EXPERIMENTAL EVALUATION OF ACOUSTIC SATURATION

BY

JASON MATTHEW SEMPSROTT

B.S., South Dakota State University, 1998
B.S., South Dakota State University, 1998

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2000

Urbana, Illinois

TABLE OF CONTENTS

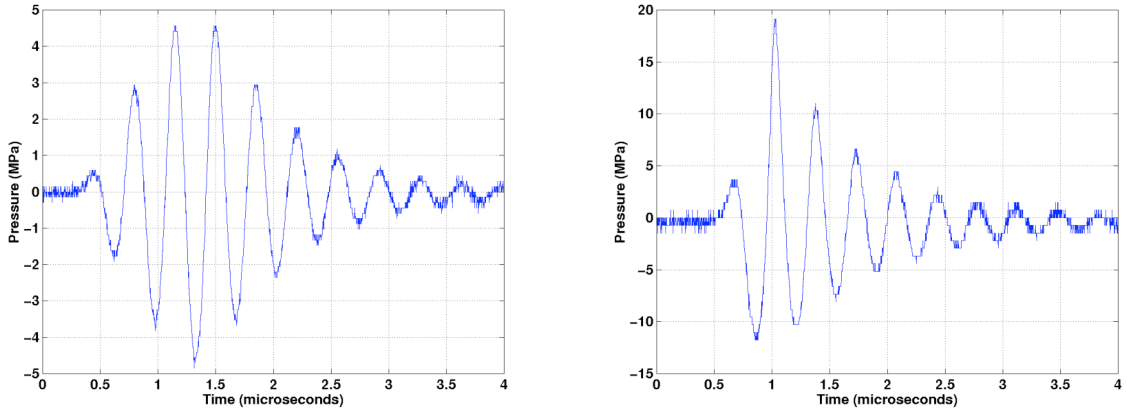
LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION TO ACOUSTIC SATURATION

Acoustic saturation states that as the input voltage to a focused ultrasonic transducer increases, there exists a limit of the acoustic pressure at the transducer's focus. The most significant phenomenon associated with saturation is the presence of nonlinear conditions in the propagating wave. As the term *nonlinear* implies, the features of an acoustic waveform change. For example, at a relatively low applied voltage, the acoustic pressure at a transducer's focus will appear sinusoidal, and the positive and negative pressures are approximately equal, as can be seen in Figure 1.1(a). However, if the input voltage is increased significantly, the positive pressure at the focus will become larger than the negative pressure at the focus, which can be seen in Figure 1.1(b). Understanding the origins of the nonlinear characteristics is the basis for understanding acoustic saturation.



(a) (b)

Figure 1.1: Time-domain acoustic waves at the focus for (a) low and (b) high input voltages.

1.1 Background

There is a need to define some of the terminology that will be used throughout this text. A system was developed to determine acoustic pressure levels at the focus of an ultrasonic field. A detailed description of the process is described in Chapter 4, but it is useful to describe some of the definitions used in that process.

Measurements of a transducer's ultrasonic pressure characteristics are made in water because of its availability and well-known characteristics [1]. The basis of the system used to find the transducer's focus is the pulse intensity integral (*PII*) [2]:

$$\mathrm{PII} = \frac{1}{\rho \cdot c}\int_0^T \left[p(t)\right]^2 dt \tag{1.1}$$

where $p(t)$ is the time-domain ultrasonic pressure waveform, ρ is the density of the propagating medium, c is the speed of sound in the medium, and T is the time interval when the received pulse at the hydrophone is nonzero [3]. Acoustic pressure waveforms can be collected at various distances from the transducer, and a series of *PII* values can be plotted as a function of axial distance from the transducer, which is shown in Figure 1.2. This is the axial profile for the transducer. By definition [2], the focus occurs at the maximum point on the derated *PII* ($PII_{.3}$) curve, which is the lower curve shown in Figure 1.2. The .3 subscript comes from the derating factor 0.3 dB/cm·MHz [2], [3]. The derating factor is an attenuation factor that can be added to

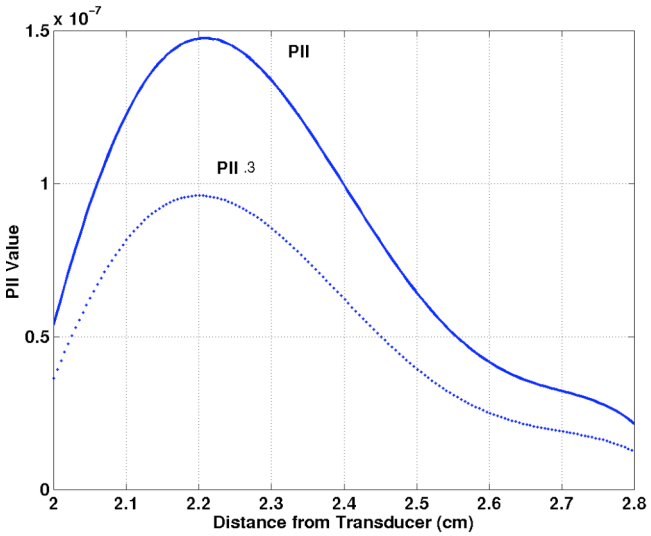


Figure 1.2: Axial profiles for *PII* and $PII_{.3}$ versus distance from the transducer.

water measurements. The derated pulse intensity integral is defined as [2]:

$$PII_{.3} = PII\left[\exp\left(-0.069 \cdot f_c \cdot z\right)\right] \tag{1.2}$$

where f_c is the center frequency, z is the distance from the transducer, and 0.069 in the exponential comes from the conversion:

$$\frac{0.3\text{dB}}{\text{cm} \cdot \text{MHz}} \bullet \frac{\text{Np}}{8.7\text{dB}} \bullet 2 = 0.069 \, {}^{\text{Np}}\!/_{\text{cm} \cdot \text{MHz}}$$

The exponential needs to be multiplied by a factor of 2 because *PII* is an intensity measurement.

Each pressure waveform collected during the measurement process has a peak positive pressure, or compressional pressure (p_c), and a peak negative pressure, or rarefactional pressure (p_r), associated with it. Similar to the *PII* plots, p_c and p_r can also be plotted as functions of distance from the source to obtain pressure profiles for the transducer. The p_c and p_r values are also derated by the derating factor and the resulting equations are

$$\begin{aligned} p_{c.3} &= p_c\left[\exp\left(-0.0345 \cdot f_c \cdot z\right)\right] \\ p_{r.3} &= p_r\left[\exp\left(-0.0345 \cdot f_c \cdot z\right)\right] \end{aligned} \tag{1.3}$$

where

$$\frac{0.3\text{dB}}{\text{cm} \cdot \text{MHz}} \bullet \frac{\text{Np}}{8.7\text{dB}} = 0.0345 \, {}^{\text{Np}}\!/_{\text{cm} \cdot \text{MHz}}$$

and f_c and z are the same as before. Axial profiles of p_c, $p_{c.3}$, p_r, and $p_{r.3}$ are shown in Figure 1.3.

Figure 1.4 shows axial profiles for $PII_{.3}$, $p_{c.3}$, and $p_{r.3}$ normalized to the maximum value of the $PII_{.3}$ profile curve. Normalization was done so that all three axial profiles could be plotted on the same graph. Once the position where $PII_{.3}$ is a maximum is determined, that position is then used to determine $p_{c.3}$ and $p_{r.3}$, which are the pressure values at the focus. For example, let the position of the maximum $PII_{.3}$ for Figure 1.2 be at 2.2 cm. The axial profiles for $p_{c.3}$ and $p_{r.3}$ are plotted in Figure 1.3 from 2 to 2.5 cm. The derated acoustic pressures at 2.2 cm would correspond to the pressures at the focus of the transducer. The $p_{r.3}$ value can then be used to determine the mechanical index (*MI*) at the transducer's focus. Mechanical index is defined as [2]:

$$MI = \frac{p_{r.3}}{\sqrt{f_c}} \tag{1.4}$$

The regulatory limit set by the Food and Drug Administration (FDA) is an *MI* of 1.9 [4].

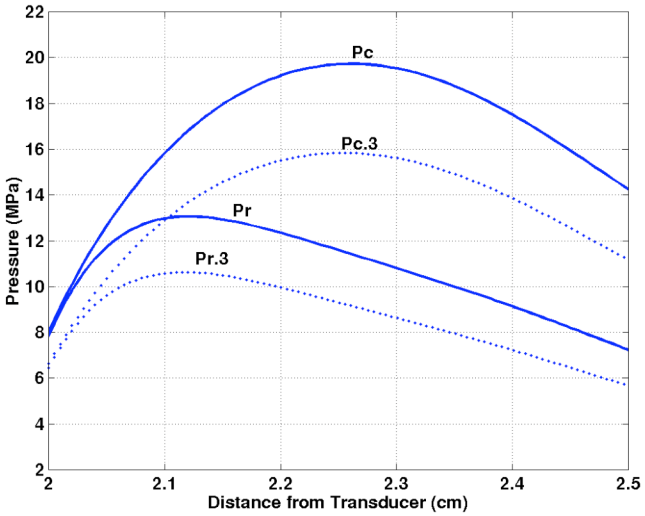


Figure 1.3: Axial profiles for p_c and p_r acoustic pressures and their derated acoustic pressures.

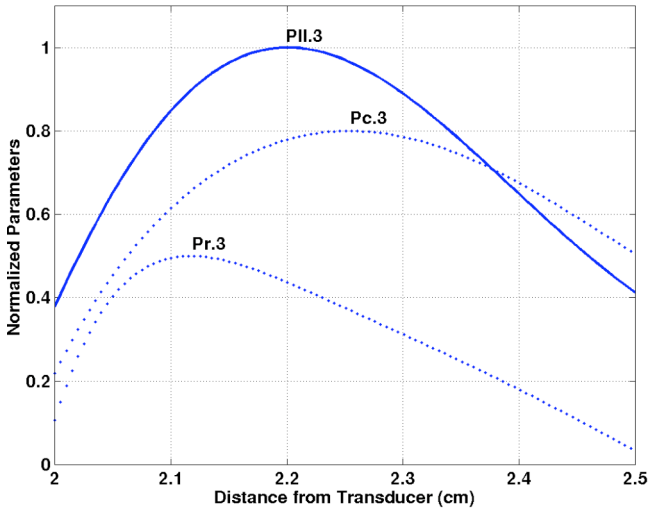


Figure 1.4: Axial profiles of $PII_{.3}$, $p_{c.3}$, and $p_{r.3}$ normalized to the same graph.

1.2 Motivation

The purpose of this thesis is to develop the theory used in predicting the saturation ultrasonic pressure levels of spherically focused transducers. From that theory, the next step is to compare the predictions to experimental results. The motivation for understanding saturation stems from the techniques used to measure a transducer's pressure fields in water, as discussed in the background section. Because water acoustic pressure measurements are derated to determine

acoustic pressures at the focus, it is necessary to understand what is happening to those measurements during nonlinear conditions in water.

Some literature states that derated water measurements at saturation levels may be underestimating acoustic pressure levels in tissue [1], [5]-[7]. Duck [5] points out that the FDA's MI limit of 1.9 is ineffective at regulating pulsed diagnostic ultrasound during highly nonlinear conditions, which is the case during acoustic saturation. This is because saturation conditions will not allow the system to operate above 1.9 for certain frequencies and focal lengths (see Fig. 3 [5]). So there exist limitations to using acoustic pressure field measurements made in water. Therefore, it is important to understand the effects associated with acoustic saturation.

This thesis can be split into two major areas: theory and experiment. In the theoretical development, the second chapter describes nonlinear propagation in and saturation of plane waves. Chapter 3 develops the theory used in predicting saturation levels of spherically focused transducers. Chapter 4 explains the system used to collect the data used in the results of Chapter 5. Chapter 5 presents a collection of data obtained from seven focused ultrasonic transducers. Chapter 6 compares the experimental results and the theoretical predictions, and discusses discrepancies found between the two. The thesis ends with a summary and discussion of possiblities for future work.

CHAPTER 2

NONLINEARITY IN PLANE WAVES

This chapter develops concepts used to decribe nonlinear propagation of plane progressive waves. Within the nonlinear developments will be the development of the acoustic shock parameter, which is used to describe the magnitude of nonlinearity in a travelling wave. The development for acoustic saturation in plane waves is also included.

2.1 Development of the Acoustic Shock Parameter

The acoustic shock parameter σ is an indicator of the magnitude of nonlinearity associated with a traveling acoustic wave. For a plane wave propagating in a lossless medium, i.e., water, there is a distance $\bar{x}$ [8] where the wave is defined as distorted. As a continuous sinusoidal wave propagates in water, it reaches a distance from the source where it no longer resembles a sinusoid, but rather a sawtooth waveform. Figure 2.1 shows the progression of a wave from pure sinusoid into a sawtooth waveform, and then the recovery back to a sinusoid. The distortion distance $\bar{x}$ in Figure 2.1 is defined as [8]:

$$\bar{x} = \frac{c_o^2}{(u_o \omega)\left(1 + \frac{B}{2A}\right)} \tag{2.1}$$

The ratio *B/A* is a measured property of the medium and is described in [8]. The value u_o is the acoustic velocity at the surface of the source, ω is the angular frequency of the source, and c_o is the speed of sound in the medium. Substituting $\omega = kc_o$, where k is the wave number, $u_o / c_o = \varepsilon$, where ε is the Mach number, and $\beta = \left(1 + \frac{1}{2}B/A\right)$, where β is the nonlinear propagation constant

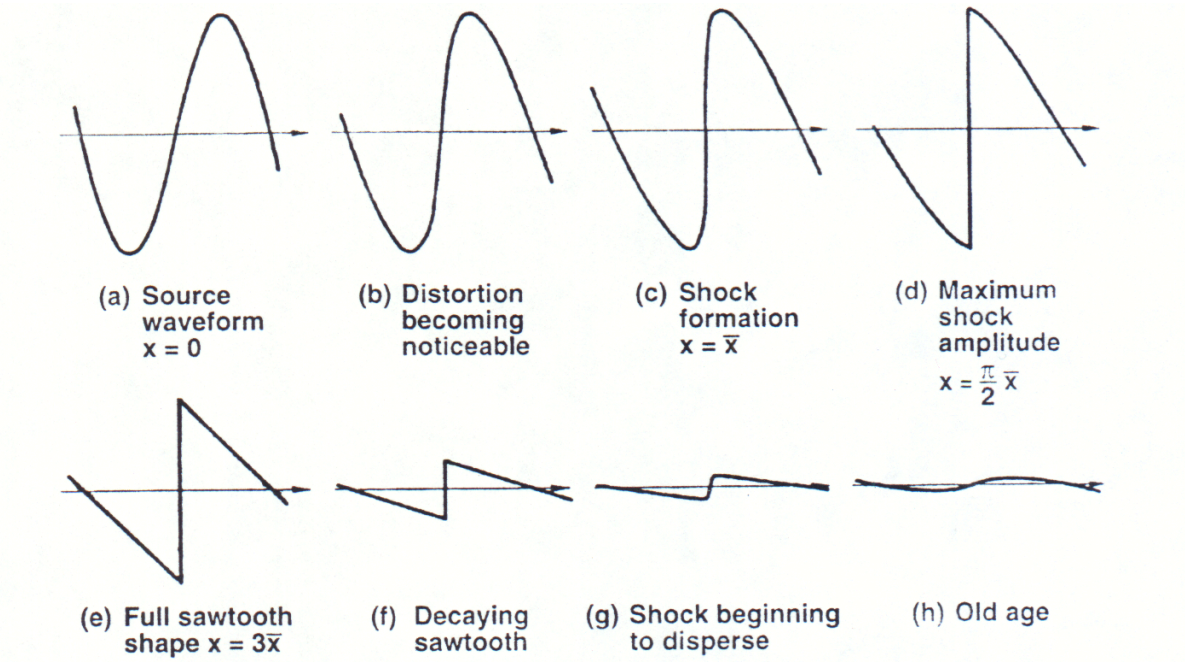


Figure 2.1: Progression of a continuous sinusoid in a liquid [8].

in a liquid [8], yields the distortion distance equation:

$$\bar{x} = \frac{1}{k \cdot \varepsilon \cdot \beta} \tag{2.2}$$

The acoustic shock parameter σ is a function of the distance x travelled by the wave and the nonlinear distance $\bar{x}$ [8]:

$$\sigma = \frac{x}{\bar{x}} = \beta \cdot \varepsilon \cdot k \cdot x \tag{2.3}$$

According to [8], σ can be divided into three regions. The first region occurs when $\sigma<1$, which means that no discontinuities or shocks are occuring in the propagating wave. The second region occurs when $1 \le \sigma \le 3$. This region is a transition region where the waveform is becoming distorted. The final region occurs when $\sigma>3$, which means that the waveform has become sawtooth in appearance. For this final region the positive portion of the waveform has caught up with the zero crossing and the zero crossing has caught up with negative protion of the wave, thus resembling a sawtooth waveform.

2.2 Fourier Analysis of a Propagating Plane Wave

A Fourier analysis of the propagating wave yields the harmonic components of the wave as a function of the shock parameter. This Fourier analysis is useful in understanding the relationship that saturation has with the shock parameter. To determine the Fourier coefficients, it is useful to start from a mathematical description of the propagating wave in the time domain [8]:

$$\frac{u}{u_o} = \sin[\omega t - kx + \sigma u / u_o] \tag{2.4}$$

Equation (2.4) can be summed into its Fourier components accordingly:

$$\frac{u}{u_o} \cong \sum_{n=1}^{\infty} B_n \sin[n(\omega t - kx)] \tag{2.5}$$

From Eq. (2.5), the next step is to find the Fourier coefficients, B_n [9]:

$$B_n = \frac{1}{\pi}\int_0^{2\pi} \sin[\omega t - kx + \sigma \frac{u}{u_o}] \sin[n(\omega t - kx)] d(\omega t - kx) \tag{2.6}$$

where the integration is with respect to (ωt - kx). A change of variable is made where $y = (\omega t - kx)$ and $\Phi = (\omega t - kx + \sigma u / u_o) = y + \sigma u / u_o$. But u/u_o is defined in Eq. (2.4), so $u/u_o = \sin \Phi$, which gives

$$\Phi = y + \sigma \cdot \sin \Phi \tag{2.7}$$

In Eq. (2.7), when $\Phi = \pi$ the variable $y = \pi$. The Fourier coefficients in Eq. (2.6) can now be written as

$$B_n = \frac{2}{\pi}\int_0^{\pi} \sin[\Phi] \sin[n \cdot y] \cdot dy \tag{2.8}$$

Equation (2.8) corresponds to Eq. (181) in Chapter 4 of [8]. Equation (2.8) can be integrated by parts using the following variables:

$$\begin{aligned} u &= \sin\Phi \rightarrow du = \cos\Phi \cdot d\Phi \\ v &= -\frac{1}{n}\cos(n \cdot y) \rightarrow dv = \sin(n \cdot y) \cdot dy \end{aligned} \tag{2.9}$$

such that the Fourier coefficients can be written

$$B_n = \frac{2}{n \cdot \pi}\left[-\sin\Phi\left(\cos n \cdot y\right)\Big|_0^{\pi} + \int_0^{\pi} \cos[\Phi] \cos[n \cdot y] \cdot d\Phi \right] \tag{2.10}$$

The first term in Eq. (2.10) is evaluated for y from zero to π. However, from Eq. (2.7) when $y = \pi$, $\Phi = \pi$, so $\sin\Phi$ becomes zero at the upper limit. Therefore the first term in Eq. (2.10) is evaluated at $y = 0$ only. In that case, $\cos(n \cdot y)$ at $y = 0$ goes to one, which leaves

$$B_n = \frac{2}{n \cdot \pi}\left[\sin\Phi\Big|_{y=0} + \int_0^{\pi} \cos[\Phi]\cos[n \cdot y] \cdot d\Phi\right] \tag{2.11}$$

Equation (2.11) is Eq. (182) in Chapter 4 of [8]. To simplify the integration of Eq. (2.11), a substitution needs to take place:

$$d(\Phi - y) = d(y + \sigma\sin\Phi - y) = d(\sigma\sin\Phi) = \sigma \cdot d(\sin\Phi) = \sigma\cos\Phi d\Phi$$

$$d(\Phi - y) = \sigma\cos\Phi d\Phi \rightarrow d\Phi = \frac{d(\Phi - y)}{\sigma\cos\Phi} \tag{2.12}$$

Substituting $d\Phi = \dfrac{d(\Phi - y)}{\sigma\cos\Phi}$ into Eq. (2.11) yields

$$B_n = \frac{2}{n \cdot \pi}\left[\sin\Phi\Big|_{y=0} + \frac{1}{\sigma}\int_0^{\pi} \cos[n \cdot y] \cdot d(\Phi - y)\right] \rightarrow$$

$$B_n = \frac{2}{n \cdot \pi}\left[\sin\Phi\Big|_{y=0} + \frac{1}{\sigma}\int_0^{\pi} \cos[n \cdot y] \cdot d\Phi - \frac{1}{\sigma}\int_0^{\pi} \cos[n \cdot y]dy\right] \tag{2.13}$$

The third term in the brackets goes to zero after the integration. The lower limit on the first integral also changes to $\Phi\big|_{y=0}$ and the resultant Fourier coefficients for the nonlinear propagating wave is

$$B_n = \frac{2}{n \cdot \pi}\left[\sin\Phi\Big|_{y=0} + \frac{1}{\sigma}\int_{\Phi|_{y=0}}^{\pi} \cos[n \cdot y] \cdot d\Phi\right] \tag{2.14}$$

The lower limits in Eq. (2.14) occur when $y = 0$ in Eq. (2.7). Letting $y = 0$ in Eq. (2.7) gives:

$$\Phi = \sigma \cdot \sin\Phi \tag{2.15}$$

A plot of Eq. (2.15) is shown in Figure 2.2. Notice that when $\sigma < 1$ the function is zero, and that it asymptotically approaches π as σ becomes large. It is now possible to evaluate the coefficients for the regions of interest. For $\sigma < 1$, the first term in Eq. (2.14) goes to zero because of the

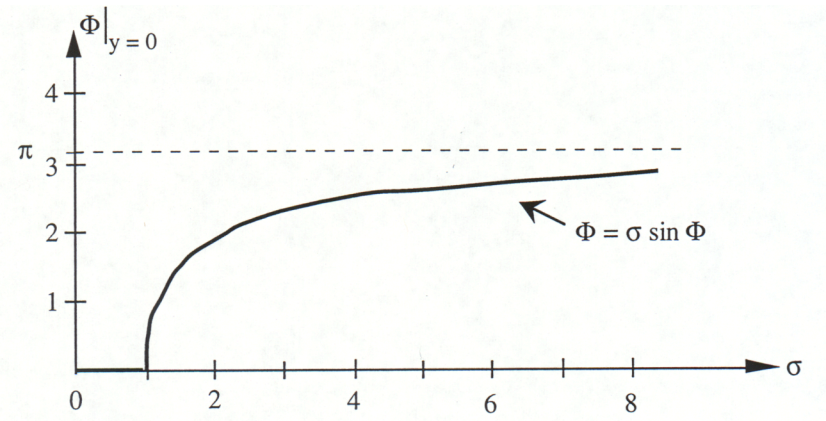


Figure 2.2: Plot of the relationship in Eq. (2.15).

characteristics shown in Figure 2.2, which leaves

$$B_n = \frac{2}{n \cdot \pi \cdot \sigma}\left[\int_0^{\pi} \cos[n \cdot y] \cdot d\Phi\right] = \frac{2}{n \cdot \pi \cdot \sigma}\left[\int_0^{\pi} \cos[n(\Phi - \sigma \sin \Phi)] \cdot d\Phi\right] \quad , \sigma < 1 \tag{2.16}$$

The result of Eq. (2.16) is the form for a Bessel's function, and it can then be written as

$$B_n = \frac{2}{n \cdot \sigma} J_n(n \cdot \sigma) \, , \, \sigma < 1 \tag{2.17}$$

where J_n is an nth order Bessel's function.

B_n can also be evaluated for $\sigma >> 1$. For this case, the integral term in Eq. (2.14) goes to zero because the lower limit approaches the upper limit, and the remaining function is

$$B_n = \frac{2}{n \cdot \pi}\left[\sin \Phi\big|_{y=0}\right] \, , \, \sigma \gg 1 \tag{2.18}$$

The next step is to solve for the $\sin\Phi$ term in Eq. (2.18). For small angles the $\sin\Phi = \Phi$, which allows the following steps to be taken:

$$\text{eq}(7) \rightarrow \Phi = \sigma \sin \Phi$$
$$\text{Let} \rightarrow \Phi = \pi - x$$
$$\sin \Phi \cong x$$
$$\pi - x = \sigma \sin \Phi = \sigma \cdot x$$
$$x = \frac{\pi}{(1 + \sigma)}$$

where x is some small angle. Substituting x into Eq. (2.18) yields

$$B_n = \frac{2}{n \cdot \pi} \sin \Phi\big|_{y=0} = \frac{2x}{n \cdot \pi} = \frac{2 \cdot \pi}{n \cdot \pi (1 + \sigma)} = \frac{2}{n(1 + \sigma)} \quad , \sigma \gg 1 \tag{2.19}$$

The result above provides the magnitude expected at very large values of the acoustic shock parameter. For the regions of $\sigma > 1$ it is possible to evaluate Eq. (2.14) numerically. Figures 2.3 and 2.4 show the harmonic components calculated from Eq. (2.14) as a function of the shock parameter. It turns out that Eq. (2.19) is a very good approximation for $\sigma > 3$. That region between 1 and 3 is the point where a wave starts to become distorted. That transition from 1 to 3 has been evaluated and is known as *Blackstock's bridging function* [8].

In summary, an acoustic shock parameter σ describes the amount of distortion associated with a propagating wave. A Fourier analysis of the propagating plane wave yielded the harmonic amplitudes. The evaluation was shown for both small ($\sigma < 1$) and large ($\sigma > 3$) shock parameters. The region $1 \leq \sigma \leq 3$ is a transition region that can be evaluated numerically.

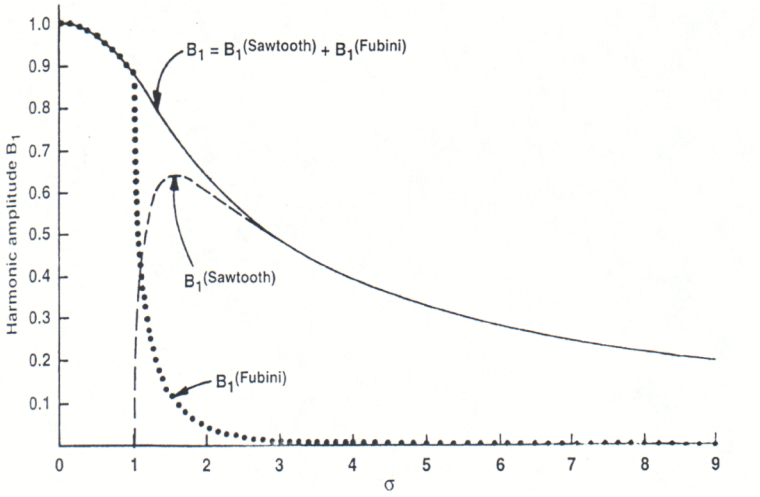


Figure 2.3: Fundamental frequency characteristics. The region for $\sigma < 1$ is known as the *Fubini* solution, and the region for $\sigma > 3$ is the *sawtooth* solution. Adding the two together in the transition region yields the continuous curve for B_1 [8].

2.3 Saturation in Plane Waves

The previous section shows the effect that a large shock parameter has on the energy available in the signal's fundamental. As σ approaches large values, the wave becomes saturated. Notice in Figure 2.4 that as σ increases the amplitude of the second and third harmonics approach that of the first (although never overtaking it). The percentage of energy in the higher harmonics increases during saturation conditions.

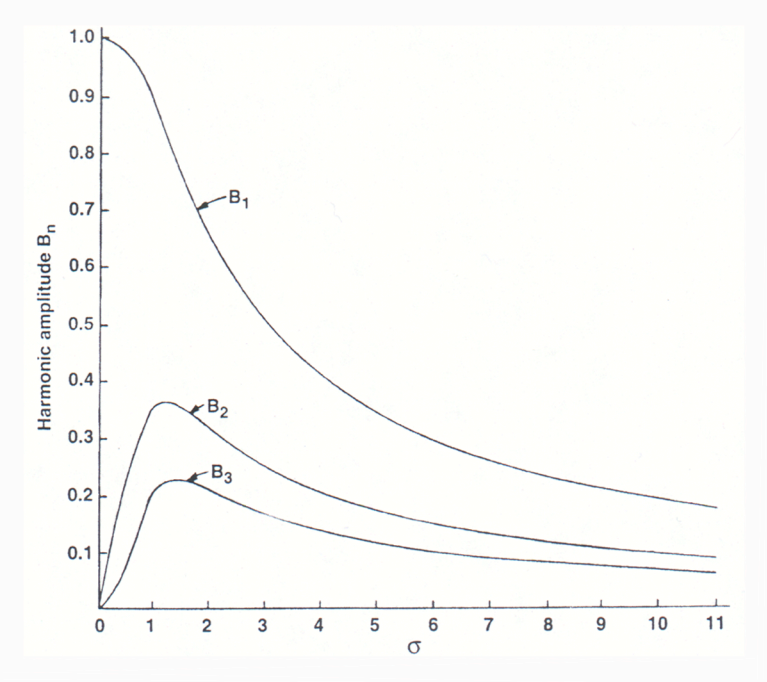


Figure 2.4: Behavior of 1st, 2nd, and 3rd harmonics as a function of shock parameter [8].

To determine the theoretical saturation pressure for a plane wave, the wave is converted into its Fourier components [8]:

$$p = \frac{2p_o}{(1+\sigma)} \sum_{n=1}^{\infty} \frac{\sin(n \cdot \omega \cdot t)}{n}, \ \sigma \ll 1 \tag{2.20}$$

which is similar to the Eq. (2.18) in the previous section. If an inverse transform of this equation is taken, the time-domain waveform is described as

$$p = \frac{p_o(\pi)}{(1+\sigma)} \tag{2.21}$$

As σ becomes quite large, $1+\sigma \approx \sigma$, and the saturation can be found by

$$p = \frac{p_o(\pi)}{\sigma} = \frac{p_o \pi}{\beta \cdot \frac{u_o}{c_o} \cdot k \cdot x} = \frac{p_o \pi \cdot \rho_o c_o^2}{\beta \cdot p_o \cdot k \cdot x} = \frac{\rho_o c_o^3}{2\beta \cdot f \cdot x} = P_{sat,p}, \ \sigma \ll 1 \tag{2.22}$$

This is the result obtained for Eq. (185) in Chapter 4 of [8], and Eq. (1) in [5]. Therefore the saturation pressure of a plane wave, $P_{sat,p}$, is a function of the medium propagation speed c_o, the medium density ρ_o, the nonlinear propagation constant β, the source frequency f, and the distance from the source x. The subscript p indicates plane wave propagation.

2.4 Summary

Plane wave propagation is the simplest scenario for analyzing nonlinear conditions. Plane waves are relatively easy to analyze mathematically, which is why time was spent developing certain ideas from them. From the analysis, it was discovered that there exists a condition when a sinusoidal waveform no longer exhibits linear conditions, i.e., that point when the acoustic shock parameter approaches a value of 1. This condition can be seen in Figure 2.1(c). If the wave continues to propagate until the shock parameter reaches 3 (Figure 2.1(e)) then the signal will no longer be sinusoidal but sawtooth in shape.

It was shown that for large shock parameters the amount of energy associated with the fundamental frequency diminishes. At large values of σ, the higher harmonics obtain a larger percentage of the overall energy. This situation led to the development of the saturation pressure equation for plane waves.

The saturation pressure for a plane wave $P_{sat,p}$ has several variables associated with it. However, the dominant variables in the equation are the source frequency f and the distance traveled x. This states that if the source frequency is increased, then the saturation pressure level will decrease. Similarly, if a wave is measured at a farther distance from the source, then the theoretical saturation level will decrease.

CHAPTER 3

SATURATION IN SPHERICALLY CONVERGING WAVES

The previous chapter developed the saturation equation for plane wave propagation. From that analysis, it is now possible to continue with the development of the saturation equation for a converging beam. Some of the results will be similar to the plane wave propagation, but with a few changes or additions. Recall from the previous chapter that the saturation equation for a plane wave is:

$$P_{sat,\,p} = \frac{\rho_o c_o^3}{2\beta \cdot f \cdot x} \tag{3.1}$$

whereas the theoretical equation for saturation in a converging wave is [5]

$$P_{sat} = \frac{\rho_o c_o^3}{2\beta \cdot f \cdot F} \cdot \frac{G}{\ln(G)} \tag{3.2}$$

Equation (3.2) is almost the same as Eq. (3.1) except for two parameters. Instead of saturation being a function of the distance travelled by the wave *x*, it is now a function of the transducer's focal length *F*. In addition to the focal length dependence, a new parameter *G* is added to account for the focusing gain of the spherical transducer. Equation (3.2) is developed in this chapter, and it is used to determine the theoretical pressure saturation level for each transducer used in the experiment.

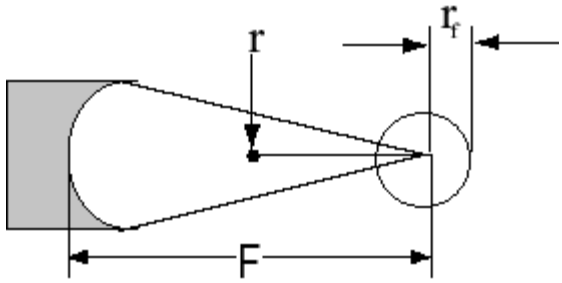
3.1 Saturation in Focused Waves

The theory of acoustic saturation for a converging wave was developed in 1959 [10], and has been analyzed recently in [5]. The work in 1959 [10] was developed for the velocity amplitude at the focus without the presence of diffraction effects. That work described an

equation for the particle velocity of an ultrasonic wave in terms of the fundamental component (cosine term in Eq. (3.3)) and the second harmonic (sine term in Eq. (3.3)), and it is given as [10]

$$v = \frac{Fv_o}{r}\cos(\omega \cdot t + k \cdot r) - \frac{\beta \cdot k}{2 \cdot r(c_o)}\left(\ln\frac{F}{r}\right) \cdot (Fv_o)^2 \sin 2(\omega \cdot t + k \cdot r) \tag{3.3}$$

where r is the distance traveled by the wave, F is the focal length, k is the wave number, v_o is the velocity at the source, c_o is the speed of sound in the medium, and β is the nonlinear propagtion



constant. Similar to plane wave propagation, the converging waves travel in the medium and eventually will begin to exhibit characteristics of nonlinearity. At some point, as is the case with saturation, the wave is assumed to become a sawtooth wave. Figure 3.1 shows the propagating wave becoming a sawtooth at the point designated r from the transducer. The value for r_f is the radius of the focus, assuming the focal region is circular.

Figure 3.1: Geometry used for determining saturation in a converging wave [5].

When the travelling wave becomes a sawtooth, the amplitude of the first harmonic is assumed to be twice the amplitude of the second harmonic [5], [10]. Taking the Fourier Transform of the right side of Eq. (3.3) and setting the first harmonic amplitude equal to twice the second harmonic amplitude gives

$$2 \cdot (F \cdot v_o) = \frac{\beta \cdot k}{c_o} \cdot \ln\left(F/r\right) \cdot (F \cdot v_o)^2 \tag{3.4}$$

$$\text{eq}(4) \Rightarrow 2 \cdot (F \cdot v_o) = \frac{\beta \cdot k}{c_o} \cdot \ln\left(F/r\right) \cdot (F \cdot v_o)^2 \Rightarrow$$

$$2 = \frac{\beta \cdot k}{c_o} \cdot \ln\left(F/r\right) \cdot (F \cdot v_o) \Rightarrow k = \frac{\omega}{c_o} \Rightarrow$$

To get the point r, at which the travelling wave becomes a sawtooth, Eq. (3.4) must be solved for r:

$$2\frac{c_o^2}{\beta\cdot\omega\cdot F\cdot v_o}=\ln\left(\frac{F}{r}\right)\Rightarrow$$
$$r=F\cdot e^{-L/F}\Rightarrow L=\frac{2\cdot c_o^2}{\beta\cdot\omega\cdot v_o} \tag{3.5}$$

This result for r is the same as Eq. (A4) in [5]. Using methods described in [11], the expression for the velocity at the focus can be written as [5], [10]

$$v_f=\frac{F\cdot v_o}{r_f}\left[1-\frac{1}{\pi}+\frac{2\beta}{\lambda\cdot c_o}Fv_o\ln\left(\frac{F}{r_f}\right)\right]^{-1} \tag{3.6}$$

If the limit is taken as the applied velocity approaches ∞, then the third term in the brackets is the determining factor, and

$$\lim_{v_o\to\infty}v_{f.lim}=\frac{F\cdot v_o}{r_f}\left[\frac{2\beta}{\lambda\cdot c_o}Fv_o\ln\left(\frac{F}{r_f}\right)\right]^{-1}=\frac{\lambda\cdot c_o}{2\beta\cdot r_f\ln\left(F/r_f\right)} \tag{3.7}$$

The distance for r_f can be expressed in terms of the source wavelength λ, the focal length F, and the transducer radius a, when the aperture angle of the transducer is small [5], [10]:

$$r_f=\frac{\lambda\cdot F^2}{\pi\cdot a^2} \tag{3.8}$$

The focusing gain, G, can be expressed as [5]

$$G=\frac{\pi\cdot a^2}{\lambda\cdot F} \tag{3.9}$$

Equation (3.9) can be substituted back into Eq. (3.8), which gives

$$r_f=\frac{F}{G} \tag{3.10}$$

Equation (3.10) can be subsituted into Eq. (3.7), which yields the velocity limit at the focus as

$$v_{f.\,lim}=\frac{\lambda\cdot c_o}{2\beta\cdot r_f\ln\left(F/r_f\right)}=\frac{\lambda\cdot c_o}{2\beta\cdot\frac{F}{G}\ln\left(F/(F/G)\right)}=\frac{c_o^2}{2\beta\cdot F\cdot f}\cdot\frac{G}{\ln(G)} \tag{3.11}$$

The velocity is converted to the saturation pressure at the focus by multiplying with the density ρ_o and speed of sound c_o, which yields the saturation equation:

$$P_{sat}=\rho_o\cdot c_o v_{f.\,lim}=\frac{\rho_o\cdot c_o c_o^2}{2\beta\cdot F\cdot f}\cdot\frac{G}{\ln(G)}=\frac{\rho_o\cdot c_o^3}{2\beta\cdot F\cdot f}\cdot\frac{G}{\ln(G)}$$

This equation is the same as Eq. (3.1) in this chapter.

3.2 Theoretical Results for the Transducers Used

From Eq. (3.2), the saturation for a spherically focused transducer can be predicted. In this experiment there were seven transducers used, as shown in Table 3.1. Each transducer used was 1.9 cm in diameter (Matec, Inc. or Panametrics, Inc.). Also shown in Table 3.1 are the four nominal focal lengths used. The diameter and nominal focal length are used to describe the transducer by its f/#, where # is the ratio of the nominal focal length to the diameter of the transducer. For example, f/3 designates a transducer with a nominal focal length three times longer than its diameter. The 9-MHz transducers included an f/3 and an f/2; the 6-MHz transducers included an f/2 and an f/1; the 3-MHz transducers included an f/2 and an f/1; and the 7.5-MHz transducer was an f/4.5. Each of the measured values in Table 3.1 were obtained using the wire technique [12]. A complete list of each transducer's characteristics is given in Appendix A.

Table 3.1: Nominal and measured transducer characteristics.

Transducer Serial #	f/#	Nominal Frequency (MHz)	Measured Frequency (MHz)	Nominal Focal Length (cm)	Measured Focal Length (cm)	Measured -6dB beamwidth (cm)
98C164	3	9	8.37	5.7	5.2	$526.7\square10^{-4}$
00064	2	9	8.23	3.8	3.9	$436.7\square10^{-4}$
98C151	2	6	5.64	3.8	4.0	$447.6\square10^{-4}$
00068	1	6	5.58	1.9	2.1	$325.8\square10^{-4}$
00059	2	3	2.83	3.8	4.3	$420.3\square10^{-4}$
98C160	1	3	2.82	1.9	2.1	$466.2\square10^{-4}$
V380*	4.5	7.5	6.55	8.55	7.1	$859.1\square10^{-4}$

* Panametrics transducer

3.2.1 Determining the gain factor

The value for the gain of a focusing transducer can be obtained using different methods. The first is simply an application of the geometry of the transducer. The value for *G* is [5], [11]

$$G = \frac{z_r}{F} = \frac{\pi(a^2)}{\lambda F} \tag{3.12}$$

where z_r is the Rayleigh distance, and *F* is the focal length. The Rayleigh distance is described by the radius of the transducer *a* and the wavelength of the source pulse λ in Eq. (3.12). Typically this value for *G* is applicable only when the aperature angle (ϕ in Figure 3.2 below) is small or when the focal length is long compared to the transducer's diameter. Four different focal lengths were used in this experiment. Table 3.2 shows the theoretical aperture angles for all four nominal focal lengths. For the focusing gain of Eq. (3.12), it would be expected that the f/4.5 and f/3 transducers would have the best approximation for *G*.

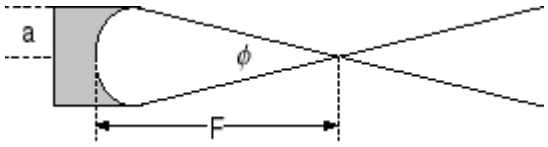


Figure 3.2: Geometry of spherically focused transducer.

Table 3.2: Theoretical aperture angles for each nominal focal length.

f/#	*F* (cm)	ϕ
4.5	8.55	12.7°
3	5.7	18.9°
2	3.8	28.1°
1	1.9	53.1°

There is another approach that may be applied to find an approximate value for the focusing gain of the transducer. This technique uses the square root of the ratio of the source area A_s to the focal area A_f [1], [13], [14]:

$$G = \sqrt{\frac{A_s}{A_f}} \tag{3.13}$$

For each transducer, the diameter is known to be 1.9 cm, which gives a radius of 0.95 cm, and $A_s = \pi \cdot (0.95)^2$. Assuming the cross-sectional focal area is circular, the -6 dB beamwidth can be used to calculate A_f. The -6 dB beamwidth is a value that is determined by using the wire technique [12]. A_f is then calculated [13], [14]:

$$A_f = \pi \cdot \left(\frac{-6 \text{ dB beamwidth}}{2} \right)^2 \bullet \left(10 \Big/ 3 \cdot \ln(10) \right)$$

The cross-sectional area of the focus is considered to be the area over which the peak-to-peak pressure is at least 36.8%, or e^{-1}, the value of the peak-to-peak pressure at the focus [13]. However, the -6 dB beamwidth is known from Table 3.1, and it can be used to determine the focal area. The factor $\left(10 \Big/ 3 \cdot \ln(10) \right)$ is introduced because the -6 dB beamwidth is used to determine the area of the focus. To get the factor the following conversion is done:

$$6 \text{ dB} \cdot (\text{factor}) = 20 \log_{10}(e^{-1})$$

$$\text{factor} = \frac{10 \log_{10}(e^{-1})}{3} \Rightarrow \log_{10}(e^{-1}) = \frac{1}{\ln(10)}$$

$$\text{factor} = \frac{10}{3 \cdot \ln(10)}$$

The value for the -6 dB beamwidth is the only value that varies for each transducer. The calculation for G is as follows:

$$G = \sqrt{\frac{A_s}{A_f}} = \sqrt{\frac{\pi \cdot (.95)^2}{\pi \cdot \left(\left(\frac{-6 \text{ dB beamwidth}}{2} \right)^2 \bullet \left(10 \Big/ 3 \cdot \ln(10) \right) \right)}} \Rightarrow$$

$$G = \frac{0.95}{\left(\frac{-6 \text{ dB beamwidth}}{2} \right)} \sqrt{\frac{3 \cdot \ln(10)}{10}} \qquad (3.14)$$

The values for G were calcualted using both methods and then the methods were compared. The results of those calculations are given in Table 3.3. The wavelength λ was calculated assuming a propagating medium with c = 1500 m/s. The results of the calculations shown in Table 3.3 show that both methods agree. To calculate the predicted acoustic saturation pressure, the value of G from Eq. (3.12) will be used. The one significant discrepancy in the gain calculations is the 2.83 MHz transducer result. This will be considered when the results are

analyzed. If for some reason the theoretical saturation does not compare with the experimental result, then the theoretical equation may need to use the gain calculated using the area technique.

Table 3.3: Table of values used to calculate G two different ways.

Frequency[1] (MHz)	λ (cm)	F[1] (cm)	G^*	-6 dB beamwidth[1] (cm)	G^{**}
8.37	0.0179	5.2	30.5	$526.7\square10^{-4}$	30.0
8.23	0.0182	3.9	39.9	$436.7\square10^{-4}$	36.1
5.64	0.0266	4.0	26.6	$447.6\square10^{-4}$	35.3
5.58	0.0269	2.1	50.2	$325.8\square10^{-4}$	48.5
2.83	0.0530	4.3	12.4	$420.3\square10^{-4}$	37.6
2.82	0.0532	1.9	28.0	$466.2\square10^{-4}$	33.9
6.55	0.023	7.1	17.4	$859.1\square10^{-4}$	18.4

*focusing gain calculated from Eq. (3.12) **focusing gain calculated from Eq. (3.14) [1]measured parameters

3.2.2 Nonlinearity in converging waves

The technique of using the cross-sectional focal area to determine the focusing gain is reliable when $\sigma < 1$ [13]. However, this σ is the description of nonlinearity in plane waves, not spherically converging waves. A new parameter σ_m was developed [13] to describe nonlinearity at the focus of medical ultrasonic transducers, and it is defined as [13], [14]:

$$\sigma_m = \frac{2\pi \cdot f \cdot \beta \cdot p_m \cdot F \cdot \ln\left(G + \sqrt{G^2 - 1}\right)}{\rho \cdot c_o^3 \sqrt{G^2 - 1}} \tag{3.15}$$

where f is the source frequency, F is the focal length, β is the nonlinear propagation coefficient, G is the focusing gain, ρ is the medium density, and c_o is the speed of sound. The value p_m is the average of p_c and p_r at the transducer focus. The average is taken because nonlinear propagation causes the peak acoustic pressure for p_c to become significantly larger than p_r.

To convert back to the acoustic shock parameter developed in Chapter 2, the following conversion is made [13]:

$$\sigma = \frac{\sigma_m}{\sin\left(\sigma_m\right)} \tag{3.16}$$

From the development of the acoustic shock parameter in Chapter 2, if σ>3, then the converging acoustic pressure wave is severely distorted.

3.2.3 Determining the theoretical saturation

The theoretical acoustic pressure saturation equation, as developed earlier, is

$$P_{sat} = \frac{\rho_o c_o^3}{2\beta \cdot f \cdot F} \cdot \frac{G}{\ln(G)}$$

The density ρ_o, the speed of sound c_o, and the nonlinear propagation constant β, were held constant at 998 kg·m^{-3}, 1500 m·s^{-1}, and 3.5, respectively, for each calculation of the theoretical P_{sat}. The measured frequencies and focal lengths from Table 3.1, and the focusing gain calculated from Eq. (3.12) were also used. Table 3.4 provides the theoretical P_{sat} for each transducer used in the experiment.

Table 3.4: Frequency, focal length, and gain used to calculate the theoretical pressure saturation level for each transducer.

f (MHz)*	F (cm)*	G**	P_{sat} (MPa)
8.37	5.2	30.5	9.9
8.23	3.9	39.9	16.2
5.64	4.0	26.6	17.3
5.58	2.1	50.2	52.6
2.83	4.3	12.4	19.5
2.82	2.1	28.0	68.3
6.55	7.1	17.4	6.3

*measured parameters **gain calculated from Eq. (3.12)

There are some interesting aspects of the saturation equation that can be seen from Table 3.4. For transducers of the same frequency, but different focal lengths, the transducer with the longest focal length has the lowest saturation level. The transducer with the shortest focal length has the highest pressure saturation level. Also, for transducers of the same focal length, the

transducer with the lowest frequency has the highest pressure saturation level. These are the relations that will be analyzed in the results. Figure 3.3 shows the effect that varying the focal length and the frequency has on the acoustic pressure saturation level.

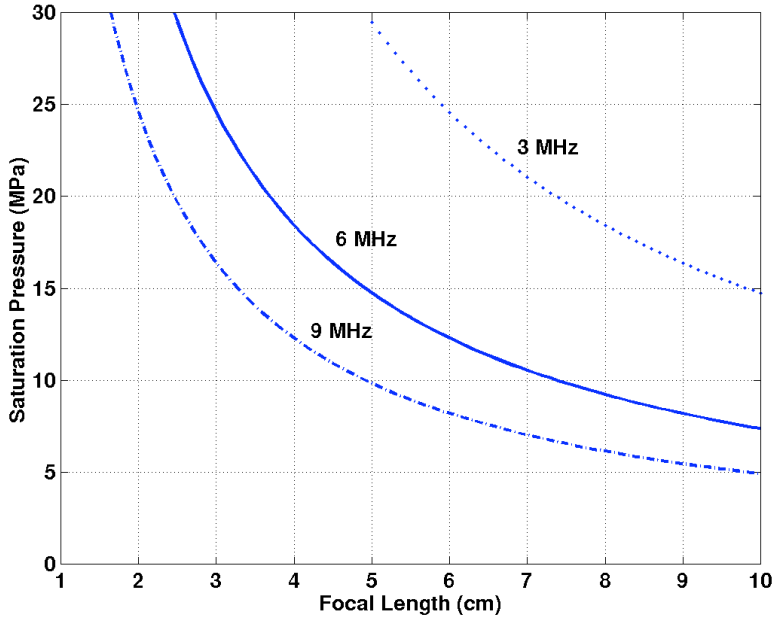


Figure 3.3: P_{sat} levels for 3 MHz (•), 6 MHz (–), and 9 MHz (•–) transducers at various focal lengths for G, ρ_o, c_o, and β equal to 30, 998 kg·m^{-3}, 1500 m·s^{-1}, and 3.5, respectively.

3.3 Summary

This chapter began by developing the acoustic pressure saturation equation for spherically focused transducers. Equation (3.2) was developed from the predicted particle velocity of the propagating wave at the focus. Two major assumptions were made in the derivation of Eq. (3.2). The first was that the focus was circular, and the second was that diffraction effects were negligible.

In Section 3.2, the theoretical saturation pressures were calculated for each transducer. To find the theoretical saturation pressures, the frequency and focal length were measured [12], the speed, density, and nonlinear propagation constant were held constant, and the focusing gain was determined from Eq. (3.12). A second method for finding the focusing gain was provided, and those results were comparable to the first method. Section 3.2 ended with calculations of the

theoretical saturation pressures for each transducer. The P_{sat} values shown in Table 3.4 will be compared to the experimental results in Chapter 5.

CHAPTER 4

EXPERIMENTAL PROCEDURES

A system was developed to find the acoustic pressures at the focus of an ultrasonic transducer. This chapter will discuss both data acquisition and data processing.

4.1 Data Acquisition

The first step in determining the acoustic pressure values at the transducer focus is finding the transducer's beam axis. The beam axis is defined as "a straight line joining points of maximum pulse intensity integral measured at several different distances in the far field" [2, p.1] To determine the beam axis, an automated procedure is used that employs a positioning system (Daedal Inc., Harrison City, PA) with ±2 μm accuracy, an oscilloscope (LeCroy Model 9354TM, Chestnut Ridge, NY), a high-powered, pulsed source (RAM5000, Ritec, Inc., Warwick, RI), a PVDF hydrophone (Marconi, Ltd., Essex, England), transducers (Matec Instruments, Inc., Hopkinton, MA, and Panametrics, Inc., Waltham, MA), a tank with degassed water, and a controlling computer (Dell Pentium-II) as seen in Figure 4.1. The Dell computer contains the C++ (Microsoft Visual C++ 5.0) software used to control the positioning system and the oscilloscope. After data collection is completed, the data are transferred to a workstation (SUN UltraSparc) for off-line analysis. The data acquisition process requires three steps: (1) manual setup, (2) automated determination of the maximum *PII* as a function of distance from the source for calculation of the beam axis, and (3) collection of RF waveforms along the beam axis.

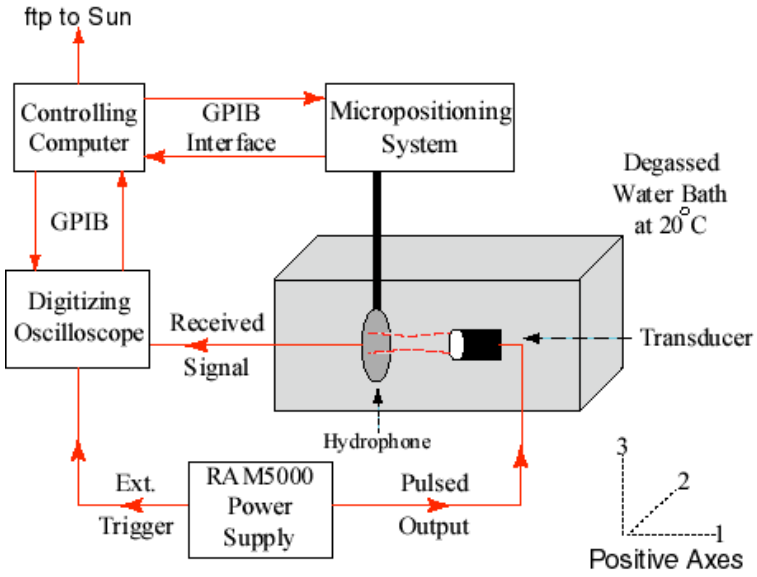


Figure 4.1: Block diagram of experimental system with positive axes convention indicated.

4.1.1 Manual setup

A one-cycle electrical pulse is applied at the transducer's center frequency. The technique for determining the center frequency is described in [12]. Determination of the beam axis begins by manually finding the pressure field's maximum signal as seen on the oscilloscope. The hydrophone is then manually positioned 500 μm in both the negative axis 2 and axis 3 directions and 2 mm in the positive axis 1 direction. Offsetting the hydrophone along axes 2 and 3 allows the automated procedure to scan those axes to where *PII* is maximized within that axis 2-3 plane. Moving the hydrophone in the positive axis 1 direction allows a max *PII* position to be found in the axis 2-3 plane between the transducer and the geometric focus. So, the hydrophone has been moved -500 μm in both the axis 2 and axis 3 directions, and +2 mm in the axis 1 direction. This will be the origin of the scan, or the (0,0,0) point, where the coordinate designation is (axis 1, axis 2, axis 3).

4.1.2 Determination of beam axis

A total of 10 axial positions of maximum *PII* are determined during the scan process, five each for axes 2 and 3. Maximum *PII* positions are collected every 2 mm along axis 1 for a total of 8 mm in the axis 1 direction. All coordinates are given in mm unless specified otherwise. Starting from (0,0,0), the hydrophone moves in 50 μm increments a total of 1 mm in the axis 2 direction in search of the signal with the maximum *PII*, and then returns to (0,0,0). The position

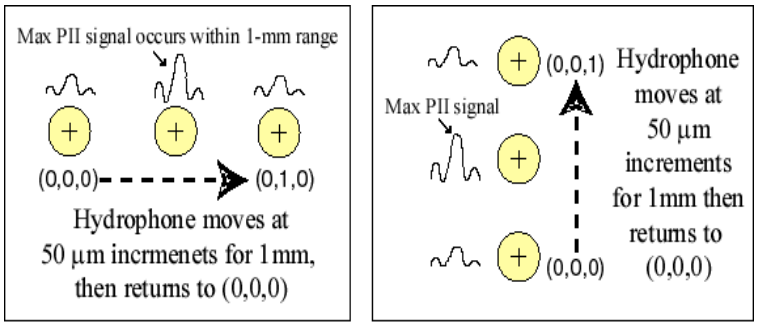


Figure 4.2: Hydrophone movement along axes -2 and -3. The hydrophone scans each axis for a maximum *PII* signal.

where the *PII* was maximized is stored for later use. The same procedure is done for the axis 3 direction. Figure 4.2 shows hydrophone movements for axes 2 and 3. Two positions exist where the *PII* is a maximum, one each for axis 2 and axis 3. The procedure in Figure 4.2 is repeated at (2,0,0), (4,0,0), (6,0,0), and (8,0,0). The two positions found in each axis 1 plane correspond to the axis 2 and axis 3 coordinates of the maximum *PII*, as shown in Figure 4.3. By definition of the beam axis [2], those 5 coordinates of maximum *PII* for each position along axis 1 can be used to create a best fit line that corresponds to the transducer's beam axis.

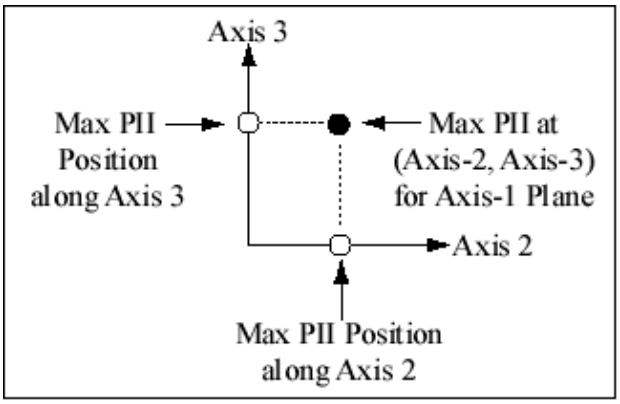


Figure 4.3: Max *PII* positions are found along axis-2 and -3 in the axis-1 plane, and those two positions correspond to the coordinates of the max *PII* in the axis-1 plane.

4.1.3 Waveform collection

After the beam axis is calculated, the hydrophone moves back to (0, x, y), where x and y are the beam axis positions in axis 2 and 3 when axis 1 is at zero. The hydrophone moves in 50-μm increments along the beam axis, and collects RF waveforms at 500 Ms/s. The waveforms are transferred to the Sun workstation where they are analyzed using Matlab®. The C++ code used for the data acquisition is contained in Appendix B.

4.2 Data Processing

In Matlab, a program calculates pressure values using voltage-to-pressure conversions provided with the calibrated hydrophone. The hydrophone was calibrated from 1 MHz up to 20 MHz. The calibration factor varies from 0.041 to 0.043 V/MPa from 1 MHz to 15 MHz. The calibration factor of 0.0425 V/MPa was used during this experiment, which is an average of the factors over the stated frequency range. The code used to analyze the data is given in Appendix C. The processing program reads in the collected data. For each waveform the program calculates the *PII*, finds the maximum (p_c) and minimum (p_r) pressures, and then plots a smooth version of all three. To show that the smoothing does not alter the data significantly, Figure 4.4 shows a typical RF data curve for the maximum p_c values and then a best fit curve to that RF data.

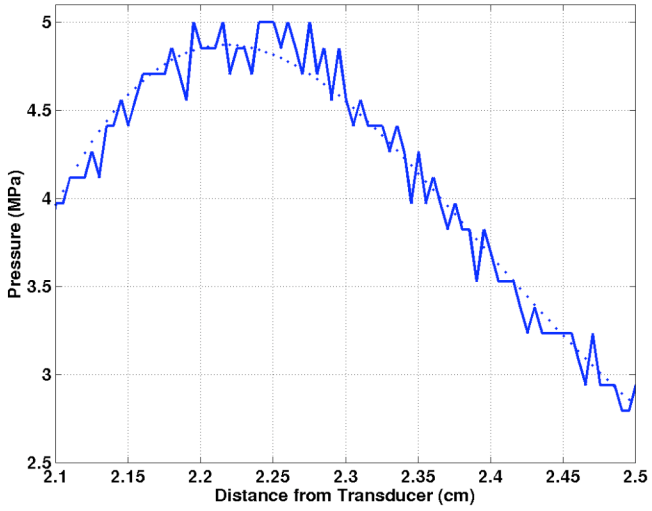


Figure 4.4: Noisy RF collected data (—) and smoothed version (·) of noisy data.

The best-fit curve is applied using two functions in Matlab. The first function is "`polyfit`," which generates a one-dimensional matrix that best fits the measured RF data collected with the hydrophone (solid line in Figure 4.4). Then the "`polyval`" function is performed to apply the dimensions of the distance from the transducer to the curve. As can be seen in Figure 4.4, the "`polyfit`" curve fits the RF data accurately. The best-fit curve maximum occurs around 4.8 MPa, and the RF data fluctuates from approximately 4.7 to 5 MPa around the polyfit maximum. The whole purpose of the smoothing is to eliminate spurious noise associated with measurement of real-time signals. The smoothing allows for the peak of the curve to be taken as the maximum value, instead of a noise spike, which may not be associated with the peak of the curve.

Also analyzed in Matlab is the waveform associated with the maximum value of the *PII* curve. This value corresponds to the location of the focus in water. An example of the waveform at the focus is shown in Figure 4.5. After the waveform at the focus is found, a Fourier analysis can be performed in order to determine the fundamental and harmonic components at the focus, as seen in Figure 4.6. This can also be done for every waveform collected, and a plot of the fundamental and its harmonics can be plotted as a function of distance from the transducer. The frequency analysis program is given in Appendix D.

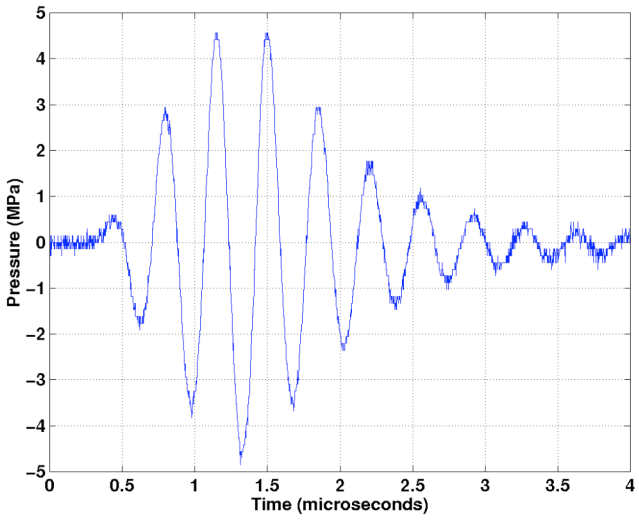


Figure 4.5: A typical linear time-domain pressure waveform at the focus.

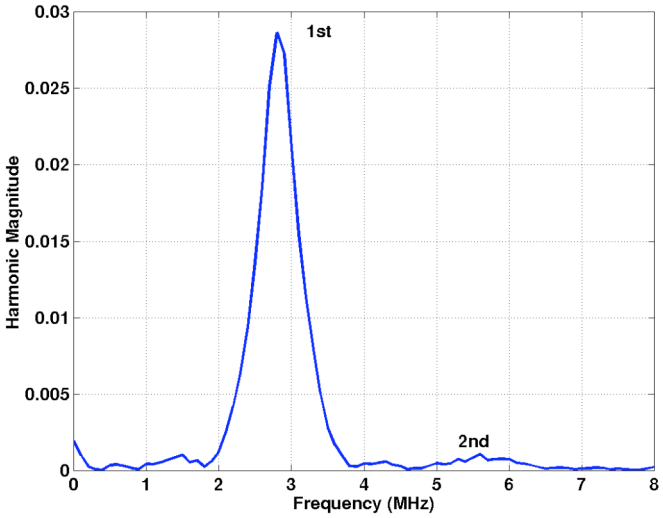


Figure 4.6: A plot of the frequency spectrum for Figure 4.5.

4.3 Summary

This data-acquisition system finds the beam axis for a spherically focused transducer. Processing the data collected along the transducer's beam axis allows determination of the overall peak values for *PII*, p_c, and p_r. Those peak pressures are analyzed in a spreadsheet and plotted as a function of applied input voltage. Those plots allow for saturation of the fields to be seen. Also included was a description of the frequency analysis of the collected waveforms. Those results allow saturation to be seen in terms of the amount of energy going into higher harmonics. All the results are presented in the next chapter.

CHAPTER 5

EXPERIMENTAL RESULTS

This chapter presents the experimental results obtained using the procedures described in Chapter 4. For each transducer used in the experiment, the input zero-to-peak voltage was varied from approximately 100 V up to 2 kV. The upper end of the applied voltage was intended to drive the ultrasonic fields at the transducer's focus into acoustic saturation. The results show measured acoustic pressure plotted versus the applied voltage for each transducer. Also included are results that show the effects of varying a transducer's focal length while keeping the frequency constant and varying the frequency while keeping the focal length constant.

Another part of the results show time-domain plots along with the corresponding frequency-domain plots. This analysis was done for both 9-MHz transducers at a low applied voltage and a high applied voltage. This will explain why applying higher and higher voltages to the transducer does not necessarily increase the acoustic pressures at the focus.

5.1 Results

A total of seven 19-mm-diameter transducers were used in this experiment. The transducers included two different focal lengths for the 9-MHz, 6-MHz, and 3-MHz transducers, and one focal length for the 7.5-MHz transducer. The applied voltage was varied from approximately 0.1 kV up to 2 kV. For each voltage setting the procedure discussed in Chapter 4 was performed. This process yielded maximum p_c and maximum p_r acoustic pressures for each applied voltage. These results were then plotted against the applied voltage. Also included in the plots is the average of the p_c and p_r values. Because of diffraction effects, the p_c is larger than p_r at high applied voltages. To compensate for the difference in the two values, an arithmetic average is taken [5], [13]. For the results of this thesis, this average peak-to-peak

pressure is what will be compared to the theoretical acoustic pressure saturation level. The results obtained from varying the applied input voltage to each transducer are shown in Figures 5.1-5.7.

Each plot shows the average p_c and p_r from the five independent trials. For each average, the standard deviation was calculated and is represented by error bars on each. Also indicated on the plots is the arithmetic average peak-to-peak pressure, which is the middle plot in each of the Figures 5.1-5.7. The standard deviation of that arithmetic average was also calculated, which is indicated by the error bars on each of the middle plots.

Acoustic saturation can be seen in Figures 5.1, 5.2, and 5.7. As the voltage reaches the higher levels the plots of the measured acoustic pressures begin to level off. The predicted saturation level for Figures 5.1 and 5.7 appear lower than the predicted value. The saturation level for Figure 5.2 appears to coincide with the predicted level.

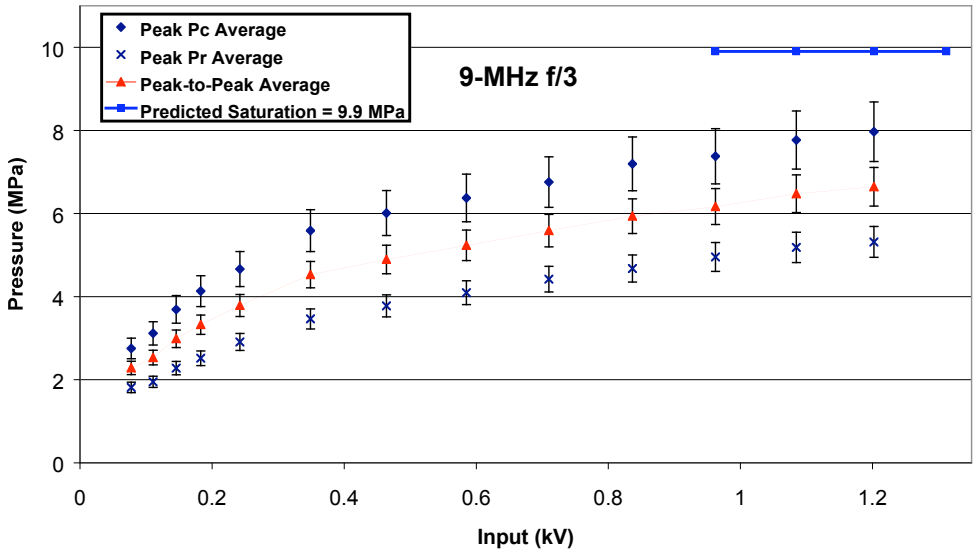
Acoustic saturation is not seen in Figures 5.3-5.6, because the applied voltage was not increased to a high enough level. The limitation in the applied voltage was due to three factors. The first factor was that an excess applied voltage could damage the transducer, the second was that the high acoustic pressures could damage the hydrophone element, and the third was that cavitation set in. The cavitation caused the received acoustic pressure signal to fluctuate. This fluctuation was enough to cause inaccurate measurements.

The 6-MHz f/2 results shown in Figure 5.3 may be approaching saturation, but the applied voltage was not increased enough to be certain. The 6-MHz f/1 transducer results shown in Figure 5.4 looks linear. Both the 3-MHz f/1 and f/2 results shown in Figures 5.5 and 5.6 look almost exponential in acoustic pressure growth. The 3-MHz f/2 results have also increased above the predicted saturation level.

5.2 Saturation in Varying Focal Lengths or Frequencies

From Eq. (3.2), it is expected that a transducer with the same frequency but a longer focal length will saturate at a lower level. Similarly, a transducer with the same focal length but a higher frequency will saturate at a lower level. Figure 5.8 shows the results for varying the focal length, and Figure 5.9 shows the results for varying the frequency. Both figures use their corresponding peak-to-peak average pressure from Section 5.1.

Figure 5.1: Peak average p_c (□) and p_r (×) along with the peak-to-peak average pressure (□) plotted



versus applied voltage for the 9-MHz f/3 transducer. The theoretical acoustic pressure saturation level (■) is 9.9 MPa.

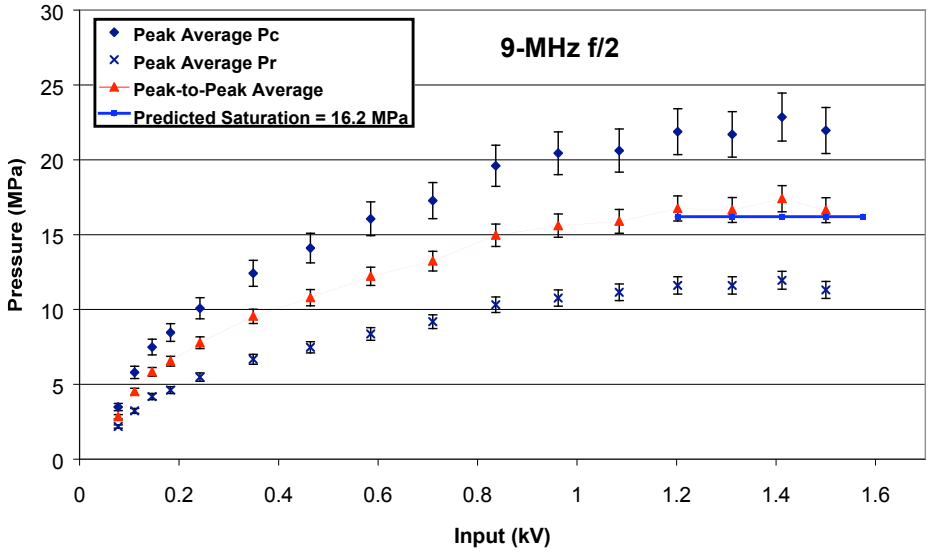
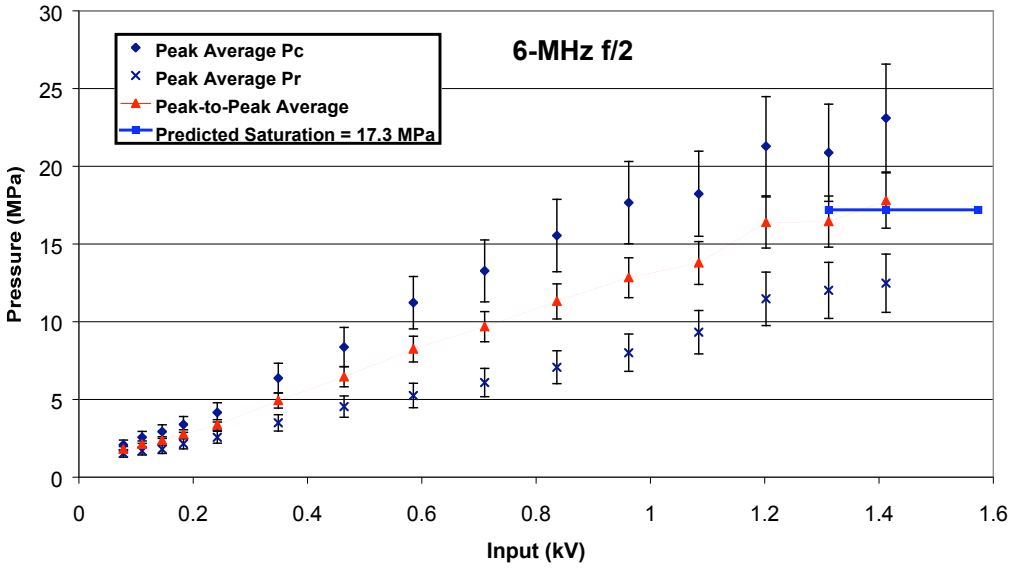


Figure 5.2: Peak average p_c (□) and p_r (×) along with the peak-to-peak average pressure (□) plotted versus applied voltage for the 9-MHz f/2 transducer. The theoretical acoustic pressure saturation level (■) is 16.2 MPa.

Figure 5.3: Peak average p_c (□) and p_r (×) along with the peak-to-peak average pressure (□) plotted



versus applied voltage for the 6-MHz f/2 transducer. The theoretical acoustic pressure saturation level (■) is 17.3 MPa.

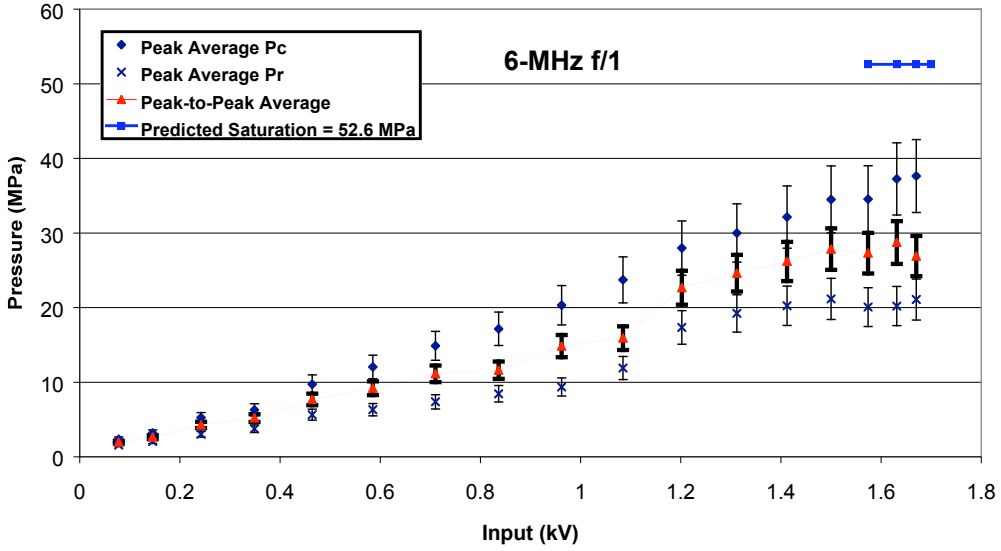


Figure 5.4: Peak average p_c (□) and p_r (×) along with the peak-to-peak average pressure (□) plotted versus applied voltage for the 6-MHz f/1 transducer. The theoretical acoustic pressure saturation level (■) is 52.6 MPa.

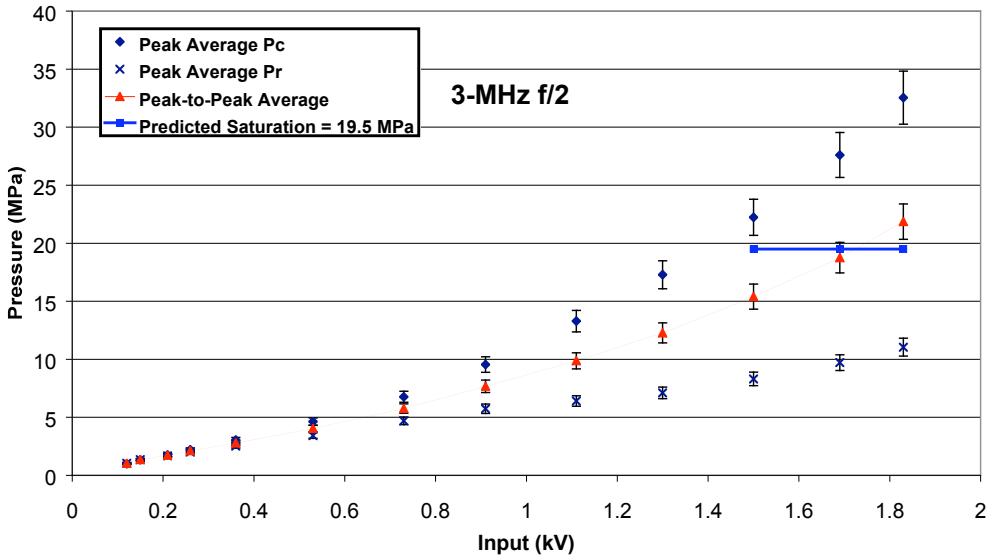
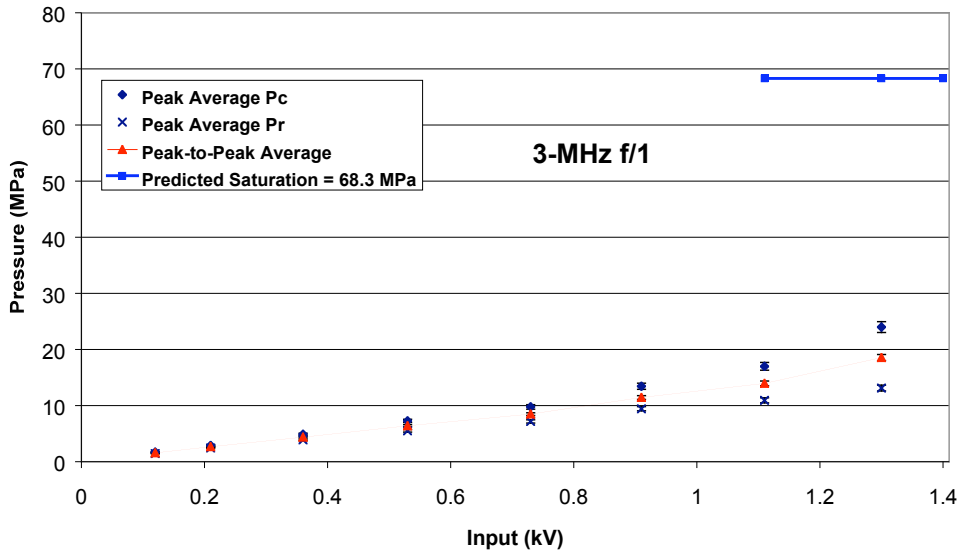


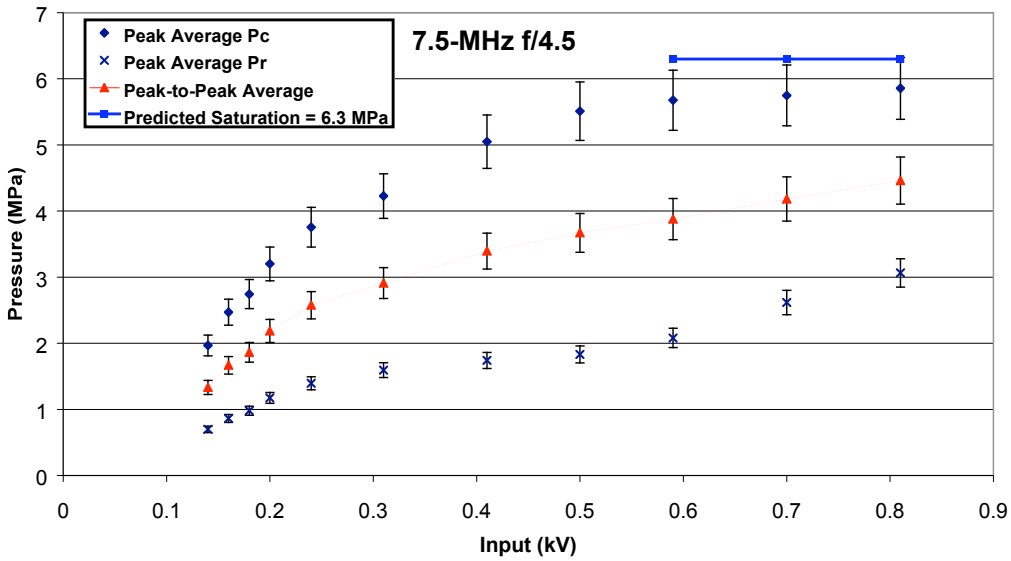
Figure 5.5: Peak average p_c (☐) and p_r (×) along with the peak-to-peak average pressure (☐) plotted versus applied voltage for the 3-MHz f/2 transducer. The theoretical acoustic pressure saturation level (■) is 19.5 MPa.

Figure 5.6: Peak average p_c (☐) and p_r (×) along with the peak-to-peak average pressure (☐) plotted



versus applied voltage for the 3-MHz f/1 transducer. The theoretical acoustic pressure saturation level (■) is 68.3 MPa.

Figure 5.7: Peak average p_c (□) and p_r (×) along with the peak-to-peak average pressure (□) plotted



versus applied voltage for the 7.5-MHz f/4.5 transducer. The theoretical acoustic pressure saturation level (■) is 6.3 MPa .

Figure 5.8 shows the results obtained from the 9 MHz f/3 and f/2 transducers. The f/3 transducer saturates at a lower level than the f/2 transducer. Figure 5.9 shows results obtained from the 9, 6, and 3 MHz, f/2 transducers. Although saturation is not seen for each transducer, the 3-MHz transducer reaches the highest measured acoustic pressures, the 6-MHz reaches the second highest, and the 9-MHz transducer reaches the lowest acoustic pressures. It is expected that this trend would continue into saturation because of the theoretical relationship describing saturation in converging waves.

5.3 Frequency Spectrum Analysis

An interesting way to look at the measured acoustic pressure waveforms is through a frequency spectrum analysis. As stated in Chapter 4, the fundamental and its harmonic components can be analyzed. This analysis provides another way for seeing the amount of nonlinear distortion and saturation associated with measured acoustic pressure waveforms. Two transducers that showed acoustic saturation effects were the 9-MHz f/3 and f/2 transducers. Two different analyses were performed on both transducer's measured data. The first analysis looks at the case when the transducer is driven by a low voltage, and the second when the transducer is driven by a high voltage.

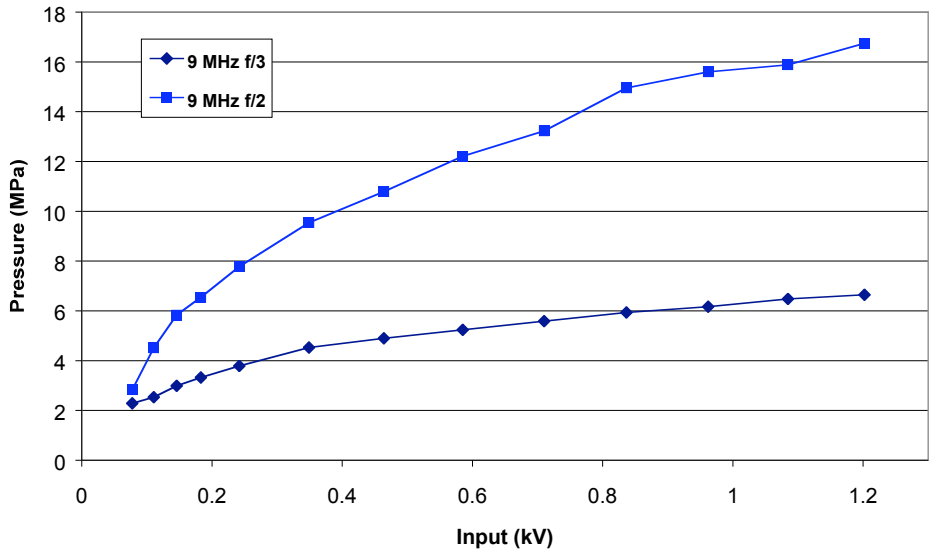


Figure 5.8: Peak-to-peak average pressure for 9-MHz f/3 (□) and 9-MHz f/2 (■) transducers plotted versus applied voltage. As expected, the f/3 transducer saturated at a lower level than the f/2.

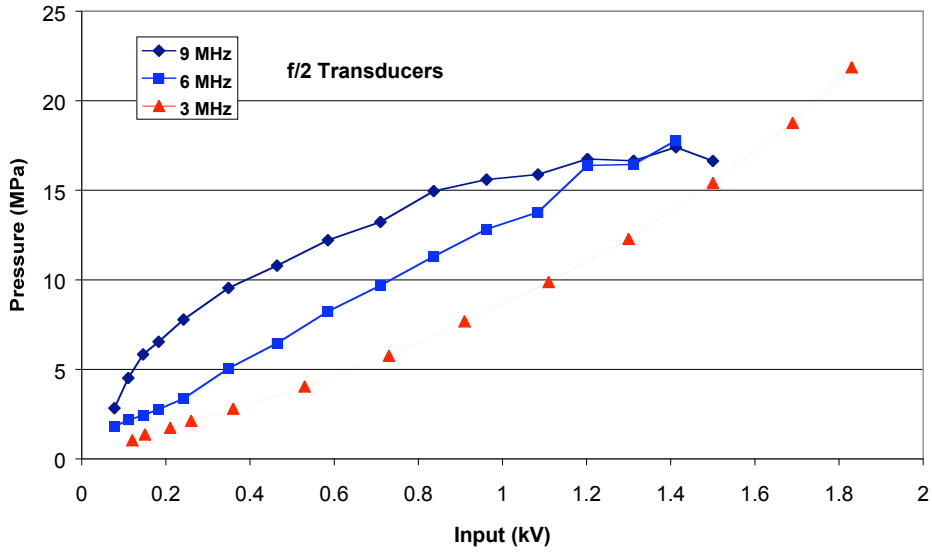
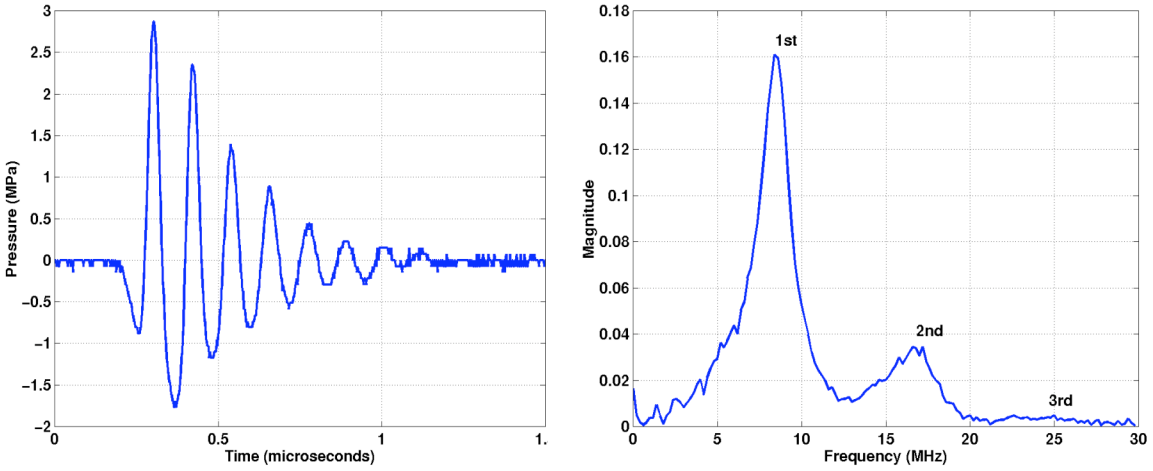


Figure 5.9: Peak-to-peak average pressure for the 9-MHz (□), 6-MHz (■), and 3-MHz (□) f/2 transducers plotted versus applied voltage. The 9-MHz transducer is saturating, the 6-MHz transducer is still rising linearly with applied voltage, and the 3-MHz transducer is rising significantly higher than both the lower frequency transducers. This is the expected trend for same focal length transducers.

Provided in this analysis is a plot of the acoustic pressure waveform at the transducer's focus in water. Performing the fast Fourier transform (FFT) on the waveform at the focus provides the magnitudes of the fundamental, second harmonic, and third harmonic. This presentation of data is similar to data presented in [15]. The magnitudes of the fundamental, second harmonic, and third harmonic can also be plotted as functions of distance from the transducer. This is similar to data presented in [16].

Figure 5.10(a) shows the waveform measured at the focus of the 9-MHz f/3 transducer with the applied voltage low. Figure 5.10(b) corresponds to the frequency spectrum of Figure 5.10(a), and it clearly shows the fundamental and second harmonic of the waveform at the focus. The third harmoic is minimal at the low applied voltage. Figure 5.11 shows the fundamental, second harmonic, and third harmonic plotted versus distance from the transducer. All magnitudes were normalized to the maximum value of the fundamental. This figure shows that the second harmonic is low compared to the fundamental, and that the third harmonic approximately zero.



(a) (b)

Figure 5.10: Time-domain acoustic pressure waveform for 9-MHz f/3 during low applied voltage conditions (a), and the frequency spectrum of the time-domain waveform (b).

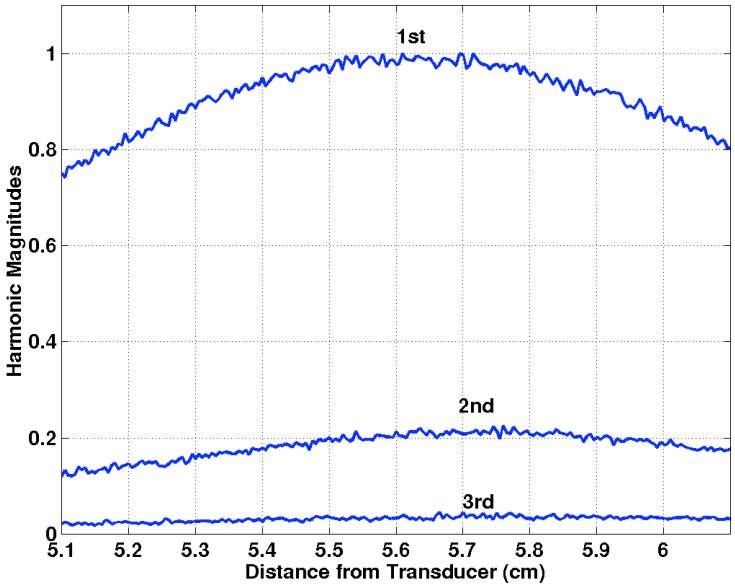
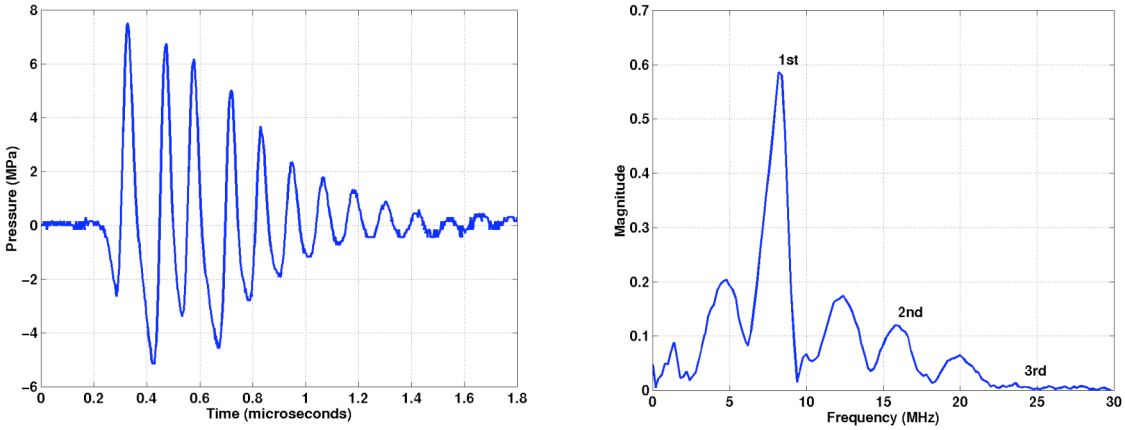


Figure 5.11: Fundamental, second, and third harmonic magnitudes for the 9-MHz f/3 transducer plotted as a function of distance from the transducer for a low applied voltage. All magnitudes normalized to the maximum magnitude value of the fundamental component.

Figures 5.12 and 5.13 are similar to Figures 5.10 and 5.11, respectively. Figures 5.12 and 5.13 are measured data collected for the 9-MHz f/3 at high applied voltages. Figure 5.12(a) shows the acoustic pressure waveform at the focus. This figure shows the highly nonlinear conditions associated with the applied high voltage. Performing the FFT on Figure 5.12(a) yields the result of Figure 5.12(b). The fundamental, second, and third harmonics are labeled in Figure 5.12(b). Figure 5.13 shows the harmonic components plotted versus the distance from the transducer. This figure shows that the second harmonic magnitude is a larger percentage of the fundamental than that shown in Figure 5.11.

Figures 5.14 and 5.15 represent data collected from the 9-MHz f/2 transducer during low applied voltages. Similar to previous figures, Figure 5.14(a) shows the acoustic pressure waveform at the focus, and Figure 5.14(b) shows the frequency spectrum. The time domain waveform in Figure 5.14(a) looks basically sinusoidal because a low applied voltage was used.

Again, the harmonic components in Figure 5.14(b) are labeled. Figure 5.15 is the plot showing harmonic magnitudes versus the distance from the transducer.



(a) (b)

Figure 5.12: Time-domain acoustic pressure waveform for 9-MHz f/3 during high applied voltage conditions (a), and the frequency spectrum of time-domain waveform (b).

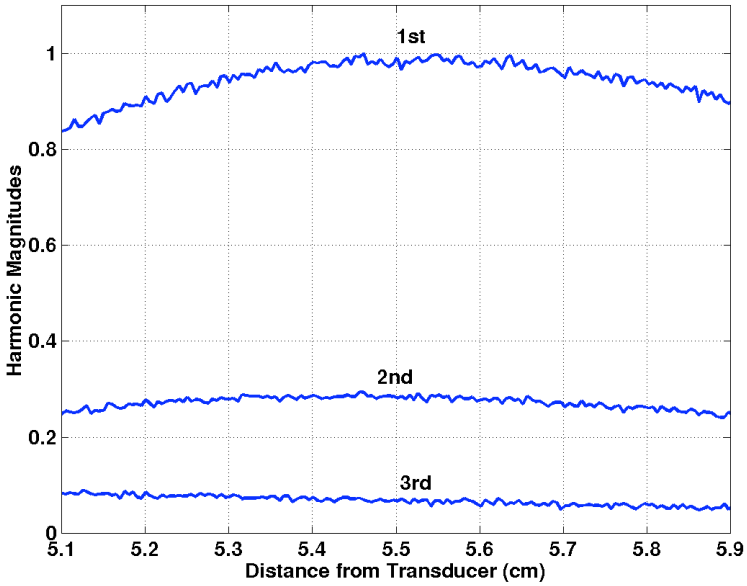
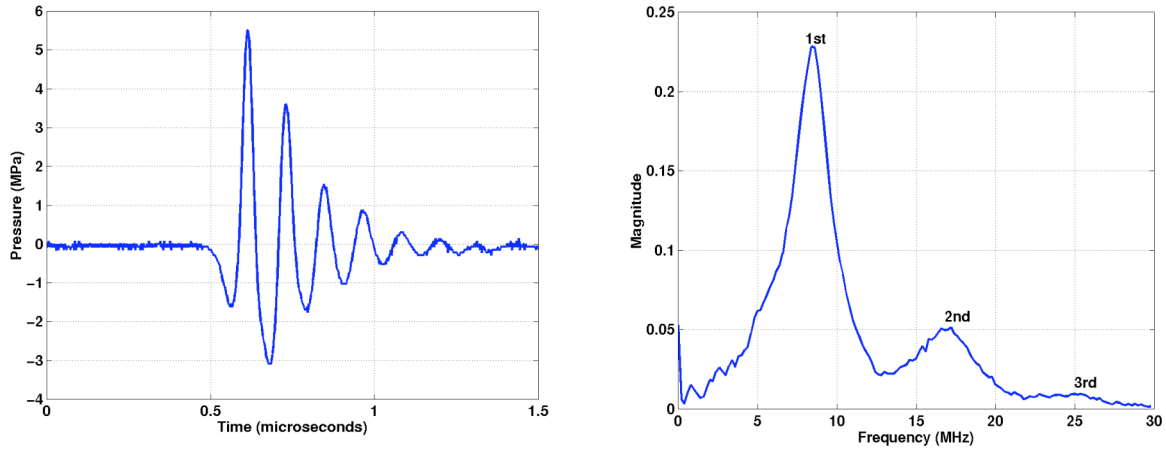


Figure 5.13: Fundamental, second, and third harmonic magnitudes for the 9-MHz f/3 transducer plotted as a function of distance from the transducer for a high applied voltage. All magnitudes normalized to the maximum magnitude value of the fundamental component.



(a) (b)

Figure 5.14: Time-domain acoustic pressure waveform for 9-MHz f/2 during low applied voltage conditions (a), and the frequency spectrum of the time-domain waveform (b).

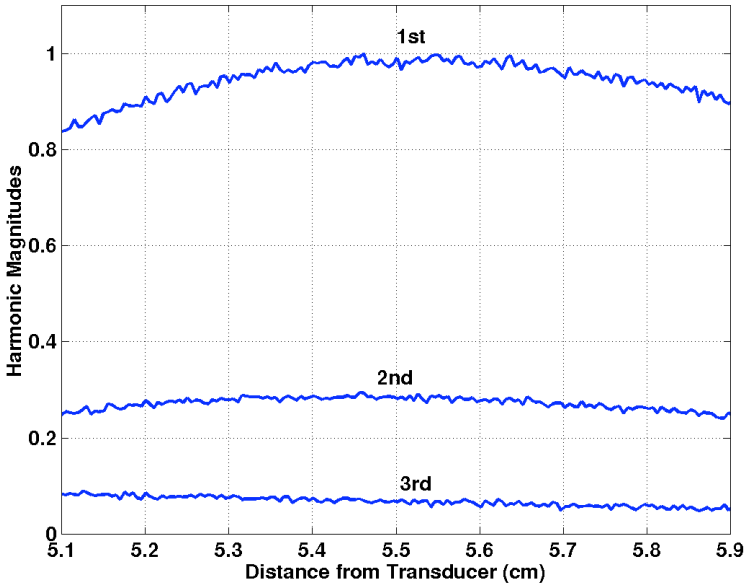
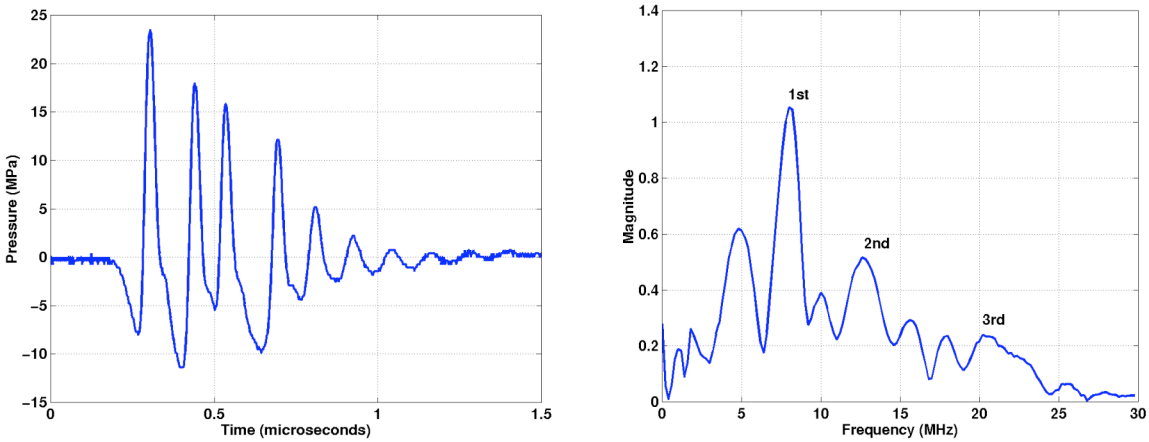


Figure 5.15: Fundamental, second, and third harmonic magnitudes for the 9-MHz f/2 transducer plotted as a function of distance from the transducer for a low applied voltage. All magnitudes normalized to the maximum magnitude value of the fundamental component.

Figures 5.16 and 5.17 are the 9-MHz f/2 data collected during high applied voltage conditions. Figure 5.16(a) shows the highly nonlinear conditions associated with the high applied voltage, and Figure 5.16(b) shows the frequency spectrum of the time domain waveform. Figure 5.17 shows that the second harmonic is approaching approximately half the magnitude of the fundamental. The third harmonic is also receiving part of the applied energy.



(a) (b)

Figure 5.16: Time-domain acoustic pressure waveform for 9-MHz f/2 during high applied voltage conditions (a), and the frequency spectrum of the time-domain waveform (b).

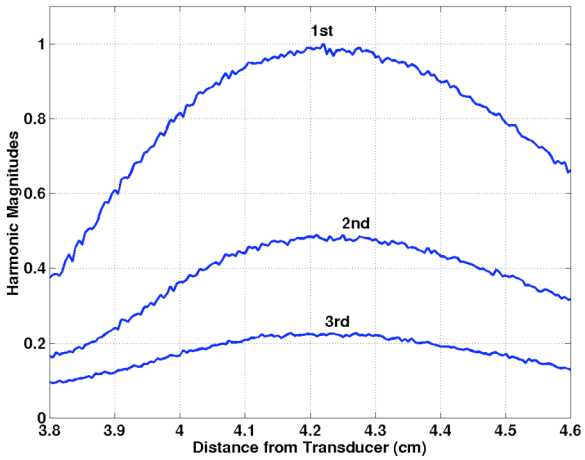


Figure 5.17: Fundamental, second, and third harmonic magnitudes for the 9-MHz f/2 transducer plotted as a function of distance from the transducer for a low applied voltage. All magnitudes normalized to the maximum magnitude value of the fundamental component.

5.4 Summary

This chapter showed results obtained for seven diffferent transducers. The 9-MHz transducers and the 7.5-MHz transducer showed acoustic saturation at the focus. The lower-frequency transducers still showed signs of increasing acoustic pressure at the focus at the high end of the applied voltage. The voltage was limited because the transducers could be damaged from the high voltage, the hydrophone could be damaged from the high acoustic pressures, or cavitation caused the waveforms to become unstable.

Figure 5.8 showed how varying the focal length while keeping the frequency constant effects the saturation level. The shorter focal length transducer saturated at a higher level than the longer focal length. Figure 5.9 showed how varying the frequency while keeping the focal length constant effects saturation level. The lower frequency transducers were reaching higher acoustic pressures. Figures 5.10-5.17 showed the frequency spectrum for both a low applied voltage and a high applied voltage for the 9-MHz f/2 and f/3 transducers. From that information, it was shown that the high applied voltages created a more nonlinear time domain pressure waveform at the focus. The frequency spectrum of that time domain waveform showed significant magnitude contributions from the second harmonics.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

This chapter discusses the results from Chapter 5 and compares those results to existing studies. This chapter also presents conclusions made from the results.

6.1 Discussion

Literature exists that analyzes the effects of acoustic saturation in spherically focused waves. In [14], a 3.5-MHz f/4 transducer was analyzed that had a focusing gain of 6.3. The experimental results for that transducer show acoustic saturation at approximately 2 MPa. If the theoretical calculations were done using the parameters stated in [14], then the P_{sat} would be approximately 5.9 MPa. This is higher than what was measured in the experiment of [14]. Another transducer analyzed was a 2.5-MHz f/4.6 transducer with a focusing gain of 2.5. However, the nonlinear propagtion was easier to attain in the higher frequency transducer, so the results focused on the first transducer. The study [14] agreed with the results of this thesis in two ways. First, the experimental results in [14] saturated at a lower level than the theoretical saturation level, which agrees with the results obtained for the 9-MHz f/3 transducer. This underestimation in the experiment may be due to the longer focal length transducers. Secondly, the fact that the lower frequency transducer was more difficult to drive into nonlinear conditions agrees with the results obtained in this thesis. Although the focusing gains are much higher in this thesis, the lower frequency transducers did not saturate because they were more difficult to drive into nonlinear conditions.

In another study [17], a 0.8-inch diameter, 450-kHz transducer was analyzed. The waves were not focused, but the ultrasonic pressure waves were measured at various distances from the

transducer. When the hydrophone was a relatively long distance away, the transducer saturated at a lower applied voltage, and the saturation pressure was lower. This agrees with experimental results presented for the two focal length results of Figure 5.8. Also shown in [17] is the difficulty associated with driving a transducer's ultrasonic fields to saturation when the hydrophone is close to the transducer. In [17], when the hydrophone was the closest to the transducer, the measured acoustic pressures continued to rise linearly with applied power. This result is similar to the 6-MHz f/2 and f/1 results shown in Figures 5.3 and 5.4. This also agrees with the P_{sat} equation because of the inverse focal length dependence on pressure saturation level.

Two studies [15], [16] looked at the magnitude levels of the fundamental, second, and third harmonics. In [16], the magnitude levels were plotted as functions of distance from the transducer. It was shown that during nonlinear conditions the second and third harmonics become more significant. In [15] measurements were made through tissue (porcine kidney). Similar to [16], plots of the fundamental, second, and third harmonic magnitudes were plotted versus distance from the transducer. The tissue made the acoustic pressure waveforms more nonlinear, which increased the magnitudes of the second and third harmonics.

In this thesis, a frequency analysis was done on the 9-MHz f/3 and f/2 transducers. The results of those plots show the effects of nonlinear propagation, which is similar to other results [15], [16]. The hydrophone did not have a high enough bandwidth to make an accurate measurement of the third harmonic components. The plots of the fundamental, second, and third harmonics in this thesis are intended to show the extent of nonlinear propagation, not necessarily the true magnitudes of the harmonics.

6.2 Conclusions

The experimental results show that some transducers saturated, whereas others did not. The higher frequency, longer focal length transducers demonstrated the most obvious acoustic saturation effects. The 9-MHz f/2 transducer had a peak-to-peak average pressure that was approximately the same as the predicted saturation level. The peak-to-peak average acoustic pressures for the 9-MHz f/3 and 7.5-MHz f/4.5 transducers were lower than the predicted levels.

The other transducers were not driven into acoustic saturation. This was due to equipment and transducer limitations. However, each transducer showed signs of nonlinearity at

the higher end of the applied voltage. Acoustic saturation is known to occur during nonlinear propagation. This implies that if the transducers could be driven harder, then they would eventually saturate. The theory states that it would be more difficult to saturate a shorter focal length, lower frequency transducer, because the pressure levels will be extremely high, as seen in Figure 3.3. This was verified in the experiment.

The results agree with the theory in many aspects. From the definition of P_{sat}

$$P_{sat} = \frac{\rho_o c_o^3}{2\beta \cdot f \cdot F} \cdot \frac{G}{\ln(G)}$$

it would be expected that longer focal length transducers of the same frequency will saturate at a lower level, which was shown in Figure 5.8. The 9-MHz f/3 saturates at a lower acoustic pressure than the 9-MHz f/2, which agrees with the P_{sat} equation. Similarly, the 6-MHz f/2 transducer was reaching lower acoustic pressures than the 6-MHz f/1 for the same applied voltages, as was the 3-MHz f/2 reaching lower acoustic pressures than the 3-MHz f/1 for the same applied voltage.

It would also be expected that transducers of the same focal length but lower frequencies will saturate at higher acoustic pressure levels. This was verified in the results of Figure 5.9. The 9-MHz f/2 saturated while the 6-MHz f/2 was still increasing. The 3-MHz f/2 was rising higher than both. There was one interesting result from this analysis. The 9-MHz transducer was reaching higher pressure levels at lower applied voltages than both the 6- and 3-MHz transducers. This may be attributed to nonlinear propagation. For example, the 9-MHz transducer saturates at a lower applied voltage, which means that the acoustic pressure fileds begin to exhibit characteristics of nonlinearity at lower applied voltages. Nonlinearity indicates that p_c is greater than p_r for spherically focused acoustic waves. The fact that p_c is greater means that the average peak-to-peak pressure will be greater for the 9-MHz transducer at lower applied voltages.

A frequency analysis was performed on both the 9-MHz f/3 and f/2 transducer fields at the focus. During applied high voltage conditions, the time-domain waveform collected at the transducer's focus looked highly nonlinear. When a frequency analysis was performed on the time-domain waveform, the magnitudes of the second and third harmonics helped show the severity of nonlinearity. For example, in Figure 5.17 the fundamental, second, and third harmonics are plotted versus distance from the 9-MHz f/2 transducer. The peak of the second

harmonic is approximately 50% the peak of the fundamental, and the peak of the third harmonic is approximately 20%. Conversely, during linear conditions in Figure 5.15, the peak of the second harmonic is approximately 30% the peak of the fundamental, and the peak of the third harmonic is approximately 10% of the fundamental.

There is an interesting effect shown in Figures 5.12 and 5.16, which are the applied high voltage waveforms collected at the focus. In Figure 5.12(b) there is a half harmonic and a 1.5 harmonic shown in the frequency spectrum and both have a higher peak magnitude than the second harmonic. Figure 5.16(b) is similar in that the half harmonic component is larger than the second harmonic. This must be an additional effect of nonlinear propagation.

The experimental results in this thesis agree with theoretical predictions. They also compare with other studies that have been conducted on acoustic saturation in spherically converging waves. The advantage of this study was that the number of transducers was not limited. Seven transducers with various frequencies and focal lengths allowed for a complete comparison of experimental results and theoretcial predictions.

.

CHAPTER 7

SUMMARY AND FUTURE WORK

7.1 Summary

This thesis provided the background to understand acoustic saturation in spherically focused transducers. It began with a development of nonlinear propagation in plane waves. From the plane wave development, it was shown that as the propagating waves approach saturation higher harmonics receive a higher percentage of the overall energy.

The next development was for the theoretical acoustic pressure saturation equation for spherically focused transducers. That equation was developed from the propagation speed of the travelling acoustic waves [10] into the more familiar P_{sat} equation [5]. For the P_{sat} equation several parameters were needed. The medium density ρ_o, the propagation speed c_o, and the nonlinear propagation constant β were all held constant for calculations of P_{sat}. The focal length F and the frequency f were dependent upon the transducer used and were given in Table 3.1. Two different methods for determining the gain factor G were given. The first method used the geometry of the transducer and its operating frequency, and the second method used the -6 dB beamwidth of the transducer. Table 3.3 shows that the two methods were comparable, so the first method was used to calculate P_{sat}. Table 3.4 shows the theoretical results for P_{sat}.

Chapter 4 discussed the methodology used in measuring the acoustic pressure levels. The system finds the beam axis, which includes the transducer's focus, and then measures pressure values along that beam axis. There are maximum acoustic pressure measurements for p_c and p_r for each waveform collected along the beam axis. Those results were plotted as functions of distance from the transducer to provide axial profiles. The peak value of the axial profiles was then plotted as a function of the zero-to-peak applied voltage. Those plots were given in Chapter 5 for each of the transducers used in the experiment. Also given in the results were frequency

analysis plots of the 9-MHz transducers. The frequency spectra gave a comparison of linear propagation and nonlinear propagation. The comparison was made by looking at the magnitudes of the second and third harmonics.

7.2 Future Work

The experimental system developed in this thesis is useful in determining the acoustic pressure levels in focused ultrasonic transducers. There are many possible experiments that could be done. It would be interesting to analyze three transducers with the same frequency and focal length and see if the saturation is the same for each. This would allow for more information to be collected for spherically focused transducers. A second possible experiment would be to change the medium of propagation. It would be expected from the theory that a more dense medium, for example, would provide a higher thoeretical P_{sat}. A third experiment would be to look at multielement transducers, not just spherically focused. It would be interesting to see what aspect of the multielement transducer dominates.

REFERENCES

[1] F. A. Duck, "Estimating in situ exposure in the presence of acoustic nonlinearity," *Journal of Ultrasound in Medicine*, vol. 18, pp. 43-53, January 1999.

[2] American Institute of Ultrasound in Medicine, *Acoustic Output Measurement and Labeling Standard for Diagnostic Ultrasound Equipment*. Laurel, MD: AIUM Publications, 1992.

[3] American Institute of Ultrasound In Medicine, *Standard for Real-Time Display of Thermal and Mechanical Acoustic Output Indices on Diagnostic Ultrasound Equipment*. Laurel, MD: AIUM Publications,1997.

[4] FDA (Food and Drug Administration), *Information for Manufacturers Seeking Marketing Clearance of Diagnostic Ultrasound Systems and Transducers*. Rockville, MD: Center for Devices and Radiological Health, US Dept. Health and Human Services, 1997.

[5] F. A. Duck, "Acoustic saturation and output regulation," *Ultrasound in Medicine and Biology*, vol. 25, pp. 1009-1018, January 1999.

[6] T. Christopher and E. L. Carstensen, "Finite amplitude distortion and its relationship to linear derating formulae for diagnostic ultrasound systems," *Ultrasound in Medicine and Biology*, vol. 22, pp. 1103-1116, August 1996.

[7] E. L. Carstensen, D. Dalecki, S. M. Gracewski, and T. Christopher, "Nonlinear propagation and the output indices," *Journal of Ultrasound in Medicine*, vol. 18, pp. 69-80, January 1999.

[8] D. T. Blackstock and M. F. Hamilton, Eds., *Nonlinear Acoustics*. San Diego: Academic Press, 1998.

[9] D. G. Zill and M. R. Cullen, *Advanced Engineering Mathematics*. Boston: Prindle, Weber & Schmidt-KENT Publishing, 1992.

[10] K. A. Naugol'nykh and E. V. Romanenko, "Amplification factor of a focusing system as a function of sound intensity," *Soviet Physics-Acoustics*, vol. 5, pp. 191-195, May 1959.

[11] K. A. Naugol'nykh and E. V. Romanenko, "On the propagation of finite-amplitude waves in a liquid," *Soviet Physics-Acoustics*, vol. 4, pp. 202-204, April 1958.

[12] K. Raum and W. D. O'Brien, Jr., "Pulse-echo field distribution measurement technique for high-frequency ultrasound sources," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 44, pp. 810-815, July 1997.

[13] D. R. Bacon, "Finite amplitude distortion of the pulsed fields used in diagnostic ultrasound," *Ultrasound in Medicine and Biology*, vol. 10, pp. 189-195, March/April 1984.

[14] F. A. Duck and M. A. Perkins, "Amplitude-dependent losses in ultrasound exposure measurement," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 35, pp. 232-241, February 1988.

[15] L. Filipczyński, T. Kujawska, R. Tymkiewicz, and J. Wójcik, "Nonlinear and linear propagation of diagnostic ultrasound pulses," *Ultrasound in Medicine and Biology*, vol. 25, pp. 285-299, March/April 1999.

[16] A. C. Baker, "A numerical study of the effect of drive level on the intensity loss from an ultrasonic beam," *Ultrasound in Medicine and Biology*, vol. 23, pp. 1083-1088, July 1997.

[17] J. A. Shooter, T. G. Muir, and D. T. Blackstock, "Acoustic saturation of spherical waves in water," *Journal of the Acoustical Society of America*, vol. 55, pp. 54-62, January 1974.

APPENDIX A
TABLE OF TRANSDUCER CHARACTERISTICS

This table lists the transducer characteristics for each transducer used in the experiment. Included in the table are nominal parameters, measured parameters, and calculated parameters along with the settings used on the Panametrics 5900 pulser/receiver.

Serial No.	98C164	00064	98C151	00068	00059	98C160	V380
Date Data Taken	6/18/98	3/22/99	7/27/98	2/23/99	2/22/99	4/12/98	9/1/99
Nominal Parameters							
Center Frequency (MHz)	9	9	6	6	3	3	7.5
Diameter (mm)	19.05	19.05	19.05	19.05	19.05	19.05	19.05
Focal Length (mm)	57.15	38.1	38.1	19.05	38.1	19.05	85.73
f/#	3	2	2	1	2	1	4.5
Measured Parameters							
Center Frequency (MHz)	8.37	8.23	5.61	5.58	2.83	2.82	6.55
~3dB Bandwidth (MHz)	1.71	1.75	0.73	0.67	0.27	0.24	4.60
Fractional Bandwidth (%)	20.39	21.28	12.93	12.02	9.55	8.55	70.23
Wavelength in Water (mm)	177.75	180.93	265.10	266.94	526.32	523.36	227.20
~20dB Pulse Duration (ns)	610.00	621.80	1414.20	2740.00	3600.00	4569.00	515.00
~6dB Beamwidth (mm)	526.70	436.70	541.40	325.80	420.30	213.10	859.10
~6dB Depth of Focus (mm)	9456.90	5465.80	6897.90	2501.10	5173.90	1890.70	15940.00
Focal Length (mm)	51.54	39.35	40.24	20.88	42.97	21.14	70.69
f1 (MHz)	7.52	7.35	5.25	5.24	2.69	2.71	4.25
f2 (MHz)	9.23	9.10	5.98	5.91	2.96	2.95	8.85
Calculated Parameters							
~6dB Beamwidth (mm)	494.38	192.09	575.72	300.77	610.18	597.13	866.69
~6dB Depth of Focus (mm)	9212.16	1366.26	8376.51	2270.38	4739.25	4564.33	22149.69
Panametrics 5900 Settings							
PRF (kHz)	2	2	2	2	2	1.25	2
Energy (mJ)	16	16	16	16	16	50	16
Damping (W)	26	26	26	26	26	15	26
High Pass Filter (MHz)	1	1	1	1	1	100	1
Low Pass Filter (MHz)	20	20	20	20	20	35	20
Attenuation (dB)	0	0	0	0	0	0	0
Gain (dB)	26	26	26	26	26	20	26
Water Temperature (deg C)	22.4	22	22.4	22.4	22	19.5	22

APPENDIX B
DATA ACQUISITION PROGRAM

This program acquires all the data needed for this thesis. It was written in Visual C++ 5.0.

```
/* -------------------
This is a program which finds the beam
axis of a transducer and then collects
RF signals along that path.  It was
created from the existing 2D program
used for developing transducer beam
scans. The program can be divided into
two parts: the first finds the beam
axis of the transducer, and the second
follows along the beam axis collecting
signals.

To do the first part, the program scans
Axis 2,finds the position of the
maximum PII, and then returns to the
start position. It then scans Axis 3
for the position of the max PII and
then returns to the start position. The
program then moves the hydrophone a
user- specified distance from the
transducer and then repeats the Axis 2
and Axis 3 scans for the max PII in
each direction.

This is done 5 times, and then a best-
fit line is calculated from the
positions of the max PII (5 each for
Axis 2 and Axis 3). This best-fit line
is the beam axis for the transducer.

With the beam axis calculated, the
hydrphone is moved back to the start
and then started on a point on the beam
axis.  The hydrophone increments at
user- specified points along the beam
axis and the RF signal is collected.

Modified by Jason Sempsrott from
programs written by Karen Topp, based
on ROWC.C by Kay Raum, and using ideas
from Nadine Smith, Kate Frazier,
and Dudley Swiney.

Two files will be created: a binary
file with extension .bin contains the
unscaled integer signal points.
The second file is an ASCII file, which
contains waveform and scan information.
-----------------------------------*/

#include <conio.h>
#include <stdio.h>
#include <string.h>

#include <stdlib.h>
#include <math.h>
#include <windows.h>
//#include "c:\qc2\include\msgraph.h"
#include <time.h>
#include "decl-32.h"

#define ERR     (1<<15)      /* Error
detected*/

#define TIMO    (1<<14)
/*Timeout*/

#define RQS     (1<<11)      /* Device
needs service  */

#define N 5/*number of points to fit*/

/* function prototypes */
void introduction(void);
void init_parameter(char [20]);
void init_files(void);
void init_device(void);
void measure(char [20]);

void line_calc(double *coord1, double
*coord2, double *line);
void scan(void);
void printMatrix(double *matrix, int
n);
//void measure2(char [20]);

FILE
       *fopen(),
       *ptr_Dat,
       *ptr1_Dat,
       *ptr2_Dat,
       *ptr_Bin;

double line2[2],    /* will contain
coefficients for axis1-axis2 line*/
       line3[2],   /* will contain
coefficients for axis1-axis3 line*/
       coordd1[N],
          coordd2[N],
          coordd3[N];

char

fileDat[50],file1Dat[50],file2Dat[50],
        fileBin[50],
        fileout[6],
        filenam[50]="d:\\data\\",
        filename[50],filename1[50],
        go1[40],
        go2[40],
        go3[40],
        gostep[30],
        comeback1[40],
        comeback2[40],
```

```
        comeback3[40],
        distance[20],
        wait1[20],
        wait2[20],
        wait3[20],
        wfm[2250],
        scopestat[100],
        scopesettle[10],
needshift1[5],  /* Do we need to shift
the time window -- y/n */
needshift2[5],  /*  for 2nd scan axis
*/
shiftdir[5],   /* Which direction to
shift window */
shiftstring[50],  /* How much do we
need to shift the time */
needavg[5],     /* For averaging trace
A */
avgstring[50],
sweeps[5],      /* How many sweeps for
averaging */
nomath[50],     /* For no math on trace
A */
        op,
        asr1[4],
        asr2[4],
        inter[20],
        ax1incr[5],    // axis1 step
size used for scanning along beam axis
        ax1tot[5],  //axis 1 total
distance for beam scan
        interval1[5];

char ax_number1,
     ax_number2;

unsigned long int   inter1,

       ax1incr_int,

       inter2,

       scopeset;

float              sos,
shifttime,shifttime1;

int     gpib0,
       dev0,
       dev1,
       dev2,
       dev3;


struct data
  {
  unsigned long int
number_point,numscan1,numscan2,numscan3
;
  float
xincr,xmult,xzero,ymult,yzero,timediv,a
xincr;
  };

struct data info;
```

```
main()
  {
  ibrsc(gpib0,1);
  introduction();
  init_parameter(distance);
  init_device();
  init_files();
  measure(distance);
  //measure2(distance);

  // compute line, and scan along it
  line_calc(coordd1, coordd2, line2);
// gives y=Ax+B
  line_calc(coordd1, coordd3, line3);
// gives z=Cx+D
  //how_many();
              // How many scans are to
be done?
  scan();

  ibloc(dev0);/* put scope back to
local mode */
  ibrsc(gpib0,0);

  return 0;
  }

void introduction()
  {
//  _clearscreen(_GCLEARSCREEN);
  printf("\n\n             2D.C  ");
  printf("\n      Scans in 2 directions
");
  printf("\n\n      Data is saved in
C:\\DATA\\  -- 'erase' your data
afterwards!  ");
  printf("\n\n\n Trace A on the LeCroy
scope will be the signal collected");
  printf("\n\n\n\n");
  }


void init_parameter(char distance1[20])
  {
  static char       //interval1[5],
                      interval2[5],
                      //distance1[20],
                      distance2[20],
                      direction1[2],
                      direction2[2],
                      stemp[4];

  unsigned long   check1, check2;


  float    dist1, dist2,ax1totl;

  float   temp, t;

  /* ---------------------------------
Calculate speed of sound in water
------------------------- */

  printf("\n\n Enter water temperature
in degrees C:  ");
```

```
  scanf("%s",stemp);
  temp=(float)atof(stemp);
  printf("\n   Temp = %f degrees
C",temp);
  t = temp/100;
  sos = (float)(1402.7 + (488*t) -
(482*t*t) + (135*t*t*t));
  printf("\n  Speed of sound in water =
%f m/s",sos);


/* -------------------------
          Set up file names
------------------------------ */

  printf("\n\n\n\n Enter a data
filename, up to 8 characters    ");
  scanf("%s",fileout);
  strcat(filenam,fileout);

  strcpy(fileBin,filenam); /* COPY name
to file for binary data */
  strcat(fileBin,".bin");  /*  ADD
".bin" to indicate binary waveform
data*/
  strcpy(fileDat,filenam); /* COPY for
.dat file */
  strcat(fileDat,".dat");  /*  ADD
".dat" to indicate the scope data */
  strcpy(file1Dat,filenam);
  strcat(file1Dat,".dat1");
  strcpy(file2Dat,filenam);
  strcat(file2Dat,".dat2");
/* -------------------------
   Info for Daedal direction axes
------------------------------ */
/* ----------------------------------
             FIRST AXIS
  -------------------------- */

 /* printf(" Enter the first scanaxis
(1-3) : ");
  ax_number1=getche();
  op=getche();*/
  //printf("\n Moving in the positive
(p) or negative (n) direction for
Daedal?: ");
  //scanf("%s", direction1);
  strcpy(direction1,"p");
  printf("\n Axis 2 and 3 stepsize (in
um)    : ");
  scanf("%s",interval1);
  printf(" Axis 2 and 3 total length
(in MM -- can use decimal)  : ");
  scanf("%s",distance1);
  check1 = atoi(distance1);
  if (check1 > 300)
     {
     printf("Cannot travel over 300
millimeters!");
     exit(0);
     }

  strcpy(inter,interval1);
  inter1=atoi(interval1);
  printf("interval1=%s\n",interval1);
  dist1=(float)atof(distance1);
/* ascii to float! */

info.numscan1=(long)(1000*dist1/inter1+
1);  /* truncates to integer */

ltoa(info.numscan1*inter1,distance1,10)
; /* for motor commands in um */

 /* switch(ax_number1)
        {
        case'1':
             dev1=ibfind("axis1");
             strcpy(go1, "MN A5 1V1
D");
             strcpy(comeback1, "MN A5
1V3 D");
             strcpy(wait1," 1R ");
             break;
        case'2':*/
             dev1=ibfind("axis2");
             strcpy(go1, "MN A5 2V1
D");
             strcpy(comeback1, "MN A5
2V3 D");
             strcpy(wait1," 2R ");
       /*    break;
           case '3':
             dev1=ibfind("axis3");
             strcpy(go1, "MN A5 3V1
D");
             strcpy(comeback1, "MN A5
3V3 D");
             strcpy(wait1," 3R ");
             break;
        default:
             printf("\n unknown
axis1");
             op=getche();
//
_clearscreen(_GCLEARSCREEN);
             exit(0);
        }*/


     {
     strcat(go1, "+");
     strcat(go1, interval1);
     strcat(go1, " G ");
     strcat(comeback1,"-");
     strcat(comeback1, distance1);
     strcat(comeback1, " G ");
     }


  /*  ---------------------------------
----------------------------
           Is the first scan axis in
the axial direction?
  -----------------------------------
--------------------------- */

  printf("\n Is the first scan axis in
the axial direction? ");
```

```
  printf("\n    i.e. Do you need the
window to shift in time (y/n) ?:  ");
  scanf("%s", needshift1);
  /* printf(" You said %s.",
needshift1); */
  if (needshift1[0] == 'y')    /*
calculate amount to shift window */
     {
     printf("\n    What direction must
the time window shift? ");
     printf("\n    i.e. positive (p) or
negative (n) ?:  ");
     scanf("%s", shiftdir);
     shifttime = (float)(2 *
((float)inter1/1e6) / sos);
     printf("\n Time shift of window
for each step is %g", shifttime);
     }


  /* ---------------------------------
------------------------------
                        SECOND AXIS
     ---------------------------------
---------------------------- */


  //printf("\n Moving in the positive
(p) or negative (n) direction for
Daedal?: ");
  //scanf("%s", direction2);
  printf("Transducer should face in the
negative axis 1 direction");
  strcpy(direction2,"n");
  printf("\n Distance between maximum
planes   (in um)      : ");
  scanf("%s",interval2);

  printf(" Axis 1 total length (in MM -
- can use decimal)  : ");
  scanf("%s",distance2);

  printf("\nEnter increment for beam
axis scan with respect to axis 1(in
um): ");
  scanf("%s",ax1incr);
  printf("\nEnter total distance for
beam axis with respect to axis 1(in mm)
");
  scanf("%s",ax1tot);


  check2 = atoi(distance2);
  if (check2 > 300)
     {
     printf("Cannot travel over 300
millimeters!");
     exit(0);
     }
  inter2=atoi(interval2);
            /*interval for finding
max*/
  ax1incr_int=atoi(ax1incr);
                   /*interval for
finding beam axis*/
  dist2=(float)atof(distance2);
/* ascii to float! */
  ax1totl=(float)atof(ax1tot);

info.numscan2=(long)(1000*dist2/inter2+
1);  /* truncates to integer */

info.numscan3=(long)(1000*ax1totl/ax1in
cr_int+1);


ltoa(info.numscan2*inter2,distance2,10)
;

   /* switch(ax_number2)
         {
         case'1': */
              dev2=ibfind("axis1");
              strcpy(go2, "MN A5 1V1
D");
              strcpy(comeback2, "MN A5
1V3 D");
              strcpy(wait2," 1R ");
              /*break;
         case'2':
              dev2=ibfind("axis2");
              strcpy(go2, "MN A5 2V1
D");
              strcpy(comeback2, "MN A1
2V3 D");
              strcpy(wait2," 2R ");
       /*     break;
         case '3':
              dev2=ibfind("axis3");
              strcpy(go2, "MN A5 3V1
D");
              strcpy(comeback2, "MN A5
3V3 D");
              strcpy(wait2," 3R ");
              break;
         default:
              printf("\n unknown
axis2");
              op=getche();
//
_clearscreen(_GCLEARSCREEN);
              exit(0);
         }*/


     strcat(go2, "-");
     strcat(go2, interval2);
     strcat(go2, " G ");
     strcat(comeback2,"+");
     strcat(comeback2, distance2);
     strcat(comeback2, " G ");


  /*  ---------------------------------
------------------------------
            Is the second scan axis in
the axial direction?
  ---------------------------------
---------------------------- */
```

```
  printf("\n Is the second scan axis in
the axial direction? ");
  printf("\n    i.e. Do you need the
window to shift in time (y/n) ?:  ");
  scanf("%s", needshift2);
  /* printf(" You said %s.",
needshift1); */
  if (needshift2[0] == 'y')    /*
calculate amount to shift window */
     {
     printf("\n    What direction must
the time window shift? ");
     printf("\n    i.e. positive (p) or
negative (n) ?:  ");
     scanf("%s", shiftdir);
     shifttime = (float)(2 *
((float)inter2/1e6) / sos);

shifttime1=(float)(2*((float)ax1incr_in
t/1e6)/sos);//shifttime for beam axis
scan
     printf("\n Time shift of window
for each step is %g", shifttime);
     }

  /* ---------------------------------
------------------------------
                how long for scope to
wait?
  -----------------------------------
------------------------------ */
  printf("\n\n Enter a 'wait time'
integer value for the scope to
settle.");
  printf("\n   (between 0 and a few
1000 --- 1000 is about 1 second):   ");
  scanf("%s",scopesettle);
  scopeset=atoi(scopesettle);

  /* ---------------------------------
------------------------------
                   Average of Trace A?
  -----------------------------------
------------------------------ */

  printf("\n\n Do you want averages of
trace A (y/n) ?: ");
  scanf("%s", needavg);
  if (needavg[0] == 'y')
     {
     printf("\n   Enter number of
sweeps for average (integer up to
1000): ");
     scanf("%s",sweeps);
     }

  }


void init_device(void)
  {
  static char            quest1[50],
                      quest2[50],
                      quest3[50];

  gpib0=ibfind("gpib0");

/* ---------------------------------
------------------------------
                  Initialize scope
  -----------------------------------
------------------------------ */

  dev0=ibfind("LECROY");   /* ID:
Lecroy */

  ibwrt(dev0,"*CLS",4);  /*  clear
status registers on scope */
  ibwrt(dev0,"WAVEFORM_SETUP SP, 0, NP,
0, FP, 0",36);
                        /* SParse=0 =>
no interval betw. pts.
                           NumPts=0 =>
send all pts.
                           FirstPoint=0
=> start reading at 1st point */

  ibwrt(dev0,"Comm_Format OFF,WORD,BIN
",25);
  ibwrt(dev0,"Comm_ForMat?",12);
  ibrd(dev0,quest1,45);
/*      printf("\n Waveform :
%s",quest1);         */

  ibwrt(dev0,"Comm_Order LO ",14);
/* LeastSigByte first */
  ibwrt(dev0,"Comm_Order?",11);
  ibrd(dev0,quest2,45);
/*      printf("\n Byte Order :
%s",quest2);           */

  ibwrt(dev0,"Comm_Header OFF ",16);
/* Header omitted */
  ibwrt(dev0,"Comm_Header?",12);
  ibrd(dev0,quest3,45);
/*      printf("\n Header :
%s",quest3);          */


/* ---------------------------------
------------------------------
               Get info for TRACE A
(Transmit signal)
   -----------------------------------
------------------------------*/

  ibwrt(dev0,"TA:INSP?
'WAVE_ARRAY_COUNT'",27);  /* TA=trace A
*/
  ibrd(dev0,wfm,1);    /* read one char
to get rid of opening quotes */
  ibrd(dev0,wfm,100);
/*      printf("\n %s",wfm);         */
  sscanf(wfm,"WAVE_ARRAY_COUNT    : %lu
",&info.number_point);

  ibwrt(dev0,"TA:INSP?
'HORIZ_INTERVAL'",25);
  ibrd(dev0,wfm,1);
```

```
  ibrd(dev0,wfm,100);
/*      printf("\n %s",wfm);         */
  sscanf(wfm,"HORIZ_INTERVAL      : %g
",&info.xincr);

  ibwrt(dev0,"TA:INSP?
'HORIZ_OFFSET'",23);
  ibrd(dev0,wfm,1);
  ibrd(dev0,wfm,100);
/*      printf("\n %s",wfm);         */
  sscanf(wfm,"HORIZ_OFFSET        : %g
",&info.xzero);

  ibwrt(dev0,"TIME_DIV?",9);
  ibrd(dev0,wfm,100);       /* note
this doesn't come back in quotes */
/*     printf("\n %s",wfm);
*/
  sscanf(wfm,"%g", &info.timediv);

  ibwrt(dev0,"TA:INSP?
'VERTICAL_GAIN'",24);
  ibrd(dev0,wfm,1);
  ibrd(dev0,wfm,100);
/*      printf("\n %s",wfm);         */
  sscanf(wfm,"VERTICAL_GAIN       : %g
",&info.ymult);

  ibwrt(dev0,"TA:INSP?
'VERTICAL_OFFSET'",26);
  ibrd(dev0,wfm,1);
  ibrd(dev0,wfm,100);
/*      printf("\n %s",wfm);         */
  sscanf(wfm,"VERTICAL_OFFSET     : %g
",&info.yzero);

  printf("\n\n DATA FOR TRACE A ");
  printf("\n\n Sample points per A-scan
: %lu",info.number_point);
  printf("\n  Xincr : %g",info.xincr);
  printf("    Xzero : %g",info.xzero);
  printf("\n  Ymult : %g",info.ymult);
  printf("    Yzero : %g",info.yzero);
  printf("    Time/div :
%g",info.timediv);

  printf("\n\n Number of scan points:
");
  printf("\n Axis[%c] scan points
:%lu",ax_number1,info.numscan1);
  printf("\n Axis[%c] scan points
:%lu",ax_number2,info.numscan2);
  printf("\n\n");


  ibwrt(dev1," F ",3);
  ibwrt(dev2," F ",3);
  }

void init_files()
  {


  if ((ptr_Bin = fopen(fileBin,"wb"))
== NULL)
       {
       printf(" error opening binary
file \n");
       exit(0);
       }

  fclose(ptr_Bin);
  }


void measure(char distance1[20])
  {
  unsigned long int
i,j,k,t,w,u,waitscope,count,

NewDistance,newinter,newdist1,blah1,bla
h2,blah3,

       blah4,extra,imax1,burn,total2,tot
al3;
  int inr,
result,imax,newimax,dummy,couch,//maxva
lue,newmaxval,

pause,data[5000],temp,temp2,temp3,temp4
,temp5[20000],coord1[2000],coord2[2000]
,coord3[2000],temp6 ;            /* for
status result from LeCroy */
  char bugme,
comeback[40],tempos1[40],tempos2[40],ne
wdist3[40],newdist[40],

newdistance1[20],start2[40],start3[40],
go3[40],
         newdistance2[20];
  float  winpos, winpost[20],
posdiv,dist1;// NewDistance;
  static char    cmd[100],      /* for
status response string */
               curv[20000],   /* MAKE
SURE ENOUGH ROOM!!! */
               posstring[60],
name[4],interval1[5]; /* string for
current window position */


  count=1;

  winpos = info.xzero +
(info.number_point*info.xincr)/2 ;
      /*  for moving time window, if
needed (window uses center of trace) */

  /* ---------------------------------
-----------------------
       If we want an average of Trace
A...
  ---------------------------------
----------------------- */
  /*if (needavg[0] == 'y')  /* then
define trace A to be an average */
  /*   {
     strcpy(avgstring,"TA:DEF
EQN,'AVGS(C1)',SWEEPS,");
     strcat(avgstring,sweeps);
```

```
ibwrt(dev0,avgstring,strlen(avgstring))
;
     }
  else                       /* no
averaging -- transmit raw trace A */
     {
     ibwrt(dev0,"CLSW",4);
     strcpy(nomath,"TA:DEF
EQN,'ZOOMONLY(C1)'");
     ibwrt(dev0,nomath,strlen(nomath));
     }

        dev3=ibfind("axis3");
        strcpy(go3, "MN A5 3V1 D");
        strcat(go3, "+");
        strcat(go3, inter);
        strcat(go3, " G ");


  for(k=0;k<=(info.numscan2-1);k++)  /*
Rows */
        {
        ibwrt(dev1," E ",3);
        ibwrt(dev3," E ",3);
       /* if ((ptr_Bin =
fopen(fileBin,"ab")) == NULL)
             {
             printf(" error appending to
transmit binary file \n");
             exit(0);
             }*/
       strcpy(newdistance1,distance1);
       strcpy(newdistance2,distance1);
       blah1=0;
       blah2=0;
       blah3=0;
       blah4=0;


        for(i=0;i<=(info.numscan1-
1);i++)  /* Scans in each row (columns)
*/

             {

for(waitscope=0;waitscope<=scopeset;wai
tscope++)
              {
              printf("\rPlease
wait...");
              }
          /* printf(" Done with one
capture ");  */
          printf(" [%lu of
%lu]",count,(info.numscan1)*(info.numsc
an2));
          count=count+1;

         /*  if (needavg[0] == 'y')  /*
then trace A an average */
         /*     {
              ibwrt(dev0,"CLSW",4);
/* clear sweeps to start average */
         /*     result = 0;
             inr = 0;
             while( (result=0x0100&inr)
!= 256)  /* 256 means math on TA is
done */
             /*   {
                strcpy(cmd,"INR?");

ibwrt(dev0,cmd,strlen(cmd));  /* what
is scope status */
             /*
ibrd(dev0,scopestat,100);
                inr = atoi(scopestat);
                /* printf("\n Waiting
for scope: string is %s", scopestat);
*/
              /* }
             }*/

          ibwrt(dev0,"TA:WF? DAT1",11);
/* Transmit wave */

ibrd(dev0,curv,(info.number_point*2));

          /* ------------------------
-----------------------------
              Go to next step and write
last curve to file
             ------------------------
-------------------------- */

             ibwrt(dev1, go1,
strlen(go1));


             temp4=0;

          for
(j=1;j<=(info.number_point*2);j=j+2)
             {
                    count=(j-1)/2;
                    // The following
four lines contain code
                    temp=(int)curv[j];
             // that changes the
character string "curv"

      temp2=(int)curv[j+1];
      // into an integer value for
manipulation.
                    temp2=temp2 << 8;
             // "curv" is eight bits
and integers are 16 bits.
                    data[count]=temp +
temp2;
      //
      fprintf(ptr1_Dat,"%d\n",data[coun
t]);   //Storing the curv in a file.

      //data[count]=data[count]/1000;


      temp3=(data[count])*(data[count])
;

                    temp4=temp3+temp4;
```

```
      //
      fprintf(ptr2_Dat,"%d\n",temp4);
// Storing the square of the curv in a
file.



      //fputc(curv[j],ptr_Bin);


             }


                   temp5[i]=temp4;
      //           printf("\nThe max
value for curve %d is
%d.\n",i,temp5[i]);



      //           fclose(ptr1_Dat);
      //           fclose(ptr2_Dat);

       }

            while (asr1[1] != 'R')
             {


ibwrt(dev1,wait1,strlen(wait1));
              ibrd(dev1,asr1,3);
              }
          asr1[1] = 'B';
          printf("\r            ");



             /* ---------------  end of
i-loop ---------------*/


temp6=temp5[0];
             for(i=1;i<=(info.numscan1-
1);i++)  /* Scans in each row (columns)
*/
             {
                   if (temp5[i]>temp6)
                   {
                   temp6=temp5[i];
                   imax=i-1;
                   }
                   newimax=imax;
             }

      dist1=(float)atof(newdistance1);


      blah1=(long)dist1; //total
distance from start to finish

      blah2=newimax*inter1;
      coord2[k]=blah2;
      extra=blah1-blah2;
```

```
      ltoa(extra,newdist,10);

      strcpy(comeback1, "MN A5 1V3 D");

      strcat(comeback1,"-");
    strcat(comeback1, newdist);
    strcat(comeback1, " G ");

      strcpy(tempos1,comeback1);



       ibwrt(dev1, comeback1,
strlen(comeback1));

      strcpy(comeback1, "MN A5 1V3 D");

      ltoa(blah2,newdistance1,10);

      strcat(comeback1,"-");
      strcat(comeback1,newdistance1);
      strcat(comeback1," G ");

      ibwrt(dev1,comeback1,strlen(comeb
ack1));



       while (asr1[1] != 'R')
         {

ibwrt(dev1,wait1,strlen(wait1));
         ibrd(dev1,asr1,3);
         }
       asr1[1] = 'B';



  /* ---------------------------------
----------------------
      If we want an average of Trace
A...
  -------------------------------------
---------------------- */
 /* if (needavg[0] == 'y')  /* then
define trace A to be an average */
 /*    {
     strcpy(avgstring,"TA:DEF
EQN,'AVGS(C1)',SWEEPS,");
     strcat(avgstring,sweeps);

ibwrt(dev0,avgstring,strlen(avgstring))
;
     }
  else*/                    /* no
averaging -- transmit raw trace A */
     {
     ibwrt(dev0,"CLSW",4);
     strcpy(nomath,"TA:DEF
EQN,'ZOOMONLY(C1)'");
     ibwrt(dev0,nomath,strlen(nomath));
     }
printf("hello");
```

```
  //for(k=0;k<=(info.numscan2-1);k++)
/* Rows */
        //{



//printf("info.numscan1=%d",info.numsca
n1);
        for(i=0;i<=(info.numscan1-
1);i++)  /* Scans in each row (columns)
*/
          {

for(waitscope=0;waitscope<=scopeset;wai
tscope++)
              {
              printf("\rPlease
wait...");
              }
           /* printf(" Done with one
capture ");  */
           printf(" [%lu of
%lu]",count,(info.numscan1)*(info.numsc
an2));
           count=count+1;

          /* if (needavg[0] == 'y')  /*
then trace A an average */
          /*    {
              ibwrt(dev0,"CLSW",4);
/* clear sweeps to start average */
          /*    result = 0;
              inr = 0;
              while( (result=0x0100&inr)
!= 256)  /* 256 means math on TA is
done */
              /*   {
                 strcpy(cmd,"INR?");

ibwrt(dev0,cmd,strlen(cmd));  /* what
is scope status */
              /*
ibrd(dev0,scopestat,100);
                 inr = atoi(scopestat);
                 /* printf("\n Waiting
for scope: string is %s", scopestat);
*/
                 /*}
              }*/

           ibwrt(dev0,"TA:WF? DAT1",11);
/* Transmit wave */

ibrd(dev0,curv,(info.number_point*2));

           /* ------------------------
-----------------------------
              Go to next step and write
last curve to file
              ------------------------
------------------------- */


             ibwrt(dev3, go3,
strlen(go3));

       strcpy(filename1,file1Dat);
              strcpy(filename,file2Dat);
              itoa(i,name,10);

              strcat(filename,name);
              strcat(filename1,name);
              temp4=0;
            for
(j=1;j<=(info.number_point*2);j=j+2)
              {
                     count=(j-1)/2;
// The following four lines contain
code
                     temp=(int)curv[j];
// that changes the character string
"curv"

       temp2=(int)curv[j+1];
// into an integer value for
manipulation.
                     temp2=temp2 << 8;
// "curv" is eight bits and integers
are 16 bits.
                     data[count]=temp +
temp2;
       //
       fprintf(ptr1_Dat,"%d\n",data[coun
t]);   //Storing the curv in a file.

       //data[count]=data[count]/1000;


       temp3=(data[count])*(data[count])
;

                     temp4=temp3+temp4;

       //fputc(curv[j],ptr_Bin);
        }
                     temp5[i]=temp4;
        }
             while (asr1[1] != 'R')
              {

ibwrt(dev3,wait1,strlen(wait1));
              ibrd(dev3,asr1,3);
              }
           asr1[1] = 'B';
           printf("\r            ");

     /* ---------------  end of i-loop
---------------*/

temp6=temp5[0];
imax1=0;
             for(i=1;i<=(info.numscan1-
1);i++)  /* Scans in each row (columns)
*/
             {

                     if (temp5[i]>temp6)
                     {
```

```
      temp6=temp5[i];
                  imax1=i-1;
                  }

                  newimax=imax1;
            }

      dist1=(float)atof(newdistance2);

      blah3=(long)dist1; //total
distance from start to finish
      blah4=newimax*inter1;
      coord3[k]=blah4;
      extra=blah3-blah4;
      ltoa(extra,newdist3,10);
      strcpy(comeback3, "MN A5 3V3 D");
      strcat(comeback3,"-");
    strcat(comeback3, newdist3);
    strcat(comeback3, " G ");
       ibwrt(dev3, comeback3,
strlen(comeback3));

      strcpy(comeback3, "MN A5 3V3 D");
      ltoa(blah4,newdistance2,10);
      strcat(comeback3,"-");
      strcat(comeback3,newdistance2);
      strcat(comeback3," G ");
      ibwrt(dev3,comeback3,strlen(comeb
ack3));



       while (asr1[1] != 'R')
         {

ibwrt(dev3,wait1,strlen(wait1));
          ibrd(dev3,asr1,3);
          }
       asr1[1] = 'B';

/*     if (needshift1[0]=='y')  /* then
put time window back */
/*          {
            winpos = info.xzero +
(info.number_point*info.xincr)/2 ;  /*
orig. position */
/*          posdiv = winpos /
(10*info.timediv) * 10;

gcvt((double)posdiv,5,posstring);
            strcpy(shiftstring, "TA:HPOS
");
            strcat(shiftstring,
posstring);
            ibwrt(dev0, shiftstring,
strlen(shiftstring) );
            }  */

       ibwrt(dev3," F ",3);

       ibwrt(dev1," F ",3);

       //fclose(ptr_Bin);
```

```
////////////////////////////////////
////////////////////////////////////
//////



       ibwrt(dev2," E ",3);

       /*while (asr2[1] != 'R')
             {
             printf("asr2=%c
\r",asr2[1]);

ibwrt(dev2,wait2,strlen(wait2));
             ibrd(dev2,asr2,3);
             }*/

       asr2[1] = 'B';

       /* ------------------------------
---------------------------
            If time shift of window is
needed for second scan axis
       ------------------------------
--------------------------- */

       if (needshift2[0]=='y')
          {
          if (shiftdir[0]=='p')
             {
                   winpos = winpos +
shifttime/2;
             }
               else
             winpos = winpos -
(shifttime);
               posdiv = winpos /
(10*info.timediv) *10;  /* pos. in div.
*/
                 /* note LeCroy needs
position in terms of divisions */

//printf("posdiv=%f\n",posdiv);

gcvt((double)posdiv,5,posstring);  /*
convert position to string */
          strcpy(shiftstring, "TA:HPOS
");
          strcat(shiftstring,
posstring);
          ibwrt(dev0, shiftstring,
strlen(shiftstring) );
          }

       ibwrt(dev2, go2, strlen(go2));
        for(w=1;w<=1000;w++)
  {
        for(i=1;i<=10;i++)
        {
              printf("i=%d\r",i);
        }
  }
```

```
        //printf("inter2=%d\n",inter2);
        coord1[k]=k*inter2;
        ibwrt(dev2," F ",3);


        }  /* --------------- end of k-
loop --------------*/



      if (needshift2[0]=='y')  /* then
put time window back */
     {
     winpos = info.xzero +
(info.number_point*info.xincr)/2 ;  /*
orig. position */
     posdiv = winpos /
(10*info.timediv) *10;
     gcvt((double)posdiv,5,posstring);
     strcpy(shiftstring, "TA:HPOS ");
     strcat(shiftstring, posstring);
     ibwrt(dev0, shiftstring,
strlen(shiftstring) );
     }

  ibwrt(dev2," E ",3);
  ibwrt(dev2, comeback2,
strlen(comeback2));
  printf("comeback2=%s",comeback2);
   for(w=1;w<=1000;w++)
  {
        for(i=1;i<=10;i++)
        {
              printf("i=%d\r",i);
        }
  }
  /*while (asr2[1] != 'R')
       {
       ibwrt(dev2,wait2,strlen(wait2));
       ibrd(dev2,asr2,3);
       }*/
  asr2[1] = 'B';

  ibwrt(dev2," F ",3);

  ibwrt(dev0,"CLSW",4);  /* clear
sweeps */
  ibwrt(dev0,"*CLS",4);  /* clear
status registers */

  total2=0;total3=0;
  for(k=0;k<=(info.numscan2-1);k++)
  {
        total2=coord2[k]+total2;
        total3=coord3[k]+total3;

  }

   total2=(long)total2/k;
   total3=(long)total3/k;

ltoa(total2,start2,10);
ltoa(total3,start3,10);

     dev1=ibfind("axis1");
        dev2=ibfind("axis2");
        dev3=ibfind("axis3");

        ibwrt(dev1," E ",3);

        ibwrt(dev3," E ",3);

        ibwrt(dev2," E ",3);

strcpy(go1,"MN A5 1V1 D");
strcat(go1,"+");
strcat(go1,comeback2);
strcat(go1," G ");

 for(w=1;w<=1000;w++)
  {
        for(i=1;i<=10;i++)
        {
              printf("i=%d\r",i);
        }
  }

strcpy(go2,"MN A5 2V1 D");
strcat(go2,"+");
strcat(go2,start2);
strcat(go2," G ");

 for(w=1;w<=1000;w++)
  {
        for(i=1;i<=10;i++)
        {
              printf("i=%d\r",i);
        }
  }
strcpy(go3,"MN A5 3V1 D");
strcat(go3,"+");
strcat(go3,start3);
strcat(go3," G ");

       ibwrt(dev1," F ",3);

       ibwrt(dev3," F ",3);

        ibwrt(dev2," F ",3);

/* cast coord data to double */
       for (i=0; i<N; i++)
       {
              coordd1[i] = (double)
coord1[i];
              coordd2[i] = (double)
coord2[i];
              coordd3[i] = (double)
coord3[i];
       }
}

/* This function calculates the
coefficients for a best fit line to
 M data points in the least squares
sense. The coefficients are for
 an equation of the form: y=Ax+B, where
A and B are constants.
 The coefficients are stored in a 2-
dim. array where the
```

```
 first element is A, and the second is
B. Arguments
 are: pointers to the arrays containing
the data points, and a pointer to
 the array which will serve as the
destination of the results.
 N is assumed to be defined as the
number of data points.
written by Adam Wunderlich, 11/1/98
*/
void line_calc(double *coord1, double
*coord2, double *line)
{

   /* local variables */
         double A[N][2],
          At[2][N],     /* transpose of
A */
          B[N],
          C[2][2] = {0},   /* C = At*A
*/
          D[2] = {0},      /* D = At*B
*/
          m,     /* multiplier */
    /* pointers to matrices */
       *aptr,
       *bptr,
       *atptr,
       *cptr,
       *dptr,

   /* temp. variables */
       ctemp,
       dtemp;

    /* counter variables */
   int i,
       j,
       k,
       r,
          l;

   aptr = &A[0][0];
   bptr = B;
   atptr = &At[0][0];
   cptr = &C[0][0];
   dptr = D;


   /* fill A */
   for (i=0; i<N; i++)
   {
      A[i][0] = coord1[i];
      A[i][1] = 1;
   }

   /* fill A transpose */
   for (i=0; i<N; i++)
   {
      for (j=0; j<2; j++)
         At[j][i] = A[i][j];
   }

   /* fill B */
   for (i=0; i<N; i++)
      B[i] = coord2[i];

   /* get At*A and put the result in C
*/
   for (k=0; k<2; k++)  /* increment
row in array At */
   {
      for (i=0; i<2; i++)  /* increment
column in array A */
    {
         for (j=0; j<N; j++) /*
increment column in At */
         {
            /* dot rows of matrix At
with columns of matrix A using */
            /* address arithmetic */
            *(cptr+ i+ k*2) +=
(*(atptr+ k*N +j)) * (*(aptr+ i +
j*2));
         }
     }
   }

   /* get At*B and put the result in D
*/
   for (k=0; k<2; k++)  /* increment
row in array At */
   {
          for (j=0; j<N; j++) /*
increment column in At */
      {
         /* dot rows of matrix At with
columns of matrix B using */
         /* address arithmetic */
         *(dptr+ k) += (*(atptr+ k*N
+j)) * (*(bptr+ j));
      }

   }
  // printf("matrix C:  \n");
   printMatrix(cptr,2);
  // printf("matrix D: \n");
   for (i=0; i < (2); i++)  /* loop
through the number of elements */
   {
   //   printf("%lf ", *(dptr+i)); /*
use address arithmetic, increment */
                                    /*
address  */
   //   printf("\n");
   }

   /* solve the normal equations Cx=D
using gaussian elimination with
      partial pivoting -- this is a 2x2
system, the result is placed
      in line */

   for (k=0; k<1; k++)
   {
      for (r=(k+1); r<2; r++)
      {
         /* partial pivoting */
         if (C[k][k] < C[r][k])
         {
```

```
            /* switch row k and row r
*/
            for (l=0; l<2; l++)
            {
               ctemp = C[r][l];
               C[r][l] = C[k][l];
               C[k][l] = ctemp;
               dtemp = D[r];
               D[r] = D[k];
               D[k] = dtemp;
            }
         }

         /* substract multiple of kth
row from rth row */
         m = (C[r][k])/(C[k][k]);
         for (j=0; j<2; j++)
                        {
            C[r][j] = (C[r][j] -
m*C[k][j]);
         }
            D[r] = D[r] - m*D[k];
      }
   }

printMatrix(cptr,2);

   for (i=0; i < (2); i++)  /* loop
through the number of elements */
   {
    //  printf("%lf ", *(dptr+i)); /*
use address arithmetic, increment */
                                  /*
address  */
   //   printf("\n");
   }

line[1] = D[1] / C[1][1];
line[0] = (D[0] - line[1]*C[0][1]) /
C[0][0];

//printf("coefficients: %lf %lf \n",
line[0], line[1]);

return;
}

/*-------------------------------------
----------------------*/
/* This function takes two arguments.
The first is a pointer to the    */
/* first element of an array.  The
second argument tells how many rows  */
/* and columns are in the array.  The
function will print the matrix to */
/* the screen, one row per line.
*/

void printMatrix(double *matrix, int n)
{
   /* declare local variables */
   int i;  /* counter variable for loop
*/

   /* print the matrix */
   printf("         ");
   for (i=0; i < (n*n); i++)  /* loop
through the number of elements */
   {
      //printf("%lf ", *(matrix+i)); /*
use address arithmetic, increment */
                                  /*
address  */
              if (((i+1)%n) == 0)  /*
if a multiple of N elements have */
                           /* been
printed, line return */
         printf ("\n         ");
   }
   printf("\n");
   return;  /* void return */
}  /* end function */

/*-------------------------------------
----------------------*/
/* This function moves the hydrophone
from the origin to the intercept
   of the line with the axis2-axis3
plane.  Then the hydrophone scans
   along the line with stepsizes and
total length as specified by the user
   with respect to axis1.  Note that I
often refer to axis1, axis2, and
   axis3, as x, y, and z respectively.
The line is defined as:
   x = (y-B)/A = (z-D)/C , where line2
= [A,B] and line3 = [C,D].
   written by Adam Wunderlich, 11/21/98
*/
void scan(void)
{

      // local variables

      long int pos[3] = {0},  //
coordinates of current position
            ax1, ax2, ax3,      //
increments for axis1, axis2, and axis3
            ax1long,
      //total beam axis 1 distance
            temp;

      unsigned long int
i,j,k,w,u,waitscope,count;
    int inr, result, axincr;
/* for status result from LeCroy */
    float  winpos, posdiv, posf[3],
magnitude, ax1f;

      char
            ax2c[5], ax3c[5],  //
increments as strings, note that these
// must be positive

      axis1[10],axis2[10],axis3[10];


    static char    cmd[100],    /* for
status response string */
```

```
                    curv[20000],    /*
MAKE SURE ENOUGH ROOM!!! */
                  posstring[60];    /*
string for current window position */

      count=1;

      winpos = info.xzero +
(info.number_point*info.xincr)/2;

      // open binary file
     if ((ptr_Bin =
fopen(fileBin,"wb")) == NULL)
            {
            printf(" error opening
binary file \n");
            exit(0);
            }

      // cast parameters specified by
user to long int
      ax1 = atol(ax1incr);
      // axis1 increment, assume
positive value
      ax1long=atol(ax1tot);
      // axis1 length
      ax1long=ax1long*1000;

      // update numscan2 for number of
scan points
      info.numscan3=ax1long/ax1 +1;
      printf("numscan3 = %ld \n",
info.numscan3);

      //  move to intercept of line
with y-z plane

      // move to y intercept

      ax2 = (long)line2[1];

      // printf("ax2=%ld\n",ax2);
      // increment is an absolute value
      temp = abs(ax2);
      ltoa(temp, ax2c, 10); // change
long to string

      ibwrt(dev1," E ",3);
      ibwrt(dev2," E ",3);
      ibwrt(dev3," E ",3);
      dev1=ibfind("axis1");
      dev2 = ibfind("axis2");
      strcpy(go2,"MN A5 2V1 D");
      strcpy(wait2," 2R ");

      if (line2[1] >= 0)
            strcat(go2,"+");
      else
            strcat(go2,"-");

      strcat(go2, ax2c);
      strcat(go2, " G ");
      ibwrt(dev2, go2, strlen(go2));

for(w=1;w<=1000;w++)
  {
        for(i=1;i<=100;i++)
        {
              printf("i=%d\r",i);
        }
  }

      // move to z intercept
      ax3 = (long)line3[1];
      //printf("ax3=%ld\n",ax3);

      temp = abs(ax3);
      ltoa(temp, ax3c, 10);
      // long to ascii
      dev3 = ibfind("axis3");
      strcpy(go3,"MN A5 3V1 D");
      strcpy(wait2," 2R ");

      if (line3[1] >= 0)
            strcat(go3,"+");
      else
            strcat(go3,"-");

      strcat(go3, ax3c);
      strcat(go3, " G ");
      ibwrt(dev3, go3, strlen(go3));


for(w=1;w<=1000;w++)
  {
        for(i=1;i<=10;i++)
        {
              printf("i=%d\r",i);
        }
  }

    // update pos[]
    pos[1] = ax2;   // update position
for axis2
      pos[2] = ax3;   // update
position for axis3

      ibwrt(dev0,"TA:WF? DAT1",11);
/* Transmit wave */
      ibrd(dev0,curv,(info.number_point
*2));

      // take curve for starting point
      for
(j=1;j<=(info.number_point*2);j++)
      {
            fputc(curv[j],ptr_Bin);
      }

      for (i=0; i<(info.numscan3 -1)
;i++)
      {
            strcpy(wait1," 1R ");
            strcpy(wait2," 2R ");
            strcpy(wait3," 3R ");
            // move to new position
            ibwrt(dev1," E ",3);
            ibwrt(dev2," E ",3);
            ibwrt(dev3," E ",3);
            strcpy(go1,"MN A5 1V1 D");
```

```
            strcpy(go2,"MN A5 2V1 D");
            strcpy(go3,"MN A5 3V1 D");
            strcat(go1,"-");
            // axis1 increment is in
negative direction, so choose other
            // increment directions
accordingly
            if (line3[0] >= 0)
                  strcat(go3,"+");
            else
                  strcat(go3,"-");

            if (line2[0] >= 0)
                  strcat(go2,"+");
            else
                  strcat(go2,"-");

            ax2=line2[0]*ax1;
            //printf("ax2=%ld\n",ax2);
            ax3=line3[0]*ax1;
            //printf("ax3=%ld\n",ax3);
            temp = abs(ax2);
            ltoa(temp, ax2c, 10);
            temp = abs(ax3);
            ltoa(temp, ax3c, 10);

            strcat(go2,ax2c);
            strcat(go2," G ");
            ibwrt(dev2, go2,
strlen(go2));  //increment axis 2

      ibwrt(dev2,wait2,strlen(wait1));
            strcat(go3,ax3c);
            strcat(go3," G ");
            ibwrt(dev3, go3,
strlen(go3)); //increment axis 3

      ibwrt(dev3,wait3,strlen(wait1));
            strcat(go1, ax1incr);
            strcat(go1," G ");
            ibwrt(dev1, go1,
strlen(go1));//increment axis 1

      ibwrt(dev1,wait1,strlen(wait1));


if (needshift2[0]=='y')
          {
          if (shiftdir[0]=='p')
             {
                  winpos = winpos +
shifttime1/2;
            }
              else
            winpos = winpos -
(shifttime1);
               posdiv = winpos /
(10*info.timediv) * 10;  /* pos. in
div. */
                 /* note LeCroy needs
position in terms of divisions */

//printf("posdiv=%f\n",posdiv);


gcvt((double)posdiv,5,posstring);  /*
convert position to string */
          strcpy(shiftstring, "TA:HPOS
");
          strcat(shiftstring,
posstring);
          ibwrt(dev0, shiftstring,
strlen(shiftstring) );
          }
    // update pos[]
       pos[0] = pos[0] - ax1;   //
update position for axis1
    pos[1] = pos[1] + ax2;  // update
position for axis2
       pos[2] = pos[2] + ax3;   //
update position for axis3

       //printf("pos[0]= %ld
       pos[1]=%ld
       pos[2]=%ld\n",pos[0],pos[1],pos[2
]);

  /*   strcpy(avgstring,"TA:DEF
EQN,'AVGS(C1)',SWEEPS,");
     strcat(avgstring,sweeps);

ibwrt(dev0,avgstring,strlen(avgstring))
;

             ibwrt(dev0,"CLSW",4);
/* clear sweeps to start average */
/*           result = 0;
             inr = 0;
             while( (result=0x0100&inr)
!= 256)  /* 256 means math on TA is
done */
/*              {
                strcpy(cmd,"INR?");

ibwrt(dev0,cmd,strlen(cmd));  /* what
is scope status */
/*
ibrd(dev0,scopestat,100);
                inr = atoi(scopestat);
                /* printf("\n Waiting
for scope: string is %s", scopestat);
*/
/*              }*/

          ibwrt(dev0,"TA:WF? DAT1",11);
/* Transmit wave */

ibrd(dev0,curv,(info.number_point*2));


      ibwrt(dev2,wait2,strlen(wait1));
      ibwrt(dev3,wait3,strlen(wait1));
      ibwrt(dev1,wait1,strlen(wait1));

            // take curve at this
position
      for
(j=1;j<=(info.number_point*2);j++)
             {
```

```
              fputc(curv[j],ptr_Bin);
              }

//     ibwrt(dev0,"CLSW",4);  /* clear
sweeps */
//    ibwrt(dev0,"*CLS",4);  /* clear
status registers */
for(w=1;w<=1000;w++)
  {
        for(u=1;u<=5;u++)
        {
              printf("u=%d\r",u);
        }
  }


       ibwrt(dev1," F ",3);
       ibwrt(dev2," F ",3);
       ibwrt(dev3," F ",3);
       }

       // put pos[] data into float form
       for (i=0; i<2 ; i++)
       {
              posf[i] = (float) pos[i];

       }

       // find magnitude of position
vector
       magnitude =
sqrt((posf[0])*(posf[0]) +
(posf[1])*(posf[1])
                          +
(posf[2])*(posf[2]));

       printf("magnitude = %f\n",
magnitude);


       // calculate actual increment
along beam axis
       // first, normalize position
vector to a unit vector
       for (i=0; i<2 ; i++)
       {
              posf[i] =
(pos[i])/magnitude;

       }

       ax1f = (float) ax1;
       printf("ax1f = %f \n",ax1f);

       // devide ax1 increment by axis1
component of posf[] to find
       // axincr
       info.axincr = ax1f / posf[0];
       printf("axincr = %f \n",
info.axincr);
////////////////////////////////////
////////////////////////////////
//Move back to beamscan start position
       ibwrt(dev1," E ",3);
       ibwrt(dev2," E ",3);
       ibwrt(dev3," E ",3);
       temp=fabs(pos[0]);
       ltoa(temp,axis1,10);
       // Axis 1 Final Position
       strcpy(go1,"MN A5 1V1 D");
       strcat(go1,"+");
       strcat(go1,axis1);
       strcat(go1," G ");
       ibwrt(dev1,go1,strlen(go1));
       //Move Axis 1 Back To Start
Position


for(w=1;w<=1000;w++)
  {
        for(i=1;i<=10;i++)
        {
              printf("i=%d\r",i);
        }
  }
       temp=fabs(pos[1]);
       ltoa(temp,axis2,10);
       strcpy(go2,"MN A5 2V1 D");
       if (pos[1]<0)
       {
             strcat(go2,"+");
       }
       else
             strcat(go2,"-");
       strcat(go2,axis2);
       strcat(go2," G ");
       ibwrt(dev2,go2,strlen(go2));

for(w=1;w<=1000;w++)
  {
        for(i=1;i<=10;i++)
        {
              printf("i=%d\r",i);
        }
  }
       temp=fabs(pos[2]);
       ltoa(temp,axis3,10);
       strcpy(go3,"MN A5 3V1 D");
       if (pos[2]<0)
       {
             strcat(go3,"+");
       }
       else
             strcat(go3,"-");
       strcat(go3,axis3);
       strcat(go3," G ");
       ibwrt(dev3,go3,strlen(go3));

for(w=1;w<=1000;w++)
  {
        for(i=1;i<=10;i++)
        {
              printf("i=%d\r",i);
        }
  }


       ibwrt(dev1," F ",3);
       ibwrt(dev2," F ",3);
       ibwrt(dev3," F ",3);
```

```
       if (needshift2[0]=='y')  /* then
put time window back */
     {
     winpos = info.xzero +
(info.number_point*info.xincr)/2 ;  /*
orig. position */
     posdiv = winpos /
(10*info.timediv) *10;
     gcvt((double)posdiv,5,posstring);
     strcpy(shiftstring, "TA:HPOS ");
     strcat(shiftstring, posstring);
     ibwrt(dev0, shiftstring,
strlen(shiftstring) );
     }

       // take absolute value of axincr
       info.axincr = (-1 * info.axincr);
  if ((ptr_Dat = fopen(fileDat,"w")) ==
NULL)
       {
       printf(" error opening data file
\n");
       exit(0);
       }
       fprintf(ptr_Dat,"%g %g %g %lu %lu
%lu %g %g %lu %g %g %lu",
    info.ymult, info.yzero, info.xincr,
info.number_point, info.numscan1,
    info.numscan2, info.xzero,
info.axincr, inter2, sos, shifttime,
       info.numscan3);
    fprintf(ptr_Dat,"\nymult yzero
xincr number_point numscan1 numscan2");
    fprintf(ptr_Dat," xzero stepsize1
stepsize2 sos timeshift numscan3\n");
    fclose(ptr_Dat);

       // note that the beam axis
increment (in microns) is in
info.axincr

       fclose(ptr_Bin);

       printf("pos[0] = %ld \n",
pos[0]);
       printf("pos[1] = %ld \n",
pos[1]);
       printf("pos[2] = %ld \n",
pos[2]);
}
```

APPENDIX C
DATA ANALYSIS PROGRAM

This program does the analysis described in Chapter 4. It was written in Matlab®.

```
% marconi.m
% computes PII, Pc, Pr and plots versus axial diastance,
% given beam axis measurements of a transducer made using a hydrophone
% this script loads both the bin and dat file
%
% written by Adam Wunderlich, Sept. 1998
% using complete5.m as a basis
% modified Dec. 1998 by Jason Sempsrott
% modified Aug. 1999 by Jason Sempsrott

clear all;
close all;

n      = input('Enter a filename: ','s');
fc     = input('Enter the center frequency of the transdcuer in MHz ');
dist   = input('Distance of hydrophone from transducer at start  (in cm)');
R      = input('Enter Ritec setting: ');
date   = input('Date of calibration: ','s');
who    = input('Person calibrating: ','s');
which  = input('Hydrophone that was used (type and Serial #):\n ','s');
factor = input('Enter the calibration factor in V/MPa: ');

bin = [n '.bin'];
dat = [n '.dat'];

fid    = fopen(dat, 'r');
A      = fscanf(fid,'%f',[12,1]);

ymult=A(1,1);       % y-scaling
yzero=A(2,1);       % DC offset
xincr=A(3,1);       % x-scaling, 1/(sampling rate)
number_point=A(4,1);       % # points in waveform
numscan1=A(5,1);    % # of scans for scan axis1
numscan2=A(6,1);    % # of scans for scan axis2
xzero=A(7,1);       % time for start of first wave form
stepsize1=A(8,1);   % step size for scan axis 1
stepsize2=A(9,1);   % stepsize for scan axis 2
sos=A(10,1);        % speed of sound
timeshift=A(11,1);  % window time shift
numscan3=A(12,1);   % number of scan points for beam axis

clear A;

% open and read .bin file
% assume LeCroy oscilloscope was used to take data

status = fclose(fid);
fid=fopen(bin,'rb','b');  % 'b' for LeCroy



%//////////////////////////////////////////////////////////////////////////
% compute PII
%//////////////////////////////////////////////////////////////////////////
PIItemp1=0;
```

```
PII=zeros(1,numscan3);
    for(j=1:numscan3)
        B=fread(fid,number_point,'short');  % B is a column vector
        B=B.*ymult;
        B=B-yzero;      % - for LeCroy
        PII(1,j)=xincr*sum(B .* B);
    end

x1 = ((0:numscan3-1)*stepsize1);     % calculates axis for axial distance
x1 = x1/10000;  % convert to cm from um
x = dist + x1;


figure
orient tall

% plot PII versus axial distance
subplot(3,1,1)
% want to apply smoothing function to curve
pPII   = polyfit(x,PII,6);
fPII   = polyval(pPII,x);
plot(x,fPII)
xlabel('Axial distance (mm)')
ylabel('PII')
grid
status = fclose(fid);
[max_fPII,x_fPII] = max(fPII);
x_fPII=dist+(x_fPII*stepsize1/10000); % axial distance of maximum Pc in cm

% also need derated water value for PII and position of maximum
PII3=fPII.*exp(-.069*fc*x);
[max_PII3,junk] = max(PII3);
x_PII3 = dist+(junk*stepsize1/10000); % axial position of max PII3


%//////////////////////////////////////////////////////////////////////////
%      relevent variables are:
%      x      = axial position from transducer
%      PII    = raw data obtained during scan
%      fPII   = smooth curve of PII
%      PII3   = derated PII values from smooth curve
%      x_PII3  = position of maximum PII3
%//////////////////////////////////////////////////////////////////////////

%//////////////////////////////////////////////////////////////////////////
% plot peak compressional pressure versus axial distance
%//////////////////////////////////////////////////////////////////////////

fid=fopen(bin,'rb','b');    % b for Lecroy

Pc=zeros(1,numscan3);
    for(j=1:numscan3)
        C=fread(fid,number_point,'short');  % C is a column vector
        C=C.*ymult - yzero;  % - for LeCroy
        Pc(1,j) = max(C);
    end

% Need to convert voltages to pressures
Pc=Pc/(factor);

subplot(3,1,2)
pPc    = polyfit(x,Pc,6);
fPc    = polyval(pPc,x);
plot(x,fPc)
xlabel('Axial distance (cm)')
```

```
ylabel('Pc (MPa)')
grid
status = fclose(fid);

%      need to find maximum Pc value and its position

[max_Pc,x_Pc]=max(fPc);
x_Pc=dist+(x_Pc*stepsize1/10000); % axial distance of maximum Pc in cm

% derated water value for Pc

Pc3    = fPc.*exp(-.0345*fc*x);

%      max value of Pc3 and its position
[max_Pc3,x_Pc3]=max(Pc3);
x_Pc3=dist+(x_Pc3*stepsize1/10000); % axial distance of maximum Pc3

Pc3_max_PII3=Pc3(1,junk); % derated Pc at the position of max PII3


%//////////////////////////////////////////////////////////////////////////
%      relevant variables include:
%            x      = axial distance
%            Pc     = original max compressional values
%            fPc    = smooth curve of Pc
%            x_Pc   = position of maximum Pc (cm)
%            max_Pc = value of maximum Pc (MPa)
%            Pc3    = derated water value for fPc (MPa)
%            x_Pc3  = position of maximum Pc3 (cm)
%            max_Pc3 = value of maximum Pc3 (MPa)
%            Pc3_max_PII3 = value of Pc3 located at max_PII3
%//////////////////////////////////////////////////////////////////////////

%//////////////////////////////////////////////////////////////////////////
% plot peak rarefractional pressure versus axial distance
%//////////////////////////////////////////////////////////////////////////

fid=fopen(bin,'rb','b');    % b for Lecroy

Pr=zeros(1,numscan3);
    for(j=1:numscan3)
        D=fread(fid,number_point,'short');  % D is a column vector
        D=D.*ymult - yzero;  % - for LeCroy
        Pr(1,j) = abs(min(D));
    end

% Need to convert voltages to pressures
Pr=Pr/(factor);

subplot(3,1,3)
pPr=polyfit(x,Pr,6);
fPr=polyval(pPr,x);
plot(x,fPr)
xlabel('Axial distance (cm)')
ylabel('Pr (MPa)')
grid
status = fclose(fid);

%      need to find maximum Pr value and its position

[max_Pr,x_Pr]=max(fPr);
x_Pr=dist+(x_Pr*stepsize1/10000); % axial distance of maximum Pr


%      derated water value for Pr
```

```
Pr3    = fPr.*exp(-.0345*fc*x);
%      max value of Pr3 and its position
[max_Pr3,x_Pr3]=max(Pr3);
x_Pr3=dist+(x_Pr3*stepsize1/10000); % axial distance of maximum Pr3

Pr3_max_PII3=Pr3(1,junk); % derated Pr at the position of max_PII3

MI = Pr3_max_PII3/(fc^.5);

%//////////////////////////////////////////////////////////////////////
%      relevant variables include:
%             x      = axial distance (cm)
%             Pr     = original max rarefactional values (MPa)
%             fPr    = smooth curve of Pr (MPa)
%             x_Pr   = position of maximum Pr (cm)
%             max_Pr = value of maximum Pr (MPa)
%             Pr3    = derated water value for fPr (MPa)
%             x_Pr3  = position of maximum Pr3 (cm)
%             max_Pr3 = value of maximum Pr3 (MPa)
%          Pr3_max_PII3 = value of Pr3 at position of max_PII3
%             MI     = mechanical index
%//////////////////////////////////////////////////////////////////////

%//////////////////////////////////////////////////////////////////////
% plot the time domain waveform at the maximum point in the PII plot
% (plots waveform at focus)
%//////////////////////////////////////////////////////////////////////

fid=fopen(bin,'rb','b');   % b for Lecroy

[max_PII,x_PII]=max(PII);     % finds axial location of focus
junky=dist+(x_PII*stepsize1/10000); % axial distance of maximum Pc in cm

temp=0;
for(j=1:x_PII-1)
    [wave]=fread(fid,number_point,'short'); % wave is col. vector

end




% check for zero points at end of data set, and count them
z=0;
for (i=1:number_point)
    if (wave(i,1) == 0)
        z = z+1;
    end
end


% plot the waveform at the focus centered around 0 V
figure

g = number_point-z;      % g is an intermediate variable

%twave=wave-mean(wave);
wavescaled=wave.*ymult - yzero;

% Want to develop the waveform for PII vs. Time
simple=0;
for(j=1:number_point)
       tempwave=wave(j,1).*ymult;
       tempwave1=tempwave.*tempwave;
       tempwave2=tempwave1+simple;
```

```
        simple=tempwave2;
        newwave(j,1)=tempwave2;
end



%twavescaled= wkeep(wavescaled,g,'l');  % truncate to get rid of zeros

timeaxis=(0:number_point-1)*xincr*1e6;
plot(timeaxis,wavescaled)
title('Waveform at Focus')
xlabel('Time (microseconds)')
ylabel('Amplitude (V)')
grid
focuspc=max(wavescaled);   %This gives the pc voltage value at the focus.
focuspc=focuspc/(factor);  % Converted volts to MPa
focuspr=abs(min(wavescaled));   %This gives the pr voltage value at the focus.
focuspr=focuspr/(factor);  % Converted volts to MPa

%//////////////////////////////////////////////////////////////////////////
%             relevant variables include:
%                           junky   = position of max_PII
%                           focuspc = Pc located at max_PII
%                           focuspr = Pr located at max_PII
%//////////////////////////////////////////////////////////////////////////

%//////////////////////////////////////////////////////////////////////////
% A plot of PII vs Time
%//////////////////////////////////////////////////////////////////////////
figure
plot(timeaxis,newwave);
xlabel('Time (microseconds)');
ylabel('PII Value');
grid

% Need to find limits for calculating the pulse duration
% This calculation uses the 90% and 10% points of the PII curve
% First find the upper and lower limits according to FIgure 3

upperlimit=max(newwave)*.9;
lowerlimit=max(newwave)*.1;

% Second, find the time value at the upper and lower limits
for(j=1:number_point)
        if(newwave(j,1) > (upperlimit))
                        timeaxisupper=timeaxis(1,j-1);
                        break;
        end
end
for(j=1:number_point)
        if(newwave(j,1) > (lowerlimit))
                        timeaxislower=timeaxis(1,j-1);
                        break;
        end
end

% Third, calculate a time for the pulse duration
pulse=(timeaxisupper-timeaxislower)*1.25;

%//////////////////////////////////////////////////////////////////////////
```

```
rawdata             = [x;PII;Pc;Pr];
smoothdata    = [x;fPII;PII3;fPc;Pc3;fPr;Pr3];
interests1    = [max_Pc;x_Pc;max_Pr;x_Pr];
interests2    = [max_Pc3;x_Pc3;max_Pr3;x_Pr3];
interests3    = [junky;focuspc;focuspr];
interests4    = [x_PII3;Pc3_max_PII3;Pr3_max_PII3];
interests5    = [MI;pulse;R];
everything    =
[R;pulse;max_Pc;x_Pc;max_Pr;x_Pr;max_Pc3;x_Pc3;max_Pr3;x_Pr3;junky;focuspc;focuspr;x_P
II3;Pc3_max_PII3;Pr3_max_PII3;MI];

n=input('Enter filename to save all data: ','s');
txtfile=[n,'.txt'];
fid=fopen(txtfile,'a');
fprintf(fid,'\n/////\n');
fprintf(fid,'Raw\n');
fprintf(fid,'x(cm)\t PII\t Pc(MPa)\t Pr(MPa)\n');
fprintf(fid,'%1.4f\t %2.4e\t %2.4f\t  %2.4f\n',rawdata);
fprintf(fid,'/////\n');
fprintf(fid,'Best Fit \n');
fprintf(fid,'x(cm)\t PII\t PII.3\t Pc(MPa)\tPc.3(MPa)\tPr(MPa)\tPr.3(MPa)\n');
fprintf(fid,'%1.4f\t %2.4e\t %2.4e\t %2.4f\t%2.4f\t%2.4f\t%2.4f\n',smoothdata);
fprintf(fid,'/////\n');
fprintf(fid,'Points of Interest\n');
fprintf(fid,'\nGlobal Information from Best Fit Data:\n');
fprintf(fid,'\nMax Pc(MPa)\tPosition(cm)\t Max Pr(MPa)\tPosition(cm)\n');
fprintf(fid,'%2.4f\t\t %1.4f\t\t %2.4f\t %1.4f\n\n',interests1);
fprintf(fid,'Max Pc.3(MPa)\tPosition(cm)\t Max Pr.3(MPa)\tPosition(cm)\n');
fprintf(fid,'%2.4f\t\t %1.4f\t\t %2.4f\t\t %1.4f\n\n',interests2);
fprintf(fid,'Information from waveform at focus (max of PII):\n');
fprintf(fid,'Position of max PII\t Pc (MPa)\t\t Pr (MPa)\n');
fprintf(fid,'%1.4f\t\t\t%2.4f\t\t\t%2.4f\n\n',interests3);
fprintf(fid,'Position, Pc.3, and Pr.3 at maximum of PII.3:\n');
fprintf(fid,'Position of max PII.3\t Pc.3 (MPa)\t\t Pr.3 (MPa)\n');
fprintf(fid,'%1.4f\t\t\t%2.4f\t\t\t%2.4f\n\n',interests4);
fprintf(fid,'Mechanical Index\t Pulse Duration (microsecs)\t RITEC Setting\n');
fprintf(fid,'%1.4f\t\t\t%1.4f\t\t\t\t%1.2f\n\n',interests5);
fprintf(fid,'Date of calibration:\n');
fprintf(fid,date);
fprintf(fid,'\nPerson who calibrated:\n');
fprintf(fid,who);
fprintf(fid,'\nHydrophone that was used (type and serial #):\n');
fprintf(fid,which);
fprintf(fid,'\nHydrophone conversion factor is (V/MPa):\n');
fprintf(fid,'%1.4f',factor);
fprintf(fid,'\nTransducer frequency(MHz)       %1.4f\t',fc);
fprintf(fid,'\n/////\n');

status=fclose(fid);

n1=input('Enter filename for points of interest only: ','s');
txtfile1=[n1,'.txt'];
fid=fopen(txtfile1,'a');
fprintf(fid,'%1.2f\t%2.4f\t%2.4f\t%2.4f\t%2.4f\t%2.4f\t%2.4f\t%2.4f\t%2.4f\t%2.4f\t%2.
4f\t%2.4f\t%2.4f\t%2.4f\t%2.4f\t%2.4f\t%2.4f\n',everything);
status=fclose(fid)

disp('Max Pc in MPa is:');
max_Pc
disp('Max Pr in MPa is:');
max_Pr
```

APPENDIX D
FREQUENCY ANALYSIS PROGRAM

This program takes select information from the program in Appendix B and does a frequency spectrum analysis. This program was written in Matlab®.

```
% trans.m
% computes the frequency spectrum for the waveform
% at the focus and determines the fundamental, second
% and third harmonics as functions of distance from the
% transducer.
% written by Jason Sempsrott
% Nov. 1999

fid=fopen(bin, 'rb','b');

for(j=1:numscan3)
       beta=fread(fid,number_point,'short');
       beta=beta.*ymult;
       beta=beta-yzero;
       for(i=1:number_point)
       these(i,j)=beta(i,1); % 'these' is every time-domain collected wave
       end

end

fclose(fid);
Ts=(timeaxis(2)-timeaxis(1));%time in seconds
Fs=1/Ts;
N=length(wavescaled);
K=Fs*(0:N-1)/N;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             Want to find frequency spectrum
%             of time domain plots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for(i=1:numscan3)
       y=fft(these(:,i),N);
       ymag=2*abs(y)/N;
       ymag(1)=ymag(1)/2;
       ymag((N/2)+1)=ymag((N/2)+1)/2;
       Mags(:,i)=ymag;     % 'Mags' is the frequency spectrum of each wave
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             Want to find harmonic components as
%             functions of distance from transducer.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for(i=1:numscan3)
       principalx=K(1:1251);
       principaly(:,i)=Mags((1:1251),i); % Want principal alias only
       fundmax(i)=max(principaly(:,i));  % Find max values of fundamental
end



for(i=1:number_point/2)
       if(principaly(i,100)==fundmax(100))
```

```
            xvalue=i;    % 'xvalue' is the array position of fundamental
      end
end

%%%%%%% Finding the second harmonics for each collected wave
lowside=(xvalue*2)-(round(xvalue/2));
highside=(xvalue*2)+(round(xvalue/2));

for(i=1:numscan3)
      junk_x=K(lowside:highside);
      junk_y=Mags((lowside:highside),i);
      secondharm(i)=max(junk_y);
end

%%%%%% Finding the third harmonic
lowside3=(xvalue*3)-(round(xvalue/2));
highside3=(xvalue*3)+(round(xvalue/2));

for(i=1:numscan3)
      junk_x=K(lowside3:highside3);
      junk_y=Mags((lowside3:highside3),i);
      thirdharm(i)=max(junk_y);
end

%%%%%%%      Normalize all harmonics to the maximum of the fundamental

fundamental=fundmax/(max(fundmax));
second=secondharm/(max(fundmax));
third=thirdharm/(max(fundmax));
figure(4)
h1=plot(x,fundamental);
hold
h2=plot(x,second);
h3=plot(x,third);
h4=gtext('1st');
h5=gtext('2nd');
h6=gtext('3rd');
set(h4,'fontsize',[13],'fontweight','bold')
set(h5,'fontsize',[13],'fontweight','bold')
set(h6,'fontsize',[13],'fontweight','bold')
set(gca,'fontsize',[13],'fontweight','bold')
xlabel('Distance from Transducer (cm)')
ylabel('Harmonic Magnitudes')
```