge pressure of 13.1 MPa

A maximum *TL* constraint of 30 dB was also applied to the two suppressor case, and the objective function was recalculated, with the resulting values shown in Figure 53. The optimal value occurs at a charge pressure pair of [2.76, 13.1] MPa, the same as the uncapped *TL* case. Similarly, the optimal charge pressure pair is the only charge pressure pair in the selected set. The objective function values of the constrained *TL* case were normalized to the maximum of the unconstrained *TL* case; and the difference between the cases is shown in Figure 54. The difference at the selected optimal charge pressure pair is 0.048. The difference is significant, but the optimal point does not shift to a different charge pressure pair. This indicates that the optimization method developed is not strongly affected by a maximum achievable *TL*. The maximum difference of normalized objective function value occurs, of 0.10, at a charge pressure pair of [13.1, 13.1] MPa, a charge pressure pair exhibiting large *TL* for system pressures of 13.8 MPa and above. However, this charge pressure pair is not in the selected set for either the *TL* cases. The differences are relative to the *TL* produced, as a charge pressure pairs with high *TL* are affected more, and the difference is not strongly dependent on TWF. Charge pressure configurations of both single pressure optimum, charge pressure pairs of [2.76, 2.76] MPa, [6.21, 6.21] MPa and [13.1, 13.1] MPa, exhibit the largest *TL* and the differences between constrained and unconstrained *TL* have local maxima at these points. In addition, the difference is larger for a two suppressor configuration than a single suppressor configuration because of a large difference between constrained and unconstrained *TL*.

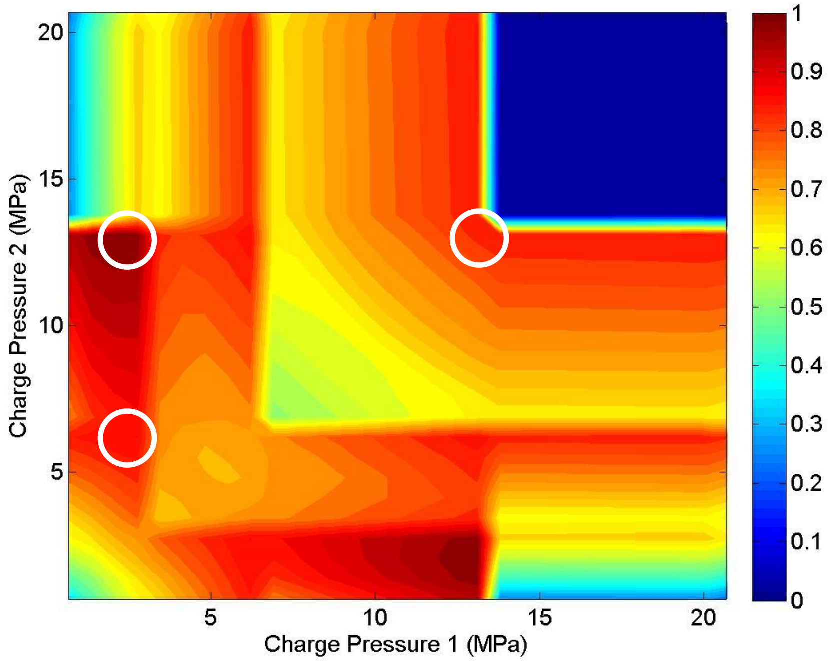

Figure 53: Objective function values: TWF Case 1; two WM-5138 Suppressors, constrained TL. Circles indicate local optima at charge pressure pairs of [2.76, 13.1] MPa, [2.76, 6.21] MPa and [13.1, 13.1] MPa respectively

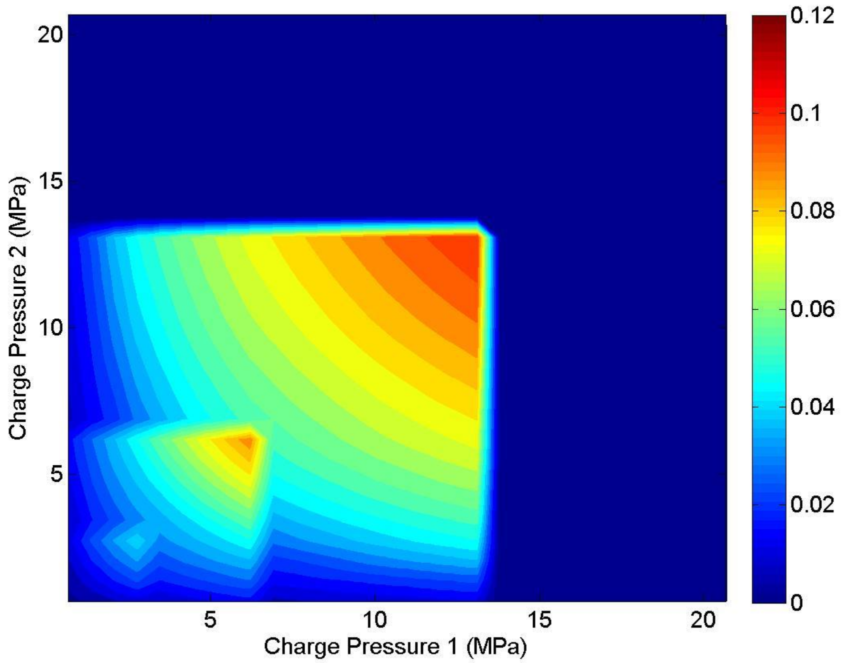

Figure 54: Difference in objective function values by imposed 30 dB *TL* constraint on TWF Case 1, two WM-5138 Suppressors

CHAPTER 6

CONCLUSIONS

A method for optimizing the charge pressure for either one or two suppressors operating in a system with a variety of pressures has been developed. The transmission loss (*TL*) behavior of the suppressors is predicted by an equivalent fluid model. Both experiment and modeling show the *TL* of the suppressors is dependent on the charge pressure ratio (CPR) and varies over frequency. The variance of *TL* across system pressure and frequency is accounted for in an objective function through use of a time weighting factor (TWF) and a frequency weighting factor (FWF). The TWF considers the time spent at each system pressure, while the FWF considers the frequency spectrum of the noise. For this work the TWFs are anticipated usages, while the FWFs are the results of experiment.

The selected set for all optimizations are made up of single pressure optima. The single pressure optimum is the optimum charge pressure for a single system pressure, and this point has been found to be close to a CPR of 0.9. For single suppressor optimizations, the optimal point is often the single pressure optimum for the largest value in the TWF; however, exceptions do occur for some TWF cases. This effect weights the objective function towards lower charge pressures.

For two suppressor optimizations both charge pressures are single pressure optimal, though rarely the same charge pressure. The optimal charge pressures are dependent on the TWF, however the selected charge pressures are not always the single pressure optimum for the system pressures corresponding to the highest two values of the TWF.

The spectral content of the noise weights the objective function through the FWF. Generally, the largest amount of noise occurs in the low frequency range. The spectral content of the noise shifts frequencies system as pressures rise. In addition, the mean value of the FWF rises with system pressure, meaning higher system pressures produce more noise. The optimal condition is not greatly affected by the FWF, however, a charge pressure configuration with low *TL* at the frequencies with high FWF values has a low objective function value.

Each system pressure causes the system to behave differently, both through its effect on CPR and effect on FWF. During use the suppressor has a fixed charge pressure, and varying the system pressure effects *TL* by changing CPR. The TWF weights the objective function towards highly used system pressures to account for these effects. The effect of the TWF is such that changing the TWF significantly change the optimal charge pressure configuration.

For some optimizations, there are multiple charge pressure configurations with objective function values in the selected set. In such cases, factors not directly captured by the objective function must be considered to determine the optimal charge pressure configuration. The first deciding factor considered is the gradient of the objective function in the neighborhood of the optimal point. A lower gradient magnitude means a given optimal point performs better as the charge pressure decreases during use. Another factor used for consideration is the air-borne noise measured in-situ for a given system.

Bladder-style suppressors only suppress fluid borne noise for the fluid within the device, but acoustic energy may take an alternative path, such as along the shell, to downstream of the device and in consequence some systems may exhibit a maximum achievable *TL*. The objective function was applied to a simulation exhibiting a hypothetical constrained maximum *TL*. The single suppressor optimization is effected less than the double suppressor optimization, as less *TL* is expected from a single suppressor than two suppressors. The objective function still predicts similar optimal points for a system with a constrained maximum *TL* as a system without a constrained maximum *TL*, such that the optimization results hold even if a system exhibits a *TL* ceiling.

6.1 Future Work

Further development of this work should include improvement in model characterization of the suppressors, with a focus on suppressors having a CPR larger than 1, improving the measurement technique and expanding the pressure resolution of the FWF. An update to the model should better predict the *TL* nulls, especially at low frequencies for two suppressor cases, especially relevant due to the predicted nulls' effects on objective function value. For this work, suppressors with a CPR larger than 1 are assumed to exhibit *TL* behavior similar to that of an expansion chamber. A better model of suppressors for this case would better inform the optimal conditions. The experimentally measured TL shows a significant amount of artifacts. Improvement needs to be made to the measurement technique to remove the artifacts in order to further inform suppressor behavior. In suppressor configurations with high predicted *TL*, transfer function coherence decreases below the necessary threshold for measurement. A method for ensuring high transfer function coherence in these cases will allow the behavior of suppressor configuration exhibiting high *TL*, specifically two suppressor configurations, to be further investigated. Improving the data acquisition will also better inform the model, enabling more accurate results. A FWF with higher frequency resolution would increase the set of single pressure optima and better reflect hydraulic systems, which work over a continuous pressure range not discrete pressures.

For two suppressor configurations, the suppressors were both assumed to be the same size. Non-identical suppressors may improve performance as suppressor geometry affects the shape of the *TL* curve allowing specific frequencies to be targeted. Future work should analyze the effect of using two dissimilarly sized suppressors. Using dissimilar suppressors may allow the shape of the *TL* curve, particularly the low frequency *TL* decrease, to better match the shape of the FWF.

Additional future work could be conducted by using the objective function to inform an optimal design of a single suppressor operating with a given FWF and a set of given TWF. In order to find an accurate optimal suppressor design, a model for predicting the FWF of a given pump operating within a system will need to be developed.

APPENDIX A

MATLAB FUNCTION FOR CALCULATION OF TRANSMISSION LOSS

```
function
TL_calc5(supNum,freq,PcVec,PsVec,fileName,r_0,r_a0,r_b,zlen,L,unitset)

%Function: TL_calc4.m
%Version: 4
%Revision: Corrects predicted transmission loss for the case when one
%suppressor is overcharged and one suppressor is undercharged
%Revision: Correction of suppressor terminology
%Revision: Allows user to omit individual dimensions and use default
values
%as well as simulate either a single suppressor set up or a double
%suppressor set up.
%         supNum - The number of suppressors in the system
%         freq - frequencies of interest
%Inputs:
 [Hz] (vector 1 x V)
%         PcVec - Series of monotonically increasing charge pressures
[psi]
%         or [Pa] (vector 1 x W)
%         PsVev - Series of monotonically increasing static pressures
[psi]
%         or [Pa] (vector 1 x X)
%         fileName - title of saved mat file (string)
%Optional Inputs:
%         r_0 - inner radius of inlet pipe [in] or [m]
%         r_a0 - outer radius of suppressor annulus [in] or [m]
%         r_b - inner radius of suppressor shell [in] or [m]
%         zlen - inner length of suppressor [in] or [m]
%         L - seperation length between suppresors [in] or [m]
%         unitset - a string of either English or Metric which
determines
%         the unit set used for the entire simulation
%Outputs:
%         A saved W x W x X x V matrix of transmission loss for every
%         condition simulated

starttime=now;

if supNum ~= 1 && supNum ~= 2
    error('Not a Simulation case')
end

%Default dimensions
dr_0 = 0.69291338512; %radius of inlet pipe
dr_a0 = 0.99212598324; %uncompressed bladder radius
dr_b = 1.64173228179; %outer radius of silencer
dzlen = 2.68503936734; %estimated effective length
```

```
dL=3.93700787; %Back to back seperation length
dunitset='English';

% Default Suppressor dimensions [in]
if nargin == 5
    r_0 = dr_0; %radius of inlet pipe
    r_a0 = dr_a0; %uncompressed bladder radius
    r_b = dr_b; %outer radius of silencer
    zlen = dzlen; %estimated effective length
    L=dL; %Back to back seperation length
    unitset=dunitset;
end

if isempty(r_0)
    r_0=dr_0;
end
if isempty(r_a0)
    r_a0=dr_a0;
end
if isempty(r_b)
    r_b=dr_b;
end
if isempty(zlen)
    zlen=dzlen;
end
if isempty(L)
    L=dL;
end
if isempty(unitset)
    unitset=dunitset;
end

switch unitset %The Default Unit set is English Units, however the code
uses metric.
    case 'English' %If English units are used they are converted into
metric
        r_0=r_0./39.3700787;
        r_a0=r_a0./39.3700787;
        r_b=r_b./39.3700787;
        zlen=zlen./39.3700787;
        L=L./39.3700787;
        PcPlot=PcVec; %#ok<NASGU> %Used on the contour plot
        PcVec=PcVec.*6894.75729;
        PsVec=PsVec.*6894.75729;
    case 'Metric'
        PcPlot=PcVec; %#ok<NASGU>
end

switch supNum
    case 1
        m=1;
        o=1;
        overall=0;
        total=length(PsVec)*length(PcVec); %The total number of
simulations being run
```

```
        TLmat=zeros(length(PcVec),length(PsVec),length(freq));
%Initializes TLmat to proper size
        for Pc1=PcVec
                for Ps=PsVec
                   TL = WM_shell(freq,Pc1,Ps,r_0,r_a0,r_b,zlen); %Calls
the function that similuates back to back suppressors
                   overall=overall+1;
                   disp(['Simulation ' num2str(overall) ' of '
num2str(total)])
                   TLmat(m,o,:)=TL; %Saves TL data in proper location
                   o=o+1;
                end
            m=m+1;
            o=1;
        end
    case 2
        n=1;
        m=1;
        o=1;
        overall=0;
        total=length(PsVec)*.5*length(PcVec)*(length(PcVec)+1)*1; %The
total number of simulations being run

TLmat=zeros(length(PcVec),length(PcVec),length(PsVec),length(freq));
%Initializes TLmat to proper size
        for Pc1=PcVec
%             PcVec2=PcVec(PcVec>=Pc1);
            for Pc2=PcVec
                for Ps=PsVec
                   TL =
WM_shell_Double(freq,Pc1,Pc2,Ps,r_0,r_a0,r_b,zlen,L); %Calls the
function that similuates back to back suppressors
                   figure; plot(freq,TL)
                   title([num2str(Pc1./6894.75729)
num2str(Pc2./6894.75729)])
                   clearvars -except freq Pc1 Pc2 Ps r_0 r_a0 r_b zlen
L o n m TLmat PcPlot PcVec PsVec unitset supNum overall starttime
fileName total TL
                   clearvars -global
                   overall=overall+1;
                   disp(['Simulation ' num2str(overall) ' of '
num2str(total)])
                   TLmat(m,n,o,:)=TL; %Saves TL data in proper location
                   clear TL
                   o=o+1;
                end
                n=n+1;
                o=1;
            end
            m=m+1;
            n=m;
            o=1;
        end
end
save(fileName,'TLmat','PcPlot','PcVec','PsVec','freq','unitset','supNum
')
```

```
endtime = now;
sec = (endtime-starttime)*60*60*24;
minu = floor(sec/60);
    sec = sec-minu*60;
hr = floor(minu/60);
    minu = minu-hr*60;
day = floor(hr/24);
    hr = hr-day*24;

timestr = 'is';
if day > 0
    timestr = [timestr ' ' num2str(day) ' days'];
end
if hr > 0
    timestr = [timestr ' ' num2str(hr) ' hours'];
end
if minu > 0
    timestr = [timestr ' ' num2str(minu) ' minutes'];
end
timestr = [timestr ' ' num2str(sec) ' seconds'];


disp(['Elapsed time for this simulation ' timestr '.'])
end


function TL = WM_shell(freq,Pc,Ps,r_0,r_a0,r_b,zlen)

starttime=now;
vs = '(v20)';
disp('*************************')
disp(['Running WM_shell.m ' vs])
disp('*************************')

global showdebug
showdebug = 0; %1 to show debug/error messages

% Initialize simulation data

% Frequencies of interest
freqsize = size(freq);
if freqsize(2) > freqsize(1)
    freq = freq.';
end
clear freqsize
freqw = freq.'*2*pi;
numharm = length(freq);

% Modes of interest
nummode = 6;
% Pc=6894.75729*Pc;
% Ps=6894.75729*Ps;

% Fluid properties
tmp = 35; %fluid temp, C
```

```
c_f = 1400; %measured sound speed in hydraulic fluid
rho_f = 865.9731; %density of hydraulic fluid
k_f = freqw./c_f; %wavenumber in free fluid
Z_f = rho_f*c_f*ones(1,length(freq)); %Specific acoustic impedance for
fluid
lambda_f = ones(1,numharm)*c_f^2*rho_f; %fluid bulk modulus

% Downstream port impedance
Zp2plus = getzp2(freq, Z_f); %downstream port is modeled as anechoic by
default

lambda_s = ones(1,numharm)*1.4*max(Ps,Pc)*exp(0.0i); %pressurized gas
bulk modulus

% Silencer dimensions
if nargin == 3
    r_0 = 0.0176; %radius of inlet pipe
    r_a0 = 0.0252; %uncompressed bladder radius
    r_b = 0.0417; %outer radius of silencer
    zlen = 0.0682; %estimated effective length
end
if Pc>=Ps
    r_b=r_a0+0.001;
end
%calculate mass of precharge gas
mass = 0.028*Pc*zlen*pi*(r_b^2-r_a0^2)/(8.314*(273+tmp));

r_a = sqrt(r_b^2-(mass/0.028*8.314*(273+tmp)/(max(Ps,Pc)*zlen*pi)));
%compressed radius
sig = (0.05)/(2*pi*r_a*zlen); %mass per area of bladder; effective
insert mass is ~50g?
rho_s = mass/(zlen*pi*(r_b^2-r_a^2)); %compressed density

%wavenumbers and sound speeds
c_L = sqrt(lambda_s/rho_s); %longitudinal sound speed
k_L = freqw./c_L; %longitudinal wave number

% Build data structure
datstruct = struct('freq',freq,'freqw',freqw,'numharm',numharm,...

'nummode',nummode,'Ps',Ps,'Pc',Pc,'tmp',tmp,'c_f',c_f,'rho_f',rho_f,...
    'k_f',k_f,'Z_f',Z_f,'Zp2plus',Zp2plus,'rho_s',rho_s,'sig',sig,...
    'c_L',c_L,'k_L',k_L,'r_0',r_0,'r_a',r_a,'r_b',r_b,'zlen',zlen,...
    'lambda_f',lambda_f,'lambda',lambda_s,'k1_rf',[],'k1_zf',[],...

'k2_rf',[],'k2_rL',[],'k2_zf',[],'k2_zL',[],'TL',[],'coef_mat',[],...
    'pcp',[],'pcm',[],'badfreq',[],'numpmode',nummode,'numlmode',0,...
    'numpint',nummode,'numlint',0,'showdebug',showdebug);
if showdebug == 1
    save simdat datstruct
end

% Run simulation and find objective function
datstruct = analyz(datstruct);
if showdebug == 1
```

```
    save simdat datstruct
end
datstruct = proc(datstruct);
TL = datstruct.TL;
if showdebug == 1
    save simdat datstruct
end

% % Plot results of simulation
% figure;
% plot(freq,TL);
% xlabel('Frequency [Hz]'), ylabel('TL [dB]')
% title([num2str(Pc/1e6) ' MPa precharge, ' num2str(Ps/1e6) ' MPa
system'])

% Time data
% Display how long the simulation took
endtime = now;
sec = (endtime-starttime)*60*60*24;
min = floor(sec/60);
    sec = sec-min*60;
hr = floor(min/60);
    min = min-hr*60;
day = floor(hr/24);
    hr = hr-day*24;

timestr = 'is';
if day > 0
    timestr = [timestr ' ' num2str(day) ' days'];
end
if hr > 0
    timestr = [timestr ' ' num2str(hr) ' hours'];
end
if min > 0
    timestr = [timestr ' ' num2str(min) ' minutes'];
end
timestr = [timestr ' ' num2str(sec) ' seconds'];


disp(['Elapsed time for this simulation ' timestr '.'])

end
% WM_shell.m
% TL = WM_shell(freq,Pc,Ps)
% Vs 20
% Modified for gas bladder silencer
% Outputs: TL = predicted transmission loss [dB] (vector n x 1)
% Inputs: freq = frequencies of interest [Hz] (vector n x 1)
%         Pc = charge pressure of bladder [Pa]
%         Ps = system pressure [Pa]

function TL = WM_shell_Double(freq,Pc1,Pc2,Ps,r_0,r_a0,r_b,zlen,L)
clear datstruct TL
starttime=now;
vs = '(v20)';
```

```
disp('*************************')
disp(['Running WM_shell.m ' vs])
disp('*************************')


% Pc1=6894.75729*Pc1;
% Pc2=6894.75729*Pc2;
% Ps=6894.75729*Ps;

P=[Pc1,Pc2];
for x=[1:2]
    clearvars -except freq Pc1 Pc2 Ps r_0 r_a0 r_b zlen L o n m TLmat
PcPlot PcVec PsVec unitset supNum overall starttime fileName total TL
datstruct x P Z_f_new E0_new F0_new Edown Fdown int_L
global showdebug
showdebug = 0; %1 to show debug/error messages

% Initialize simulation data

% Frequencies of interest
freqsize = size(freq);
if freqsize(2) > freqsize(1)
    freq = freq.';
end
clear freqsize
freqw = freq.'*2*pi;
numharm = length(freq);

% Modes of interest
nummode = 6;

% Fluid properties
tmp = 35; %fluid temp, C
c_f = 1400; %measured sound speed in hydraulic fluid
rho_f = 865.9731; %density of hydraulic fluid
k_f = freqw./c_f; %wavenumber in free fluid
Z_f = rho_f*c_f*ones(1,length(freq)); %Specific acoustic impedance for
fluid
lambda_f = ones(1,numharm)*c_f^2*rho_f; %fluid bulk modulus

% Downstream port impedance
if x==1
    Zp2plus = getzp2(freq, Z_f); %downstream port is modeled as
anechoic by default
elseif x==2
    Zp2plus = Z_f_new;
end

lambda_s = ones(1,numharm)*1.4*max(Ps,P(x))*exp(0.0i); %pressurized gas
bulk modulus

% Silencer dimensions (in meters)
% r_0 = 0.0176; %radius of inlet pipe
% r_a0 = 0.0252; %uncompressed bladder radius
% r_b = 0.0417; %outer radius of silencer
% zlen = 0.0682; %estimated effective length

%calculate mass of precharge gas
mass = 0.028*P(x)*zlen*pi*(r_b^2-r_a0^2)/(8.314*(273+tmp));

r_a = sqrt(r_b^2-(mass/0.028*8.314*(273+tmp)/(max(Ps,P(x))*zlen*pi)));
%compressed radius
sig = (0.05)/(2*pi*r_a*zlen); %mass per area of bladder; effective
insert mass is ~50g?
rho_s = mass/(zlen*pi*(r_b^2-r_a^2)); %compressed density

%wavenumbers and sound speeds
c_L = sqrt(lambda_s/rho_s); %longitudinal sound speed
k_L = freqw./c_L; %longitudinal wave number

% Build data structure
datstruct = struct('freq',freq,'freqw',freqw,'numharm',numharm,...

'nummode',nummode,'Ps',Ps,'Pc',P(x),'tmp',tmp,'c_f',c_f,'rho_f',rho_f,.
..
    'k_f',k_f,'Z_f',Z_f,'Zp2plus',Zp2plus,'rho_s',rho_s,'sig',sig,...
    'c_L',c_L,'k_L',k_L,'r_0',r_0,'r_a',r_a,'r_b',r_b,'zlen',zlen,...
    'lambda_f',lambda_f,'lambda',lambda_s,'k1_rf',[],'k1_zf',[],...

'k2_rf',[],'k2_rL',[],'k2_zf',[],'k2_zL',[],'TL',[],'coef_mat',[],...
    'pcp',[],'pcm',[],'badfreq',[],'numpmode',nummode,'numlmode',0,...
    'numpint',nummode,'numlint',0,'showdebug',showdebug);
if showdebug == 1
    save simdat datstruct
end

if P(x)>=Ps
    r_b=r_a0;
    datstruct = struct('freq',freq,'freqw',freqw,'numharm',numharm,...

'nummode',nummode,'Ps',Ps,'Pc',P(x),'tmp',tmp,'c_f',c_f,'rho_f',rho_f,.
..
    'k_f',k_f,'Z_f',Z_f,'Zp2plus',Zp2plus,'rho_s',rho_s,'sig',sig,...
    'c_L',c_L,'k_L',k_L,'r_0',r_0,'r_a',r_a,'r_b',r_b,'zlen',zlen,...
    'lambda_f',lambda_f,'lambda',lambda_s,'k1_rf',[],'k1_zf',[],...

'k2_rf',[],'k2_rL',[],'k2_zf',[],'k2_zL',[],'TL',[],'coef_mat',[],...
    'pcp',[],'pcm',[],'badfreq',[],'numpmode',nummode,'numlmode',0,...
    'numpint',nummode,'numlint',0,'showdebug',showdebug);
    datstruct = analyz(datstruct);
    datstruct = proc(datstruct);
else
    datstruct = analyz(datstruct);
    datstruct = proc(datstruct);
end

if showdebug == 1
    save simdat datstruct
end
if showdebug == 1
    save simdat datstruct
```

```
end
if x==1
    int_L = L; %length between silencers
    E0_new =
ones(1,datstruct.numharm).*exp(1i*int_L*datstruct.k1_zf(:,1).');
    F0_new =
datstruct.coef_mat(1,:).*exp(1i*int_L*datstruct.k1_zf(:,datstruct.nummo
de+1).');
    rho_f = datstruct.rho_f;
    c_f = datstruct.c_f;
    Z_f_new = (E0_new+F0_new).*(rho_f.*c_f)./(E0_new-F0_new);
    Edown=datstruct.coef_mat(datstruct.nummode*3+1,:);
    Fdown=datstruct.coef_mat(datstruct.nummode*4+1,:);
elseif x==2
    Aup=ones(1,datstruct.numharm);
    Bup=datstruct.coef_mat(1,:);
    Adown=datstruct.coef_mat(datstruct.nummode*3+1,:).*exp(-
1i*datstruct.k1_zf(:,1).'*int_L);
    Edown_new=Adown.*Edown;
    Fdown_new=Adown.*Fdown;
end
end %the for loop ends here
TL=20*log10(abs((Aup.^2-Fdown_new.^2)./(Aup.*Edown_new-
Bup.*Fdown_new)));

% figure;
% plot(freq,TL)
% title([num2str(Pc1./6894.75729),'    ' num2str(Pc2./6894.75729)])

% Time data
% Display how long the simulation took
endtime = now;
sec = (endtime-starttime)*60*60*24;
minu = floor(sec/60);
    sec = sec-minu*60;
hr = floor(minu/60);
    minu = minu-hr*60;
day = floor(hr/24);
    hr = hr-day*24;

timestr = 'is';
if day > 0
    timestr = [timestr ' ' num2str(day) ' days'];
end
if hr > 0
    timestr = [timestr ' ' num2str(hr) ' hours'];
end
if minu > 0
    timestr = [timestr ' ' num2str(minu) ' minutes'];
end
timestr = [timestr ' ' num2str(sec) ' seconds'];


disp(['Elapsed time for this simulation ' timestr '.'])
end
```

```
%%
%analyz
% by K Marek, revised May 2011
% Solves eigenfunctions and returns orthogonal wavenumbers
% This gives us the unique axial wavenumber for each mode.

function datstruct = analyz(datstruct)

global showdebug
numharm = datstruct.numharm;
nummode = datstruct.nummode;
k_f = datstruct.k_f;
rho_f = datstruct.rho_f;
r_0 = datstruct.r_0;
k_L = datstruct.k_L;
rho_s = datstruct.rho_s;
sig = datstruct.sig;
r_a = datstruct.r_a;
r_b = datstruct.r_b;
lambda_f = datstruct.lambda_f;
lambda = datstruct.lambda;
freqw = datstruct.freqw;



% Solution loop 1

% Use a Newton-Raphson method to solve for k_zf using initial value
% generated by robust root finder.  k1 for k outside silencer region.

disp('Begin root solving')
indcmax = 200;
fguess_all = 0.001;
deltaguess_all = 0.0001;

k1_zf = zeros(numharm,nummode*2); %1st half pos. travelling wave; 2nd
half neg.
k1_rf = k1_zf;

disp('Find initial pipe roots');
k1_zf(1,:) = findroots_nl([],nummode,r_0,k_f(1),freqw(1),...
    lambda_f(1),rho_f,fguess_all,deltaguess_all);
disp('Find subsequent pipe roots');

for inda = 2:numharm
    problem = 'none';
    for indb = 1:nummode*2
        indc = 1;
        convb = 0;

        guess = k1_zf(inda-1,indb);
        % Negative travelling wave might have negative wavenumber of
pos.
        if indb > nummode
            if abs(k1_zf(inda-1,indb) + k1_zf(inda-1,indb-nummode)) <
2*deltaguess_all
```

```
                guess = -k1_zf(inda,indb-nummode);
            end
        end

        [fguess,fprime] =
eigm(guess,freqw(inda),0,lambda_f(inda),0,rho_f,0,r_0,r_0);

        if(fguess == 0)
            convb = 1;
        end
        while(convb == 0)
            newguess = guess-fguess/fprime;
            deltaguess = abs(newguess-guess);

            guess = newguess;
            [fguess,fprime] =
eigm(guess,freqw(inda),0,lambda_f(inda),0,rho_f,0,r_0,r_0);

            if(indc > indcmax)
                if showdebug == 1
                disp(['Too many iterations, harmonic ' num2str(inda)
...
                    ', mode ' num2str(indb) ' in numeric 2-D ' ...
                    'compression wave solution, section 1']);
                end
                problem = 'iterations';
            end
            if((abs(fguess) < fguess_all) && (deltaguess <
deltaguess_all))
                convb = 1;
                for indd = 1:indb-1
                    if(abs(guess-k1_zf(inda,indd)) < 2*deltaguess_all)
                        if showdebug == 1
                        disp(['Duplicate solution found, harmonic ' ...
                            num2str(inda) ', modes ' num2str(indd) '
and '...
                            num2str(indb) ', section 1']);
                        end
                        problem = 'duplicates';
                    end
                end
            end
            indc = indc+1;
            if ~strcmp(problem,'none')
                convb = -1;
            end
        end
        if abs(imag(guess)) < deltaguess_all
            guess = real(guess);
        end
        k1_zf(inda,indb) = guess;
    end
    % Make sure negative and positives haven't switched up
    for indb = 1:nummode
        if (imag(k1_zf(inda,indb))) > 0 || ((imag(k1_zf(inda,indb)) ==
0) && real(k1_zf(inda,indb)) < 0)
```

```
            if abs(k1_zf(inda,indb)+k1_zf(inda,indb+nummode)) <
2*deltaguess_all
                k1_zf(inda,indb) = -k1_zf(inda,indb);
                k1_zf(inda,indb+nummode) = -k1_zf(inda,indb+nummode);
            else
                problem = 'signs1';
            end
        end
        if (imag(k1_zf(inda,indb+nummode)) < 0) || ...
                ((imag(k1_zf(inda,indb+nummode)) == 0) &&
(real(k1_zf(inda,indb+nummode)) > 0))
            problem = 'signs2';
        end
    end
    % Call robust root finder if problems were encountered
    if ~strcmp(problem,'none')
        disp(problem)
        disp(k1_zf(inda,:))
        disp('Trying findroots_nl.m')
        k1_zf(inda,:) =
findroots_nl(k1_zf(inda,:),nummode,r_0,k_f(inda),freqw(inda),lambda_f(i
nda),...
            rho_f,fguess_all,deltaguess_all);
    end
end

% now have k1_zf from above
for indb = 1:(2*nummode)
    k1_rf(:,indb) = sqrt(k_f.^2 - (k1_zf(:,indb)).'.^2);
end

% Solution loop 2

% Use a Newton-Raphson method to solve for k_zf using initial value
% generated by root finding function.  k2 for k inside silencer region.

indcmax = 200;
fguess_all = 0.001;
deltaguess_all = 0.0001;
maxnums = 500; %Related to e^maxnums becomes to hard to work with

k2_zf = zeros(numharm,nummode*2);
k2_rf = k2_zf;
k2_rL = k2_zf;

disp('Find initial silencer roots');
k2_zf(1,:) =
findroots([],nummode,r_a,r_b,k_f(1),k_L(1),freqw(1),lambda_f(1),...
    lambda(1),rho_f,rho_s,sig,fguess_all,deltaguess_all);
disp('Find subsequent silencer roots');

for inda = 2:numharm
    problem = 'none';
    for indb = 1:nummode*2
        indc = 1;
        convb = 0;

        guess = k2_zf(inda-1,indb);
        % Negative travelling wave might have negative wavenumber of
pos.
        if indb > nummode
            if abs(k2_zf(inda-1,indb) + k2_zf(inda-1,indb-nummode)) <
2*deltaguess_all
                guess = -k2_zf(inda,indb-nummode);
            end
        end

        [E,dE] =
eigm(guess,freqw(inda),lambda(inda),lambda_f(inda),rho_s,rho_f,sig,r_a,
r_b);
        if r_a == r_b
            fguess = E;
            fprime = dE;
            clear E dE %added
        else
            fguess = det(E);
            fprime = trace(adjugate2(E)*dE);
        end

        if(fguess == 0)
            convb = 1;
        end
        while(convb == 0)
            newguess = guess-fguess/fprime;
            deltaguess = abs(newguess-guess);

            oldfguess = fguess;
            guess = newguess;

            [E,dE] =
eigm(guess,freqw(inda),lambda(inda),lambda_f(inda),rho_s,rho_f,sig,r_a,
r_b);
            if r_a == r_b
                fguess = E;
                fprime = dE;
                clear E dE %add
            else
                fguess = det(E);
                fprime = trace(adjugate2(E)*dE);
            end

            if(indc > indcmax)
                if showdebug == 1
                disp(['Too many iterations, harmonic ' num2str(inda)
...
                    ', mode ' num2str(indb) ' in numeric 2-D ' ...
                    'compression wave solution, section 2']);
                disp(['guess = ' num2str(guess)]);
                disp(['fguess = ' num2str(oldfguess) ...
                    '; deltaguess = ' num2str(deltaguess)]);
                disp(['fprime = ' num2str(fprime)]);
                end
                problem = 'iterations';
            end
            if (abs(fguess) < (deltaguess_all/2*abs(fprime))) &&
(deltaguess < deltaguess_all)
                convb = 1;
                for indd = 1:indb-1
                    if(abs(guess-k2_zf(inda,indd)) < 2*deltaguess_all)
                        if showdebug == 1
                        disp(['Duplicate solution found, harmonic ' ...
                            num2str(inda) ', modes ' num2str(indd) '
and '...
                            num2str(indb) ', section 2']);
                        end
                        problem = 'duplicates';
                    end
                end
                if abs(real(guess)) > maxnums
                    problem = 'Too large';
                end
            end
            indc = indc+1;
            if ~strcmp(problem,'none')
                convb = -1;
            end
        end
        if abs(imag(guess)) < deltaguess_all
            guess = real(guess);
        end
        k2_zf(inda,indb) = guess;
    end
    % Make sure negative and positives haven't switched up
    for indb = 1:nummode
        if (imag(k2_zf(inda,indb))) > 0 || ((imag(k2_zf(inda,indb)) ==
0) && real(k2_zf(inda,indb)) < 0)
            if abs(k2_zf(inda,indb)+k2_zf(inda,indb+nummode)) <
2*deltaguess_all
                k2_zf(inda,indb) = -k2_zf(inda,indb);
                k2_zf(inda,indb+nummode) = -k2_zf(inda,indb+nummode);
            else
                problem = 'signs';
            end
        end
        if (imag(k2_zf(inda,indb+nummode))) < 0 || ...
                ((imag(k2_zf(inda,indb+nummode)) == 0) &&
real(k2_zf(inda,indb+nummode)) > 0)
            problem = 'signs';
        end
    end
    % Call robust root finder if problems were encountered
    if ~strcmp(problem,'none')
        disp(problem)
        disp('Trying findroots.m')
        k2_zf(inda,:) =
findroots(k2_zf(inda,:),nummode,r_a,r_b,k_f(inda),k_L(inda),freqw(inda)
,...
lambda_f(inda),lambda(inda),rho_f,rho_s,sig,fguess_all,deltaguess_all);
    end
end

% now have k2_zf from above
for indb = 1:(2*nummode)
    k2_rf(:,indb) = sqrt(k_f.^2 - (k2_zf(:,indb)).'.^2);
    k2_rL(:,indb) = sqrt(k_L.^2 - (k2_zf(:,indb)).'.^2);
end
k2_zL = k2_zf;

disp('Root finding complete')

clear inda indb indc indd

%% Put new data into datstruct

datstruct.k1_rf = k1_rf;
datstruct.k1_zf = k1_zf;

datstruct.k2_rf = k2_rf;
datstruct.k2_zf = k2_zf;
datstruct.k2_rL = k2_rL;
datstruct.k2_zL = k2_zL;

end

%%
% findroots
% by K Marek
%
% This root finder uses intelligently placed seed points to find a
given
% number of roots of the eigenfunction.  It then uses the argument
% principle to verify that all the roots in the given area have been
found.
% If problems occur, they are addressed automatically and the function
% tries again.


function froots = findroots(seeds,numroots,r_a,r_b,k_f,k_L,w,...
    lambdaf,lambda,rho_f,rho_s,sig,fguess_all,deltaguess_all)

if r_a == r_b %no liner case
    froots =
findroots_nl(seeds,numroots,r_a,k_f,w,lambdaf,rho_f,fguess_all,deltague
ss_all);
    clearvars -except freq Pc1 Pc2 Ps r_0 r_a0 r_b zlen L o n m TLmat
PcPlot PcVec PsVec unitset supNum overall starttime fileName total TL
froots
    return;
end
```

```
global showdebug

% Initialize variables
allgood = 0; % if all desired roots have been found
trynum = 1; % how many tries did it take?
problem = 'none';
lfactor = 1000; % how many segements per length to check phase
rmin = 0; rmax = 0; imin = 0; imax = 0;
slen = length(seeds);
deltaguess_all_orig = deltaguess_all;
dupnum = 1;
maxnums = 100; %Related to e^maxnums becomes to hard to work with
k_Tp = k_L; %unused
immin = 0;
immax = 0;

while allgood == 0
    % Initial point guesses, based on trynum and problem
    if trynum == 1
        %compression roots lie mostly on imaginary axis
        froots = zeros(1,numroots*8+6);
        froots(1:3) = [k_f k_Tp k_L]*1.01; %not quite on the value
because they are removable discontinuities
        for irt = 1:numroots
            k = max(k_L,k_f);
            k_r = (irt+0.5)*pi/(r_b+r_a*(rho_f/rho_s-1));
            guess = sqrt(k^2-k_r^2);
            if imag(guess) > 0
                guess = -guess;
            end
            froots(3+4*irt) = guess;
            if irt == 1
                froots(4:6) = (1:3)*guess/2;
            else
                froots((4*irt):(4*irt+2)) = froots(4*irt-1)+(1:3)*...
                    (guess-froots(4*irt-1))/2;
            end
        end
        froots((4*numroots+4):end) = -froots(1:(4*numroots+3));
        immin = min(imag(froots));
        immax = max(imag(froots));
    else
        if strcmp(problem,'too_few')
            % Keep converged roots, add some more to the end
            imin = min(imin,immin)*(1+(2*(trynum-1)/length(froots)));
            imax = max(imax,immax)*(1+(2*(trynum-1)/length(froots)));
            if (rmin == 0) || ~isfinite(rmin)
                rmin = (0.1+(trynum-1))*min(-real([k_f,k_L]));
            end
            if (rmax == 0) || ~isfinite(rmax)
                rmax = (0.1+(trynum-1))*max(real([k_f,k_L]));
            end
            if (imin == 0) || ~isfinite(imin)
                imin = (0.1+(trynum-1))*rmin;
            end
            if (imax == 0) || ~isfinite(imax)
```

```
                imax = (0.1+(trynum-1))*rmax;
            end
            problem = 'missed_some'; %fill in the gaps...
        end
        if strcmp(problem,'missed_some')
            % Choose new root guess locations
            % Be sure they will eventually cover the whole selection
            newroots = zeros(1,((trynum^2+trynum+1)*length(froots)));
            newroots(1:length(froots)) = froots;
            irt = length(froots)+1;
            for i_im = 1:trynum*length(froots)
                ival = imin + (imax-
imin)*i_im/(trynum*length(froots)+1);
                for i_re = 1:(trynum+1)
                    rval = rmin + (rmax-rmin)*i_re/(trynum+2);
                    newroots(irt) = rval+1i*ival;
                    irt = irt+1;
                end
            end
            froots = newroots;
        end
    end

    problem = 'none'; %reset problem flag

    % Find roots using gradient method from all initial points
    % Use a Newton-Raphson method to solve for k_zf using initial guess
    % generated by some method above.  k2 for k inside silencer region.

    indcmax = 200;

    for indb = 1:length(froots)
        indc = 1;
        convb = 0;

        guess = froots(indb);

        [E,dE] = eigm(guess,w,lambda,lambdaf,rho_s,rho_f,sig,r_a,r_b);
        fprime = trace(adjugate2(E)*dE);
        fguess = det(E);
        if(fguess == 0)
            convb = 1;
        end
        while(convb == 0)
            newguess = guess-fguess/fprime;
            deltaguess = abs(newguess-guess);

            guess = newguess;

            [E,dE] =
eigm(guess,w,lambda,lambdaf,rho_s,rho_f,sig,r_a,r_b);
            fguess = det(E);
            fprime = trace(adjugate2(E)*dE);

            if(indc > indcmax)
```

```
                convb = -1;
                problem = 'iterations';
            end
            if((abs(fguess) < (deltaguess_all/2*abs(fprime))) &&
(deltaguess < deltaguess_all))
                convb = 1;
            end
            indc = indc+1;
        end
        froots(indb) = guess;
        if convb == -1
            froots(indb) = -1i*inf;
            problem = 'none';
        end
    end

    % Delete non-converged roots
    inda = 0;
    newroots = zeros(size(froots));
    for indb = 1:length(froots)
       if isfinite(froots(indb))
           newroots(indb-inda) = froots(indb);
       else
           inda = inda+1;
       end
    end
    froots = newroots(1:length(froots)-inda);
    % Delete roots thought to be numerical artifacts
    inda = 0;
    newroots = zeros(size(froots));
    for indb = 1:length(froots)
        if (abs(real(froots(indb))) <
max(abs(real([k_f,k_L,k_Tp,min(imag(froots)),max(imag(froots))])))) 
&&...
                (max(abs(imag([sqrt(k_f^2-froots(indb)^2),sqrt(k_L^2-
froots(indb)^2)]))) < maxnums) &&...
                (abs(real(froots(indb))) < maxnums)
            newroots(indb-inda) = froots(indb);
        else
            inda = inda+1;
        end
    end
    froots = newroots(1:length(froots)-inda);

    % Sort roots by type
    rpnum = 0;
    rmnum = 0;
    cpnum = 0;
    cmnum = 0;
    rproots = zeros(1,length(froots));
    rmroots = rproots;
    cproots = rproots;
    cmroots = rproots;
    for indb = 1:length(froots)
        if abs(imag(froots(indb))) < deltaguess_all
            if real(froots(indb)) > 0
                rpnum = rpnum+1;
                rproots(rpnum) = real(froots(indb));
            else
                rmnum = rmnum+1;
                rmroots(rmnum) = real(froots(indb));
            end
        else
            if imag(froots(indb)) < 0
                cpnum = cpnum+1;
                cproots(cpnum) = froots(indb);
            else
                cmnum = cmnum+1;
                cmroots(cmnum) = froots(indb);
            end
        end
    end
    rproots = rproots(1:rpnum);
    rmroots = rmroots(1:rmnum);
    cproots = cproots(1:cpnum);
    cmroots = cmroots(1:cmnum);

    % Sort each type of root
    newroots = zeros(1,rpnum);
    for indb = 1:rpnum
        [value,index] = max(real(rproots));
        newroots(indb) = rproots(index);
        rproots(index) = -inf;
    end
    rproots = newroots;
    newroots = zeros(1,rmnum);
    for indb = 1:rmnum
        [value,index] = min(real(rmroots));
        newroots(indb) = rmroots(index);
        rmroots(index) = inf;
    end
    rmroots = newroots;
    newroots = zeros(1,cpnum);
    for indb = 1:cpnum
        [value,index] = max(imag(cproots));
        newroots(indb) = cproots(index);
        cproots(index) = -1i*inf;
    end
    cproots = newroots;
    newroots = zeros(1,cmnum);
    for indb = 1:cmnum
        [value,index] = min(imag(cmroots));
        newroots(indb) = cmroots(index);
        cmroots(index) = 1i*inf;
    end
    cmroots = newroots;

    % Eliminate duplicate points
    if rpnum > 1
        dcount = 0;
        newroots = rproots;
        for indb = 2:rpnum
```

```
            found_duplicate = 0;
            indb1 = 1;
            while (indb1 < indb) && (found_duplicate == 0)
                if abs(rproots(indb)-rproots(indb1)) <
(2*dupnum*deltaguess_all)
                    dcount = dcount+1;
                    found_duplicate = 1;
                end
                indb1 = indb1+1;
            end
            if found_duplicate == 0
                newroots(indb-dcount)=rproots(indb);
            end
        end
        rproots=newroots(1:(rpnum-dcount));
        rpnum = rpnum-dcount;
    end
    if rmnum > 1
        dcount = 0;
        newroots = rmroots;
        for indb = 2:rmnum
            found_duplicate = 0;
            indb1 = 1;
            while (indb1 < indb) && (found_duplicate == 0)
                if abs(rmroots(indb)-rmroots(indb1)) <
(2*dupnum*deltaguess_all)
                    dcount = dcount+1;
                    found_duplicate = 1;
                end
                indb1 = indb1+1;
            end
            if found_duplicate == 0
                newroots(indb-dcount)=rmroots(indb);
            end
        end
        rmroots=newroots(1:(rmnum-dcount));
        rmnum = rmnum-dcount;
    end
    if cpnum > 1
        dcount = 0;
        newroots = cproots;
        for indb = 2:cpnum
            found_duplicate = 0;
            indb1 = 1;
            while (indb1 < indb) && (found_duplicate == 0)
                if abs(cproots(indb)-cproots(indb1)) <
(2*dupnum*deltaguess_all)
                    dcount = dcount+1;
                    found_duplicate = 1;
                end
                indb1 = indb1+1;
            end
            if found_duplicate == 0
                newroots(indb-dcount)=cproots(indb);
            end
        end
        cproots=newroots(1:(cpnum-dcount));
        cpnum = cpnum-dcount;
    end
    if cmnum > 1
        dcount = 0;
        newroots = cmroots;
        for indb = 2:cmnum
            found_duplicate = 0;
            indb1 = 1;
            while (indb1 < indb) && (found_duplicate == 0)
                if abs(cmroots(indb)-cmroots(indb1)) <
(2*dupnum*deltaguess_all)
                    dcount = dcount+1;
                    found_duplicate = 1;
                end
                indb1 = indb1+1;
            end
            if found_duplicate == 0
                newroots(indb-dcount)=cmroots(indb);
            end
        end
        cmroots=newroots(1:(cmnum-dcount));
        cmnum = cmnum-dcount;
    end

    % Check that enough roots were found (problem = 'too_few')
    if min(rpnum+cpnum,rmnum+cmnum) < numroots
        problem = 'too_few';
        froots = [rproots cproots rmroots cmroots];
        imax = 1.1*max(imag([froots,-k_f,-k_L,-k_Tp,0.05i]));
        imin = 1.1*min(imag([froots,k_f,k_L,k_Tp,-0.05i]));
        rmax = min([1.1*max(real([froots,k_f,k_L,k_Tp])),maxnums]);
        rmin = max([1.1*min(real([froots,-k_f,-k_L,-k_Tp])),-maxnums]);
        numrootsp = rpnum+cpnum;
        numrootsm = rmnum+cmnum;
        proots = [rproots cproots];
        mroots = [rmroots cmroots];
    else
        proots = [rproots cproots];
        mroots = [rmroots cmroots];
        if min(imag(proots)) == 0
            imin = -1;
            numrootsp = numroots;
        else if length(proots) == numroots
                if numroots == 1
                    imin = 1.1*min([imag(proots(numroots)),-.05]);
                    numrootsp = numroots;
                else
                    imin = imag(proots(numroots)) + ...
                        0.1*(imag(proots(numroots))-
imag(proots(numroots-1)));
                    if abs(imag(proots(numroots)-proots(numroots-1))) <
5*deltaguess_all
                        imin = imag(proots(numroots))*1.1;
                    end
                    numrootsp = numroots;
                end
            else
```

```
                numrootsp = numroots;
                pdone = 0;
                for indd = (numroots+1):length(proots)
                    if (imag(proots(indd)) >
1.01*imag(proots(numrootsp)))
                        numrootsp = indd;
                        imin = 1.02*imag(proots(indd));
                    else
                        if (pdone == 0)
                            imin = imag(proots(numrootsp)) +
0.1*(imag(proots(indd)) - ...
                                imag(proots(indd-1)));
                            pdone = 1;
                        end
                    end
                end
            end
        end
        if max(imag(mroots)) == 0
            imax = 1;
            numrootsm = numroots;
        else if length(mroots) == numroots
                if numroots == 1
                    imax = 1.1*imag(mroots(numroots));
                    numrootsm = numroots;
                else
                    imax = imag(mroots(numroots)) + ...
                        0.1*(imag(mroots(numroots))-
imag(mroots(numroots-1)));
                    if abs(imag(mroots(numroots)-mroots(numroots-1))) <
5*deltaguess_all
                        imax = imag(mroots(numroots))*1.1;
                    end
                    numrootsm = numroots;
                end
            else
                numrootsm = numroots;
                mdone = 0;
                for indd = (numroots+1):length(mroots)
                    if (imag(mroots(indd)) <
1.01*imag(mroots(numroots)))
                        numrootsm = indd;
                        imax = 1.02*imag(mroots(indd));
                    else
                        if (mdone == 0)
                            imax = imag(mroots(numrootsm)) +
0.1*(imag(mroots(indd)) - ...
                                imag(mroots(indd-1)));
                            mdone = 1;
                        end
                    end
                end
            end
        end
        froots = [proots(1:numrootsp) mroots(1:numrootsm)];
        rmax = min([1.1*max(real([froots,k_f,k_L,k_Tp])),maxnums]);
        rmin = max([1.1*min(real([froots,-k_f,-k_L,-k_Tp])),-maxnums]);
```

```
    end

    if max(abs([imax,imin])) > 5000
        imax = 5000;
        imin = -5000;
        problem = 'missed_some';
    end

    % Use argument principle to determine if all contained roots were
found
    if strcmp(problem,'none') || strcmp(problem,'too_few') %continue if
no problems so far
        legnum = 1; %four legs to traverse entire rectangle
        % Leg 1: [rmin imax] to [rmin imin]
        % Leg 2: [rmin imin] to [rmax imin]
        % Leg 3: [rmax imin] to [rmax imax]
        % Leg 4: [rmax imax] to [rmin imax]
        prevpoint = [rmin imax];
        guess = prevpoint(1) + 1i*prevpoint(2);
        init_arg =
180/pi*angle(det(eigm2(guess,w,lambda,lambdaf,rho_s,rho_f,sig,r_a,r_b))
);
        prev_arg = init_arg;
        numturns = 0; %number of loops the argument makes about origin
        dpoint = [0 1/lfactor*(imin-imax)]*1/numroots;
        dpointtemp = dpoint;
        dptcount = 0;
        finished = 0;
        darg_all = 5;
        while finished == 0
            % Ensure each point doesn't vary too much in argument
            goodpoint = 0;
            while goodpoint == 0
                if dptcount > 0
                    thispoint = prevpoint+dpointtemp;
                    dptcount = dptcount-1;
                else
                    thispoint = prevpoint+dpoint;
                    dpointtemp = dpoint;
                end
                if (legnum == 1) && (thispoint(2) < imin)
                    thispoint(2) = imin;
                end
                if (legnum == 2) && (thispoint(1) > rmax)
                    thispoint(1) = rmax;
                end
                if (legnum == 3) && (thispoint(2) > imax)
                    thispoint(2) = imax;
                end
                if (legnum == 4) && (thispoint(1) < rmin)
                    thispoint(1) = rmin;
                end
                guess = thispoint(1) + 1i*thispoint(2);
                point_arg =
180/pi*angle(det(eigm2(guess,w,lambda,lambdaf,rho_s,rho_f,sig,r_a,r_b))
);
```

```
                if (abs(point_arg - prev_arg) < darg_all) || ...
                        ((abs(point_arg - prev_arg) < (darg_all + 360))
&& ...
                        (abs(point_arg - prev_arg) > (360 - darg_all)))
                    goodpoint = 1;
                else
                    if dptcount == 0
                        dptcount = 10;
                        dpointtemp = dpoint/10;
                    else
                        dptcount = 10*dptcount;
                        dpointtemp = dpointtemp/10;
                    end
                end
                if (goodpoint == 0) && (dptcount > 10^7)
                    problem = 'Duplicates/check';
                    dupnum = dupnum+2;
                    lfactor = lfactor*1.5;
                    if showdebug == 1
                    disp('Discontinuous check path in findoots.m')
                    disp(['At ' num2str(thispoint) ', leg '
num2str(legnum)]);
                    end
                    goodpoint = 1;
                    finished = 1;
                end
            end
            % Find the change in angle, record
            if abs(point_arg - prev_arg) < darg_all
                numturns = numturns + (point_arg - prev_arg)/360;
            else
                if abs(point_arg - prev_arg + 360) < darg_all
                    numturns = numturns + (point_arg - prev_arg +
360)/360;
                else
                    if abs(point_arg - prev_arg - 360) < darg_all
                        numturns = numturns + (point_arg - prev_arg -
360)/360;
                    else
                        if showdebug == 1
                        disp('AAAA!!!! Terrible error in
findroots!!!!')
                        end
                    end
                end
            end

            % Change the leg number if appropriate
            if (thispoint(1) == rmin) && (thispoint(2) == imin)
                legnum = 2;
                dpoint = [1/lfactor*(rmax-rmin) 0]*1/numroots;
                if dptcount > 0
                    dpointtemp = dpoint/dptcount;
                end
            end
            if (thispoint(1) == rmax) && (thispoint(2) == imin)
                legnum = 3;
                dpoint = [0 1/lfactor*(imax-imin)]*1/numroots;
                if dptcount > 0
                    dpointtemp = dpoint/dptcount;
                end
            end
            if (thispoint(1) == rmax) && (thispoint(2) == imax)
                legnum = 4;
                dpoint = [1/lfactor*(rmin-rmax) 0]*1/numroots;
                if dptcount > 0
                    dpointtemp = dpoint/dptcount;
                end
            end
            % Check if we're back to the starting point
            if (thispoint(1) == rmin) && (thispoint(2) == imax)
                finished = 1;
            end
            prevpoint = thispoint;
            prev_arg = point_arg;
        end
        if strcmp(problem,'none') && (abs(numturns + 0 - numrootsp -
numrootsm) > 0.05) %numturns + numpoles - numzeros = 0
            if (numturns + 0 - numrootsp - numrootsm) > 0.05
                problem = 'missed_some';
                dupnum = 1;
            else
                if (numrootsp+numrootsm) < (2*numroots)
                    problem = 'too_few';
                else
                    problem = 'Duplicates/check';
                    dupnum = dupnum+2;
                    lfactor = lfactor*1.5;
                end
            end
        end
    end

    if strcmp(problem,'none')
        allgood = 1;
        froots = [proots(1:numroots) mroots(1:numroots)];
    else
        % Return error if too many tries.
        if trynum > 20
            disp('Too many tries in findroots!')
            disp('Exiting on error.')
            allgood = -1;
            froots = [];
        end
        if showdebug == 1
        disp(problem)
        end
        trynum = trynum + 1;
    end
end
% Roots found!
if allgood == 1
```

```
    disp(['Roots found in ' num2str(trynum) ' tries!'])
end
end

%%
% findroots_nl
% by K Marek
%
% This root finder uses intelligently placed seed points to find a
given
% number of roots of the eigenfunction.  It then uses the argument
% principle to verify that all the roots in the given area have been
found.
% If problems occur, they are addressed automatically and the function
% tries again.  This version finds roots when no liner is present (test
% section or inlet/outlet pipes); r_0 is the outer pipe diameter in any
% case.


function froots = findroots_nl(seeds,numroots,r_0,k_f,w,lambda_f,rho_f,
...
    fguess_all,deltaguess_all)

% global showdebug

% Initialize variables
allgood = 0; % if all desired roots have been found
trynum = 1; % how many tries did it take?
problem = 'none';
lfactor = 1000; % how many segements per length to check phase
rmin = 0; rmax = 0; imin = 0; imax = 0;
slen = length(seeds);
froots = zeros(1,numroots*8+2+slen);
deltaguess_all_orig = deltaguess_all;

while allgood == 0
    % Initial point guesses, based on trynum and problem
    if trynum == 1
        froots(1) = k_f;
        for irt = 1:numroots
            k_rf = (irt+0.5)*pi/r_0;
            guess = 1i*imag(sqrt(k_f^2-k_rf^2));
            if imag(guess) > 0
                guess = -guess;
            end
            froots(1+4*irt) = guess;
            if irt == 1
                froots(2:4) = (1:3)*guess/4;
            else
                froots((4*irt-2):(4*irt)) = froots(4*irt-3)+(1:3)*...
                    (guess-froots(4*irt-3))/4;
            end
        end
        froots(4*numroots+2:(end-slen)) = -froots(1:4*numroots+1);
        froots((end-slen+1):end) = seeds;
    else
```

```
        if strcmp(problem,'missed_some')
            % Choose new root guess locations
            % Be sure they will eventually cover the whole selection
            newroots = zeros(1,((trynum^2+trynum+1)*length(froots)));
            newroots(1:length(froots)) = froots;
            irt = length(froots)+1;
            for i_im = 1:trynum*length(froots)
                ival = imin + (imax-
imin)*i_im/(trynum*length(froots)+1);
                newroots(irt) = 1i*ival;
                for i_re = 1:trynum
                    rval = rmin + (rmax-rmin)*i_re/(trynum+1);
                    newroots(irt) = rval+1i*ival;
                    irt = irt+1;
                end
            end
            froots = newroots;
        end
        if strcmp(problem,'too_few')
            % Keep converged roots, add some more to the end
            newroots = [froots zeros(1,8*trynum)];
            newroots(length(froots)+(1:(4*trynum))) = ...
                (1i*imax*(1+(1:(4*trynum))/(1+length(froots)/2)));
            newroots(length(froots)+4*trynum+(1:(4*trynum))) = -
newroots(length(froots)+(1:(4*trynum)));
            froots = newroots;
        end
    end

    problem = 'none'; %reset problem flag

    % Find roots using gradient method from all initial points
    % Use a Newton-Raphson method to solve for k_zf using initial guess
    % generated by some method above.  k2 for k inside silencer region.

    indcmax = 200;
    if trynum > 5
        deltaguess_all = deltaguess_all_orig*5/trynum;
    end

    for indb = 1:length(froots)
        indc = 1;
        convb = 0;

        guess = froots(indb);

        [fguess,fprime] = eigm(guess,w,0,lambda_f,0,rho_f,0,r_0,r_0);

        if(fguess == 0)
            convb = 1;
        end
        while(convb == 0)
            newguess = guess-fguess/fprime;
            deltaguess = abs(newguess-guess);
            guess = newguess;
            [fguess,fprime] =
eigm(guess,w,0,lambda_f,0,rho_f,0,r_0,r_0);
            if(indc > indcmax)
                convb = -1;
                problem = 'iterations';
            end
            if((abs(fguess) < (deltaguess_all/2*abs(fprime))) &&
(deltaguess < deltaguess_all))
                convb = 1;
            end
            indc = indc+1;
        end
        froots(indb) = guess;
        if convb == -1
            froots(indb) = -1i*inf;
            problem = 'none';
        end
    end

    % Delete non-converged roots
    inda = 0;
    newroots = zeros(size(froots));
    for indb = 1:length(froots)
       if isfinite(froots(indb))
           newroots(indb-inda) = froots(indb);
       else
           inda = inda+1;
       end
    end
    froots = newroots(1:length(froots)-inda);

    % Sort roots by type
    rpnum = 0;
    rmnum = 0;
    cpnum = 0;
    cmnum = 0;
    rproots = zeros(1,length(froots));
    rmroots = rproots;
    cproots = rproots;
    cmroots = rproots;
    for indb = 1:length(froots)
        if abs(imag(froots(indb))) < deltaguess_all
            if real(froots(indb)) > 0
                rpnum = rpnum+1;
                rproots(rpnum) = real(froots(indb));
            else
                rmnum = rmnum+1;
                rmroots(rmnum) = real(froots(indb));
            end
        else
            if imag(froots(indb)) < 0
                cpnum = cpnum+1;
                cproots(cpnum) = froots(indb);
            else
                cmnum = cmnum+1;
                cmroots(cmnum) = froots(indb);
            end
        end
    end
    rproots = rproots(1:rpnum);
    rmroots = rmroots(1:rmnum);
    cproots = cproots(1:cpnum);
    cmroots = cmroots(1:cmnum);

    % Sort each type of root
    newroots = zeros(1,rpnum);
    for indb = 1:rpnum
        [value,index] = max(real(rproots));
        newroots(indb) = rproots(index);
        rproots(index) = -inf;
    end
    rproots = newroots;
    newroots = zeros(1,rmnum);
    for indb = 1:rmnum
        [value,index] = min(real(rmroots));
        newroots(indb) = rmroots(index);
        rmroots(index) = inf;
    end
    rmroots = newroots;
    newroots = zeros(1,cpnum);
    for indb = 1:cpnum
        [value,index] = max(imag(cproots));
        newroots(indb) = cproots(index);
        cproots(index) = -1i*inf;
    end
    cproots = newroots;
    newroots = zeros(1,cmnum);
    for indb = 1:cmnum
        [value,index] = min(imag(cmroots));
        newroots(indb) = cmroots(index);
        cmroots(index) = 1i*inf;
    end
    cmroots = newroots;

    % Eliminate duplicate points
    if rpnum > 1
        dcount = 0;
        newroots = rproots;
        for indb = 2:rpnum
            found_duplicate = 0;
            indb1 = 1;
            while (indb1 < indb) && (found_duplicate == 0)
                if abs(rproots(indb)-rproots(indb1)) <
(2*deltaguess_all)
                    dcount = dcount+1;
                    found_duplicate = 1;
                end
                indb1 = indb1+1;
            end
            if found_duplicate == 0
                newroots(indb-dcount)=rproots(indb);
            end
        end
        rproots=newroots(1:(rpnum-dcount));
        rpnum = rpnum-dcount;
    end
    if rmnum > 1
        dcount = 0;
        newroots = rmroots;
        for indb = 2:rmnum
            found_duplicate = 0;
            indb1 = 1;
            while (indb1 < indb) && (found_duplicate == 0)
                if abs(rmroots(indb)-rmroots(indb1)) <
(2*deltaguess_all)
                    dcount = dcount+1;
                    found_duplicate = 1;
                end
                indb1 = indb1+1;
            end
            if found_duplicate == 0
                newroots(indb-dcount)=rmroots(indb);
            end
        end
        rmroots=newroots(1:(rmnum-dcount));
        rmnum = rmnum-dcount;
    end
    if cpnum > 1
        dcount = 0;
        newroots = cproots;
        for indb = 2:cpnum
            found_duplicate = 0;
            indb1 = 1;
            while (indb1 < indb) && (found_duplicate == 0)
                if abs(cproots(indb)-cproots(indb1)) <
(2*deltaguess_all)
                    dcount = dcount+1;
                    found_duplicate = 1;
                end
                indb1 = indb1+1;
            end
            if found_duplicate == 0
                newroots(indb-dcount)=cproots(indb);
            end
        end
        cproots=newroots(1:(cpnum-dcount));
        cpnum = cpnum-dcount;
    end
    if cmnum > 1
        dcount = 0;
        newroots = cmroots;
        for indb = 2:cmnum
            found_duplicate = 0;
            indb1 = 1;
            while (indb1 < indb) && (found_duplicate == 0)
                if abs(cmroots(indb)-cmroots(indb1)) <
(2*deltaguess_all)
                    dcount = dcount+1;
                    found_duplicate = 1;
                end
                indb1 = indb1+1;
            end
            if found_duplicate == 0
                newroots(indb-dcount)=cmroots(indb);
            end
        end
        cmroots=newroots(1:(cmnum-dcount));
        cmnum = cmnum-dcount;
    end

    % Check that enough roots were found (problem = 'too_few')
    if min(rpnum+cpnum,rmnum+cmnum) < numroots
        problem = 'too_few';
        froots = [rproots cproots rmroots cmroots];
        imax = 1.1*max(imag([froots,-k_f,0.05i]));
        imin = 1.1*min(imag([froots,k_f,-0.05i]));
        rmax = min(1.1*max(real([froots,k_f])));
        rmin = max(1.1*min(real([froots,-k_f])));
        numrootsp = rpnum+cpnum;
        numrootsm = rmnum+cmnum;
    else
        proots = [rproots cproots];
        mroots = [rmroots cmroots];
        if min(imag(proots)) == 0
            imin = -1;
            numrootsp = numroots;
        else if length(proots) == numroots
                if numroots == 1
                    imin = 1.1*min([imag(proots(numroots)),-.05]);
                    numrootsp = numroots;
                else
                    imin = imag(proots(numroots)) + ...
                        0.1*(imag(proots(numroots))-
imag(proots(numroots-1)));
                    numrootsp = numroots;
                end
            else
                numrootsp = numroots;
                pdone = 0;
                for indd = (numroots+1):length(proots)
                    if (imag(proots(indd)) >
1.01*imag(proots(numrootsp)))
                        numrootsp = indd;
                        imin = 1.02*imag(proots(indd));
                    else
                        if (pdone == 0)
                            imin = imag(proots(numrootsp)) +
0.1*(imag(proots(indd)) - ...
                                imag(proots(indd-1)));
                            pdone = 1;
                        end
                    end
                end
            end
        end
        if max(imag(mroots)) == 0
            imax = 1;
            numrootsm = numroots;
        else if length(mroots) == numroots
                if numroots == 1
                    imax = 1.1*imag(mroots(numroots));
                    numrootsm = numroots;
                else
                    imax = imag(mroots(numroots)) + ...
                        0.1*(imag(mroots(numroots))-
imag(mroots(numroots-1)));
                    numrootsm = numroots;
                end
            else
                numrootsm = numroots;
                mdone = 0;
                for indd = (numroots+1):length(mroots)
                    if (imag(mroots(indd)) <
1.01*imag(mroots(numroots)))
                        numrootsm = indd;
                        imax = 1.02*imag(mroots(indd));
                    else
                        if (mdone == 0)
                            imax = imag(mroots(numrootsm)) +
0.1*(imag(mroots(indd)) - ...
                                imag(mroots(indd-1)));
                            mdone = 1;
                        end
                    end
                end
            end
        end
        froots = [proots(1:numrootsp) mroots(1:numrootsm)];
    end

    rmax = 1.1*max(real([froots,k_f]));
    rmin = 1.1*min(real([froots,-k_f]));
    % Use argument principle to determine if all contained roots were
found
    if strcmp(problem,'none') || strcmp(problem,'too_few') %continue if
no problems so far
        legnum = 1; %four legs to traverse entire rectangle
        % Leg 1: [rmin imax] to [rmin imin]
        % Leg 2: [rmin imin] to [rmax imin]
        % Leg 3: [rmax imin] to [rmax imax]
        % Leg 4: [rmax imax] to [rmin imax]
        prevpoint = [rmin imax];
        guess = prevpoint(1) + 1i*prevpoint(2);
        init_arg =
180/pi*angle(eigm2(guess,w,0,lambda_f,0,rho_f,0,r_0,r_0));
        prev_arg = init_arg;
        numturns = 0; %number of loops the argument makes about origin
        dpoint = [0 1/lfactor*(imin-imax)]*1/numroots;
        dpointtemp = dpoint;
        dptcount = 0;
        finished = 0;
        darg_all = 5;
        while finished == 0
            % Ensure each point doesn't vary too much in argument
            goodpoint = 0;
            while goodpoint == 0
                if dptcount > 0
                    thispoint = prevpoint+dpointtemp;
                    dptcount = dptcount-1;
                else
                    thispoint = prevpoint+dpoint;
                    dpointtemp = dpoint;
                end
                if (legnum == 1) && (thispoint(2) < imin)
                    thispoint(2) = imin;
                end
                if (legnum == 2) && (thispoint(1) > rmax)
                    thispoint(1) = rmax;
                end
                if (legnum == 3) && (thispoint(2) > imax)
                    thispoint(2) = imax;
                end
                if (legnum == 4) && (thispoint(1) < rmin)
                    thispoint(1) = rmin;
                end
                guess = thispoint(1) + 1i*thispoint(2);
                point_arg =
180/pi*angle(eigm2(guess,w,0,lambda_f,0,rho_f,0,r_0,r_0));

                if (abs(point_arg - prev_arg) < darg_all) || ...
                        ((abs(point_arg - prev_arg) < (darg_all + 360))
&& ...

                        (abs(point_arg - prev_arg) > (360 - darg_all)))
                    goodpoint = 1;
                else
                    if dptcount == 0
                        dptcount = 10;
                        dpointtemp = dpoint/10;
                    else
                        dptcount = 10*dptcount;
                        dpointtemp = dpointtemp/10;
                    end
                end
                if (goodpoint == 0) && (dptcount > 10^7)
                    problem = 'missed_some';
                    if showdebug == 1
                    disp('Discontinuous check path in findoots.m')
                    disp(['At ' num2str(thispoint) ', leg '
num2str(legnum)]);
                    end
                    goodpoint = 1;
                    finished = 1;
                end
            end
            % Find the change in angle, record
            if abs(point_arg - prev_arg) < darg_all
                numturns = numturns + (point_arg - prev_arg)/360;
            else
                if abs(point_arg - prev_arg + 360) < darg_all
```

```
                    numturns = numturns + (point_arg - prev_arg +
360)/360;
                else
                if abs(point_arg - prev_arg - 360) < darg_all
                    numturns = numturns + (point_arg - prev_arg -
360)/360;
                else
                    if showdebug == 1
                    disp('AAAA!!!! Terrible error in
findroots_nl.m!!!!')
                    end
                end
                end
            end

            % Change the leg number if appropriate
            if (thispoint(1) == rmin) && (thispoint(2) == imin)
                legnum = 2;
                dpoint = [1/lfactor*(rmax-rmin) 0]*1/numroots;
                if dptcount > 0
                    dpointtemp = dpoint/dptcount;
                end
            end
            if (thispoint(1) == rmax) && (thispoint(2) == imin)
                legnum = 3;
                dpoint = [0 1/lfactor*(imax-imin)]*1/numroots;
                if dptcount > 0
                    dpointtemp = dpoint/dptcount;
                end
            end
            if (thispoint(1) == rmax) && (thispoint(2) == imax)
                legnum = 4;
                dpoint = [1/lfactor*(rmin-rmax) 0]*1/numroots;
                if dptcount > 0
                    dpointtemp = dpoint/dptcount;
                end
            end
            % Check if we're back to the starting point
            if (thispoint(1) == rmin) && (thispoint(2) == imax)
                finished = 1;
            end
            prevpoint = thispoint;
            prev_arg = point_arg;
        end
        if strcmp(problem,'none') && (abs(numturns + 0 - numrootsp -
numrootsm) > 0.05) %numturns + numpoles - numzeros = 0
            if (numturns + 0 - numrootsp - numrootsm) > 0.05
                problem = 'missed_some';
            else
                problem = 'too_few';
            end
        end
    end

    if strcmp(problem,'none')
        allgood = 1;
        froots = [proots(1:numroots) mroots(1:numroots)];
    else
        % Return error if too many tries.
        if trynum > 20
            disp('Too many tries in findroots_nl!')
            disp('Exiting on error.')
            allgood = -1;
            froots = [];
        end
        if showdebug == 1
        disp(problem)
        disp(froots)
        end
        trynum = trynum + 1;
    end
end
% Roots found!
if allgood == 1
    disp(['Roots found in ' num2str(trynum) ' tries!'])
end
clearvars -except freq Pc1 Pc2 Ps r_0 r_a0 r_b zlen L o n m TLmat
PcPlot PcVec PsVec unitset supNum overall starttime fileName total TL
froots
end


%%
%eigm
% by K Marek
% June 2011
%
% Provides the matrix whose determinant is the eigenfunction.  Also
% provides the matrix where each element is the derivative wrt/kx of
the
% corresponding 'eigenmatrix' elements.  This should save processing
time
% over calculating it all more than once.

function [E,dE] = eigm(kz,w,lambda,lambdaf,rho_s,rho_f,sig,r_a,r_b)

choice1 = 2; %2 for compressive liner, 3 for no liner

%test for no liner condition (also for up/down-stream pipe section)
if r_a == r_b
    choice1 = 3;
    clear E dE
end

% Derive needed quantities
c_L = sqrt(lambda/rho_s);
c_f = sqrt(lambdaf/rho_f);
kL = w/c_L;
kf = w/c_f;
krL = sqrt(kL^2-kz^2);
krf = sqrt(kf^2-kz^2);

%Bessel functions are re-used enough that it's probably worth just
```

```
%calculating them once.
j0fa = besselj(0,krf*r_a);
j1fa = besselj(1,krf*r_a);

if choice1 < 3
    j0La = besselj(0,krL*r_a);
    j1La = besselj(1,krL*r_a);
    y0La = bessely(0,krL*r_a);
    y1La = bessely(1,krL*r_a);
    j0Lb = besselj(0,krL*r_b);
    j1Lb = besselj(1,krL*r_b);
    y0Lb = bessely(0,krL*r_b);
    y1Lb = bessely(1,krL*r_b);
end

if choice1 == 2 %just compression waves
    % Populate matrix
    E = zeros(3);
    %row 1: continuity of radial displacement at r=a
    E(1,1) = krf/w*j1fa;
    E(1,2) = -krL/w*j1La;
    E(1,3) = -krL/w*y1La;
    %row 2: continuity of stress/pressure at r=a (add limp mass sheet)
    E(2,1) = kf^2/w^2*lambdaf*j0fa - krf*sig*j1fa;
    E(2,2) = -kL^2/w^2*lambda*j0La;
    E(2,3) = -kL^2/w^2*lambda*y0La;
    %row 3: zero radial displacement at r=b
    E(3,2) = -krL/w*j1Lb;
    E(3,3) = -krL/w*y1Lb;


    %Populate derivative matrix
    dE = zeros(3);
    %row 1
    dE(1,1) = 1/w*-kz*r_a*j0fa;
    dE(1,2) = 1/w*kz*r_a*j0La;
    dE(1,3) = 1/w*kz*r_a*y0La;
    %row 2
    dE(2,1) = (1/w^2)*lambdaf*kf^2*kz*r_a/krf*j1fa + sig*kz*r_a*j0fa;
    dE(2,2) = (1/w^2)*(-lambda*kL^2*kz*r_a/krL*j1La);
    dE(2,3) = (1/w^2)*(-lambda*kL^2*kz*r_a/krL*y1La);
    %row 3
    dE(3,2) = 1/w*kz*r_b*j0Lb;
    dE(3,3) = 1/w*kz*r_b*y0Lb;
end

if choice1 == 3 %no liner
    % Populate matrix
    E = zeros(1);
    %row 1: continuity of radial displacement at r=a
    E(1,1) = krf/w*j1fa;

    %Populate derivative matrix
    dE = zeros(1);
    %row 1
    dE(1,1) = 1/w*-kz*r_a*j0fa;
```

```
end

end

%%
%eigm2
% by K Marek
% June 2011
%
% Provides the matrix whose determinant is the eigenfunction.  Omits
% derivative matrix.

function [E] = eigm2(kz,w,lambda,lambdaf,rho_s,rho_f,sig,r_a,r_b)

choice1 = 2; %2 for compressive liner, 3 for no liner

%test for no liner condition (also for up/down-stream pipe section)
if r_a == r_b
    choice1 = 3;
    clear E
end

% Derive needed quantities
c_L = sqrt(lambda/rho_s);
c_f = sqrt(lambdaf/rho_f);
kL = w/c_L;
kf = w/c_f;
krL = sqrt(kL^2-kz^2);
krf = sqrt(kf^2-kz^2);

%Bessel functions are re-used enough that it's probably worth just
%calculating them once.
j0fa = besselj(0,krf*r_a);
j1fa = besselj(1,krf*r_a);

if choice1 < 3
    j0La = besselj(0,krL*r_a);
    j1La = besselj(1,krL*r_a);
    y0La = bessely(0,krL*r_a);
    y1La = bessely(1,krL*r_a);
    j1Lb = besselj(1,krL*r_b);
    y1Lb = bessely(1,krL*r_b);
end

if choice1 == 2 %just compression waves
    % Populate matrix
    E = zeros(3);
    %row 1: continuity of radial displacement at r=a
    E(1,1) = krf/w*j1fa;
    E(1,2) = -krL/w*j1La;
    E(1,3) = -krL/w*y1La;
    %row 2: continuity of stress/pressure at r=a (add limp mass sheet)
    E(2,1) = kf^2/w^2*lambdaf*j0fa - krf*sig*j1fa;
    E(2,2) = -kL^2/w^2*lambda*j0La;
    E(2,3) = -kL^2/w^2*lambda*y0La;
    %row 3: zero radial displacement at r=b
    E(3,2) = -krL/w*j1Lb;
    E(3,3) = -krL/w*y1Lb;
end

if choice1 == 3 %no liner
    % Populate matrix
    E = zeros(1);
    %row 1: continuity of radial displacement at r=a
    E(1,1) = krf/w*j1fa;
end

end

%%
%adjugate2
% by K Marek
%
% Finds matrix adjugate

function B = adjugate2(A)

[m,n] = size(A);
if (m ~= n) || (n < 2)
 error('Matrix A should be size n x n with n >= 2.')
end

C = zeros(n);
for ind1 = 1:n
    A1 = [A(1:ind1-1,:); A(ind1+1:n,:)];
    for ind2 = 1:n
      A2 = [A1(:,1:ind2-1), A1(:,ind2+1:n)];
      C(ind1,ind2) = (-1)^(ind1+ind2)*det(A2);
    end
end
B = C.';
end

%%
%bint
%by K Marek, June 2010
%
% This function integrates J0 or Y0 of k*r.
% It takes a series approximation of the Struve function until it
appears
% to converge.
% char 'type' must be 'j' for besselj or 'y' or 'n' for bessely.

function int_ans = bint(kr,r0,r1,type)

%% int(r*J0) method
if (kr == 0) && (type == 'j')
    int_ans = 0.5*(r1^2-r0^2);
    return
end
```

```
if r0 == r1
    int_ans = 0;
    return
end
if type == 'j'
    int_ans = r1/kr*besselj(1,kr*r1) - r0/kr*besselj(1,kr*r0);
else
    if type == 'y' || type == 'n'
        int_ans = r1/kr*bessely(1,kr*r1) - r0/kr*bessely(1,kr*r0);
    else
        int_ans = 0;
        disp('Invalid input format to bint.m.')
        return
    end
end

end

%%
%getzp2
%by K Marek
%March 2010
%
%This file may be used to import experimental data to get downstream
%port impedance. By default it gives downstream reflectionless
condition.

function Zp2 = getzp2(freq, Z_f);

%If reflectionless coefficient is to be used:
    Zp2 = Z_f;
    return

% Else load data from a file
% Do stuff here

end %end function getzp2

%%
%proc
%by K Marek, June 2010
% Mode matching - generates correct relative amplitudes for each mode,
from
% which TL calculation is derived

function datstruct = proc(datstruct)

disp('Finding modal amplitudes and TL');

global showdebug

freq = datstruct.freq;
freqw = datstruct.freqw;
numharm = datstruct.numharm;
nummode = datstruct.nummode;
```

```
k1_rf = datstruct.k1_rf;
k1_zf = datstruct.k1_zf;
r_0 = datstruct.r_0;
k2_rf = datstruct.k2_rf;
k2_rL = datstruct.k2_rL;
k2_zf = datstruct.k2_zf;
r_a = datstruct.r_a;
r_b = datstruct.r_b;
zlen = datstruct.zlen;
rho_f = datstruct.rho_f;
rho_s = datstruct.rho_s;
sig = datstruct.sig;
c_f = datstruct.c_f;
lambda_f = datstruct.lambda_f;
lambda = datstruct.lambda;
Zp2plus = datstruct.Zp2plus;
kf = datstruct.k_f;
kL = datstruct.k_L;

numpmode = datstruct.numpmode;
numlmode = datstruct.numlmode;
numpint = datstruct.numpint;
numlint = datstruct.numlint;

k1fp = k1_rf(:,1:nummode);
k1fm = k1_rf(:,(nummode+1):2*nummode);
k1zp = k1_zf(:,1:nummode);
k1zm = k1_zf(:,(nummode+1):2*nummode);
k2fp = k2_rf(:,1:nummode);
k2fm = k2_rf(:,(nummode+1):2*nummode);
k2Lp = k2_rL(:,1:nummode);
k2Lm = k2_rL(:,(nummode+1):2*nummode);
k2zp = k2_zf(:,1:nummode);
k2zm = k2_zf(:,(nummode+1):2*nummode);

% 2-D compression/shear wave method
R_1 = zeros(numharm,1);
T_1 = R_1;
TL = R_1;
F_1 = R_1;
F_2 = R_1;
badfreq = zeros(1,numharm);
coef_mat = zeros(3*numpmode+2*nummode,numharm);

pcp = zeros(3,nummode,numharm);
pcm = pcp;

for inda1 = 1:numharm

    % Calculate propagation coefficients for each mode
    for inda2 = 1:nummode
        Ep =
eigm2(k2zp(inda1,inda2),freqw(inda1),lambda(inda1),lambda_f(inda1),rho_
s,rho_f,sig,r_a,r_b);
```

```
        Em =
eigm2(k2zm(inda1,inda2),freqw(inda1),lambda(inda1),lambda_f(inda1),rho_
s,rho_f,sig,r_a,r_b);
        temp1 = -Ep(:,1);
        Elen = length(temp1);
        temp1 = temp1(1:(Elen-1),1);
        Ep = Ep(1:(Elen-1),2:Elen);
        pcptemp = [1; linsolve(Ep,temp1)];
        pcp(:,inda2,inda1) = pcptemp/sqrt(abs(pcptemp.'*pcptemp));
%normalize
        if (rcond(Ep) < eps) || ~isfinite(rcond(Ep))
            badfreq(inda1) = 1;
            if showdebug == 1
            disp(['Ill conditioned matrix freq '
num2str(freq(inda1))...
                ', harm ' num2str(inda1) '.']);
            disp(Ep)
            disp(temp1)
            disp(pcp)
            end
        end
        temp1 = -Em(1:(Elen-1),1);
        Em = Em(1:(Elen-1),2:Elen);
        pcmtemp = [1; linsolve(Em,temp1)];
        pcm(:,inda2,inda1) = pcmtemp/sqrt(abs(pcmtemp.'*pcmtemp));
%normalize
        %Order of coeffeicients: F,L1,L2,T1,T2.
        clear temp1 pcptemp pcmtemp
    end

    % Set up integral matrices
    intP1p = zeros(numpint,nummode);
    intP2p = zeros(numpint,nummode);
    intP1m = zeros(numpint,nummode);
    intP2m = zeros(numpint,nummode);
    intU1p = zeros(numpint,nummode);
    intU2p = zeros(numpint,nummode);
    intU1m = zeros(numpint,nummode);
    intU2m = zeros(numpint,nummode);

    for inda3 = 1:nummode %mode number

        %Some useful constants we'd like to calculate only once
        temp1p = bint(k1fp(inda1,inda3),0,r_0,'j');
        temp2p = bint(k2fp(inda1,inda3),0,r_a,'j');
        temp1m = bint(k1fm(inda1,inda3),0,r_0,'j');
        temp2m = bint(k2fm(inda1,inda3),0,r_a,'j');

        %Calculate where to look at pressure continuity and zero shear
        %stress
        rp1 = r_0*(1:numpint)/numpint; %where we look at pressure
        rp3 = r_b*(1:numpint)/numpint; %and at displacement

        for inda2 = 1:max([nummode,numpint,numlint]) %iterate integrals
across port faces
            if inda2 <= numpint %Pressure matching
```

```
                intP1p(inda2,inda3) = -
kf(inda1)^2*lambda_f(inda1)*bint(k1fp(inda1,inda3),0,rp1(inda2),'j');
                intP1m(inda2,inda3) = -
kf(inda1)^2*lambda_f(inda1)*bint(k1fm(inda1,inda3),0,rp1(inda2),'j');
                if rp1(inda2) <= r_a
                    intP2p(inda2,inda3) = -
kf(inda1)^2*lambda_f(inda1)*pcp(1,inda3,inda1)*bint(k2fp(inda1,inda3),0
,rp1(inda2),'j');
                    intP2m(inda2,inda3) = -
kf(inda1)^2*lambda_f(inda1)*pcm(1,inda3,inda1)*bint(k2fm(inda1,inda3),0
,rp1(inda2),'j');
                else
                    %compression waves
                    intP2p(inda2,inda3) = -
kf(inda1)^2*lambda_f(inda1)*pcp(1,inda3,inda1)*temp2p + ...
                        (-kL(inda1)^2*lambda(inda1))*...

(pcp(2,inda3,inda1)*bint(k2Lp(inda1,inda3),r_a,rp1(inda2),'j') ...
                        +
pcp(3,inda3,inda1)*bint(k2Lp(inda1,inda3),r_a,rp1(inda2),'y'));
                    intP2m(inda2,inda3) = -
kf(inda1)^2*lambda_f(inda1)*pcm(1,inda3,inda1)*temp2m + ...
                        (-kL(inda1)^2*lambda(inda1))*...

(pcm(2,inda3,inda1)*bint(k2Lm(inda1,inda3),r_a,rp1(inda2),'j') ...
                        +
pcm(3,inda3,inda1)*bint(k2Lm(inda1,inda3),r_a,rp1(inda2),'y'));
                    if showdebug == 1
                    disp('Note: liner overlaps port!');
                    end
                end
            end

            if inda2 <= numpint
                if rp3(inda2) <= r_0
                    intU1p(inda2,inda3) = -1i*k1zp(inda1,inda3)*...
                        bint(k1fp(inda1,inda3),0,rp3(inda2),'j');
                    intU1m(inda2,inda3) = -1i*k1zm(inda1,inda3)*...
                        bint(k1fm(inda1,inda3),0,rp3(inda2),'j');
                else
                    intU1p(inda2,inda3) = -1i*k1zp(inda1,inda3)*temp1p;
                    intU1m(inda2,inda3) = -1i*k1zm(inda1,inda3)*temp1m;
                end
                if rp3(inda2) <= r_a
                    intU2p(inda2,inda3) = -
1i*k2zp(inda1,inda3)*pcp(1,inda3,inda1)*...
                        bint(k2fp(inda1,inda3),0,rp3(inda2),'j');
                    intU2m(inda2,inda3) = -
1i*k2zm(inda1,inda3)*pcm(1,inda3,inda1)*...
                        bint(k2fm(inda1,inda3),0,rp3(inda2),'j');
                else
                    intU2p(inda2,inda3) = -
1i*k2zp(inda1,inda3)*pcp(1,inda3,inda1)*temp2p + ...
                        (-
1i*k2zp(inda1,inda3))*(pcp(2,inda3,inda1)*bint(k2Lp(inda1,inda3),r_a,rp
3(inda2),'j')...
+pcp(3,inda3,inda1)*bint(k2Lp(inda1,inda3),r_a,rp3(inda2),'y'));
                    intU2m(inda2,inda3) = -
1i*k2zm(inda1,inda3)*pcm(1,inda3,inda1)*temp2m + ...
                        (-
1i*k2zm(inda1,inda3))*(pcm(2,inda3,inda1)*bint(k2Lm(inda1,inda3),r_a,rp
3(inda2),'j')...

+pcm(3,inda3,inda1)*bint(k2Lm(inda1,inda3),r_a,rp3(inda2),'y'));
                end
            end
        end
    end

    %Now fill up linear equations matrix with boundary conditions
    matA = zeros(4*numpint+2*numlint+numpmode,2*nummode+3*numpmode);
    matB = zeros(4*numpint+2*numlint+numpmode,1);

    for inda2 = 1:numpint
        %'A' coefficients: matB vector
        matB(inda2) = -intU1p(inda2,1); %p1 disp
        matB(2*numpint+2*numlint+inda2) = -intP1p(inda2,1); %p1
pressure
    end

    for inda2 = 1:nummode
        %KEY:
        %First rows: displacement at port 1
        %Second rows: displacement at port 2
        %Third rows (if applicable): shear stress at port 1
        %Fourth rows (if applicable): shear stress at port 2
        %Fifth rows: pressure at port 1
        %Sixth rows: pressure at port 2
        %Seventh rows: impedance condition for port 2

        if inda2 <= numpmode

            %'B' coefficients
            matA(1:numpint,inda2) = intU1m(:,inda2); %p1 disp
            matA(2*numpint+2*numlint+(1:numpint),inda2) =
intP1m(:,inda2); %p1 pressure

            %'E' coefficients
            matA(numpint+(1:numpint),numpmode+2*nummode+inda2) = ...
                -intU1p(:,inda2); %p2 disp

matA(3*numpint+2*numlint+(1:numpint),numpmode+2*nummode+inda2) = ...
                -intP1p(:,inda2); %p2 pressure
            if inda2==1
                % Zp2 only for plane wave modes; rest are evanescent
and
                % shouldn't have any reflections
                matA(4*numpint+2*numlint+1,numpmode+2*nummode+inda2) =
...
```

```
1i*freqw(inda1)*Zp2plus(inda1)*intU1p(numpint,inda2) +
intP1p(numpint,inda2);
            end

            %'F' coefficients
            matA(numpint+(1:numpint),2*numpmode+2*nummode+inda2) = ...
                -intU1m(:,inda2); %p2 disp

matA(3*numpint+2*numlint+(1:numpint),2*numpmode+2*nummode+inda2) = ...
                -intP1m(:,inda2); %p2 pressure
            if inda2==1
                % Zp2 only for plane wave modes; rest are evanescent
and
                % shouldn't have any reflections
                matA(4*numpint+2*numlint+1,2*numpmode+2*nummode+inda2)
= ...

1i*freqw(inda1)*Zp2plus(inda1)*intU1m(numpint,inda2) +
intP1m(numpint,inda2);
            else
                %set all other F coefficients to zero

matA(4*numpint+2*numlint+inda2,2*numpmode+2*nummode+inda2) = 1;
            end
        end

        %'C' coefficients
        matA(1:numpint,numpmode+inda2) = -intU2p(:,inda2); %p1 disp
        matA(numpint+(1:numpint),numpmode+inda2) = ...
            intU2p(:,inda2)*exp(-1i*k2zp(inda1,inda2)*zlen); %p2 disp
        matA(2*numpint+2*numlint+(1:numpint),numpmode+inda2) = -
intP2p(:,inda2); %p1 pressure
        matA(3*numpint+2*numlint+(1:numpint),numpmode+inda2) = ...
            intP2p(:,inda2)*exp(-1i*k2zp(inda1,inda2)*zlen); %p2
pressure

        %'D' coefficients
        matA(1:numpint,numpmode+nummode+inda2) = ...
            -intU2m(:,inda2)*exp(1i*k2zm(inda1,inda2)*zlen); %p1 disp
        matA(numpint+(1:numpint),numpmode+nummode+inda2) =
intU2m(:,inda2); %p2 disp
        matA(2*numpint+2*numlint+(1:numpint),numpmode+nummode+inda2) =
...
            -intP2m(:,inda2)*exp(1i*k2zm(inda1,inda2)*zlen); %p1
pressure
        matA(3*numpint+2*numlint+(1:numpint),numpmode+nummode+inda2) =
intP2m(:,inda2); %p2 pressure


    end


    %Re-scale things!
    for ind5 = 1:(4*numpint+2*numlint+numpmode)
```

```
        temp = max(abs(matA(ind5,:)));
        matA(ind5,:) = matA(ind5,:)/temp;
        matB(ind5) = matB(ind5)/temp;
    end


    matS = linsolve(matA,matB); %solution matrix
    temp = size(matA);
    if temp(1) == temp(2)
        if (rcond(matA) < eps) || ~isfinite(rcond(matA))
            if showdebug == 1
            disp(['Ill conditioned matrix freq '
num2str(freq(inda1))...
                '. harm ' num2str(inda1) '.']);
            end
            badfreq(inda1) = 1;
        end
    end


    R_1(inda1) = matS(1);
    T_1(inda1) = matS(2*nummode+numpmode+1);
    F_1(inda1) = T_1(inda1)*((Zp2plus(inda1)-rho_f*c_f)/ ...
        (Zp2plus(inda1)+rho_f*c_f));
    F_2(inda1) = matS(2*nummode+2*numpmode+1); %unused; F_2 = F_1 in
theory
    TL(inda1) = -20*log10(abs((T_1(inda1)-R_1(inda1)*F_1(inda1))/...
        (1-F_1(inda1)^2)));


    coef_mat(:,inda1)=matS;

    datstruct.TL = TL;
    datstruct.coef_mat = coef_mat;
    datstruct.pcp = pcp;
    datstruct.pcm = pcm;
    datstruct.badfreq = badfreq;
    datstruct.numpmode = numpmode;
    datstruct.numlmode = numlmode;
    datstruct.numpint = numpint;
    datstruct.numlint = numlint;
end

end
```

APPENDIX B

WEIGHTING FUNCTION

```
function WeightingOpt2(fileName,W,D,div)

%Function: WeightingOpt.m
%Version: 2
%Revisions: Handles new input for either a single suppressor or double
%suppressor configuration
%Inputs:
%       fileName - the saved file from TL_calc that includes the freq
(vector 1 x V),
%       PcVec (vector 1 x W), PsVec (vector 1 x X), PcPlot (vector 1 x
W),
%       unitset, subNum and TLmat (matrix W x W x X x V)
%       W - Frequency weighting factor, the min frequency and max
frequency
%       must be the same as the input to TL_calc but the resolution may
be
%       different. WeightingOpt will interpolate the vectors to make
them
%       the same dimension. W must have a row for each system pressure.
%       (vector X x Y)
%       D - Time weighting factor, must have a non-negative entry for
each
%       system pressure (vector 1 x X)
%Optional Inputs:
%       div - The number of color bands on the contour plot. A higher
%       number gives better resolution but slows down processing.
Default
%       value is 40.
%Outputs:
%        The optimal charge pressure is displayed in the command window
%        A contour plot should the value of the objective function for
all
%        charge pressure pairings

if nargin==3
    div=40; %Default amount of color divisions
end
eval(['load ' fileName])
% Data File check, ensures that all need variables are in the file.
if exist('TLmat','var')==0
    error('No Transmission Loss matrix in fileName (other variables may
be missing as well)')
elseif exist('supNum','var')==0
    error('No suppressor number in fileName (other variables may be
missing as well)')
elseif exist('PcPlot','var')==0
    error('No plot vector in fileName (other variables may be missing
as well)')
elseif exist('PcVec','var')==0
    error('No charge vector in fileName (other variables may be missing
as well)')
```

```
elseif exist('freq','var')==0
    error('No frequency vector in fileName (other variables may be
missing as well)')
elseif exist('unitset','var')==0
    error('No unit set in fileName')
end


% Input Check, Ensures that every input is the correct size
[Wrow Wcol]=size(W);
LPs = length(PsVec);
LD = length(D);
if Wrow == LPs && LPs == LD
elseif Wrow == LPs && LPs ~= LD
    error('Every Static Pressure does not have a time weighting
factor')
elseif Wrow ~= LPs && LPs == LD
    error('Every Static Pressure does not have a frequency weighting
factor')
elseif Wrow ~= LPs && LPs ~= LD
    error('Every Static Pressure does not have a time weighting factor
and a frequency weighting factor')
end
switch supNum
case 2
    %Weights each frequency using the frequency weighting factor,
creating the Wmat
    %matrix which is 3 dimensional (W x W x X)
    Wmat=zeros(length(PcVec),length(PcVec),length(PsVec)); %initializes
the Wmat matrix
    for xx=1:length(PcVec)
        for yy=1:length(PcVec)
            for zz=1:length(PsVec)

TLint=interp1(freq,reshape(TLmat(xx,yy,zz,:),size(freq)),linspace(min(f
req),max(freq),Wcol)); %Interpolates TL for a given condtion to the
same size as W
                Wmat(xx,yy,zz)=mean(W(zz,:).*TLint,2); %weights each
frequency, takes the average and puts it into the correct location
            end
        end
    end


    %Weights each system pressure using the time weighting factor,
creating the
    %Dmat matrix which is 2 dimensional (W x W)
    D=D./sum(D);
    Dmat=zeros(length(PcVec),length(PcVec));
    for xx=1:length(PcVec)
        for yy=1:length(PcVec)
            Dmat(xx,yy)=sum(D.*reshape(Wmat(xx,yy,:),size(D)),2);
        end
    end
    DmatN=Dmat+Dmat.'-diag(diag(Dmat)); %Only half of the coniditions
are simulated
    %since the results are symmetric, this line fills in the matrix
    step=(max(max(DmatN))-min(min(DmatN)))/div;%This determines the
number of
    % color steps on the graph currently there are 40. Fewer does not
show
    % enough resolution while more slows down processing.

    [aa bb]=max(DmatN);
    [~, dd]=max(aa);
    Char1=PcPlot(dd);
    Char2=PcPlot(bb(dd));

    DmatN=DmatN./max(max(DmatN)); %Normalization

    % figure %
    % plot(PcVec./1e6,diag(DmatN))
    % xlabel('Charge Pressure (MPa)')
    % ylabel('Magnitude of Objective Function')
    % axis([min(PcVec./1e6),max(PcVec./1e6),0,1])
    figure
    contourf(PcPlot,PcPlot,DmatN,div,'LineStyle','none');%
    colorbar
    switch unitset
        case 'English'
            xlabel('Charge Pressure 1 (psi)')
            ylabel('Charge Pressure 2 (psi)')
            title( ['The optimal charge pressure pairing is
'num2str(Char1) ' psi and ' num2str(Char2) ' psi.'])
        case 'Metric'
            xlabel('Charge Pressure 1 (Pa)')
            ylabel('Charge Pressure 2 (Pa)')
            title(['The optimal charge pressure pairing is '
num2str(Char1) ' psi and ' num2str(Char2) ' psi.'])
    end

    switch unitset %Displays the optimal condition in the command
window, depending on unit set
        case 'English'
            DisMat=['Static Pressures: ';'Normalized  D:    '];

DisMat2=num2str([round(PsVec./6894.75729);round(100*D)./100]);
            disp(['The optimal charge pressure pairing is '
num2str(Char1) ' psi and ' num2str(Char2) ' psi.'])
            disp([DisMat DisMat2]) %Displays the normalized Time
Weighting Factor
        case 'Metric'
            DisMat=['Static Pressures: ';'Normalized  D:    '];
            DisMat2=num2str([round(PsVec);round(100*D)./100]);
            disp(['The optimal charge pressure pairing is '
num2str(Char1) ' Pa and ' num2str(Char2) ' Pa.'])
            disp([DisMat DisMat2]) %Displays the normalized Time
Weighting Factor
    end
case 1
```

```
    %Weights each frequency using the frequency weighting factor,
creating the Wmat
  %matrix which is 3 dimensional (W x W x X)
  Wmat=zeros(length(PcVec),length(PsVec)); %initializes the Wmat
matrix
  for xx=1:length(PcVec)
      for zz=1:length(PsVec)

TLint=interp1(freq,reshape(TLmat(xx,zz,:),size(freq)),linspace(min(freq
),max(freq),Wcol)); %Interpolates TL for a given condtion to the same
size as W
        Wmat(xx,zz)=mean(W(zz,:).*TLint,2); %weights each
frequency, takes the average and puts it into the correct location
      end
  end



  %Weights each system pressure using the time weighting factor,
creating the
  %Dmat matrix which is 2 dimensional (W x W)
  D=D./sum(D);
  Dmat=zeros(length(PcVec),1);
  for xx=1:length(PcVec)
      Dmat(xx)=mean(D.*reshape(Wmat(xx,:),size(D)),2);
  end

  Dmat=Dmat./max(max(Dmat));
  figure
  plot(PcPlot.',Dmat)
  ylabel('Magnitude of Objective Function')
  axis([min(PcPlot),max(PcPlot),0,1])
  [aa bb]=max(Dmat);
  Char1=PcPlot(bb);
    switch unitset %Displays the optimal condition in the command
window, depending on unit set
    case 'English'
      Tmat=['Static Pressures: ';'Normalized  D:    '];

Tmat2=num2str([round(PsVec./6894.75729);round(100*D)./100]);
      disp(['The optimal charge pressure is ' num2str(Char1) '
psi.'])
      disp([Tmat Tmat2])
    case 'Metric'
      Tmat=['Static Pressures: ';'Normalized  D:    '];
      Tmat2=num2str([round(PsVec);round(100*D)./100]);
      disp(['The optimal charge pressure pairing is '
num2str(Char1) ' Pa.'])
      disp([Tmat Tmat2])
    end
      switch unitset
    case 'English'
      xlabel('Charge Pressure 1 (psi)')
      title(['The optimal charge pressure pairing is '
num2str(Char1) ' psi.'])
    case 'Metric'
      xlabel('Charge Pressure 1 (Pa)')
```

```
            title(['The optimal charge pressure pairing is '
num2str(Char1) ' psi.'])
        end
end
```

REFERENCES

[1] E. Arendt. *Pulsation Absorbing Device*. USPTO Pat. No. 4,759,387 (1988).

[2] J.C. Shiery. *Noise Suppressor*. USPTO Pat. No. 5,732,741 (1998).

[3] J. Gary M. Jenski and J.C. Shiery. *Noise Suppressor*. USPTO Pat. No. 5,735,313 (1998).

[4] K.A. Marek, E.R. Gruber, and K.A. Cunefare, *Linear Analytical Model for a Pressurized Gas Bladder Style Hydraulic Silencer.* International Journal of Fluid Power, 2012. **Submitted**.

[5] D.N. Johnston and K.A. Edge, *A Test Method for Measurement of Pump Fluid-Borne Noise Characteristics*, in *International Off-Highway & Powerplant Congress and Exposition*. 1991: Milwaukee, Wisconsin.

[6] L.E. Kinsler, et al., *Fundamentals of Acoustics*. 4th ed. 1999: John Wiley & Sons, Inc.

[7] A. Selamet and Z.L. Ji, *Acoustic Attenuation Performance of Circular Expansion Chambers With Extended Inlet/Outlet.* Journal of Sound and Vibration, 1999. **223**(2): p. 197-212.

[8] A.D. Pierce, *Acoustics: An Introduction to Its Physical Properties and Applications*. 1989, Melville, NY: Acoustical Society of America.

[9] R. Kirby and A. Cummings, *Prediction of the bulk acoustic properties of fibrous materials at low frequencies.* Applied Acoustics, 1999. **56**(2): p. 101-125.

[10] R. Wilkes. *Noise Reduction in Hydraulic Systems*. in *Inter-Noise 95*. 1995. Newport Beach, CA, USA.

[11] E.R. Gruber, et al., *Optimization of Single and Dual Suppressors Under Varying Load and Pressure Conditions.* International Journal of Fluid Power, 2012. **Submitted**.

[12] N.E. Earnhart, K.A. Marek, and K.A. Cunefare. *Evaluation of hydraulic silencers*. in *NoiseCon10*. 2010. Baltimore, MD.

[13] A.F. Seybert and D.F. Ross, *Experimental determination of acoustic properties using a two-microphone random-excitation technique.* Journal of the Acoustical Society of America, 1976. **61**(5): p. 1362-1370.

[14] J.Y. Chung and D.A. Blaser, *Transfer function method of measuring in-duct acoustic properties. I. Theory.* Journal of the Acoustical Society of America, 1980. **68**(3): p. 907-913.

[15] J.Y. Chung and D.A. Blaser, *Transfer function method of measuring in-duct acoustic properties. II. Experiment.* Journal of the Acoustical Society of America, 1980. **68**(3): p. 914-921.

[16] C.W.S. To and A.G. Doige, *A transient testing technique for the determination of matrix parameters of acoustic systems, I: Theory and principles.* Journal of Sound and Vibration, 1979. **62**(2): p. 207-222.

[17] C.W.S. To and A.G. Doige, *A transient testing technique for the determination of matrix parameters of acoustic systems, II: Experimental procedures and results.* Journal of Sound and Vibration, 1979. **62**(2): p. 223-233.

[18] T.Y. Lung and A.G. Doige, *A time-averaging transient testing method for acoustic properties of piping systems and mufflers with flow.* Journal of the Acoustical Society of America, 1982. **73**(3): p. 867-876.

[19] E. Kojima and K.A. Edge. *Experimental determination of hydraulic silencer transfer matrices and assessment of the method for use as a standard test procedure*. in *Innovations in Fluid Power, 7th Bath International Fluid Power Workshop*. 1994. University of Bath, UK: Research Studies Press Ltd.

[20] K.K. Lau, D.N. Johnston, and K.A. Edge. *Fluid borne noise characteristics of hydraulic filters and silencers*. in *Innovations in Fluid Power, 7th Bath International Fluid Power Workshop*. 1994. University of Bath, UK: Research Studies Press Ltd.

[21] J.E. Drew, D.K. Longmore, and D.N. Johnston, *Measurement of the longitudinal transmission characteristics of fluid-filled hoses.* Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering, 1997. **211**(3): p. 219-228.

[22] D.N. Johnston, D.K. Longmore, and J.E. Drew, *A technique for the measurement of the transfer matrix characteristics of two-port hydraulic components.* Fluid Power Systems and Technology, 1994. **1**: p. 25-33.

[23] ISO-15086-1, *Hydraulic fluid power - Determination of fluid-borne noise characteristics of components and systems*, in *Introduction*. 2001, International Standards Organization: Geneva, Switzerland.

[24] ISO-15086-2, *Hydraulic fluid power - Determination of fluid-borne noise characteristics of components and systems*, in *Measurement of speed of sound in a fluid in a pipe*. 2000, International Standards Organization: Geneva, Switzerland.

[25] ISO-15086-3, *Hydraulic fluid power - Determination of the fluid-borne noise characteristics of components and systems*, in *Measurement of hydraulic impedance*. 2008, International Standards Organization: Geneva, Switzerland.

[26] A. Selamet, et al., *Analytical approach for sound attenuation in perforated dissipative silencers.* Journal of the Acoustical Society of America, 2005. **115**(5): p. 2091-2099.

[27] F.D. Denia, et al., *Acoustic attenuation performance of perforated dissipative mufflers with empty inlet/outlet extensions.* Journal of Sound and Vibration, 2007. **302**: p. 1000-1017.

[28] I. Lee, A. Selamet, and N.T. Huff, *Impact of perforation impedance on the transmission loss of reactive and dissipative silencers.* Journal of the Acoustical Society of America, 2006. **120**(6): p. 3706-3713.

[29] A. Selamet, et al., *Analytical approach for sound attenuation in perforated dissipative silencers.* Journal of the Acoustical Society of America, 2004. **115**(5): p. 2091-2099.