Development of a Directional Wave Gage for Short Sea Waves

by

Nicole Marie Suoja

B.S., Mechanical Engineering
Washington State University, 1994

Submitted to the Department of Ocean Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Ocean Engineering

at the
Massachusetts Institute of Technology
June 1996

Signature of Author .....
Department of Ocean Engineering
May 10, 1996

Certified by .....
Jerome H. Milgram
Professor of Ocean Engineering
Thesis Supervisor

Accepted by .....
A. Douglas Carmichael
Professor of Ocean Engineering
Chairman, Department of Ocean Engineering Graduate Committee

Development of a Directional Wave Gage for Short Sea Waves

by

Nicole Marie Suoja

Submitted to the Department of Ocean Engineering on June 10, 1996
in Partial Fulfillment of the Requirements for the degree of
Master of Science in Ocean Engineering

ABSTRACT

Understanding sea-surface interactions is an important subject of study within the oceanographic community. In particular, the amplitude, direction, and general behavior of high frequency waves play significant roles in radar imaging and remote sensing. Waves in the frequency range of interest to remote sensing systems are four to five orders of magnitude lower in energy than waves of primary oceanographic engineering interest, such as general sea swell. These high frequency waves of interest have wavelengths of approximately 1 to 30 centimeters. Unfortunately, most currently existing *in situ* wave measuring instruments do not measure the amplitude and directional distribution of these small-wavelength, low-energy waves. The few tha t can measure them are costly and require very low sea states in which to operate.

Thus, there is need for an instrument to measure the directional and energy content of the high frequency waves that is deployable in variable sea states. An instrument which can accomplish these tasks has been developed. Its initial design and lab testing were conducted by previous MIT student David Chen (Chen, 1994). The practical use of this instrument in the ocean environment and its continuing evolution are the topics of this thesis.

The theory behind the use of a 64-wire surface-piercing circular wave gage has now been fully developed. The previous gage design and analysis code have been further optimized to provide increased ease of calibration and use in the ocean environment and to improve noise rejection. The instrument has been subsequently tested in a wave tank to insure its accuracy in comparison with tests performed using a laser slope gage and off a pier to insure its usability. The data analysis code has been refined and tested with known inputs and the results compare favorably with independent measurements. Finally, the instrument has been tested in open water conditions. The results from these tests have been compared to theoretical predictions of fully developed wind driven waves.

Thesis Supervisor: Dr. Jerome H. Milgram
Title: Professor of Ocean Engineering

ACKNOWLEDGEMENTS

I would like to thank a number of people and organizations who have gone beyond the call of duty as supervisors, mentors, and friends to help me get where I am today.

First and foremost, I would like to thank Professor Jerome Milgram for his help, his guidance, and his support; all of which I have needed in various quantities at different times. He has challenged me to be the best I can be (which can be an onerous task) and I thank him wholeheartedly for it.

I would also like to thank the group of students in the Marine Instrumentation and Computation Lab for their support and help. Greg Thomas has helped me appreciate the hydrodynamics related to my work and has always been there as a friend and mentor when I needed suggestions from 'someone who has gone through all this before'. John Mass has provided countless hours of his time and patient explanations when I was out of my element working with computers and electronics. Jess Riggle also was a good person to talk to when I needed another student to share my woes and celebrations with. Bill Milewski and Bill "the Dude" Ramsey have only been in the lab for a short period of time, but have become friends to laugh with and share a good beer with (while discussing the finer points of potential flow calculations, of course....) Cindy Dernay was also a part of the 'lab family' for about a year and she became a close friend as well as a very dependable source of help around the Institute. Jessica Forbess, who was only a member of our lab group for one summer as a UROP student, reminded me what it was like to be an undergraduate. She was hard-working and very dependable.

Outside of the MIT environment I would like to extend a thank you to my family for always being there to support, encourage, and believe in me when I felt like I just couldn't do any more. Also, since joining the MIT community, I have met a gentleman who has changed my life: Mr. Albert Tervalon. He has come to understand me very well, and when he didn't understand, he still smiled and nodded. His love and support have meant the world to me.

Finally I would like to thank various organizations which have made my experimentation with the circular wave gage possible: Greene Marine, Woods Hole Oceanographic Institute, ICF Kaiser, and Sea Grant.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

There is much that is still not understood about the properties of the ocean surface and their effect on remote sensing systems such as satellite radar imaging. In particular, satellite imaging systems are dependent on the scattering behavior of high-frequency sea-surface waves which ride on the lower-frequency sea swell. To further understand the role these high-frequency waves play in scattering incident microwaves from satellites or aircraft, the directional wave number spectra of these high-frequency waves need to be measured. The directional spectrum provides information about the energy distribution both in frequency and direction of the surface waves.

One example showing the need for studying the sea wave scattering relationship is that some synthetic aperture radar (SAR) images of the sea surface show less sensitivity to look direction than expected (Milgram, 1988). Based on microwave backscatter theory this would seem to indicate that the directional wave number spectra of the high-frequency scattering waves are broad in direction. However, observation and preliminary measurements show that the wave number spectra of well-developed sea surfaces tend to be *highly* directional and dominated by the primary wind direction. The question then arises as to whether the microwave backscatter theory is an incomplete model for describing sea surface scattering behavior. To fully address this incongruity and the further questions it brings about, it is necessary to monitor the sea surface *in situ* during remote sensing operations to obtain accurate ground-truth information.

There are many instruments available which can measure the directional wave number spectrum of surface waves *in situ*. Most, however, are not designed to operate in the range of frequencies of interest to remote sensing systems. Here, waves that are Bragg-resonant with L-, C-, and X-band radar are considered. Their frequency range is between

2 and 10 Hz, which corresponds approximately to waves with wavelengths between thirty and one centimeter, respectively. Even some that can obtain spectra at these frequencies cannot discern direction. Currently, there are only two well-known instruments which have been used to accomplish this task. The first is a scanning laser slope gage developed by Daniel Kwoh at TRW which has been used to collect open ocean spectra (Yuen, 1991) . This instrument, however, cannot measure waves with frequencies above five hertz, is expensive, and can be operated only in low sea states. The second is an instrument developed at the Woods Hole Oceanographic Institute by Erik Bock (Bock & Hara, 1995), (Hara, Bock & Lyzenga, 1994), which is also a scanning laser slope gage. While it does possess the ability to measure frequencies up to thirty-four hertz, it has drawbacks similar to those of the TRW instrument due to cost and sea-state operational limits.

The objective of this work, therefore, is to produce an instrument which can measure the directional spectra of high frequency waves and can be used in a range of sea states. It must also be easy to accurately calibrate and produce reliable and repeatable results.

The development of an instrument to meet these objectives was initiated by former Massachusetts Institute of Technology (MIT) student David Chen (Chen, 1994). This previous work focused on the theoretical development of the wave gage. Its primary goal was to show that a resistive wire wave gage and its post-processing programs could be designed and built. This was accomplished and the instrument was successfully tested in the Marine Instrumentation and Computation Lab at MIT. The next logical step in the research was to modify the instrument so that it could be used in the ocean environment to accurately and repeatedly produce directional spectra. This involved some modifications to the instrument itself, the development of an accurate calibration system, and further development of the post-processing analysis code. These modification activities and the subsequent experimental tests performed with the circular wave gage are the subjects of this thesis.

2. THEORY

The following discussion includes brief descriptions of the basic theory used in the development of the directional wave gage. The first two sections are not directly related to the instrument's design, but help in understanding the framework in which the instrument can be used and the validity of the output directional spectra. The third section about spectral analysis contains a description of the core of the signal processing method used to generate a directional spectrum from the sea-surface elevation data.

2.1 BACKSCATTER THEORY

A primary mechanism for return of incident microwaves from the ocean surface to SAR receivers is known as Bragg backscatter. This theory proposes that at moderate incidence angles, 30° to 60°, the primary return or reflection from incident microwaves comes from constructive interference of the reflected signal. This fundamentally makes sense because at incidence angles greater than a few degrees, most of the incident signal will be reflected away from the receiver, rather than towards it. The resulting return signal, therefore, is very weak and will be detectable only if the reflected return signals constructively interfere with one another.

To produce this constructive interference, the incident microwave wavelength and the wavelength of the sea-surface waves must be related in a very specific way. The geometric relationship between these waves is shown in Figure 2.1.

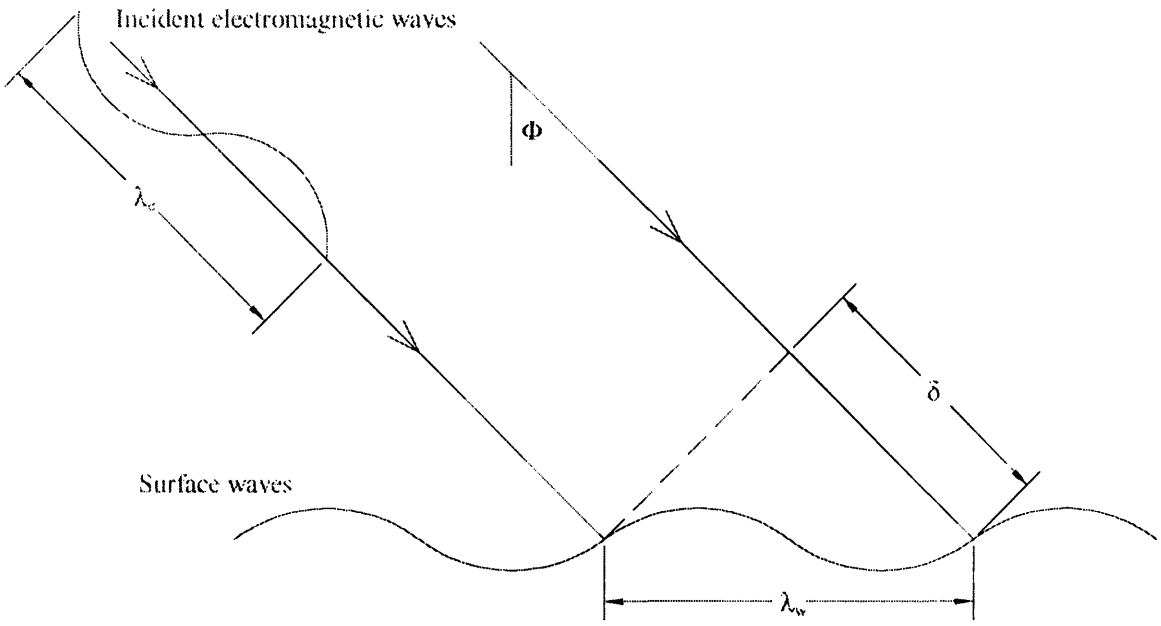

FIGURE 2.1: Bragg Backscatter Geometry

This relationship can be described as follows. Given an incident electromagnetic wave of wavelength λ_e, for first mode constructive interference to occur, the length δ must be related to λ_e such that

$$2\delta = \lambda_e . \tag{2.1}$$

For a radar incidence angle of Φ, as shown in Figure 1, the wavelength of a surface wave, λ_w, is related to the distance δ by

$$\delta = \lambda_w \sin(\Phi) . \tag{2.2}$$

Therefore, the surface wave wavelength is related to the incident electromagnetic signal wavelength by

$$\lambda_w = \lambda_e / [2 \sin(\Phi)] . \tag{2.3}$$

For X, C, and L-band radar signals, the approximate wavelength of the primary scattering waves on the surface can be calculated and is shown in Table 2.1 with radar look angles of 60° and 45° from vertical.

TABLE 2.1: Incident Electromagnetic Wave Relation to Surface Wavelengths for First Mode Interference

Radar Band	Incident Wavelength, λ_e	Surface Wavelength, λ_w	
		$\phi = 60°$	$\phi = 45°$
L - band	≈ 20 cm	11.5 cm	14.1 cm
C - band	≈ 6 cm	3.5 cm	4.2 cm
X - band	≈ 1.5 - 2 cm	1.0 cm	1.2 cm

Many SAR radar imaging systems operate in the X, C, or L-band range. Since the circular wave gage is intended as a ground-truthing device for imaging systems, the information contained in the table above shows why the instrument was specifically designed to measure waves with wavelengths between one and thirty centimeters.

2.2 SURFACE WAVES

In order to properly interpret the spectral plots the wave gage produces it is important to understand the underlying phenomena of surface waves. These are the natural waves developed on the sea surface generated primarily by the wind. The high-frequency waves this wave gage is designed to measure are at the upper end of the range of waves typically found on the sea surface. In particular, these high-frequency waves contain significantly less energy than typical oceanographic engineering waves of interest. This difference in energy of four or five orders of magnitude is why most wave gages designed to measure more energetic waves cannot resolve them; the waves simply don't contain enough energy. Figure 2.2 shows this energy density relationship. The electronics and software of this instrument, therefore, must significantly filter the input wave signal to insure that the high-energy, low-frequency waves which would destroy the dynamic range of the instrument are effectively filtered out.

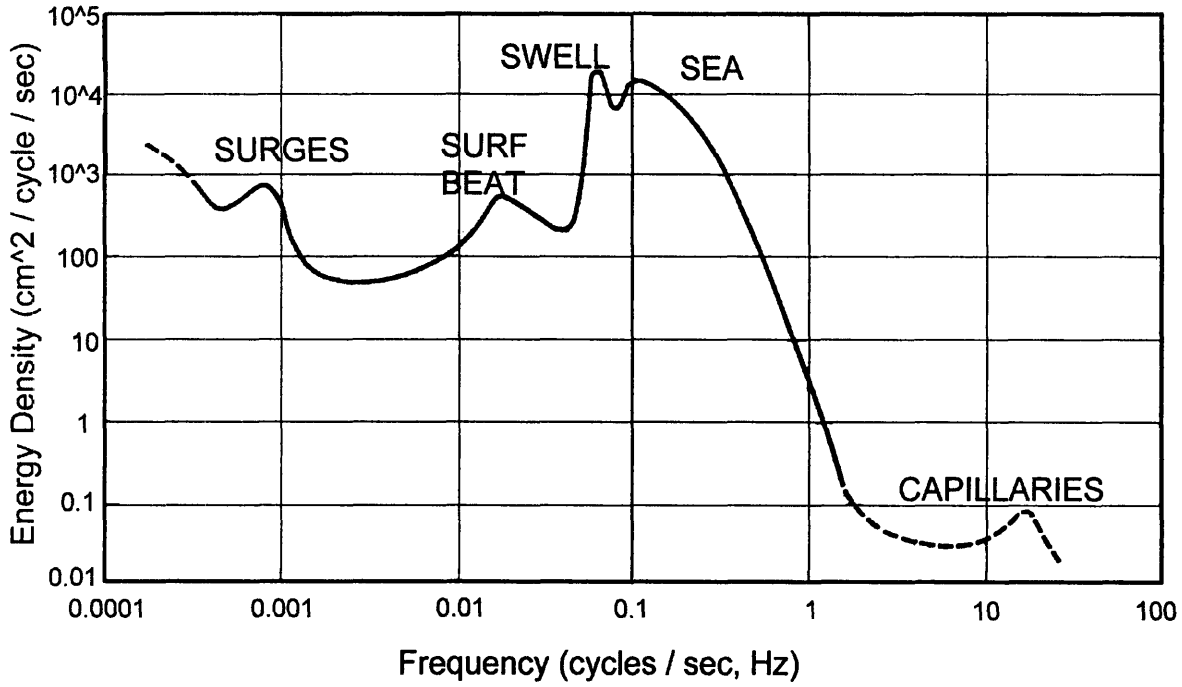

FIGURE 2.2: Energy Density in Sea Surface Waves as a Function of Frequency (Comstock, 1967)

Many measurements of gravity waves in the open ocean have been collected and used to develop models which accurately predict wave growth, propagation, and decay. These models also predict sea spectra. The Pierson-Moskowitz model (Pierson & Moskowitz, 1964) is one trade standard, but it does not predict wave spectra at high frequencies. The Bjerkaas and Riedel modification of the Pierson model (Bjerkaas & Riedel, 1979) includes some important corrections and simplifications and extends the model into the high frequency range the circular wave gage has been developed to measure. For this reason and the fact that its results have been verified by considerable experimental data, the Bjerkaas and Riedel model will be used as a basis of comparison for results of the spectral measurements obtained with the wave gage. This model predicts a two-and-one-half decibel (dB) decrease per decade of the wave number spectrum as a function of wave number in the range of 0.1 to 3.0 rad/cm. This is illustrated in Figure 2.3, which shows the relationship between wave number spectra and the wave number.

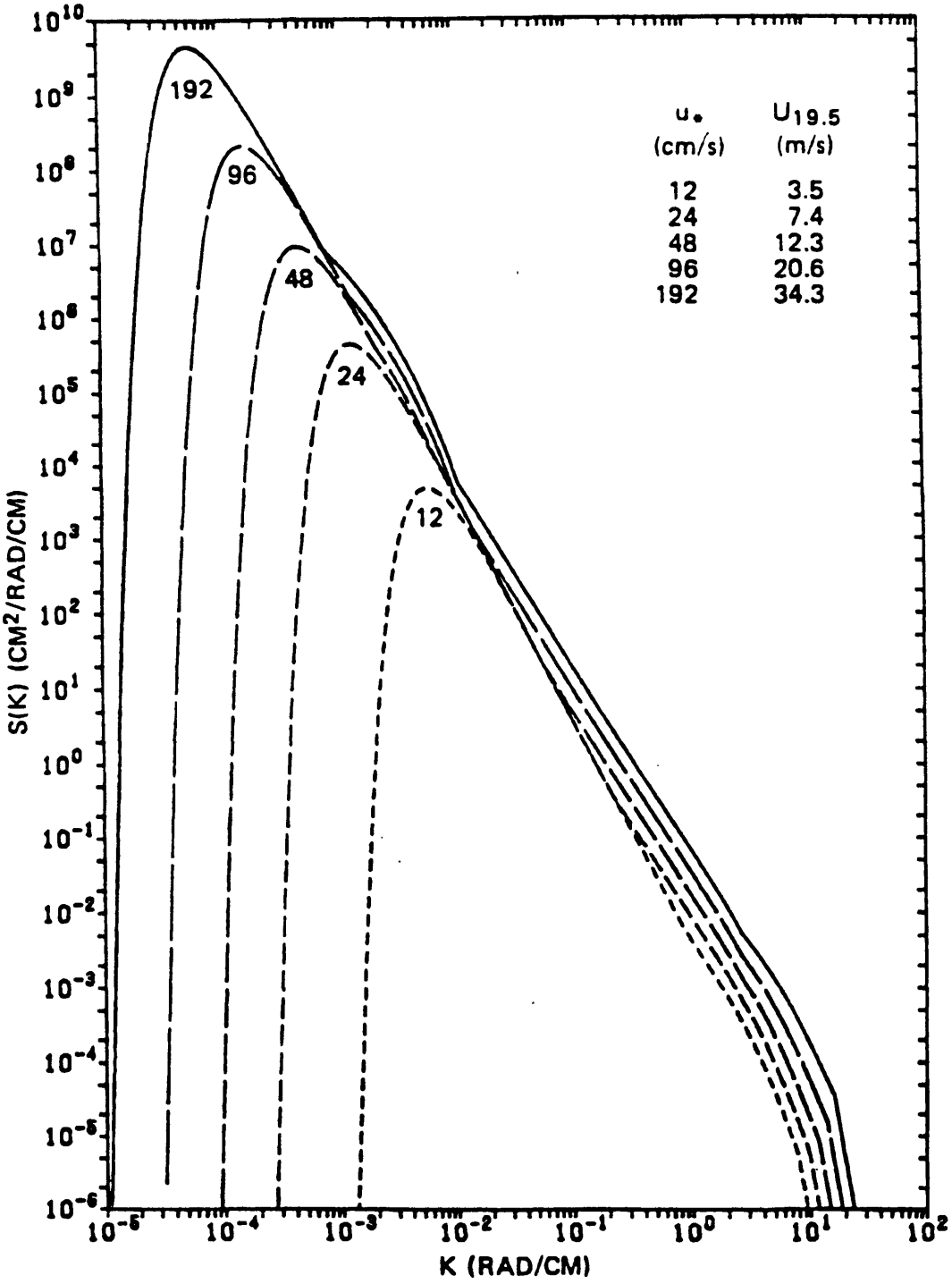

FIGURE 2.3: Bjerkaas and Riedel Wave Number Spectra for Fully Developed Wind Driven Waves (Bjerkaas & Riedel, 1979)

The wave number of a wave is related its frequency by the following relationship (valid only in deep water):

$$\omega^2 = gk + \frac{tk^3}{\rho}, \tag{2.4}$$

where ω is the frequency of the wave in radians per second,
g is the gravitational constant,
k is the wave number,
t is the surface tension, and
ρ is the density of water.

The first term on the right-hand side of equation 2.4 represents the contribution of gravity to the wave frequency. The second term represents the surface tension contribution. The surface tension contribution becomes insignificant below wave frequencies of about 30 rad/s or 5 Hz.

2.3 SPECTRAL ANALYSIS

One of the primary difficulties in obtaining directional spectra of the sea surface is the physical difficulty in obtaining a spatially two-dimensional measurement of the sea surface elevation. The input signal must also be measured over time at every sampling point to allow temporal averaging if the spatial dimensions of the measured region are modest. Some systems can obtain one of the two requirements, either spatial sampling (two-dimensional and evenly spaced), or temporal sampling (time-series), but not both. One feature of the analysis used by the circular wave gage is that evenly spaced data points are not required. Instead of using an evenly spaced grid, a more modest spatial sampling is used. The following analysis describes the method used to obtain the directional spectrum of the sea surface if the autocorrelation can be calculated at some known spatial lags, $(\vec{x}, \vec{y})$.

2.3.1 The Autocorrelation

Let R(x,y) be the windowed autocorrelation of an input signal. The autocorrelation of the input signal is assumed to occupy a region in autocorrelation space bounded by

$$|x| < L_x \text{ and } |y| < L_y \,, \tag{2.5}$$

where L_x and L_y are the horizontal and vertical distances spanned by spatial differences in the sample points. Outside this region the smoothing window forces R(x,y) to zero.

Let R(x,y) be represented by an infinite sum of its Fourier coefficients. Therefore, for $|x| \le L_x$ and $|y| \le L_y$,

$$R(x,y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} A_{nm} e^{\frac{i\pi mx}{L_x}} e^{\frac{i\pi ny}{L_y}} \,, \tag{2.6}$$

where A_{nm} are the corresponding Fourier coefficients.

Since R(x,y) must be symmetric, $A_{-m-n} = A_{mn}$. The individual terms of equation 2.6 can now be rewritten as,

$$\begin{aligned} & A_{mn} e^{\frac{i\pi mx}{L_x}} e^{\frac{i\pi ny}{L_y}} + A_{-m-n} e^{\frac{-i\pi mx}{L_x}} e^{\frac{-i\pi ny}{L_y}} = \\ & 2A_{mn} \cos\left(\frac{\pi mx}{L_x} + \frac{\pi ny}{L_y}\right) = \\ & 2A_{mn}\left[\cos\left(\frac{\pi mx}{L_x}\right)\cos\left(\frac{\pi ny}{L_y}\right) - \sin\left(\frac{\pi mx}{L_x}\right)\sin\left(\frac{\pi ny}{L_y}\right)\right] . \end{aligned} \tag{2.7}$$

Expanding this and rewriting the whole summation from equation 2.6 gives

$$\begin{aligned} R(x,y) &= \sum_{n=-\infty}^{\infty} A_{n0} \cos\left(\frac{\pi ny}{L_y}\right) + \sum_{m=1}^{\infty} \sum_{n=-\infty}^{\infty} 2A_{nm}\left[\cos\left(\frac{\pi mx}{L_x}\right)\cos\left(\frac{\pi ny}{L_y}\right) - \sin\left(\frac{\pi mx}{L_x}\right)\sin\left(\frac{\pi ny}{L_y}\right)\right] \\ &= A_{00} + 2\sum_{n=1}^{\infty} A_{n0} \cos\left(\frac{\pi ny}{L_y}\right) + \sum_{m=1}^{\infty} \sum_{n=-\infty}^{\infty} 2A_{nm}\left[\cos\left(\frac{\pi mx}{L_x}\right)\cos\left(\frac{\pi ny}{L_y}\right) - \sin\left(\frac{\pi mx}{L_x}\right)\sin\left(\frac{\pi ny}{L_y}\right)\right] . \end{aligned} \tag{2.8}$$

For the wave gage, the vectors between any two gage elements have a prescribed set of values (x_p,y_p) for $p=1,2,...,p_{max}$. The autocorrelation function evaluated at the location of each of these vectors is approximated as the time average of the products of the surface elevations at the two corresponding gage elements. Since discrete values of the autocorrelation function, $R(x_p,y_p)$, can be calculated from measured surface elevation data, then the series from equation 2.8 can be approximated by its truncated form:

$$R(x_p, y_p) \cong A_{00} + 2\sum_{n=1}^{N-1} A_{n0} \cos(\frac{\pi n y_p}{L_y}) + 2\sum_{m=1}^{M-1}\sum_{n=-(N-1)}^{N-1} A_{nm}[\cos(\frac{\pi m x_p}{L_x})\cos(\frac{\pi n y_p}{L_y}) - \sin(\frac{\pi m x_p}{L_x})\sin(\frac{\pi n y_p}{L_y})] . \quad (2.9)$$

The unknowns in this equation are the Fourier coefficients, which are the A_{mn} 's. There are M(2N-1)-N+1 of these unknown coefficients. If enough values of the autocorrelation are known, this system of equations can be solved for the A_{mn}'s. In other words, if

$$p_{max} > M(2N-1)-N+1 , \quad (2.10)$$

then one can use a minimum mean square error matrix solver technique to find the Fourier coefficients of the autocorrelation matrix.

2.3.2 The Directional Spectrum

The classical method of determining a two-dimensional spectrum of a two-dimensional signal is to take the Fourier transform of the autocorrelation of that signal. For a continuous signal this corresponds to computing the following integral,

$$S(k_x, k_y) = \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} R(x,y)e^{-i(k_x x + k_y y)}dx\,dy , \quad (2.11)$$

where R(x,y) is the spatial autocorrelation of the signal. If the representation of the autocorrelation from equation 2.6 is substituted for the continuous autocorrelation in equation 2.11, the spectrum can be rewritten in the form

$$S(k_x, k_y) = \frac{1}{(2\pi)^2}\sum_{n=-\infty}^{\infty}\sum_{m=-\infty}^{\infty}\int_{-L_y}^{L_y}\int_{-L_x}^{L_x} A_{mn} e^{\frac{i\pi m x}{L_x}} e^{\frac{i\pi n y}{L_y}} e^{-ik_x x} e^{-ik_y y}dx\,dy , \quad (2.12)$$

which is rewritten as:

$$S(k_x, k_y) = \frac{1}{(2\pi)^2}\sum_{n=-\infty}^{\infty}\sum_{m=-\infty}^{\infty}\int_{-L_y}^{L_y}\int_{-L_x}^{L_x} A_{mn} e^{ix(\frac{\pi m}{L_x}-k_x)} e^{iy(\frac{\pi n}{L_y}-k_y)}dx\,dy . \quad (2.13)$$

The integrals can now be evaluated by quadrature, giving

$$S(k_x,k_y)=\frac{1}{(2\pi)^2}\sum_{n=-\infty}^{\infty}\sum_{m=-\infty}^{\infty}A_{mn}\frac{2\sin[L_x(\frac{\pi m}{L_x}-k_x)]}{(\frac{\pi m}{L_x}-k_x)}\cdot\frac{2\sin[L_y(\frac{\pi n}{L_y}-k_y)]}{(\frac{\pi n}{L_y}-k_y)}\,. \qquad (2.14)$$

Simplifying this result gives

$$S(k_x,k_y)=\frac{4}{(2\pi)^2}\sum_{n=-\infty}^{\infty}\sum_{m=-\infty}^{\infty}A_{mn}\frac{\sin(\pi m-L_xk_x)}{(\pi m-L_xk_x)}\cdot\frac{\sin(\pi n-L_yk_y)}{(\pi n-L_yk_y)}\,. \qquad (2.15)$$

This equation can be used to calculate the spectrum at any wave number values, k_x and k_y, assuming the A_{mn} coefficients are known. For particular values of k_x and k_y the formula simplifies even further, where

for $k_x=\frac{\pi m'}{L_x}$,

$$S(k_x,k_y)=\frac{4}{(2\pi)^2}\sum_{n=-\infty}^{\infty}A_{m'n}L_xL_y\frac{\sin(n\pi-L_yk_y)}{(n\pi-L_yk_y)} \qquad (2.16)$$

and for $k_y=\frac{\pi n'}{L_y}$,

$$S(k_x,k_y)=\frac{4}{(2\pi)^2}\sum_{m=-\infty}^{\infty}A_{mn'}L_xL_y\frac{\sin(m\pi-L_xk_x)}{(m\pi-L_xk_x)}\,. \qquad (2.17)$$

Furthermore, if $k_x=\frac{\pi m'}{L_x}$ and $k_y=\frac{\pi n'}{L_y}$ then

$$S(k_x,k_y)=\frac{4}{(2\pi)^2}A_{m'n'}L_xL_y\,. \qquad (2.18)$$

From equation 2.9 the A_{mn} coefficients can be calculated for m = 1, 2, ... ,M-1 and n = -(N-1), -(N-1) + 1, ... ,0, ... ,(N-1).

3. INSTRUMENT DESIGN OPTIMIZATION

The current circular wave gage has continually evolved through improvement on previous work and implementation of new ideas about how to make the instrument more user friendly, more cost effective, and more accurate. The current development focuses on the effort to use and test the wave gage in its proper environment, the ocean. This chapter describes the current physical design of the gage, the testing and development previously performed, the changes and modifications effected in this stage of the gage's development cycle, and the data flow through the analysis code which produces the final output plot of the directional spectrum.

3.1 INSTRUMENT LAYOUT

The circular wave gage consists of three major components shown in Figure 3.1.

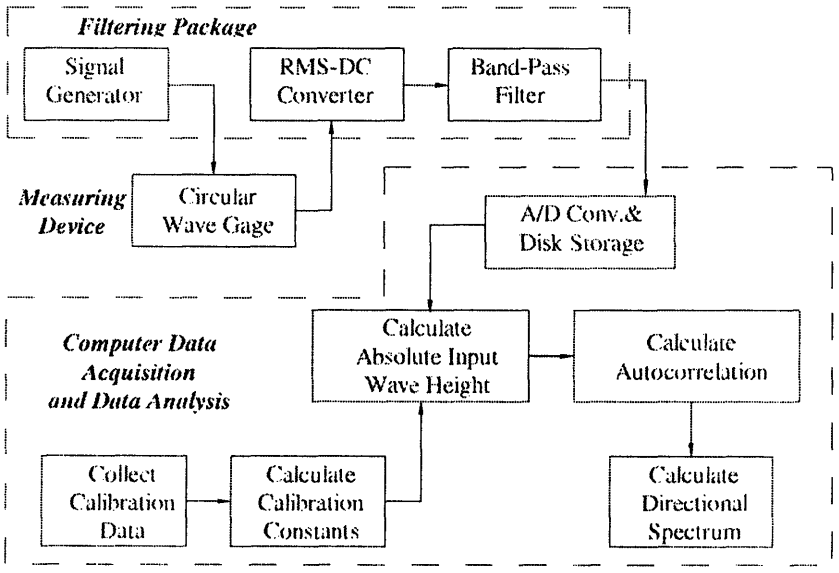

FIGURE 3.1: Components of Circular Wave Gage

The first component of the system is the measuring device, which stands approximately a meter and a half high. Its top plate is made of acrylic to act as an electrical insulator and its baseplate is made of stainless steel to act as a conductor. A 3/8 inch diameter, 1.3 meter-long stainless steel center post is used to separate these two plates and act as the conductor from the base plate to the electrical ground of the instrument electronics. An array of 64 34-gauge nichrome wires are attached vertically between the top and base plates. They are arranged in a circle with a diameter of 20 centimeters and have a resistance per length of 17 ohms/ft. Each is wound and twist-tied to a stainless steel hitch pin at the top of the gage and soldered to a small stainless steel spring at the base. The springs hold the wires taut and minimize the wire breakage when debris contacts the wires. Each spring is soldered to a stainless steel thumb screw which is connected to the base of the gage with a nut. The nut is tightened on the bottom of the base plate and holds the shoulder of the thumb screw in constant contact with the base plate. Water-tight electrical connectors on the top plate allow signals to be transferred back to the filtering package. This configuration is shown in detail in Figure 3.2.

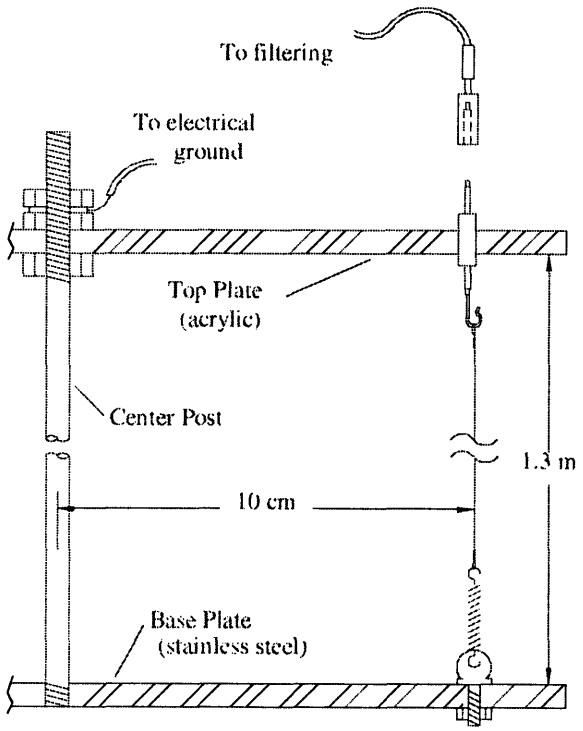

FIGURE 3.2: Wave Gage Wire Connection Diagram

The operation of the circular wave gage is very straightforward. Each of the resistive wires on the circular gage acts as a voltage divider, as shown in Figure 3.3.

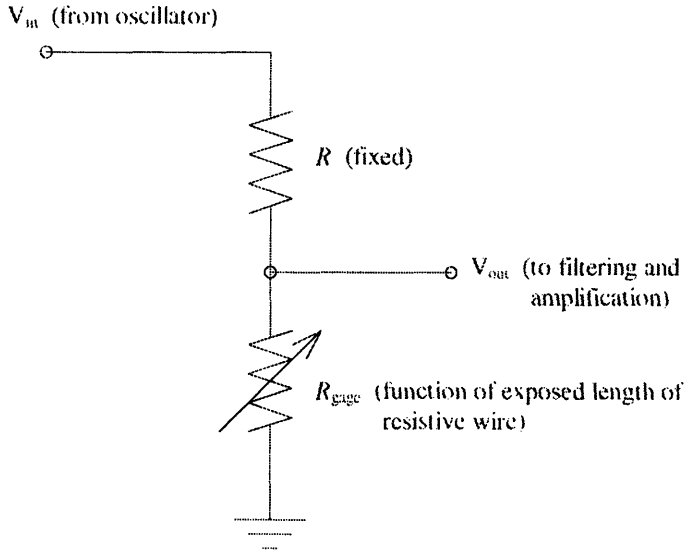

FIGURE 3.3: Voltage Divider Relationship for Wave Gage

The relationship between the input and output voltages of a voltage divider is mathematically represented by

$$V_{out} = \frac{R_{gage}}{R + R_{gage}} \cdot V_{in} \ . \tag{3.1}$$

If R is significantly greater than R_{gage}, this relationship reduces to

$$V_{out} = \frac{R_{gage}}{R} \cdot V_{in} \ . \tag{3.2}$$

Since R is fixed, the output voltage is linearly related to the input voltage by R_{gage} . Furthermore, the seawater, because it is very conductive, acts almost as a short circuit between the point where each resistive wire pierces the water surface and the center post of the gage. This center post is connected to the electrical ground of the electronics package. Therefore, as the level of the seawater on each of the wires changes, the effective resistance of each of the wires, R_{gage} , changes. As the effective resistance changes, the voltage measured across the voltage divider changes. The voltage measured

across each wire is, therefore, linearly related to the sea-surface elevation at that point. This linear relationship allows the filtered output signal from the wave gage to be analyzed and used to reconstruct the input surface elevations, which is the goal of the physical instrument design.

The effectiveness of the wave gage is somewhat limited by the fact that the voltage along the wire does not immediately become zero at the sea-surface interface as illustrated in Figure 3.4. If the water level is near the bottom of the gage, the effective resistance across the gage no longer remains linearly related to the sea-surface elevation. The linear relationship holds true only if the primary path to ground is through the sea water. The water level across the gage must, therefore, remain at least 15 to 20 cm from the base of the instrument.

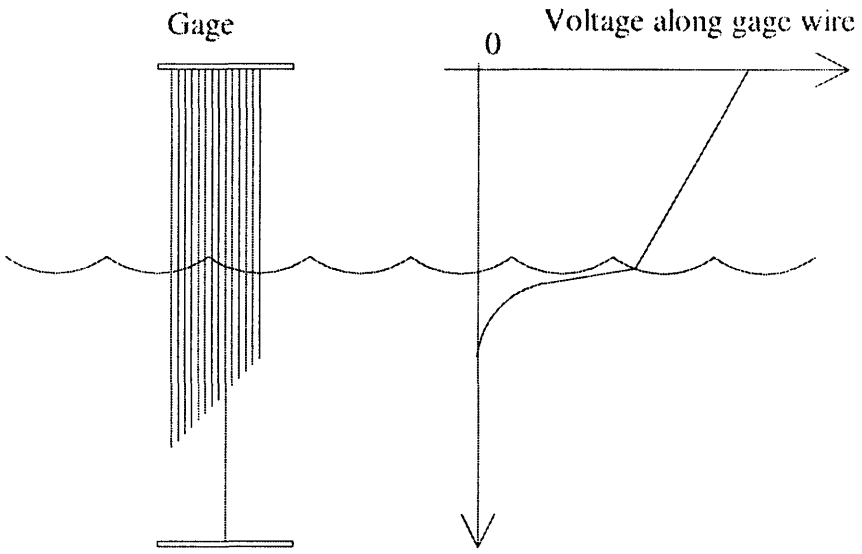

FIGURE 3.4: Voltage Along Each Wave Gage Wire as a Function of Water Level

It is also important to note that the input voltage sent to the wave gage is an oscillating signal. This is used because a DC voltage along a wire submersed in water will hydrolyze the water in contact with it. The resulting bubbles forming along the wires would constantly alter their effective resistance, which is definitely not desired. A high-frequency

(10 kHz) wave is, therefore, used as a carrier wave for the lower frequency measured input wave signal. The input signal is subsequently converted to an analog voltage by an RMS-DC converter which is part of the electronic system.

The filtering package consists of a number of internal parts. The first is an intermediate filter board. This board acts as a junction between the raw input signal, the wave gage electronics, and the computer data acquisition system. It is a router for the 64 data signals as well as a final low-pass filter between the conditioning electronics and the computer. The raw input voltages from the gage are passed directly to four conditioning boards, each of which conditions sixteen of the channels. The conditioning consists of a pre-amplifier, an RMS-DC converter, a trim offset, a series of low- and high-pass filters, and a final adjustable output gain. Figure 3.5 shows the configuration of this system. A detailed description of each of these components is given by Chen (1994).

The final physical component of the system is the computer data acquisition system. Once the raw signals have been properly filtered, they are sampled by two A/D boards installed in a Dell 486-66 computer. One of the boards is a Computer Boards model DAS-16, which samples channels 0 through 15, and the second is a Computer Boards model DAS-48, which samples the remaining channels in sequential order. The sampled voltage signals are stored on the hard drive of the computer for further signal processing.

3.2 PREVIOUS WORK

The current modifications and continuing development of the directional wave gage are built primarily upon the work of previous MIT student David Chen and are presented in his Master's Thesis (Chen, 1994). The important results regarding the design of the saltwater wave gage are summarized here along with comments regarding their accuracy and application to the current development.

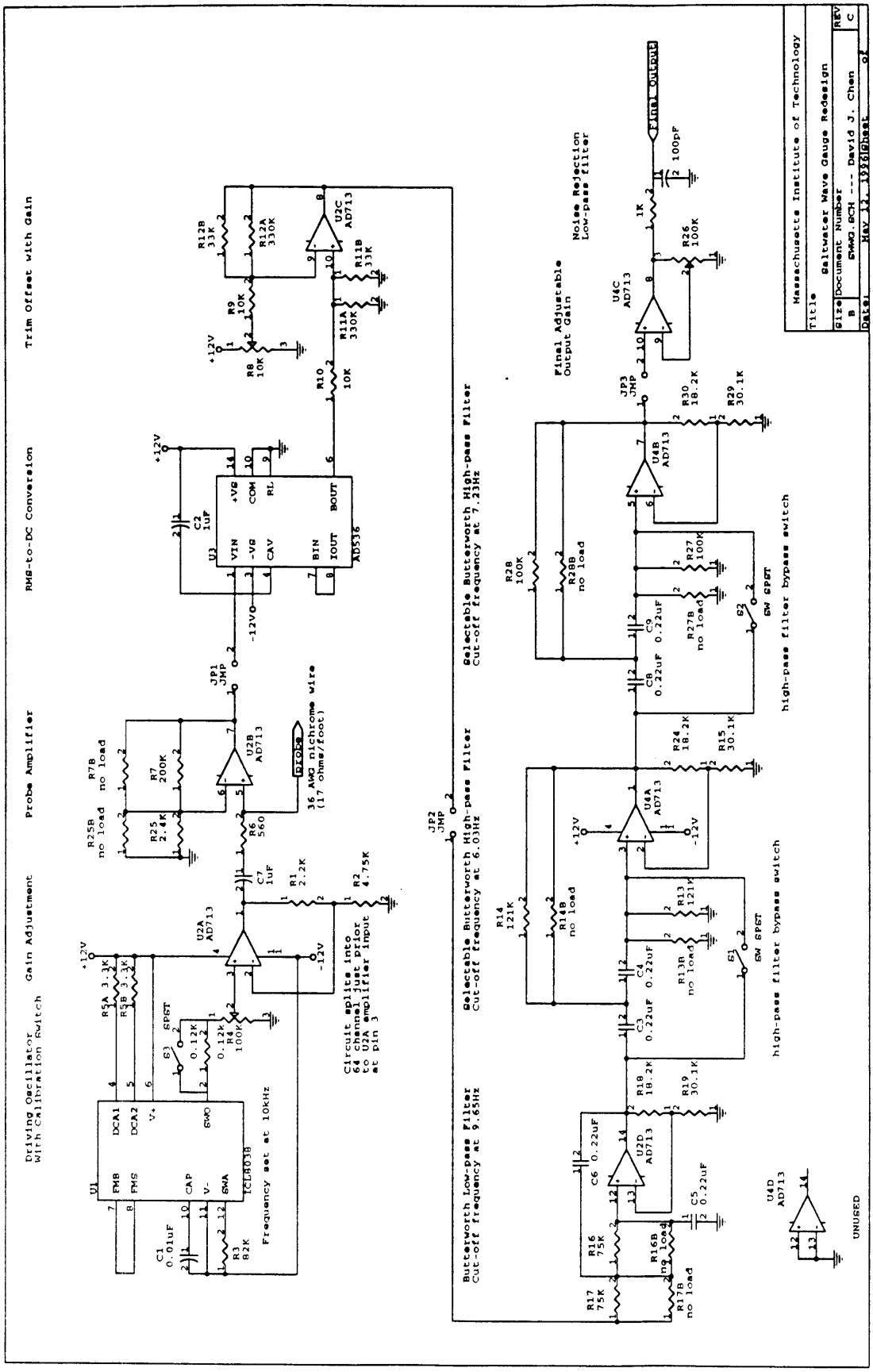

FIGURE 3.5: Circular Wave Gage Circuit Diagram

Chen presented a detailed analysis of the linear relationship between the effective resistance of a submerged resistive wire and the height of the saltwater on this wire. He then developed a circuit theory and mechanical design using this model. The circuit shown in Figure 3.5 is Chen's design. Minor modifications have been made to this design which will be discussed later in this Chapter.

Part of the currently used data analysis code was also developed by Chen and other students working with him. A data acquisition program was developed which sampled the 64 output voltages measured by the four circuit boards via the two data acquisition boards previously described. A DC calibration system was also developed which measured the voltage output of the wave gage at various static positions in a flat saltwater bath with the filtering mechanism effectively bypassed. This was done because the filtering system removes all DC input signals, and if there are not any waves, the filtered output will be zero. The slope of the resulting relationship between voltage and amount of submerged wire was approximately linear and was used as a calibration constant for each channel.

During actual wave measurements, Chen used a data acquisition sampling rate of 20 Hz and data was collected for 26 seconds. The collected voltage file was transformed into a file of wave heights in inches by multiplying each wire's voltage signal by its respective calibration constants. Any mean values of the channels were then subtracted out.

The calibration mechanism developed by Chen as described in his thesis is inadequate to accurately determine the height of input waves, because the filtering mechanism which he by-passes during his calibration procedure is not constant over time or between channels. The only way for the actual wave heights of a sine wave input to be determined would be for the output voltage signals to be multiplied not only by the calibration constants he measured, but also by the corresponding filter amplitude at the particular frequency of the input wave. This may, indeed, be what he did to achieve the amplitude consistency and resolution he reported in his thesis, but it is not described anywhere therein. This is one of the primary reasons that a new calibration scheme was developed.

The rest of the signal processing was performed in three steps. The first was a configuration step which formatted the wave height information so that it was compatible with the subsequent data analysis code. The second stage involved computing the two-dimensional time-averaged autocorrelation. The final stage produced the directional spectrum from the autocorrelation.

Tests were conducted to evaluate the immersed wire voltage relationship assumptions used in the design of the saltwater wave gage. The voltage along the wire was confirmed to be linear along the portion of the wire not immersed in the water. Also, along the submerged portion of the wire the voltage decayed exponentially. These results confirm the theoretical predictions and are the basis of Figure 3.4.

Tests were also performed which showed the dependence of the calibration constants on water salinity. Chen concluded that, aside from a DC offset, the calibration slopes would be approximately constant as long as the salinity was at least 10 ppt. It was also determined that the cleanliness of the wires played a role in the calibration constants. Over time, dirty wires caused the calibration slopes to become somewhat non-linear.

Chen determined that the gage calibration would not be significantly altered by the water's salinity as long as the salinity exceeded 10 ppt, where standard sea water has a salinity of nearly 35 ppt. These conclusions justify the use of salt water with a salinity of only 15 ppt for the tests performed in this research.

The frequency response curves for each channel were also compared. This comparison confirmed the assumption that they were all nearly identical in shape.

Extensive tests were performed to analyze how sensitive the system was to noise. It was concluded that some of the components of the system were very susceptible to interference by unwanted crosstalk and radio-frequency interference from power lines.

Using a mechanical oscillator in a 'bucket' of saltwater the gage was tested to insure that the amplitude and frequency content of the processed output signals was correct. It was found that the precision of the gage in this controlled environment was consistently within 5 percent of the expected output. The accuracy of the instrument depended on the oscillation frequency. For lower frequencies the amplitudes were within 1 percent of the actual values, but as the frequency increased, the amplitudes became less accurate. Explanations for this pointed to the testing apparatus, not the instrument itself.

Testing similar to the bucket testing was also performed in a wave tank. It was shown that the accuracy and precision tests were repeatable, with a slightly decreased signal-to-noise ratio for low amplitude input signals. This decrease was explained by the fact that the noise appeared to remain constant while the wave gage output signal varied. If the output signal was small, the signal-to-noise ratio decreased. This effect indicated the existence of a noise limitation in the system.

The remaining performance criterion for the wave gage was that it accurately discern directional information from the input signals. These tests were also performed in a wave tank. The directional information was plotted using TECPLOT. It was determined that the wave number resolution was accurate and the directional content of the signal was preserved. The wave number resolution limit of the gage was calculated from geometric limitations related to the size of the wave gage. Analysis of the instrument resolution beyond this investigation was not undertaken.

The accuracy and precision estimates made by Chen must be carefully weighed because the method of calibration he used is not either accurately or adequately described in his thesis. His analytic results, therefore, cannot be treated as absolute. Furthermore, the smoothing window used in the calculation of the directional spectrum has a width of just over 0.3 rad/cm in wave number. His determination that the wave number resolution is nearly $\pi/20$ rad/cm (0.16 rad/cm) is, thus, difficult to understand.

Chen concluded that the directional wave gage was an effective tool for measuring both the energy and directional content in short waves on the sea surface. He further suggested that some modifications be made to increase the signal-to-noise ratio and to improve the ruggedness of the instrument.

His suggestions were used as a starting point for the modifications made during the further development of the wave gage.

3.3 GAGE MODIFICATIONS

Many changes to the existing physical design of the circular wave gage were implemented to increase its functionality and to fill in gaps left in the previous work. The most important of these concerned modifications to permit and facilitate on-the-water use of the instrument. The concept for the use of the instrument had been proven, but it needed to be tested in the real-world environment. To accomplish this task there were several issues that needed to be addressed. One was improving the physical ground connections on the gage itself. The second was increasing the signal-to-noise ratio in the system. It was too sensitive to external noise sources and internal crosstalk between channels. The data acquisition system also needed to be streamlined to insure that aliasing was effectively reduced. Finally, the addition of a compass was proposed to aid in relating the final directional spectrum output to the actual wind direction and gage position in a global reference frame.

3.3.1 Grounding Issues

One of the problems consistently encountered during the onset of this project was the inability to keep all of the channels electrically grounded. Most of the difficulty came from the connections on the submerged portion of the wave gage. Because the base stainless steel plate acts as the ground for the current traveling through the wires, if the connection between the wires and this plate is not consistent, the output voltage signal will fluctuate.

This connection, however, is very difficult to maintain because of the highly corrosive environment of sea water. Many different attempts were made to increase the durability of this connection. One important criterion that had to be met was that the wires had to be easily replaceable because they sometimes broke or stretched during use. Furthermore, the diameter of the nichrome wire could not be significantly modified to help alleviate this problem because its resistance per length, which was determined by its diameter, had to remain at approximately the design magnitude. Also, the spring and any connecting mechanism had to be corrosion resistant to prevent degradation in the salt-water environment. After many trials using various soldering techniques and multiple spring-to-base connectors, a solution was finally found. The sping had to be silver-soldered to both the nichrome wire and a thumb screw. The thumb screw had a shoulder on it which could be tightened against the base plate with a nut tightened on the bottom of the gage. This configuration was shown in Figure 3.2. The soldering process took longer to perform than was desired, but it proved to be a good solution to the grounding issues and allowed for easy replacement of broken wires.

A further suggestion was proposed to improve the grounding problem by removing the ground connection at the base of the gage altogether. Instead of slightly improving the problem, why not simply remove it? This would have been, of course, easier to implement. The problem with this method was that the effective resistance of the path through the sea water to the ground grew non-linearly to near infinite values near the bottom of the gage. The assumption, then, that the resistance across the gage is significantly less than the resistance across the top of the voltage divider would be violated. This, in turn, would mean that the linearity assumption upon which the instrument previously operated, and which is described in equation 3.2, would no longer hold. If this relationship no longer held, completely new methods for analyzing the output data would have had to be developed. It was not the purpose of this research to completely redevelop the operating and processing system used by the gage, so it was rejected as a possible solution to the grounding problem.

3.3.2 Noise Reduction

Another important issue that needed to be addressed was the noise in the system. The voltage signals generated by the sea-surface input are very small and, thus, can easily be corrupted by noise introduced into the system prior to the filtering and amplification processes. There are many locations where this noise could enter the system, and the problem is compounded by the fact that there are sixty-four channels which have to be shielded from noise instead of just one or two. The primary locations for noise input were assumed to be the previously unshielded cables from the gage itself to the electronics and the cables from the electronics to the computer, which acted as antennae. To remove this source of noise, properly shielded cables were installed.

Another source of noise was found to be the four sine wave signal generators used to create the high-frequency carrier wave sent to the wave gage. Each of these oscillators generated a carrier wave for sixteen of the channels. It was determined that the four oscillators were not operating at exactly the same frequency. The cross-talk generated by this small difference in frequencies was adding high-frequency noise to the data signal and corrupting it. This was remedied by removing three of the oscillators and driving all sixty-four channels with a single oscillator. The remaining oscillator functions at the upper bound of its operational limits, but its use significantly reduces the noise measurable in the system.

3.3.3 Data Acquisition

Aliasing was also a potential problem. In order to ensure that aliasing was adequately eliminated, the characteristics of the input signal needed to be fully anticipated. The data sampling rate then needed to be set at least two times as high as the highest wave frequency expected in the input signal (also known as the Nyquist frequency). In this case, the highest wave frequency of interest lies around 10 Hz, but the highest wave frequency expected on the sea surface could be as high as 16 or 20 Hz. To ensure that the information in the frequency range of interest not be corrupted, therefore, a sampling frequency of 32 Hz is used. Some aliasing may still occur from input frequencies above

16 Hz, but it should affect only the range of frequencies well above the 10 Hz interest level.

3.3.4 Addition of Compass

To incorporate absolute direction into the final directional spectrum a compass was added to the system. The compass is a KVH flux-gate model which would be mounted on the top of the gage in the instrument's final form. Currently, it is housed in a separate case which is manually aligned with a particular channel on the gage and fixed to the platform that the gage is being suspended from. If the gage platform rotates, then the compass also rotates. The data acquisition program reads the compass heading before every scan of the wave gage wires. This information is then used to renumber the channels so that the channel closest to the North heading becomes channel zero. This allows the directional information of the input signal to be retained even if the gage is dynamically rotated.

3.4 CALIBRATION

A key issue that must be addressed for the effective use of any instrument is how to accurately calibrate it. Without proper calibration, the output of the instrument will not represent the wave field. Obtaining an accurate calibration procedure for this wave gage is accomplished in two steps. The first of these is modeling the relationship between the output of the instrument and the input. In this case the output of the instrument is a time series of voltages for each channel and the input is the actual ocean surface wave height time series at each of the wire locations. The relationship between these two signals can be represented in the frequency domain using transform techniques and is known as the frequency response. The second step is to compare this model with the actual relationship by generating a known input to the system and measuring the resulting output.

3.4.1 Use of the Calibration Constant

The time domain representation of a single channel is shown in Figure 3.6.

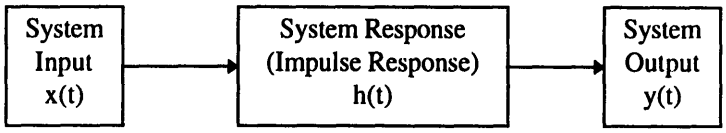

FIGURE 3.6: Time Domain Signal Flow

One can look at this system not only in the time domain as shown above, but also in the frequency domain. The frequency domain characterization, however, is an easier tool to use for understanding the relationships between input and output signals. To transform the time signals into the frequency domain the Fourier transform has to be used. The Fourier transform is defined as

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi ft} dt \ . \tag{3.3}$$

Note that the result of this integration can (and often does) produce a complex (real plus imaginary) result. Transforming the system described in Figure 3.6 into the frequency domain gives the result in Figure 3.7.

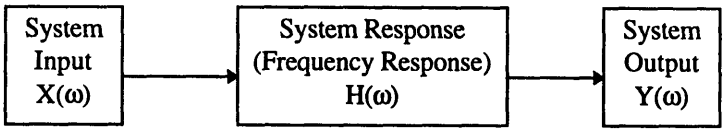

FIGURE 3.7: Frequency Domain Signal Flow

It can be shown using basic signal processing theory and the definition of the Fourier transform given in equation 3.3 that the following relationship is true for any causal system of this form (Newland, 1993):

$$Y(\omega) = H(\omega)\, X(\omega) \ . \tag{3.4}$$

Therefore, if both the input and output of a system were known in time, they could be transformed into the frequency domain and used to generate the frequency response function $H(\omega)$. Once the frequency response has been calculated, it can be used with the system output to accurately reconstruct the input signal. This is the goal of calibration.

For any analog electronic system, like the wave gage, the shape of the frequency response function, $H(\omega)$, can be modeled using the circuit values of the various circuit elements that make up the frequency-dependent portions of the system. In the case of the wave gage, these elements would be the resistors and capacitors used to create the low- and high-pass filters. This model can predict the shape of the frequency response, but it cannot accurately predict the magnitude or the exact location of the peak of the frequency response. The reason for this is that the values of the circuit elements have some uncertainty associated with them. For instance, resistors are generally purchased with specific tolerances such as 1 or 5 percent. The *exact* values of all the circuit elements are not known. More important, the values of these elements can and do change with temperature. The modeled theoretical frequency response cannot, therefore, be used alone to fully represent the characteristics of the system.

The determination of the correction factor needed to adjust the magnitude of the theoretical frequency response so that it matches the actual frequency response is defined as the calibration constant for each channel. Before the absolute magnitude can be determined, however, the correct phase relationship must be established. In other words, before the theoretical frequency response is multiplied by a calibration factor to correct its amplitude, its shape must be properly shifted along the frequency axis so that its peak lines up with the peak of the actual frequency response. This is accomplished by comparing the theoretical frequency response with an actual frequency response determined by sending the system a known input and measuring the corresponding output. If the input and output are known, the frequency response can be determined using the relationship shown in equation 3.4. The easiest way to generate a known input is to induce a step into the system. There is a unique relationship between the step response and the frequency

response which can be determined using Fourier analysis methods (Newland, 1993). This relationship is

$$H_s(\omega) = Y_s(\omega) \cdot \frac{i\omega}{stepheight} \ , \tag{3.5}$$

where $Y_s(\omega)$ is the step response of the system and the stepheight is the length in centimeters of nichrome wire that is shorted out to generate this step. In actuality, the step is made by stepping the voltage instead of the resistance. The relationship between the two will be described in the next section which details the generation of the calibration constants. The exact timing of the step is not important because the purpose of comparing the measured frequency response with the theoretical frequency response is only to properly determine the shape by comparing each of their magnitudes. In fact, this method uses the circuit values as variables and fits the equation for the theoretical frequency response to the measured frequency response for each channel. These fitted frequency response functions, $G_i(\omega)$, now have all the proper phase properties and are the proper shape, but do not necessarily have the proper absolute magnitude.

Since, for each channel, the only difference between $G_i(\omega)$ and the actual transfer function, $H_i(\omega)$, by definition is some unknown constant, then

$$H_i(\omega) = A_i G_i(\omega) \quad \text{for } i = 1,2,\ldots,64 \ , \tag{3.6}$$

where A_i are called the calibration constants and i represents the channel the constant is associated with. For any output of the i^{th} channel, $Y_i(\omega)$, therefore, the input can be recreated using equation 3.4:

$$Y_i(\omega) = H_i(\omega)X_i(\omega) = A_i G_i(\omega)X_i(\omega) \tag{3.7}$$

so

$$X_i(\omega) = Y_i(\omega) / [\, A_i \, G_i(\omega)], \tag{3.8}$$

where $Y_i(\omega)$ can be calculated from its corresponding time series using a Fourier transform, and A_i has units of cm/V .

3.4.2 Generation of Calibration Constants

The procedures for generating the calibration constants were briefly described in the previous section. The details, however, are somewhat involved. The steps involved are as follows: inciting a known amplitude step into the system, measuring the response of the system, calculating the system frequency response from the measured output, fitting the theoretical frequency response to the calculated frequency response, and then generating the calibration constants for each channel by comparing the integral of the measured and fitted frequency response function to the normalized theoretical frequency response.

A step is applied to the system by stepping the voltage across the wave gage. Recall that the wave gage acts as the bottom half of a voltage divider, as shown previously in Figure 3.3. Further recall that the input voltage is related to the output voltage by

$$V_{out} = \frac{R_{gage}}{R} \cdot V_{in} \ , \tag{3.2}$$

where R has a fixed value much greater than R_{gage}. There are two ways that a step input can be generated. An instantaneous change in resistance can be made (using a switch, for instance) or an instantaneous change in voltage can be made. Based on equation 3.2, these two kinds of steps will produce the same result. The voltage step is used because it is easier to implement.

The voltage step is accomplished by adding a switch just before the wave gage, which shorts a resistor. When the switch is thrown open, the high-frequency carrier wave voltage must pass through a resistor which reduces its voltage by a small amount as shown in Figure 3.8. This happens before the circuit splits into the 64 separate channels. It is assumed that this voltage step, which is identical for all channels, causes the same voltage drop across the gage for every wire. This, of course, requires that all the wave gage wires have nearly identical properties. To verify this assumption that the voltage step across each wire is the same, the voltage step was measured for 48 of the 64 channels. The voltage step was within 10 mV of its approximate value of 230 mV on every channel. This difference in voltage step value is less than 5 percent of the total step height. It was

further determined that those channels that had either the lowest or highest voltage step values were not correlated to the lowest or highest calibration constants. These results validate the assumption that the voltage step is nearly the same across all the channels.

The time history response of the system is measured using the program calib1.c, which writes the output of the gage as sampled by the A/D converter to a file.

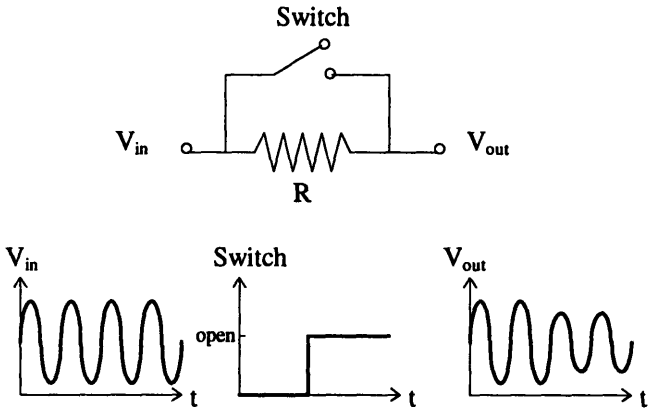

FIGURE 3.8: Voltage Step Configuration

The actual frequency response is generated using the step output data, $y_s(t)$. Recall that the relationship between the step response and the transfer function is

$$H_s(\omega) = Y_s(\omega) \cdot \frac{i\omega}{stepheight} \tag{3.5}$$

The step height was a measure of how many centimeters of wire resistance the voltage step used during the calibration procedure corresponded to. This was determined by measuring the steady-state output voltage of the RMS-DC converter before and after the voltage step was induced on a particular channel. The voltage switch was then left open and a 'short' was connected to a portion of one of the wires on the wave gage. The 'short' was adjusted until the voltage output of the RMS-DC converter was identical to the voltage output obtained after throwing the voltage switch. The amount of wire that was shorted on the wave gage wire was the height of the step used in all subsequent

calculations. Experimental results show that the voltage step corresponds to shorting approximately 2.8 cm of wire.

The actual frequency response, $G_{i,a}(\omega)$, must now be used to generate a best-fit theoretical frequency response, $G_{i,th}(\omega)$. This is accomplished using a best fit program. The program used to accomplish this is a Chi-square minimization routine which chooses the best values for the variables in the theoretical frequency response equation that will result in the best fit between the two functions. The variables that can be modified are each of the resistors in the filters, each of the capacitors in the filters, and the time constant associated with the RMS-DC converter, which acts like an additional low-pass filter. It is believed that the primary cause of the difference between the theoretical and the measured frequency responses is due to temperature change induced variations in the capacitors and the time constant of the RMS-DC converter. To simplify calculations it was assumed that since all the capacitors in the filters are identical, they would all change in the same manner if environmental conditions began to affect them. A more disciplined discussion of the calibration drift is included at the end of this section. Now, the theoretical frequency response function that is being fitted to the actual data is no longer a function of over 15 variables, but is a function of only three; one is the value of the capacitor in all the filters, the second is the value of the time constant associated with the RMS-DC converter, and the third is the magnitude of the function. The fitting program takes initial guesses for these three variables and finds best-fit values for them. These best-fit values are then used to generate a new frequency response function that more accurately describes the system. The plot in Figure 3.9 shows an example of the original frequency response function with the approximate values of the resistors, capacitors, and time constant as well as the fitted frequency response function with its modified capacitor and time constant values.

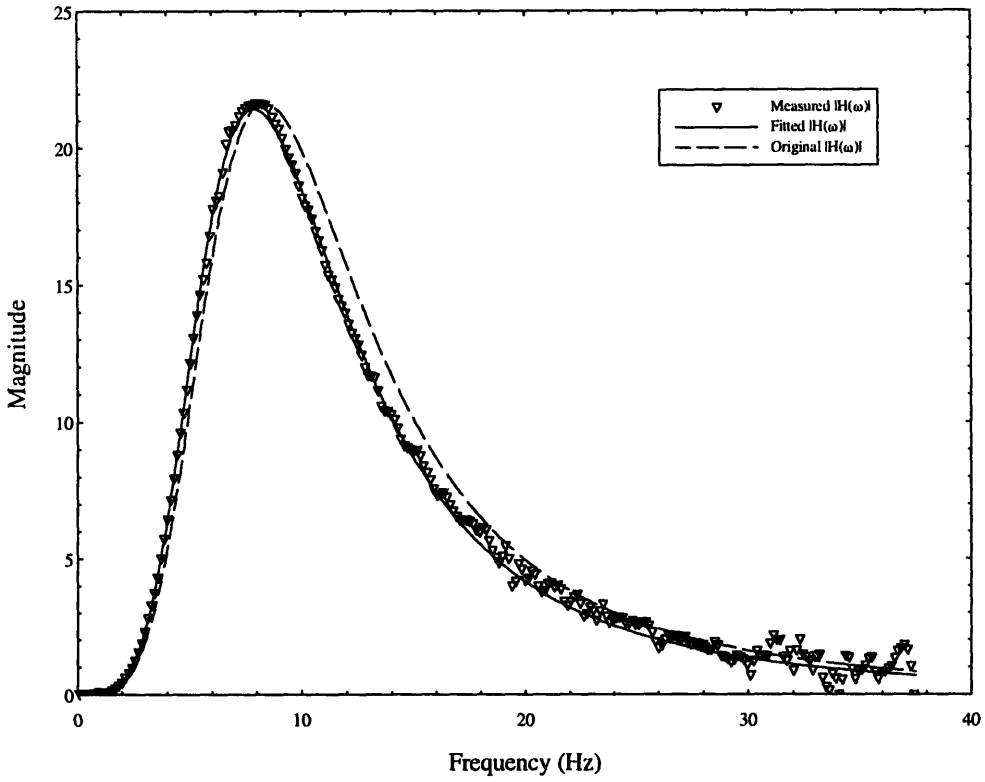

FIGURE 3.9: Sample of Fit of Frequency Response Function for Channel 0

The calibration constants are generated by comparing the area underneath the square of the fitted theoretical frequency response, $A^2_{i,th}$, and the area under the square of the measured and frequency response from the step input, $A^2_{i,m}$,

$$A^2_{i,th} = \int_{12.6 rad/s}^{62.8} |G_{i,th}(\omega)|^2 \, d\omega \tag{3.9}$$

$$A^2_{i,m} = \int_{12.6 rad/s}^{62.8} |G_{i,a}(\omega)|^2 \, d\omega \quad , \tag{3.10}$$

where

$$G_{i,a}(\omega) = A_i G_{i,th}(\omega) \tag{3.11}$$

from equation 3.6. The area comparison is bandlimited between 12.6 rad/s (2 Hz) and 62.8 rad/s (10 Hz) to prevent noise from affecting the results and because this is the particular range of interest for the instrument. The fact that the step input is not of unit magnitude is also taken into account by dividing the result of the area comparisons by the

step height. So the calibration constants, A_i, are generated using the following relationship:

$$A_i = \frac{\sqrt{(A_{i,m}^2 / A_{i,th}^2)}}{stepheight} \quad i = 0,1,2,\ldots,63 \; . \tag{3.12}$$

3.4.3 Calibration Drift

During the initial development of the calibration procedures it was assumed that the calibration constants were fairly stable over time. In other words, if calibration constants were collected right after the wave gage was turned on, five or six hours later, or days later, they would all be approximately the same. Small differences were expected, which is why calibration procedures were developed that would allow a user to take multiple calibration data sets during a single set of experiments. However, it was found after some testing that the calibration data sets drifted significantly over time. After some experimentation it was determined that the circuit elements were heating up and this was causing the drift. Figure 3.10 shows the drift of the average peak frequency of the frequency response function over time caused by the heating of the circuit elements. This experiment was performed in the lab.

This heating is caused by the housing used to enclose the electronics. Each of the four electronics boards, each having sixteen channels worth of electronics, is covered by a Plexiglas panel to prevent water or other foreign debris from affecting the performance of the board. The amplifier chips on each board heat up significantly during use to some steady-state operating temperature. The Plexiglas panels act as insulators which help raise the ambient temperature surrounding all of the elements on the circuit board. This rise in temperature causes a change in the values of the circuit elements. If the circuit elements change their values then the frequency response of the system also changes. This temperature change affects the calibration.

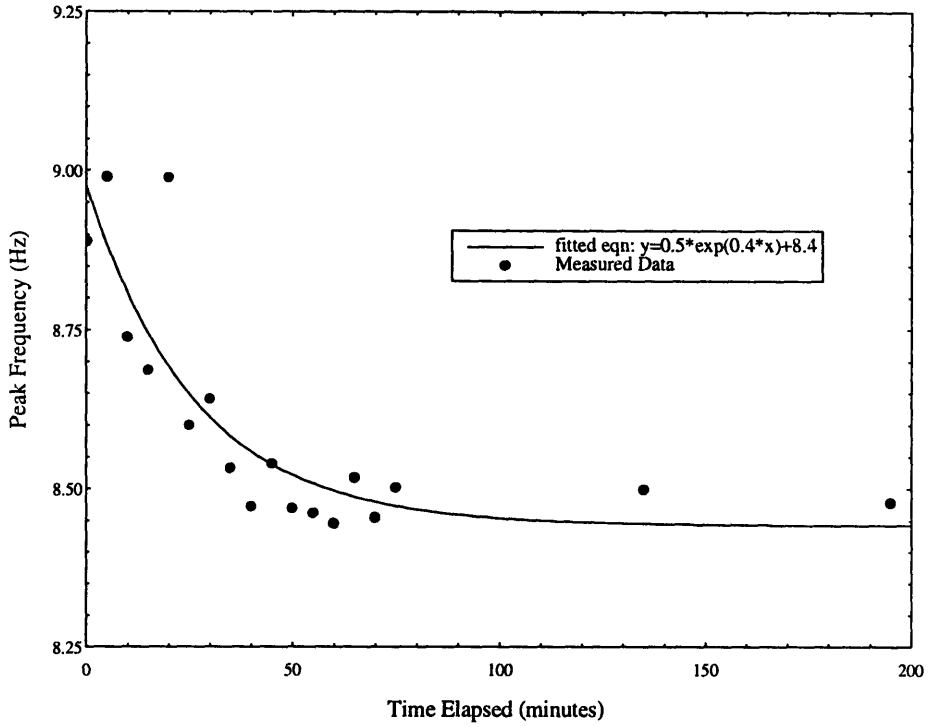

FIGURE 3.10: Location of Peak Frequency of Wave Gage Transfer Function Over Time

This hypothesis was tested using various methods, including artificially heating up a channel and measuring its calibration constant during this process. It was determined that the circuit elements most affected by the temperature were the capacitors. They are used, therefore, as one of the variables in the theoretical frequency response best fit calculations. The temperature drift was most effectively verified during one of the on-the-water tests performed on Deer Island during December. The ambient temperature was below 32° F and with wind-chill was not more than 15° F during the experiment. The average calibration constant, instead of asymptotically approaching a higher value than it had when the gage was turned on after being removed from the warm car and being properly assembled, approached a lower value.

This problem cannot easily be remedied without replacing all of the current capacitors with ones having a higher heat tolerance, or by building a well-ventilated and constant-temperature electronic housing. Neither of these options was feasible given the time and monetary constraints on the project. The current solution to this problem is to let the

instrument warm up for at least an hour before use and to calibrate both before and after each test to insure consistency.

3.5 ANALYSIS CODE

Each analysis step in the computer data acquisition portion of the circular wave gage process diagram shown in Figure 3.1 is performed by a separate program. They could all easily have been combined into one analysis program, but it was more convenient to keep them separate, as this made them each easier to debug and test for accuracy.

Some of the core code had already been at least functionally developed. This included the data acquisition program as well as the autocorrelation calculation program and the directional spectrum calculation program. The autocorrelation code was essentially unmodified. The data acquisition code, however, was altered to institute a higher data sampling rate than was used by Chen. The directional spectrum code was also modified to generate additional output files containing integrals of the spectrum as a function of wave number and radial slices of the spectrum to show directionality of the input. Some of the analysis in the spectrum code was also modified to fix minor calculation errors in the previous analysis method. Furthermore, three new pieces of code were created which incorporate the new calibration system and the additional compass information.

The following sections contain brief descriptions of each of the segments of the analysis code, including both the previously generated code and the new programs. These descriptions are not meant to expand on the discussion of the signal processing and calibration methods described earlier in this chapter, but are instead meant as a guide for those who would like to incorporate similar programming into their own work or for further understanding of the process used to fully analyze the collected wave gage data. Additional comments on program-specific information are included in the code itself, which can be found in the Appendices of this thesis.

3.5.1 Data Acquisition: cknic.c

This program collects the output data from the wave gage during its operation. It first prompts the user for an output file name, the range of wave gage channels with which the user wants to collect data, and the total number of scans the user wants to collect at a scan rate of 32 Hz. It then collects a full scan of all sixty-four channels, thirty-two times a second until the number of scans the user has indicated have been collected. Each scan of all 64 channels takes approximates 5 milliseconds, which is essentially instantaneous compared to the time scale of the input signal. The slew is discussed further in Chapter 4. For further discussion of this program please refer to Chen (1994).

INPUT: name for output data set, range of wave gage channels to collect (0-63), number of scans to collect at 32 Hz.
OUTPUT: user-named data set
This program is included in Appendix A.

3.5.2 Calibration Data Collection: calib1.c

This program is nearly identical to ck32.c. The primary difference is that it automatically collects five seconds of data at 50 Hz instead of prompting the user for this input. It also collects eight data sets instead of one. Four data sets are collected which contain the output of the gage when it is not given any input. The other four data sets contain the output of the instrument from a voltage step generated by the user throwing a switch on the gage during the five seconds of data collection.

INPUT: no input files or information required
OUTPUT: step1.dat, step2.dat, step3.dat, step4.dat, zero1.dat, zero2.dat, zero3.dat, zero4.dat
This program is included in Appendix B.

3.5.3 Calibration Constant Calculation: calcon11.for

This program calculates the calibration constant for each channel. It first reads in the step and zero data from program calib.c. It then determines if any of the channels did not receive the signal by looking at their peak output voltage value compared to what is expected. If a channel did not receive the signal it is not used in any further calculations and its calibration constant is set to zero. The Fourier transform of the data set from each channel is then taken using the program fftjm.for. The four step input data sets for each working channel are then averaged together. The result is divided by the frequency to get the frequency response function, bandlimited between 2 and 10 Hz to eliminate noise, and then squared. The area underneath this new function is compared to the area underneath the squared fitted theoretical frequency response function after it has been bandlimited over the same frequency range. The constant of proportionality between these two areas is calculated and then printed to an output file. This proportionality constant is the calibration constant. The program also prints out the real and imaginary parts of the fitted theoretical frequency response functions for use in the subsequent program in the files step1.cal and step2.cal, respectively. Various run diagnostics and channel-specific information are also printed to the screen and to secondary output files.

Incorporated in this program is the generation of the fitted frequency response function using a program from *Numerical Recipes in FORTRAN* (Press,1992) called mrqmin.f which performs the least squares fit procedure described in section 3.4.2. This program, in turn, calls mrqcof.for, covsrt.for, gaussj.for (Press,1992) and funcs.for .

INPUT: output of calib.c (step1.dat ... zero4.dat)
OUTPUT: const.cal, step.cal, step1.cal, step2.cal, stats.out, oldom.out, newom.out, avdata.out
EXTERNAL FUNCTIONS AND SUBROUTINES: fftjm.for, funcs.for, mrqmin.for, mrqcof.for, covsrt.for, gaussj.for
This program is included in Appendix C.

3.5.4 Preprocessing of Raw Data: autop1.for

This program uses the calibration constants and the total 'fitted' frequency response function to transform the collected wave gage data from a time history of voltages to a time history of sea-surface elevation (in centimeters). First, the wave gage data set and the real and imaginary parts of the frequency response function are read. Then the Fourier transform of each channel of the data set is computed. These data are then divided by the frequency response function and divided by the calibration constant as discussed in the calibration section. This result is then inverse Fourier transformed. This result is the sea-surface elevation time history, which is then stored in the autoin.dat file. During this procedure any channels that are excessively noisy or not receiving signals are replaced with interpolated data from the surrounding channels. Finally, the data are renumbered so that whichever channel is facing due north is always channel zero. This is done to retain cohesive directional information throughout the data set, as the testing platform or the wave gage itself may have moved during the course of the test. This also allows an absolute reference to be used for the directional characteristics of the final spectral output.

INPUT: const.cal, step1.cal, step2.cal, user named data set (from ck32.c), user input cutoff frequency for Fourier transform
OUTPUT: autoin.dat
This program is included in Appendix D.

3.5.5 Calculation of Autocorrelation: sw_auto.c

This program calculates the autocorrelation of the sea-surface elevation data. This is done by multiplying each data point of a particular scan by every other data point in that scan. Each of these autocorrelation values is a function of the difference in locations of each of the data points used to generate it. The ensemble average of autocorrelation of each of the 4096 scans is then calculated and printed to a file. Unlike an autocorrelation calculated over time or in one dimension, the autocorrelation calculated in this program is

a function of translation in two dimensions, x and y. A more explicit description of this program is contained in Chen (1994).

INPUT: output of autoprp9.for (autoin.dat), number of scans the user wants to evaluate
OUTPUT: correl.mas
This program is included in Appendix E.

3.5.6 Calculation of Directional Spectrum: dfsum13.for

This program calculates the directional spectrum of an input data set which contains the magnitude of the autocorrelation of the data set and the locations at which these autocorrelation values are known. First the autocorrelation data set is convolved with a smoothing window. Next the truncated Fourier series coefficients are calculated for the autocorrelation data set. Finally, these Fourier coefficients are used to calculate the directional spectrum at particular wavenumbers in both the North-South direction and the East-West direction. This method is fully described in Chapter 2.

In addition to generating the directional spectrum, the program also produces an angle spectrum and a point spectrum. The angle spectrum is a representation of the spectrum along various slices taken along a particular radial direction from the origin. The point spectrum is calculated by integrating the two-dimensional spectrum over a particular wave number. These are used to compare the results of the spectral calculations to other experimentally or theoretically generated spectra data collected by other researchers.

INPUT: output of sw_auto.c (correl.mas)
OUTPUT: s.tec, z.tec, sk.tec, anglspec.out, ptspec.out
This program is included in Appendix F.

4. DESIGN / INSTRUMENT VERFICATION

An integral part of the design process is the verification of the instrument's performance. This task is divided into two parts for the circular wave gage, one being the signal processing verification and the other being the electronics and gage design verification. This can be done because the data collection and a signal processing portions of the instrument are inherently separable. Each of the two parts can be tested for its precision and limitations which can then be used to characterize the overall performance of the instrument.

4.1 SIGNAL PROCESSING

Various methods are used to determine the accuracy of the two primary signal processing programs which calculate the autocorrelation of the input signal and its directional spectrum. The first involves simulating an input of known properties. This input is then processed and the results are compared to expected results which can be calculated using the known input. A second method involves testing only the directional spectrum program by using an alternate, more traditional, two-dimensional Fourier transform approximation. A comparison of the variance of each wire prior to processing and the final variance computed using the full spectrum plot is also used to determine global properties of the processing code. Consideration is also given to the resolution limits imposed by the smoothing windows in the calculation methods.

4.1.1 Simulated Input

The first method used a simulation program which generated a sine wave input sampled as it would have been if it had been collected by the wave gage. The simulation program

which generates this input, swsim_n2.c, is included in Appendix G. Four test cases were run with a range of wave numbers between 0.4 and 4.0 rad/cm and a constant amplitude of approximately 1 cm. The direction of each simulated input was from due west, which corresponds to a right-to-left progression across the page for the autocorrelation plots. This output was then processed using the autocorrelation and directional spectrum programs. The two-dimensional autocorrelation and the directional spectrum for each of these tests are shown in Figures 4.1 and 4.2, respectively.

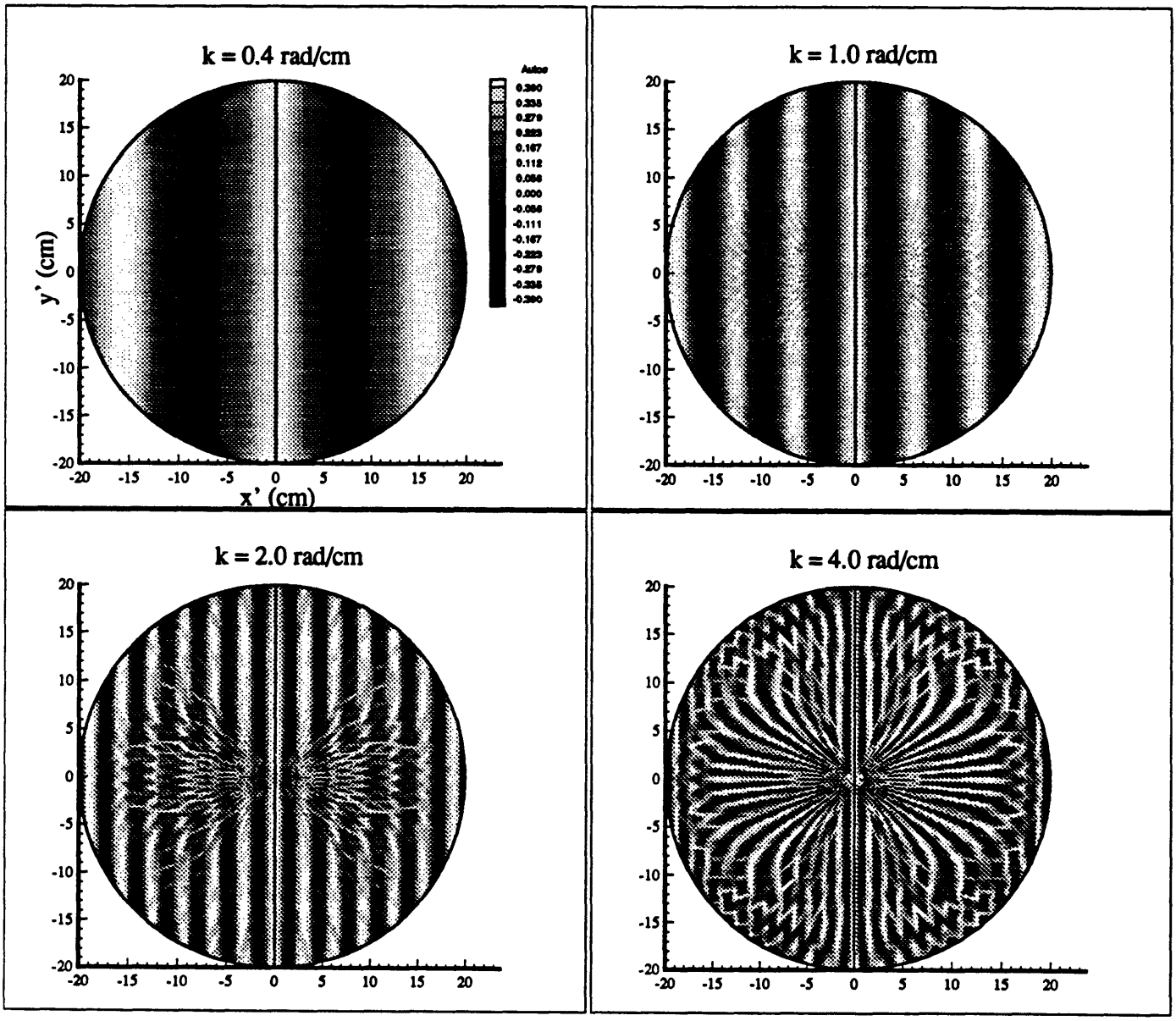

FIGURE 4.1: Autocorrelations of Simulated Sine Wave Inputs

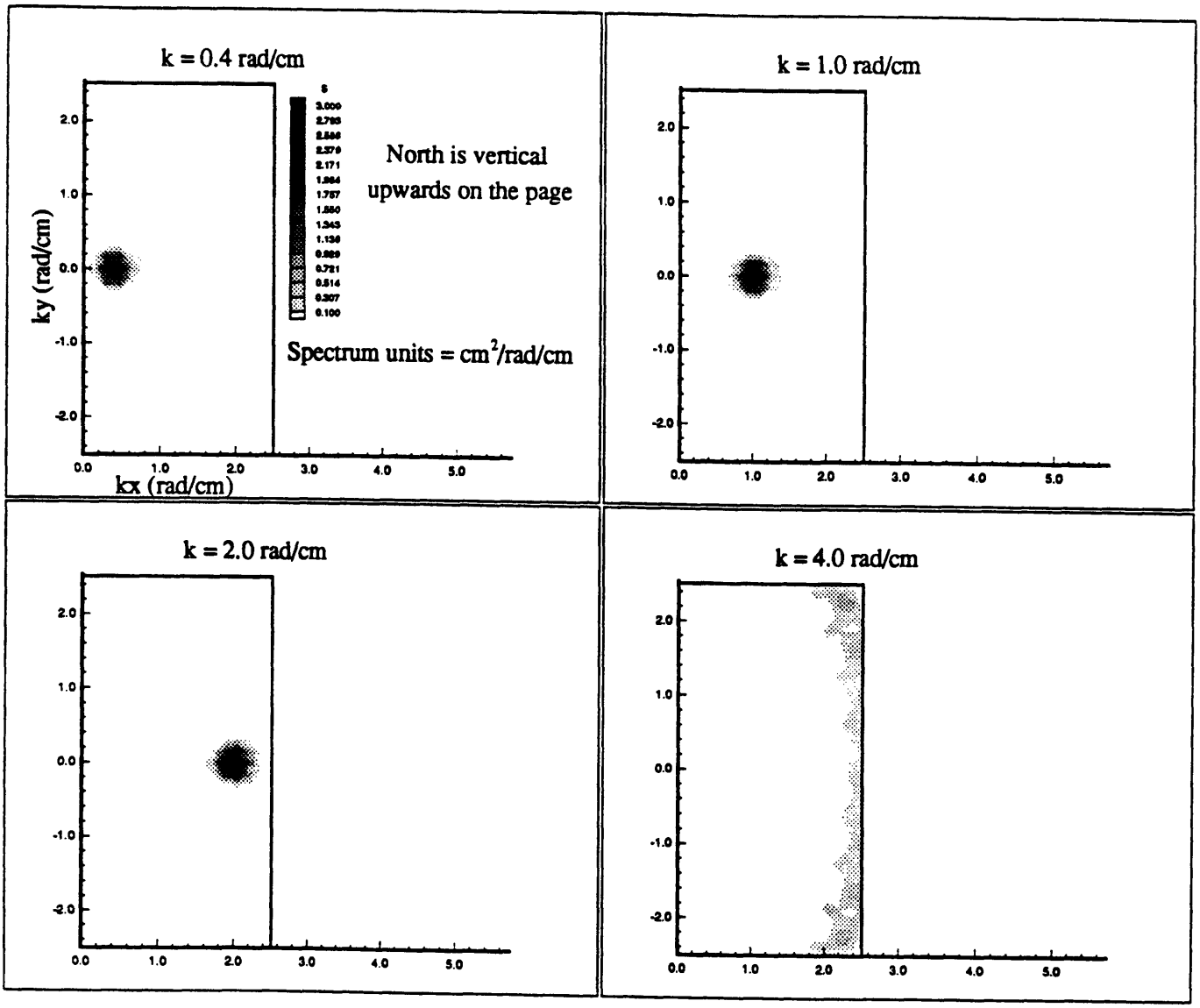

FIGURE 4.2: Directional Spectra of Simulated Sine Wave Inputs

The autocorrelation plot for a simple sine wave input should also be sinusoidal in nature. The top left autocorrelation plot shown in Figure 4.1 does, indeed, show a sinusoidal signal. The autocorrelation program does, thus, generated the expected output. The distortion which becomes noticeable at a wave number of about 2 rad/s is due to the effect of both the discrete nature of the sampling and the finite spacing between each of the wires on the instrument. It can be seen from Figure 4.1 that for a wave of constant amplitude, the instrument's ability to accurately reproduce the autocorrelation is reduced as the wave

number or frequency of the input wave increases. At waves numbers much above 3 rad/s the coherence of the autocorrelation function is degraded so much that it cannot be used to reproduce any semblance of the actual spectrum of the input signal.

The directional spectrum was generated by fitting a truncated two-dimensional sine and cosine wave series to the above autocorrelation plots after they have been windowed. Chapter 2 includes a detailed description of the fitting process, and a later section of this chapter contains a discussion of the windows used to smooth the data. In the cases where the autocorrelation was not significantly distorted, the directional spectrum plots should show a very distinct peak at the wave number corresponding to the input wave number. Each peak should be located on the x-axis, where the x-axis corresponds to an east-west direction and vertical upwards on the page corresponds to due north for these plots. As can be seen from Figure 4.2, the directional spectra of the lowest three wave numbers have these characteristics, but the 4.0 rad/s wave number input shows significant distortion. It is clear that the majority of this error can be ascribed to the distortion in the autocorrelation. At this high of a wave number, the wavelength is shorter than the wave gage array can resolve without excessive aliasing. It should be noted that the energy in the system has been preserved for each test, even in the 4 rad/cm spectral plot, but it simply is not distributed properly if the autocorrelation is significantly distorted.

4.1.2 2-D Fourier Transform Approximation

The second method used to verify the output of the directional spectrum program calculations was the traditional method of using a two-dimensional fast Fourier transform. This method is not used in the current wave gage signal processing because the autocorrelation is not known at evenly-spaced points. It can, however, be used for approximate comparisons. This was done by using the unevenly-spaced autocorrelation data to generate an interpolated approximation of the autocorrelation at evenly spaced points. This evenly-spaced approximation was then used as the input to a traditional two-dimensional Fourier transform. The directional spectrum generated in this manner was not altogether accurate because of the approximations made by interpolating the

autocorrelation, but it could be used to show that the actual method used by the wave gage processing code has described the simulated wave field. The directional spectrum on the left in Figure 4.3 was generated using the wave gage directional spectrum code. The spectrum on the right in Figure 4.3 was generated using an approximated autocorrelation and a two-dimensional Fourier transform. The two plots are shown on identical scales.

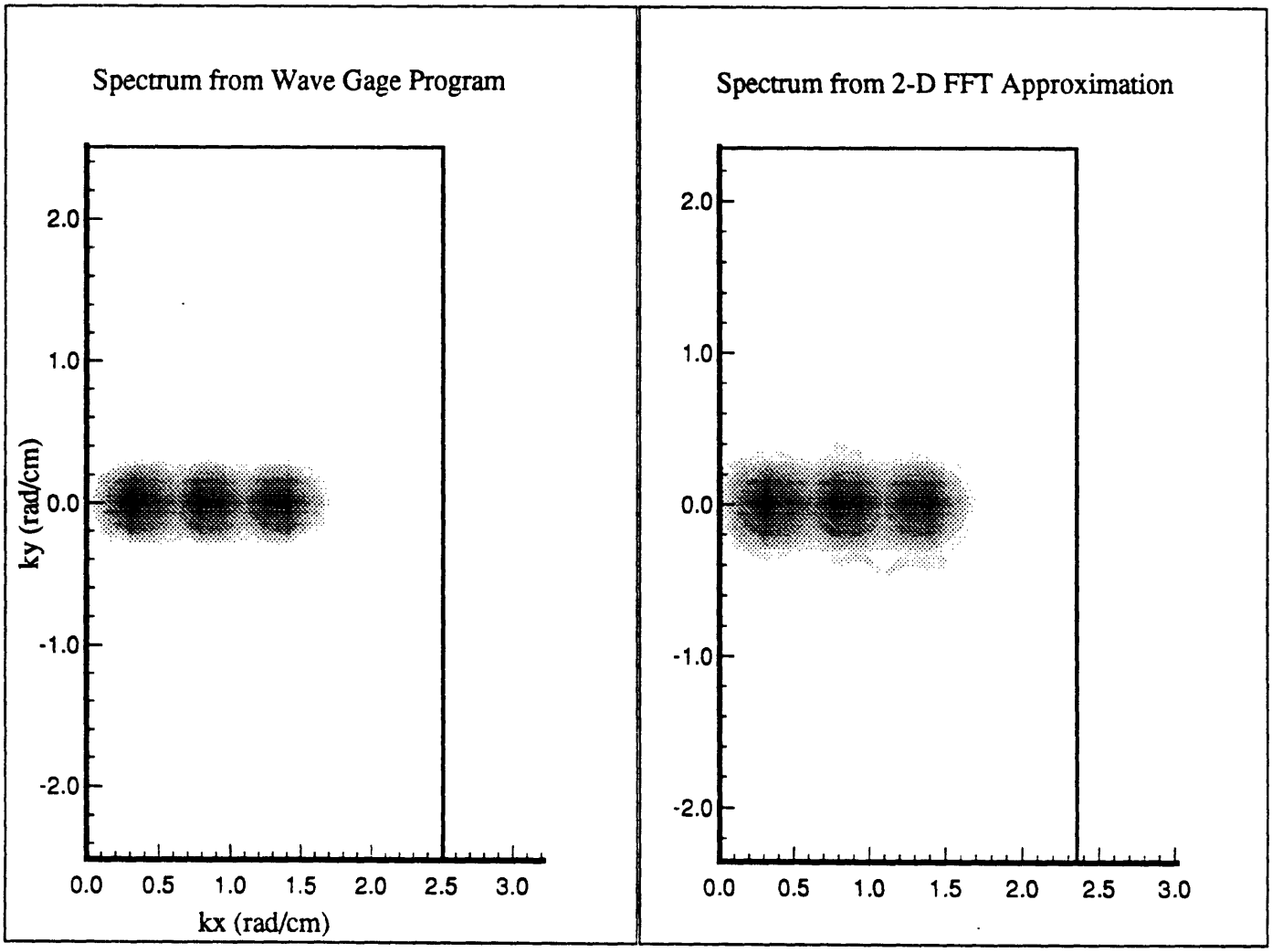

FIGURE 4.3: Comparison of Directional Spectra

It can be seen that both contain nearly identical information. It should also be noted that for more complex spectra, the two-dimensional approximation is less agreeable than Figure 4.3 would predict. The important conclusion to draw is that the method presented

in Chapter 2 for calculating the directional spectrum gives a very similar result to the well-established 2-D FFT method.

4.1.3 Variance Comparison

Another check to insure that the data processing code is working properly is to compare the output statistics of very easy to characterize input, such as sine waves. To accomplish this, five tests were run with sine wave inputs of various amplitudes and frequencies. For each test the average variance of the 64 individual wires' calibrated output signal was calculated using the equation for a discrete signal with N sample points

$$\sigma_{ave}^2 = \frac{1}{64}\sum_{1}^{64}[\frac{1}{N}\sum_{i=1}^{N}(x_i - x')^2] \tag{4.1}$$

where x' is the mean value of the discrete process $x_i(t)$. This value was calculated before the calibrated output was processed using the autocorrelation and directional spectrum programs. The variance can also be calculated by integrating the wave number spectrum of a signal or by calculating its autocorrelation at the origin such that

$$\sigma^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} S(k_x, k_y)dk_x dk_y = R(x=0, y=0) \tag{4.2}$$

or

$$\sigma^2 = \int_{0}^{2\pi 3.0}\int_{0}^{} |k|\, S(|k|,\alpha)\, dk\, d\alpha \tag{4.3}$$

where $S(k_x, k_y)$ is the two-dimensional wave number spectrum as a function of directional wave number, $S(k,\alpha)$ is the frequency spectrum as a function of the magintude of the wave number and angle, and $R(x,y)$ is the two-dimensional autocorrelation. The variance for each of the sine wave input tests can, thus, be calculated before the spectral processing and compared to the value after the processing. Table 4.1 shows the results of these tests.

TABLE 4.1: Variance of Signal Before and After Processing

	Input Signal		Variance (cm^2)			
Test #	k (rad/cm)	Amp. (cm)	Single wire ave.	*R(0,0)*	Integral of Spectrum	% Loss
1	0.4	0.94	0.446	0.446	0.440	1.3
2	1.0	0.94	0.446	0.446	0.445	0.2
3	2.0	0.94	0.446	0.446	0.445	0.2
4	1.0	0.66	0.219	0.219	0.218	0.5
5	1.0	0.38	0.0714	0.0714	0.0713	0.1

These data show that there was no energy lost in the calculation of the autocorrelation, but that there was a very small amount of energy lost during the calculation of the directional spectrum. The amount of energy lost remained below 1.5% regardless of amplitude or frequency of the input wave. Using a discrete Fourier transform of the autocorrelation, there should be no energy loss between the autocorrelation and the spectrum. However, the spectral processing code represents the aucorrelation by a *truncated* Fourier series. As is well known, any truncated series will not have the exact energy of an infinite series which represents the same input, it will be slightly less. There is also some energy loss due to the integration technique used to calculate the area under the discrete spectrum. The integration method used was a trapezoidal rule with a wave number spacing (radially) of 0.05 rad/cm and an angular spacing of 1.66°. Both of the losses mentioned could be reduced by increasing the number of Fourier coefficients used and increasing the integral resolution, but each of these sacrifices the speed at which the code can be processed.

4.1.4 Window Resolution

There is an important resolution limit imposed on the processed data which is due to the smoothing window used in the directional spectrum program. Recall from Chapter 2 that the spectrum is related to the autocorrelation by the equation

$$S(k_x,k_y)=\frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}R(x,y)e^{-i(k_x x+k_y y)}dx\,dy \qquad (2.11)$$

where $R(x,y)$ is the windowed autocorrelation. The windowed autocorrelation can be related to the actual autocorrelation by

$$R(x,y) = B(x,y)R_a(x,y) \tag{4.4}$$

where $R_a(x,y)$ is the actual continuous autocorrelation and $B(x,y)$ is the smoothing window in autocorrelation space. The smoothing window $B(x,y)$ can be further broken into two parts

$$B(x,y) = B'(x,y)W(x,y) \tag{4.5}$$

where $B'(x,y)$ is a set of N impulses at locations (x_n,y_n) in autocorrelation space and $W(x,y)$ is the window used to smooth the effects of the finite data set. The $B'(x,y)$ portion of the window is due to the discrete nature of the autocorrelation which is known only at the given (x_n,y_n) locations as described in Chapter 2.

The width of the total smoothing window $B(x,y)$ in wave number space limits the wave number resolution of the output spectrum because it, effectively, averages the energy underneath it. Figure 4.4 shows the normalized Fourier transform of the total smoothing window as a function of wave number where the actual three-dimensional window is this projection revolved around the vertical axis. Figure 4.4 shows that the directional spectrum output will be limited in wave number resolution by nearly 0.3 rad/cm.

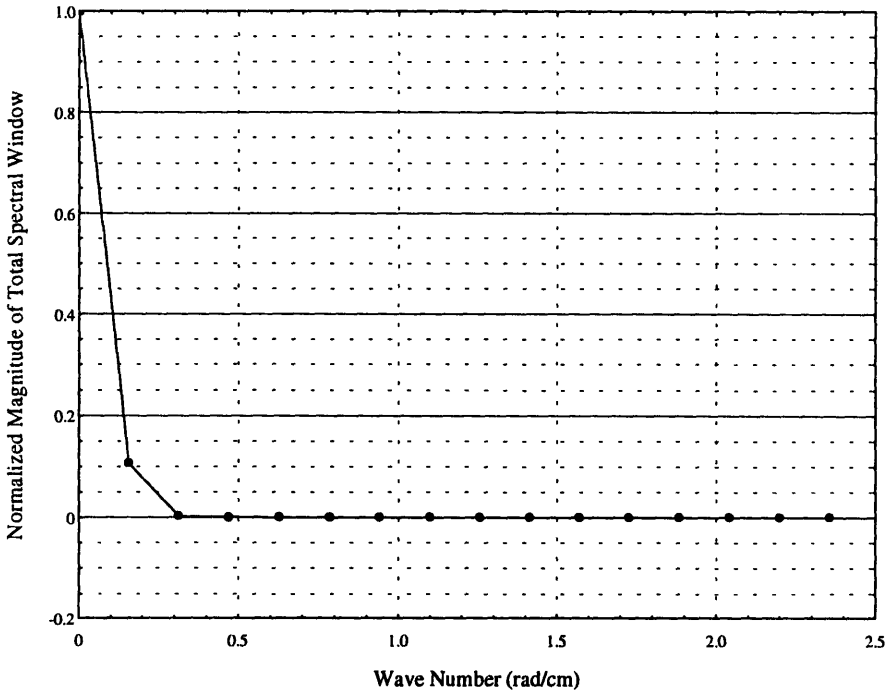

FIGURE 4.4: Total Smoothing Window in Wave Number Space

4.2 ELECTRONICS / GAGE

Testing the measurement device of the wave gage required a different approach than the testing of the signal processing code. The signal processing code either generated the correct directional spectrum given a particular input or it did not. The only potential errors were computational. On the other hand, the measurement device, the circular gage and electronics, contained many more potential sources of error. These could have included things as fundamental as incorrectly measuring the input to misinterpreting the output. To fully understand the operation of the measurement device, thus, one must characterize its limitations. Various aspects of the electronics were tested to understand the instrument's limitations. These included the effects of the high-pass filters on the signal, the slew associated with attempting to sample 64 channels instantaneously, and the signal to noise ratio at various frequencies. Furthermore, the total ability of the circular wave gage to measure waves correctly, also known as precision, was measured and is probably the most important metric obtained.

4.2.1 Filter Effects

The two high-pass filters in the electronics are some of the most important components of the circular wave gage because they set the dynamic range of the instrument. It is very important, thus, to verify that the filters are affecting the measured signal as expected. To test the high-pass filters three experiments were run using various combinations of the filters, the first using neither of the filters, the second using the first filter, and the third using both the first and the second filter (the first filter has a cutoff frequency of approximately 6 Hz and the second has a cutoff of 7.2 Hz). The three experiments used the same approximate wave input and the data was sampled at 200 Hz to insure adequate resolution of the voltage signals. Figure 4.5 shows the results of these three experiments where the signals were shifted in time to facilitate their comparison.

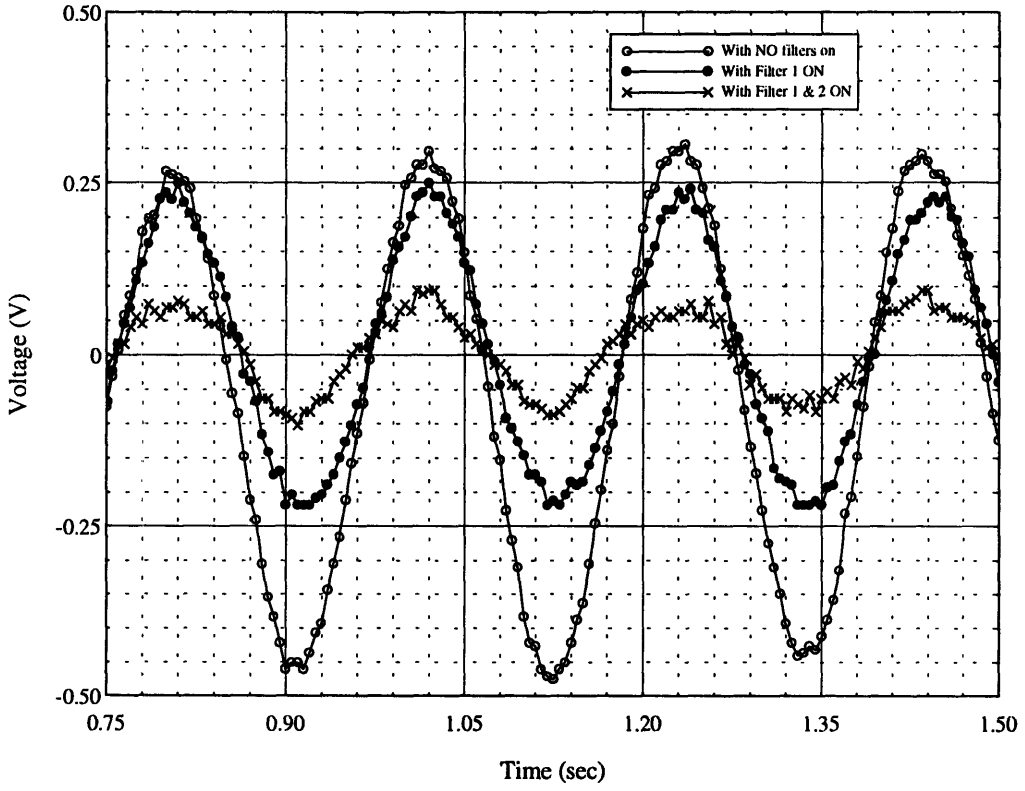

FIGURE 4.5: Raw Voltage Data Collected With and Without High-Pass Filters

The largest amplitude signal was the one collected without the two high-pass filters and it showed a low frequency modulation as one might expect without low frequency filtering. As each of the high-pass filters were turned on, the 5 Hz input signal was reduced in magnitude first by about 60% and then by about 44%, from an amplitude of approximately 0.35 V to 0.22 V and then to 0.1 V, respectively. These reductions in magnitude compare well with the magnitude changes predicted by the normalized high-pass filter responses shown in Figure 4.6.

The apparently larger noise seen on the data signal with both filters turned on in Figure 4.6 is caused by the scaling effect of the low pass filter on the input signal which, by definition, does *not* scale the high frequency portions of the signal. The effect of this noise on the system is discussed in a later section of this chapter.

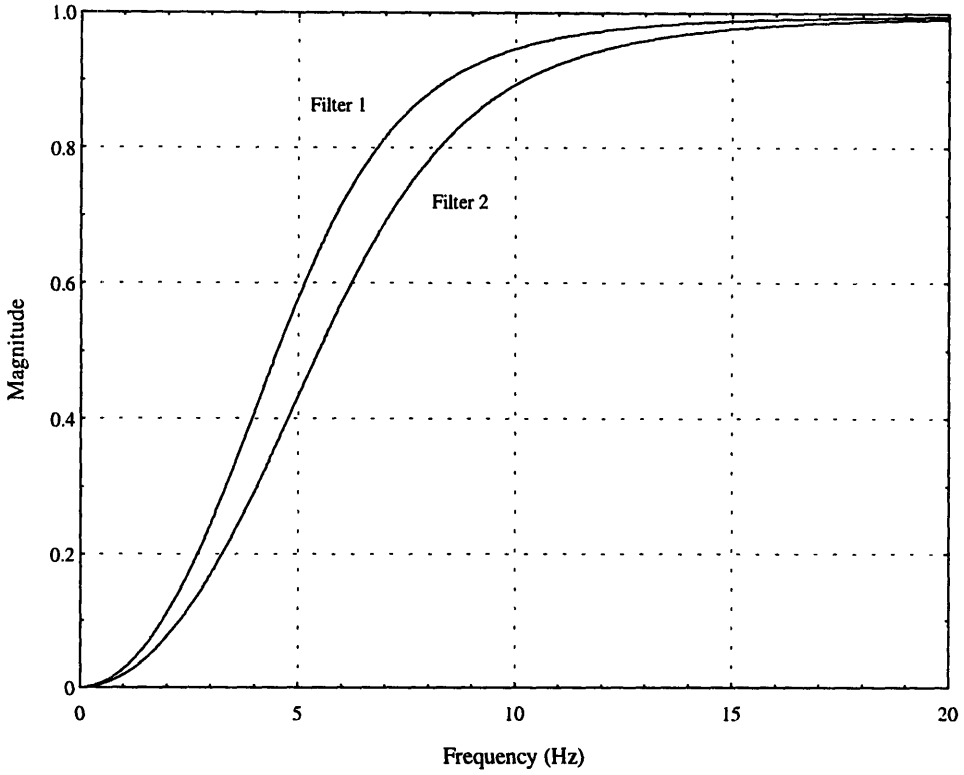

FIGURE 4.6: Wave Gage Normalized High-Pass Filter Magnitudes

4.2.2 Slew

An important characteristic of the data sampling system is the slew. This is particularly important because of the assumption made by the autocorrelation code that each of the input scans is instantaneous. To determine if this assumption is valid an experiment was run using a signal generator linked directly to the A/D input boards. Each of the channels received an identical sine wave input and were sampled with the identical code used to sample the standard experimental input at 32 Hz. Figure 4.7 shows the results from this test on the first and last channels sampled, channels 0 and 63, respectively.

The output from channel 63 appears to occur 'before' the output from channel 0 because by the time channel 63 is sampled its voltage has increased (or decreased) slightly due to the time that has elapsed since channel 0 was sampled. Based on Figure 4.7 the slew between the first and last channels is only approximately 0.005 second (1/200th of a

second). The phase difference between the channels in the autocorrelation caused by this slew is negligible, therefore the nearly instantaneous scan assumption is well-founded.

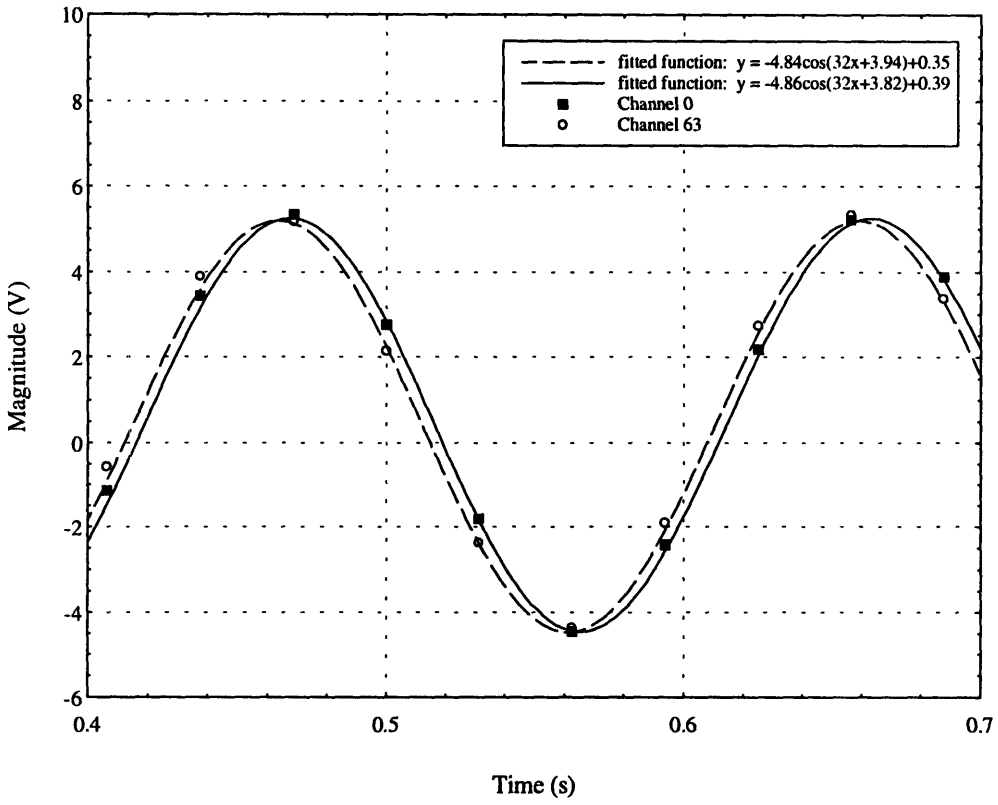

FIGURE 4.7: Slew Between First and Last Channels Sampled for Identical Sinusoidal Input

4.2.3 Signal to Noise Ratio

Noise in the sampled input signal from electronic or other sources can cause significant errors in the resulting directional spectrum. The total average voltage noise level measured on the gage is between 50 and 80 mV peak-to-peak. It has a somewhat broad low-level spectral content, but a significant portion of the noise is due the incomplete filtering of some of the 10 kHz carrier wave. Because the data is only sampled at 32 Hz, this high frequency energy gets folded back down into the lower frequency bins and corrupts the data. The amount of corruption depends on the strength of the signal. An

input signal which is only 60 mV peak-to-peak will be significantly affected by the noise, while a signal which is 2 V peak-to-peak will not be significantly affected. The magnitude of the input signal is a function of its amplitude, with high frequency waves having much lower average amplitudes than low frequency waves. As an example, an input sine wave of amplitude 0.03 mm has an approximate peak-to-peak voltage of 140 mV. The signal to noise ratio for this data, therefore, is only about 2.

In order to partially remedy this situation, the noise spectrum can be subtracted from the output spectrum. This can be accomplished by subtracting the noise spectrum from the frequency spectrum of the collected data just before it is calibrated, where the noise spectrum is generated from the noise time series collected during calibration. For a real system this method will not remove all of the noise, but it should remove most of it.

4.2.4 Precision

The circular wave gage measures four properties of the input wave signal: its amplitude, its wave number, its frequency, and its direction. The precision of the amplitude and frequency were determined by comparing the output of the circular wave gage to laser slope gage wave measurements. The testing setup and calibration procedures used for the laser slope gage measurements are described in Appendix H. The wave number is not a completely independent measure. For the tests described in this section, it can be related to the frequency through the dispersion relationship. For high frequency waves traveling on top of long waves, as would be measured in the ocean environment, however, there are a number of other factors including the convection of the short waves by the long waves, which affect this relationship. These conditions cannot be modeled well in the laboratory conditions. It is, therefore, not appropriate to speak of wave number resolution in this context. The discussion of wave number resolution relative to the signal processing was, however, discussed in the first section of this chapter. In the same light, the precision of the absolute directional information measured by the circular wave gage is a function only of the accuracy of the placement of the freestanding flux-gate compass. Because this is not physically associated with the proper functioning of the wave gage, only one test was

performed to insure that the directional information measured by the circular wave gage was near the expected input wave direction.

Five experiments were run to determine the precision of the instrument at various amplitudes and frequencies. The frequencies encompassed a major portion of range the instrument is expected to encounter in the field, but the amplitude range was limited to amplitudes below 5 mm because of the limitations of the laser slope gage to measure waves larger than this. Table 4.2 shows the wave amplitude and frequency measurements made by the laser slope gage and the circular wave gage. Appendix I contains a sample time series wave height from a single wire on the circular wave gage as well as the laser slope gage data from these tests.

The amplitude and frequency information given in Table 4.2 is an average over all the wires for each test. The purpose for this is to show that each wire individually can measure the waves to within specific limits. It was shown in section 4.1.1 that the autocorrelation and directional spectrum code introduce some error into the final spectral results. This section, however, is meant to characterize only the ability of the gage itself to measure waves, so the amplitude and frequency information were calculated before the individual wire data was combined to generate the directional spectrum.

TABLE 4.2: Frequency and Amplitude Precision Data

	Laser Slope Gage Data		Circular Wave Gage Data		*Differences*	
Test #	Freq. (Hz)	Ampl. (cm)	Freq. (Hz)	Ampl. (cm)	*Freq. (Hz)*	*Ampl. (cm)*
1	**3.125**	**0.264**	**3.13**	**0.258**	***0.005***	***-0.006***
2	**3.883**	**0.206**	**3.90**	**0.189**	***0.017***	***-0.017***
3	**4.391**	**0.155**	**4.40**	**0.129**	***0.009***	***-0.026***
4	**4.961**	**0.113**	**4.97**	**0.082**	***0.009***	***-0.031***
5	**5.617**	**0.029**	**5.41**	**0.028**	***-0.207***	***-0.001***

There was generally good agreement between the laser slope gage measurements and the circular wave gage measurements. Table 4.2 shows that the circular wave gage amplitude measurement was always within 0.35 mm of the laser slope gage measurement. Similarly,

the frequencies measured by the circular wave gage were always within 0.2 Hz of the laser slope gage measurements. The frequency difference for test 5 is larger than the rest because the signal to noise ratio at that frequency for waves at such low amplitudes is significant (see previous section on Signal to Noise Ratio). Locating the precise peak frequency for this data set, thus, becomes more difficult and prone to error. Likewise, the lower than expected amplitude error shown in test 5 is most likely a result of errors due to the signal to noise ratio. Based on these results, it can be concluded that each wire on the circular wave gage can measure fairly accurately both the actual wave height and the frequency of input wave signals, however the instrument error increases as the amplitude of the input waves decreases.

A directional measurement was also calculated using the data from test 3. The azimuthal direction in which the input waves traveled was measured using a hand-held compass and determined to be approximately SSE (or 160°N) . This was compared with direction measured from the directional spectrum plot shown in Figure 4.8 which was also approximately SSE. It is difficult to determine an absolute direction measurement from the plot, but the maximum peak located by using the raw spectral data occurred at 169° which is well within a 15° accuracy range. The true directional precision of the circular wave gage is probably much better because significant errors were possible in the placement of the free-standing flux gate compass and in the measurement made using the hand-held compass. These errors would be reduced if the flux gate compass were fixed to the circular wave gage.

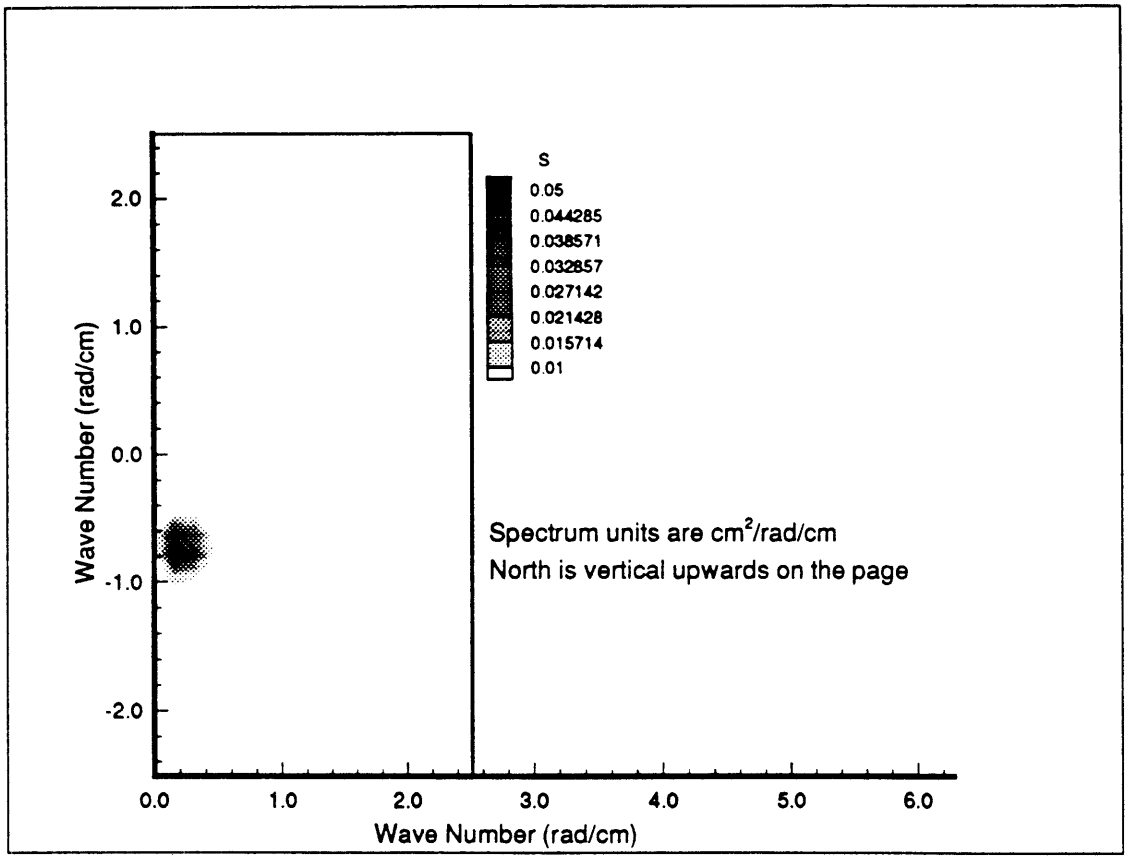

FIGURE 4.8: Directional Spectrum of Sine Wave at 4.4 Hz ($|k| \cong 0.8$ rad/cm)

5. EXPERIMENTAL DATA COLLECTION, RESULTS, AND DISCUSSION

Once the circular wave gage and its signal processing programs were fully tested in the lab environment, the next step was to test the instrument in its intended environment: the ocean. This effort was the prime focus of the current development. Therefore a number of experiments were planned to accomplish this task. This chapter describes the test setup and platform, presents the results, and discusses the meaning and validity of the results from two of the experiments performed.

5.1 EXPERIMENTAL SETUP

The two experiments described in this chapter were conducted in Casco Bay, Maine on August 17 and September 26 of 1995. They were performed off of a catamaran platform operated by Greene Marine, a local marine company. The circular wave gage was rigidly suspended below two planks laid across the 3.5 meter span between the bows of the two hulls. The catamaran was propelled by a 'push-boat' with an outboard motor which kept a nearly constant upwind speed of 1 to 2 knots to allow for steerageway during the data collection. The catamaran was directed into the wind during data collection to prevent significant reflection from the hull wake from disrupting the wave field at the gage.

During the experiments the electronics and computer were powered by a Honda generator. Several 128 second data sets were collected over the the time span of a few hours on each day of testing. An anemometer and a hand-held compass were used to collect average wind speed and direction during each data set. Calibration data sets were collected before and after the instrument was suspended between the catamaran hulls.

5.2 RESULTS

The plots shown in Figure 5.1 are the directional spectra calculated from the measured data and viewed in TECPLOT for four of the data sets collected on August 17. The wind was blowing 4.5 - 5.5 knots from the northeast during the four tests shown.

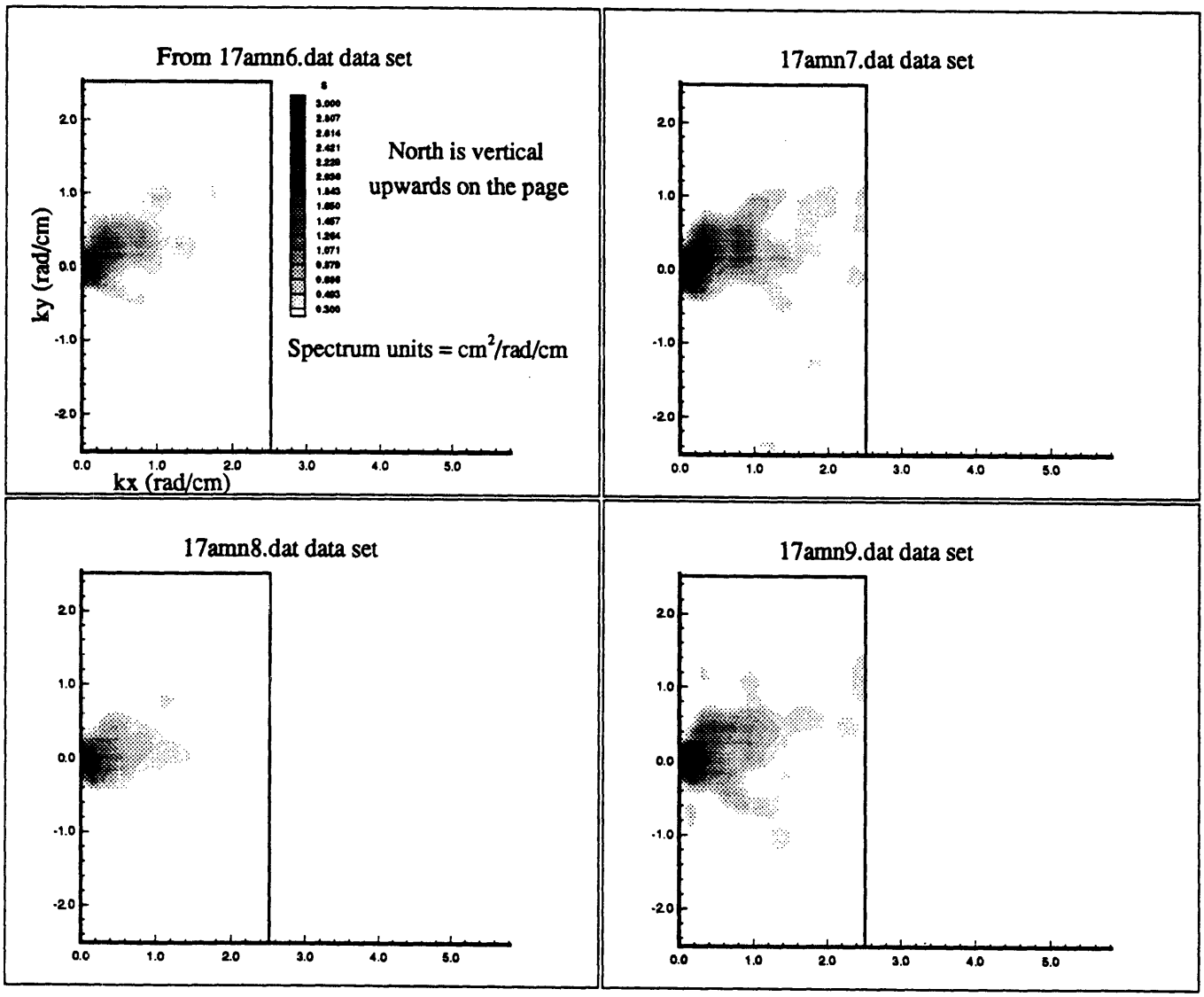

FIGURE 5.1: Directional Spectra from August 17 Experiment

Only one side of each spectrum is shown because the other half of the spectrum is symmetric through the origin, so it does not contain any new information. Furthermore,

the $S(k_x,k_y)$ spectra shown in Figure 5.1 are one-sided spectra. In other words, all of the energy in the wave field is contained in the single side of the spectra shown.

Using the directional spectrum information, a wave number spectrum was generated for each of the data sets using the following relationship

$$S(|k|) = \int_0^{\pi} |k|\, S(|k|,\alpha) d\alpha \tag{5.1}$$

where $S(k,\alpha)$ is the spectrum as a function of wave number and angle, which can easily be generated from the $S(k_x,k_y)$ spectrum.. The integral is calculated over only half of the full 2π range because the spectra are one-sided as described above. Figure 5.2 shows the wave number spectra as well as the Bjerkaas and Riedel theoretical prediction for a wind friction velocity near what would be expected from the wind conditions during the experiment. The wave number spectrum shown in Figure 5.2 has been smoothed with an averaging window which is approximately 0.1 rad/cm wide.

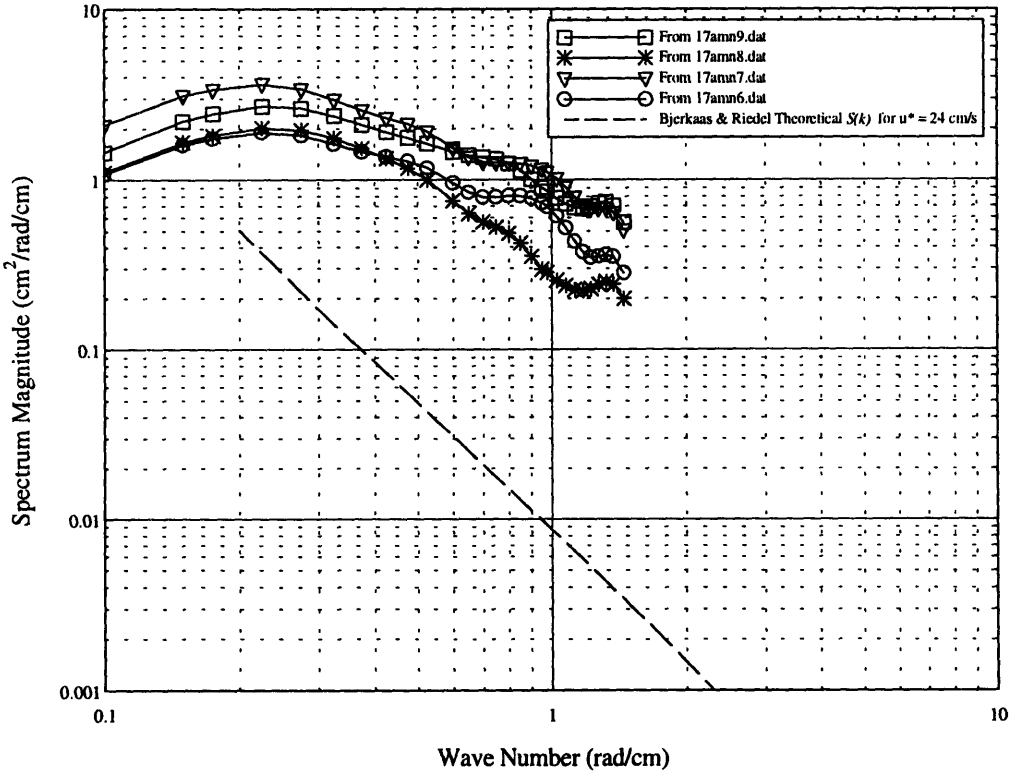

FIGURE 5.2: Wave Number Spectra from August 17 Experiment

The single wire frequency spectrum was also calculated. It used one channel of the calibrated wave height versus time data, $x_i(t)$, and was generated using the equation

$$S(\omega) = \left|\Im\{x_i(t)\}\right|^2 \tag{5.2}$$

where $\Im\{x(t)\}$ is the smoothed Fourier transform of the wave height versus time data. The smoothing was done in EasyPlot using a sliding data window which was 11 data points wide. The frequency spectra using channel 0 from each of the four data sets are shown in Figure 5.3. Frequencies below 14 rad/s are not shown because, as mentioned in Chapters 3 and 4, they are electronically and digitally filtered out.

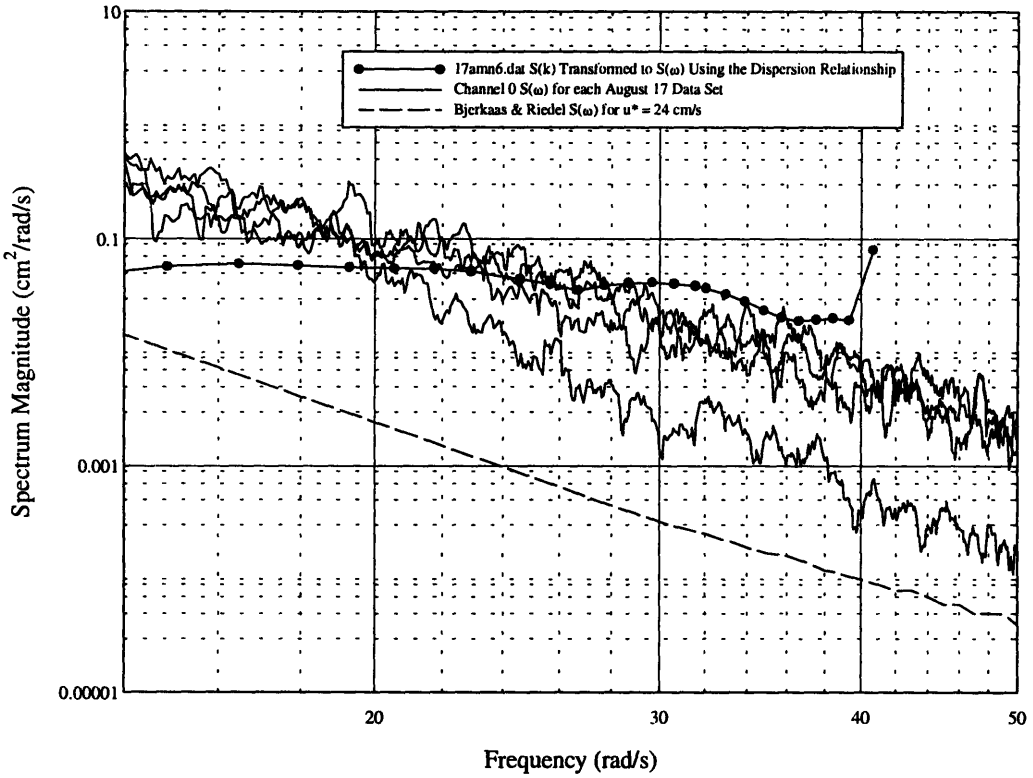

FIGURE 5.3: Frequency Spectra from the August 17 Experiment for Channel 0 of Each Data Set

Figure 5.4 shows the power spectral density along various 'slices' of the directional spectrum for the 17amn7.dat data set relative to the primary wind direction. This plot is

fairly representative of the power spectral density relationships as a function of angle off the wind for all of the August 17th data.

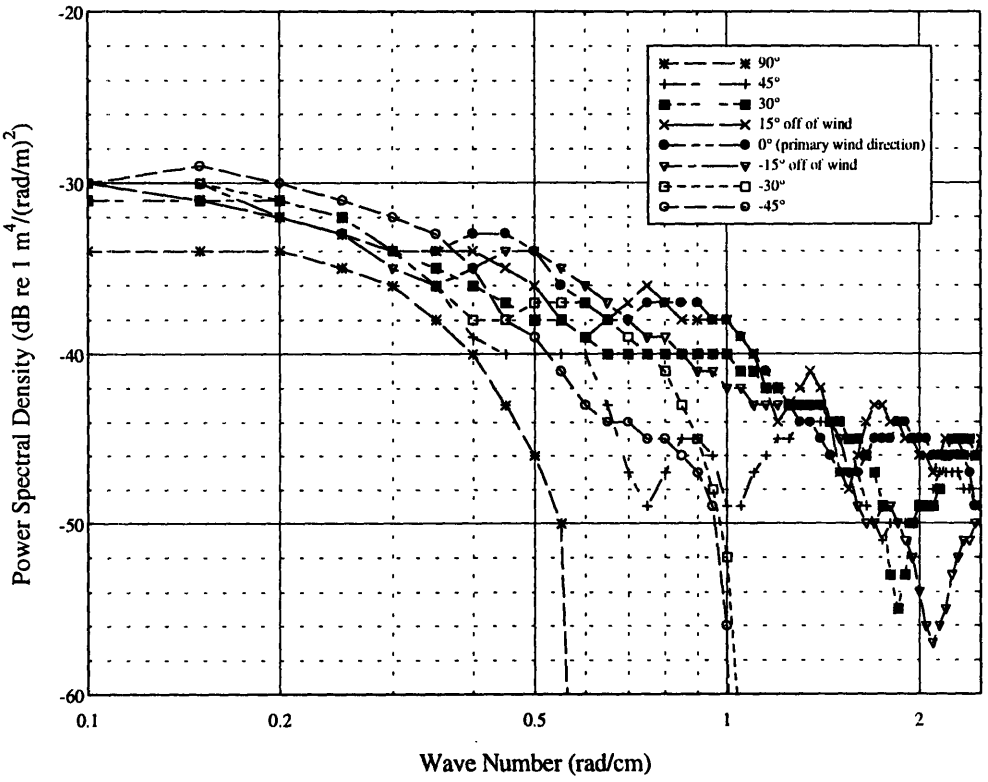

FIGURE 5.4: Power Spectral Density for 17amn7.dat Data Set

The wind for the second experiment on September 26 was blowing from the southeast at between 6 and 7 knots. The directional spectra plots for several of the data sets are shown in Figure 5.5, the wave number spectra are shown in Figure 5.6, and the frequency spectra from channel 0 of each data set are shown in Figure 5.7. Each plot is generated in the same fashion as described for the August 17 experimental results.

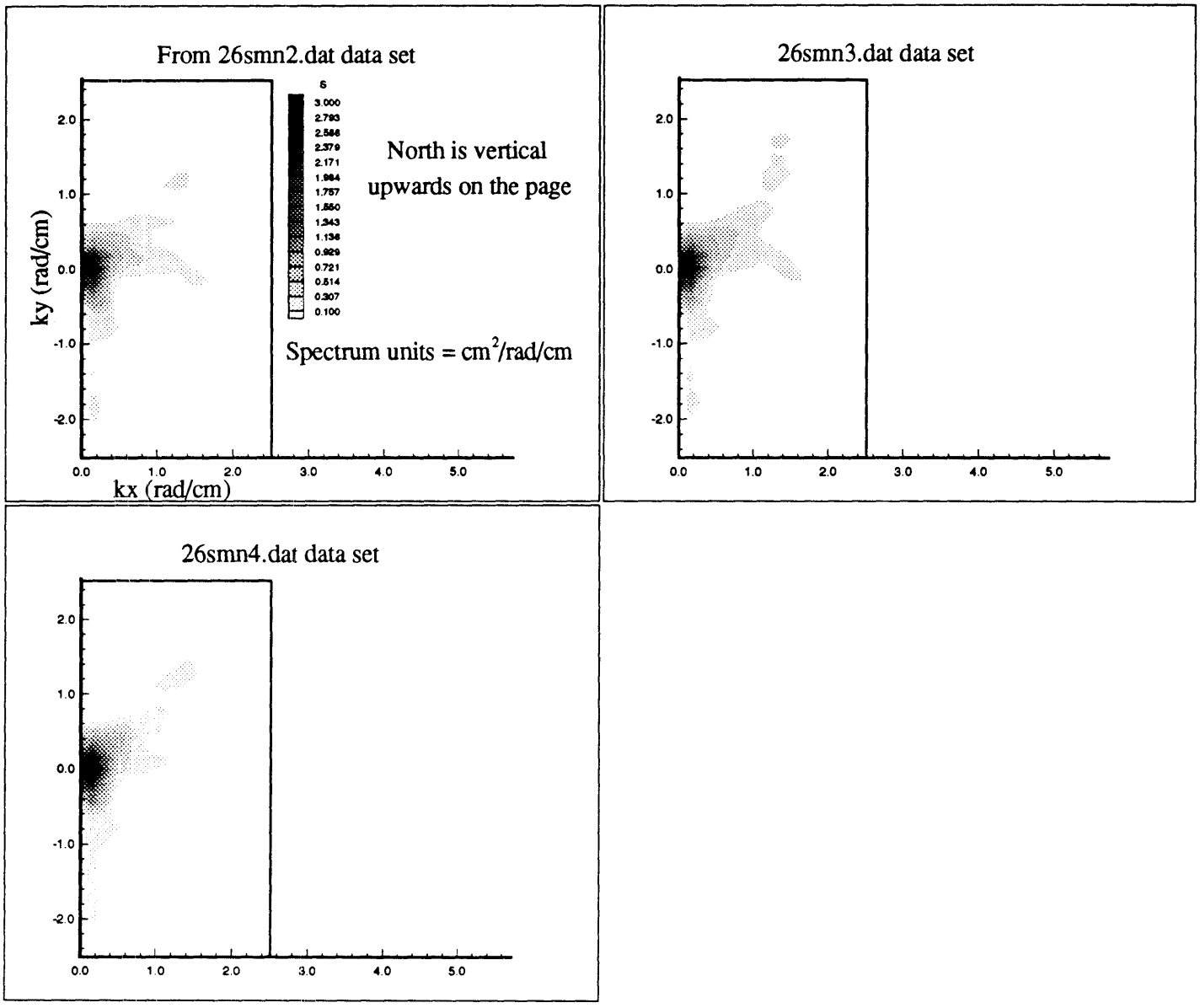

FIGURE 5.5: Directional Spectra from September 26 Experiment

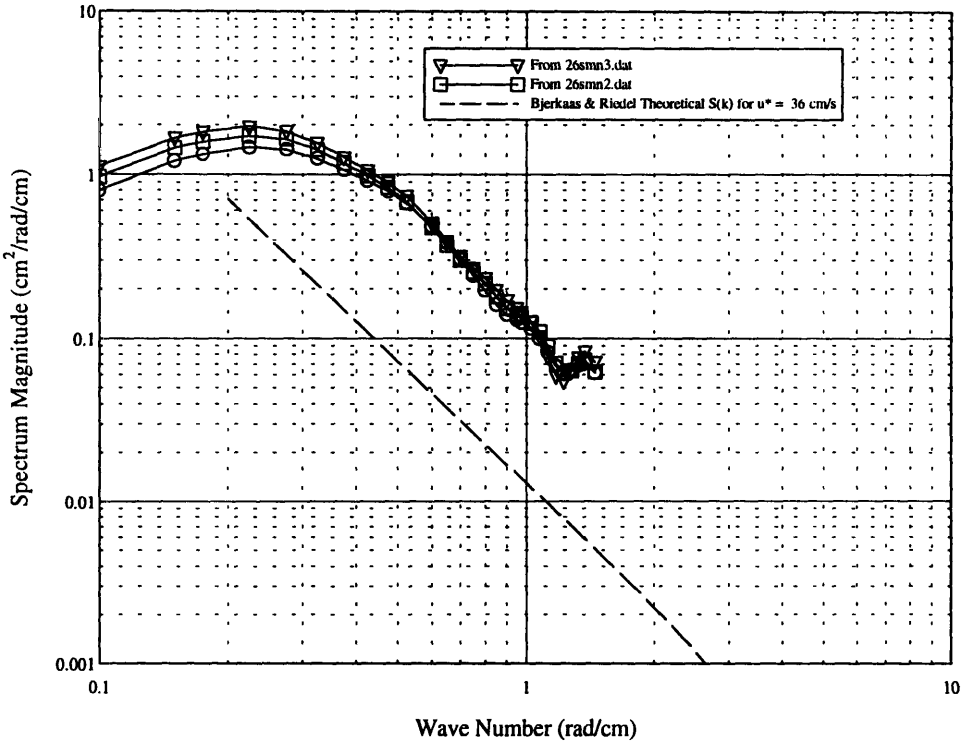

FIGURE 5.6: Wave Number Spectra from September 26 Experiment

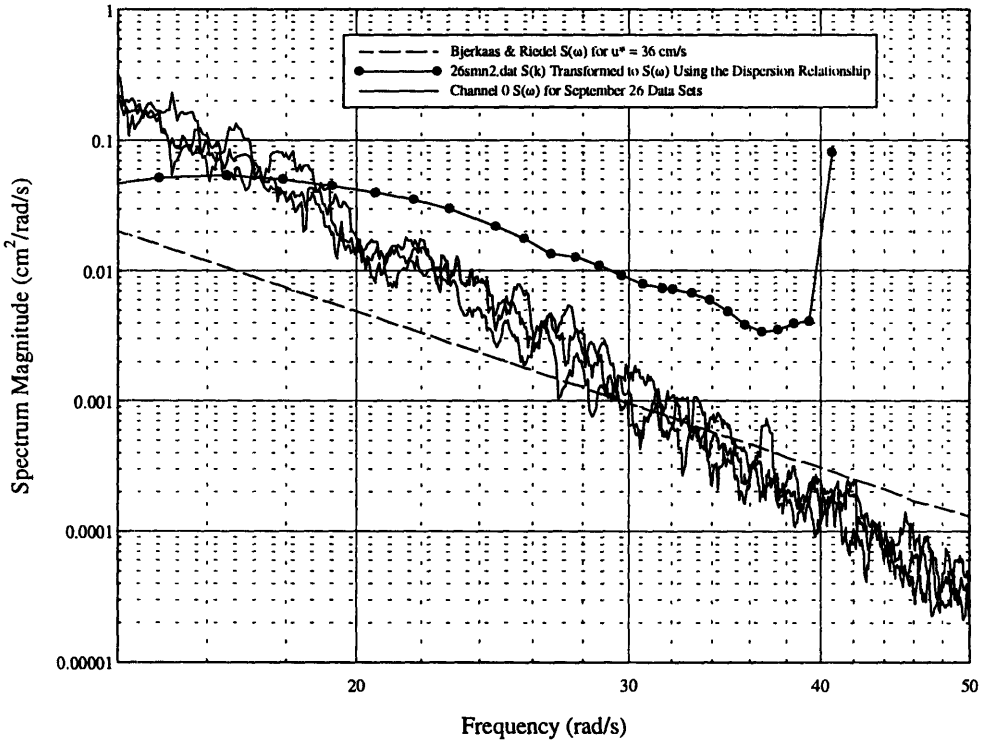

FIGURE 5.7: Frequency Spectra from the September 26 Experiment for Channel 0 of Each Data Set

5.3 DISCUSSION

The August 17 directional spectra show an energy spread which is in approximately the same direction as the wind, as expected. The September 26 directional spectra show a bi-directional distribution centered around the wind direction. Photographs of the sea surface on that day, however, show that the short waves were very bi-directional in nature. These results support the conclusion that the directional wave gage does indeed measure the directions of the short sea waves correctly.

The single wire frequency spectra differ somewhat from the Bjerkaas and Riedel model in amplitude, but not very much in slope. This could be explained either by the fact that neither of the sea states during these experiments was fully developed, which is what the Bjerkaas and Riedel model represents, or by the fact that the experiments were performed in a coastal, relatively shallow region.

The wave number spectra for each of the experiments present further curious results because their relationship to the Bjerkaas and Riedel theoretical prediction is different than it is in the frequency spectra plots. As can be seen from the frequency spectra plots, if a measured wave number spectrum is transformed into a frequency spectrum using only the dispersion relationship, it does not resemble the measured single wire frequency spectrum. This difference is most likely due to the fact that the high frequency waves are affected by two factors which the dispersion relation does not take into account: a short wave traveling on top of a long wave will have its frequency shifted as a consequence of the orbital velocities of the long wave, and the high frequency components of the underlying Stokes waves travel at the velocity of the low frequency component of the Stokes wave, not at the velocity predicted by the dispersion relationship. These two factors make the energy transformation from one spectrum to the other not as straightforward as might be expected.

These results lead to the conclusion that the Bjerkaas and Riedel frequency spectrum which is relatively well documented and experimentally verified cannot be used solely as a basis for the wave number spectrum using only the dispersion relationship transformation.

In addition to the Bjerkaas and Riedel model, there is also some data which was collected by TRW which is shown in Figure 5.8. It shows the power spectral density as a function of wave number and direction from the primary wind direction. This data, however, shows only a 5dB difference between waves traveling in the along-wind direction and those traveling perpendicular to the wind direction. As can be seen in Figure 5.4, the data collected on August 17th shows a much more significant directionality than the TRW result. There could be a number of reasons for this related to the sea state and location of the TRW experiments versus the Casco Bay experiments

There are two other apparent differences between the TRW results and those collected with the circular wave gage. The first is the difference in magnitude of the power spectral density plots. This is due to the fact that the reference magnitude that TRW used to normalize their spectral density plot is not known. The difference in magnitude is, thus, understandable. The other difference is the average slope of the data. The TRW results show a spectral slope of nearly k^{-4}, or 40 dB per decade, which is much greater than Bjerkaas and Riedel theory theory which predicts a slope of only about k-2.5, or 25 dB per decade. One the other hand, the circular wave gage results show a slope of only about k^{-1}, or 10 dB per decade. The circular wave gage results shown in Figure 5.4 are, however, consistent with the data shown in Figure 5.2 which also show an average slope of only about k-1. The slope difference in Figure 5.2 was discussed previously.

Without further experimental studies, it is difficult to draw any significant conclusions from the comparison between these two data sets except to say that it is believed that the high frequency waves the circular wave gage measures are more directional than has been effectively shown before.

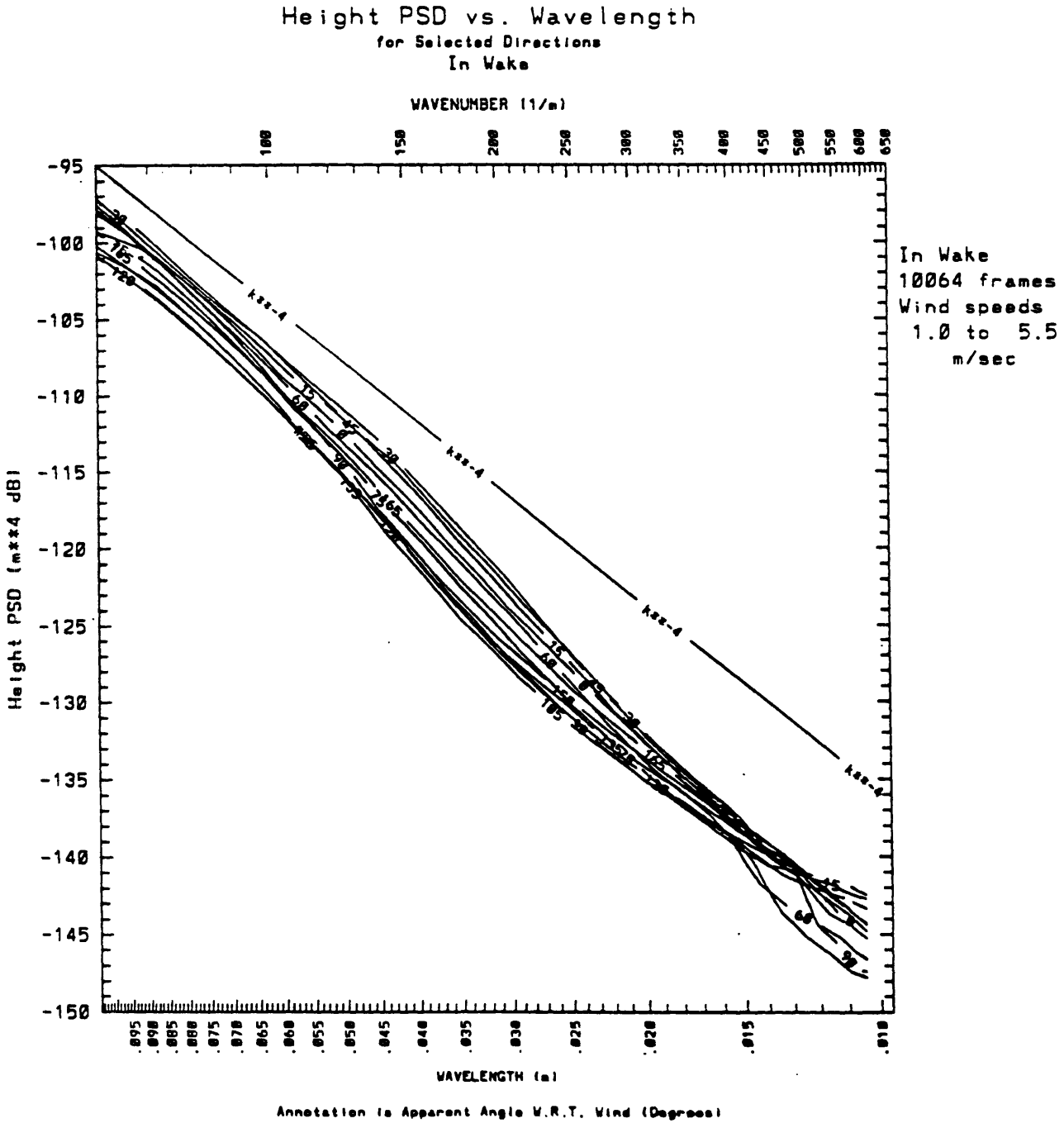

FIGURE 5.8: TRW Power Spectral Density Directional Dependence Plot (Yuen, 1991)

6. CONCLUSIONS

The circular wave gage was designed to measure the directional spectrum of high frequency waves on the ocean surface. Electronic and signal processing modifications were made to the previously designed instrument to accomplish this task. Furthermore, an easy-to-use calibration technique was added to allow for accurate and repeatable calibration of the instrument in relatively constant temperature environments. Based on the performance tests completed in the Marine Instrumentation and Computation Laboratory wave tank and the Casco Bay field experiments, it is concluded that the instrument can measure high frequency directional spectra. It is suggested, however, that the instrument be tested in conjunction with another directional wave measuring device at least once to insure that the spectral amplitude measurements are being properly calculated for at-sea conditions.

6.1 INSTRUMENT LIMITATIONS

In its current configuration the circular wave gage does have some limitations: it is subject to calibration drift over time due to temperature changes of the electronic components, it cannot be used in very large sea states, unless it is on a buoy that follows the long waves, and it cannot yet be deployed for autonomous data collection. Perhaps an even more relevant limitation at this stage of the instrument's development is that it has not been tested simultaneously with another directional wave measuring instrument to verify its experimental results. This is particularly important given the differences found between the experimental data and both the TRW results and the theoretical Bjerkaas and Riedel model.

6.2 RECOMMENDATIONS

There are a number of suggestions which can be offered both to remedy the current instrument limitations and to improve on the current design.

The calibration drift could be reduced or eliminated by housing the electronics in a temperature controlled environment. This may not be an issue once the instrument has been modified for autonomous field deployment if the electronics package is submerged.

The dynamic range of the instrument could be improved by modifying the analog filtering system. The new system would need to have a steeper transfer function slope between 2 and 10 Hz to further reduce the energy in the low frequency waves and magnify the energy in the low frequency waves. Bjerkaas and Riedel predict a spectral frequency dependence of approximately ω^{-4}, therefore, developing a filter which carries the existing ω^{4} dependence to higher frequencies could flatten the instrument spectrum significantly.

Autonomous deployment was not intended to be a part of the current development. It is, however, the next logical step in the evolution of the circular wave gage. This step would require a complete repackaging of the electronics, a new connector system to get the signal from the wave gage back to the electronics package, development of a buoy on which to mount the instrument, and additions to the current data collection software to allow for programmable data collection triggering. The electronics and signal processing may also need to be streamlined to meet any power limitations imposed by operating from a limited power source.

Developing an instrument which does not have many contemporary counterparts is difficult because reliable ground-truthing is difficult to obtain. Furthermore, there is a limited or nonexistent archive of collected data against which the instrument's results can be compared. For these reasons, it is imperative that the circular wave gage be further tested with another directional wave gage. Plans to accomplish this in the Fall of 1996 are

being developed in coordination with Dr. Erik Bock at the Woods Hole Oceanographic Institute who has developed an optical directional wave gage.

There are a number of questions that have not been adequately answered by this research that deserve further investigation. One of these is the effect of the center post on the measured directional spectra. The center post on the gage was removed for all of the verification studies which are presented in Chapter 4 because at the time it was believed that the center post could have been significantly affecting the spectra. It was also mentioned in Chapter 4 that to reduce the signal to noise ratio the noise spectrum should be subtracted from the voltage signal spectrum. This is not currently incorporated into the signal processing, but it should be included in the next iteration of its development. It is no longer believed that the effect is significant, but a detailed analysis should be pursued. Another issue that needs further attention is the further reduction of noise in the system. It was determined that the channels are still sensitive to noise as described in Chapter 4. Some of this can be digitally removed, but attempting to reduce the source rather than remove the result is a more attractive and generally more effective method of dealing with the problem. It is recommended, therefore, that more effort be given to this issue.

After further instrument testing and development it is predicted that this instrument will be very useful for measuring not only high frequency surface wave spectra, but also for experimentally determining the frequency and wave number relationship for which very little experimental data currently exists.

6.3 SUMMARY OF GAGE CHARACTERISTICS

Size: Circular wire gage total height ≈ 1.4 m

Circular wire gage total diameter ≈ 25 cm

Number of channels: 64

Frequency range: ≈ 2 to 10 Hz waves

Resolution in wave number: ≈ 0.3 rad/cm

Resolution in direction: depends on accuracy of placing of compass (compass res. ≈1.5°)

Amplitude resolution: ≈ 0.35mm

Rate of data collection: 32 Hz (i.e. one scan of all 64 channels occurs 32 times a second)

Total time of data collection for one data set: 128 seconds

Rate of calibration data collection: 75.02 Hz

BIBLIOGRAPHY

Bjerkaas, A. W., Riedel, F. W., "Proposed Model for the Elevation Spectrum of a Wind-Roughened Sea Surface", *Tech. Memo.* , The Johns Hopkins University, APL TG 1328, Laurel, Maryland, December 1979.

Blackman, R. B. and Tukey, J. W., *The Measurement of Power Spectra from the Point of View of Communications Engineering*, Dover Publications, Inc., New York, 1958.

Bock, Erik J. and Hara, Tetsu, "Optical Measurements of Capillary-Gravity Wave Spectra Using a Scanning Laser Slope Gauge," *Journal of Atmospheric and Oceanic Technology*, Vol. 12, No. 2, April 1995, pp. 395-403.

Chen, David J., Designing Wave-measuring Instruments, SM thesis, Massachusetts Institute of Technology, 1994.

Comstock, John P. (ed.), *Principles of Naval Architecture*, The Society of Naval Architects and Marine Engineers, New York, 1967. (p607-627)

Hara, Tetsu, Bock, Erik J., and Lyzenga, David, "In situ Measurements of Capillary-Gravity Wave Spectra Using a Scanning Laser Slope Gage and Microwave Radars," *Journal of Geophysical Research*, Vol. 99, No. C6, June 1994, pp. 12,593-12,602.

Newland, D. E., *An Introduction to Random Vibrations, Spectral and Wavelet Analysis*, 3rd ed., Longman Scientific and Technical, New York, 1993.

Milgram, Jerome H., "Theory of Radar Backscatter from Short Waves Generated by Ships, with Application to Radar (SAR) Imagery," *Journal of Ship Research*, Vol. 32, No. 1, March 1988, pp. 54-69.

Press, William H., Teukolsky, Saul A., Vetterling, William T., Plannery, Brian P., *Numerical Recipes in FORTRAN*, 2nd Ed., Cambridge University Press, New York, 1992, pp. 519-524, 678-683.

Pierson, W. J. and Moskowitz, L., "A Proposed Spectral Form for Fully Developed Wind Seas Based on the Similarity Theory of S. A. Kitaigorodskii," *Journal of Geophysical Research*, Vol. 69, 1964.

Yuen, H., "Archive of Wave Slope Gauge Data," in *ONR Ship Wake Detection Program Review Meeting*, September 26, 1991.

APPENDIX A: cknic.c Program

```
#include <stdio.h>
#include <conio.h>
#include <graph.h>

#define BASE_DAS48 0x380
#define BASE_DAS16 0x300
#define DELAY 100

float getval_das16(int channel, int gain);
float getval_das48(int channel, int gain);
int wait_loop(void);
void setup_counter0(void);
extern void mscl_dasg(int *, int *, int *);  /* DECLARE CALL structure */

char *buffer_in_com1;
char *buffer_out_com1;      /* place to get next character from */
char *buffer_in_com2;
char *buffer_out_com2;      /* place to get next character from */
char word[50],w1[50];
char word2[25],w2[25];

main()
{
    int    channel, gain, hi_c, low_c,head;
    int    i, gain_das16, gain_das48;
    int    num_samp, sample_count;
    long   dummy;
    char   type, foo[80], out_file[12];
    float  Data[64];
    FILE   *foof;

 if (setup_serial_port() == 0)    /* OPEN ports,  COM1 & COM2 */
    _outtext("\nSetting Up COM Ports: ... ok\n");
 else {
    printf("\nERROR Setting Up COM1: or COM2: ...");
    cleanup();
    }

    printf("\nAny key to continue");
      type = getche();

    gain_das48 = 8;
    gain_das16 = 1;
    _clearscreen(_GCLEARSCREEN);
    _displaycursor(_GCURSOROFF);
    setup_das16();
    setup_counter0();
    sample_count = 0;
```

```
_settextcolor(12);
_outtext("\n\n\n          Data Display Mode -- Hit Any Key To Exit...");
_settextcolor(14);

  while (!kbhit()) {
  _settextposition(3,5);
  read_com_port();

  for (i=0;  i<16; i++)  Data[i]  = getval_das16(i, gain_das16);
  for (i=16; i<64; i++)  Data[i]  = getval_das48(i-16, gain_das48);

  _settextposition(6,0);
    for (i=0; i < 64; i=i+4) {
        if (i == 16) _outtext("\n");
        sprintf(foo,
             "Ch[%02d]: %6.2fV    Ch[%02d]: %6.2fV    Ch[%02d]: %6.2fV Ch[%02d]: %6.2fV   \n",
       i, Data[i], i+1, Data[i+1], i+2, Data[i+2], i+3, Data[i+3]);
        _outtext(foo);

    }
for (dummy=0; dummy<1000000; dummy++);
}
getch();

_settextcolor(11);
_outtext("\n\nEnter Output Filename: ");
scanf("%s", out_file);

if((foof = fopen(out_file, "wb"))==NULL){
    printf("\n\nerror opening file out_file");
    exit(0);
    }

low_c = 0;
hi_c = 63;

_outtext("\n\nEnter # of Samples @ 32 Hz: ");
scanf("%d", &num_samp);

while (sample_count < num_samp) {
  /*** go into wait mode ***/
  wait_loop();
  /**** get the  compas heading ***/
  buf_getword_com1();
    head = 0;
    for(i=1;i<44;i++){
     if((word[i] =='$') && (head == 0)){
     sprintf(w2, "%c%c%c%c%c",
     word[i+1],word[i+2],word[i+3],word[i+4],word[i+5]);
     fprintf(foof, "%s ",w2);
     head++;
     }
    }
  /**** get the  data ***/
```

```
      for (i=0;  i<16; i++)  Data[i]  = getval_das16(i, gain_das16);
      for (i=16; i<64; i++)  Data[i]  = getval_das48(i-16, gain_das48);
      /**** print the  data ***/
      for (i=low_c; i<=hi_c; i++) fprintf(foof, "%6.3f ", Data[i]);
      fprintf(foof, "\n");
      if ((sample_count % 100) == 0) printf("\rCount = %d of %d", sample_count,num_samp);
      ++sample_count;
      }

     printf("\rCount = %d of %d", sample_count,num_samp);
   _outtext("\n\n");
   cleanup();
}

cleanup(){
   release_serial_ports();
   _settextcolor(0);
   _setcolor(0);
   _setvideomode(_DEFAULTMODE);
   fcloseall();
   exit(0);
   }

float getval_das48(int channel, int gain){
   int hi, low;
   unsigned long x;
   float val;

   /* Gain    Range    */
   /*  1     0-10v    */
   /*  3     0-5v     */
   /*  5     0-2.5v   */
   /*  7     0-1.25v  */
   /*  8     +/-10v   */
   /*  0     +/-5v    */
   /*  2     +/-2.5v  */
   /*  4     +/-1.25v */
   /*  6     +/-0.625v */

   outp(BASE_DAS48+3,gain);       /* Set Gain */
   outp(BASE_DAS48+2,channel);    /* Set channel */
   for (x=1; x<DELAY; x++);
   outp(BASE_DAS48+1,0);          /* Start 12 Bit Conversion */
   while (inp(BASE_DAS48+2) > 127); /* Wait for DONE bit to be cleared */
   low = inp(BASE_DAS48);         /* Read Low Nibble */
   hi  = inp(BASE_DAS48+1);       /* Read MSByte */
   val = hi*16.0 + low/16.0;  /* Add bytes and right-shift low */
   val = val - 2048.0;
   switch (gain) {
      case 0:
      case 1:
         val = val / 409.6;
         break;
      case 2:
```

```
        case 3:
            val = val / 819.2;
            break;
        case 4:
        case 5:
            val = val / 1638.4;
            break;
        case 6:
        case 7:
            val = val / 3276.8;
            break;
        case 8:
            val = val / 204.8;
            break;
    }
    return val;
}

setup_das16(){
    /* First, test for board's existence by writing and reading a reg */
    int    error, value;
    int    Mode, Params[16], Flag;
    error = 0;
    value = 0;
    Mode=0;                    /* SETUP for Mode 0 */
    Params[0]=BASE_DAS16;          /* Base address */
    Params[1]=7;                /* 7 Interrupt Level */
    Params[2]=1;                /* 3 DMA Level */
    mscl_dasg(&Mode, Params, &Flag);  /* EXECUTE Mode 0 */
    if (Flag !=0)               /* Error if Flag is not zero */
    {
        printf("\n\nMode 0 Error Flag = %d\n", Flag);
        error = 1;
    }
    outp(BASE_DAS16+9, 3);
    value = inp(BASE_DAS16+9);
    if (value != 3) error = 1;
    outp(BASE_DAS16+9, 2);
    value = inp(BASE_DAS16+9);
    if (value != 2) error = 1;
    outp(BASE_DAS16+9, 0);
        if (error) {
        printf("\nNo DAS-16 Board Found at address %d",
        BASE_DAS16);
        exit(0);
    } else
    printf("          DAS-16 Board Found at address %d", BASE_DAS16);
}

float getval_das16(int channel, int gain){
    int hi, low, polarity;
    unsigned long x;
    float val;
    outp(BASE_DAS16+11,gain);     /* Set Gain */
```

```
    outp(BASE_DAS16+2,(channel*16)+channel); /* Set channel via mux limits*/
    for (x=1; x<DELAY; x++);
    outp(BASE_DAS16,0);           /* Start 12 Bit Conversion */
    while (inp(BASE_DAS16+8) > 127);/* Wait for DONE bit to be cleared */
    low = inp(BASE_DAS16);        /* Read Low Nibble */
    hi  = inp(BASE_DAS16+1);      /* Read MSByte */
    val =(float)hi*16.0 + (float)low/16.0;  /* Add bytes and right-shift low */
    val = (val-2048)/(204.8*(float)gain); /* For bipolar operation*/
    return val;
}

void setup_counter0(){
    int          Mode, Params[16], Flag;
    outp(BASE_DAS16+10, 2);  /* Enable 100kHz clock to ctr 0 */
    Mode = 10;
    Params[0] = 3;
    mscl_dasg(&Mode, Params, &Flag);  /* Setup Square Wave config ctr 0 */
    Mode = 11;
    Params[0] = 3125;  /* Set divisor: freq = 100,000/3125 = 32 Hz */
    mscl_dasg(&Mode, Params, &Flag);  /* Setup Square Wave config ctr 0 */
}

int wait_loop(void){
      int    Mode, Params[16], Flag, prevsamp, peak1, peak2;
      int k, count, start1,start2,bye;
      peak1 = 0;
      peak2 = 0;
    start1 = 0;
    start2 = 0;
    while(!peak2){
     Mode = 12;
     Params[0] = 1;  /* Latch before Read */
     mscl_dasg(&Mode, Params, &Flag);  /* Read Counter 0 */
     if (Params[1] > 32767) Params[1] = Params[1]-65537;
     count = Params[1];
          if((count == 0))                start1 = 1;
     if((start1 == 1) && (count > 500))  peak1 = 1;
     if((count == 0)  && (peak1 == 1))   start2 = 1;
     if((start2 == 1) && (count > 500))  peak2 = 1;
      }
}

read_com_port(){
    int i;
     buf_getword_com1();
     for(i=1;i<49;i++){
      if((word[i] =='$')){
       _settextposition(3,30);
       sprintf(w2, "HEADING: %c%c%c%c%c",
       word[i+1],word[i+2],word[i+3],word[i+4],word[i+5]);
       _outtext(w2);
       }
      }
}
```

APPENDIX B: calib.c Program

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

#define BASE_DAS48 0x380
#define BASE_DAS16 0x300
#define DELAY 100

float getval_das16(int channel, int gain);
float getval_das48(int channel, int gain);
int read_couter0(void);
void setup_counter0(void);
extern void mscl_dasg(int *, int *, int *);  /* DECLARE CALL structure */

main()
{
    int    channel, gain, hi_c, low_c;
    int    i, gain_das16, gain_das48, num, num1;
    long   num_samp, sample_count, dummy, div, trigger;
    char   foo[80], out_file[12];
    float  Data[64];
    FILE   *foof;

    gain_das48 = 8;
    gain_das16 = 1;
    _clearscreen(_GCLEARSCREEN);
    _displaycursor(_GCURSOROFF);

    setup_das16();
    setup_counter0();
    sample_count = 0;

    _settextcolor(12);
    _outtext("\n\n\n          Data Display Mode -- Hit Any Key To Exit...");
    _settextcolor(14);

      while (!kbhit()) {

      for (i=0;  i<16; i++)  Data[i]  = getval_das16(i, gain_das16);
      for (i=16; i<64; i++)  Data[i]  = getval_das48(i-16, gain_das48);

      _settextposition(6,0);
        for (i=0; i < 64; i=i+4) {
            if (i == 16) _outtext("\n");
            sprintf(foo,
          "Ch[%02d]: %6.2fV    Ch[%02d]: %6.2fV    Ch[%02d]: %6.2fV    Ch[%02d]: %6.2fV   \n",
    i, Data[i], i+1, Data[i+1], i+2, Data[i+2], i+3, Data[i+3]);
            _outtext(foo);
```

```
        }
    for (dummy=0; dummy<1000000; dummy++);
    }

    getch();
    printf("hit any key to start calib loop");

    getch();

/*** Collect calibration data - 225 samples at 75 Hz ***/

   _settextcolor(11);
   for (num=1; num<=8; num++)
    {
    if (num==1) {
      strcpy(out_file,"zero1.dat");
      num1 = 1;
      }
    if (num==2) {
      strcpy(out_file,"step1.dat");
      num1 = 1;
      }
    if (num==3) {
      strcpy(out_file,"zero2.dat");
      num1 = 2;
      }
    if (num==4) {
      strcpy(out_file,"step2.dat");
      num1 = 2;
      }
    if (num==5) {
      strcpy(out_file,"zero3.dat");
      num1 = 3;
      }
    if (num==6) {
      strcpy(out_file,"step3.dat");
      num1 = 3;
      }
    if (num==7) {
      strcpy(out_file,"zero4.dat");
      num1 = 4;
      }
    if (num==8) {
      strcpy(out_file,"step4.dat");
      num1 = 4;
      }

/** if odd **/
    if (!(num % 2) == 0)
    {
    printf("\n\nPress Enter when ready to take zero data set %d\n",num1);
    getch();
    }
```

```
/** if even **/
    else
    {
    printf("\n\nPress Enter when ready to take step data set %d\n",num1);
    printf("\nAfter pressing Enter, flip step switch within 2 seconds\n");
    getch();
    }

    foof = fopen(out_file, "w");
    low_c = 0;
    hi_c = 63;
    num_samp = 225;
    div = 1250;
    trigger = div - 100;
    sample_count =0;
    while (sample_count < num_samp) {
      printf("%d\r",sample_count);
      while (read_counter0() > trigger);
 /* wait until counter gets to a specific value before running */

      for (i=0;  i<16; i++)  Data[i]  = getval_das16(i, gain_das16);
      for (i=16; i<64; i++)  Data[i]  = getval_das48(i-16, gain_das48);
      for (i=low_c; i<=hi_c; i++) fprintf(foof, "%6.3f ", Data[i]);
      fprintf(foof, "\n");

      while (read_counter0() < trigger); /* if sampling gets done before counter terminates, wait... */

      ++sample_count;
      }

    printf("\nDone collecting data.  Count = %d", sample_count);

    fclose(foof);
    _settextcolor(10);
    _outtext("\n\n");
    }
}

float getval_das48(int channel, int gain){
    int hi, low;
    unsigned long x;
    float val;

    /* Gain    Range    */
    /*  1     0-10v    */
    /*  3     0-5v     */
    /*  5     0-2.5v   */
    /*  7     0-1.25v  */
    /*  8     +/-10v   */
    /*  0     +/-5v    */
    /*  2     +/-2.5v  */
    /*  4     +/-1.25v */
    /*  6     +/-0.625v */
```

```
    outp(BASE_DAS48+3,gain);        /* Set Gain */
    outp(BASE_DAS48+2,channel);     /* Set channel */
    for (x=1; x<DELAY; x++);
    outp(BASE_DAS48+1,0);           /* Start 12 Bit Conversion */
    while (inp(BASE_DAS48+2) > 127); /* Wait for DONE bit to be cleared */
    low = inp(BASE_DAS48);          /* Read Low Nibble */
    hi  = inp(BASE_DAS48+1);        /* Read MSByte */
    val = hi*16.0 + low/16.0;  /* Add bytes and right-shift low */
    val = val - 2048.0;

    switch (gain) {
        case 0:
        case 1:
            val = val / 409.6;
            break;
        case 2:
        case 3:
            val = val / 819.2;
            break;
        case 4:
        case 5:
            val = val / 1638.4;
            break;
        case 6:
        case 7:
            val = val / 3276.8;
            break;
        case 8:
            val = val / 204.8;
            break;
    }
    return val;
}

setup_das16(){
    /* First, test for board's existence by writing and reading a reg */
    int    error, value;
    int    Mode, Params[16], Flag;
    error = 0;
    value = 0;
    Mode=0;                         /* SETUP for Mode 0 */
    Params[0]=BASE_DAS16;           /* Base address */
    Params[1]=7;                    /* 7 Interrupt Level */
    Params[2]=1;                    /* 3 DMA Level */
    mscl_dasg(&Mode, Params, &Flag);  /* EXECUTE Mode 0 */
    if (Flag !=0)                   /* Error if Flag is not zero */
    {
        printf("\n\nMode 0 Error Flag = %d\n", Flag);
        error = 1;
    }

    outp(BASE_DAS16+9, 3);
    value = inp(BASE_DAS16+9);
```

```
    if (value != 3) error = 1;

    outp(BASE_DAS16+9, 2);
    value = inp(BASE_DAS16+9);
    if (value != 2) error = 1;

    outp(BASE_DAS16+9, 0);
        if (error) {
        printf("\nNo DAS-16 Board Found at address %d",
        BASE_DAS16);
        exit(0);
    } else
    printf("            DAS-16 Board Found at address %d", BASE_DAS16);
}

float getval_das16(int channel, int gain){
    int hi, low, polarity;
    unsigned long x;
    float val;
    outp(BASE_DAS16+11,gain);      /* Set Gain */
    outp(BASE_DAS16+2,(channel*16)+channel);  /* Set channel via mux limits*/
    for (x=1; x<DELAY; x++);
    outp(BASE_DAS16,0);           /* Start 12 Bit Conversion */
    while (inp(BASE_DAS16+8) > 127);/* Wait for DONE bit to be cleared */
    low = inp(BASE_DAS16);        /* Read Low Nibble */
    hi  = inp(BASE_DAS16+1);      /* Read MSByte */
    val =(float)hi*16.0 + (float)low/16.0;   /* Add bytes and right-shift low */
    val = (val-2048)/(204.8*(float)gain); /* For bipolar operation*/
    return val;
}

void setup_counter0(){
    int          Mode, Params[16], Flag;
    outp(BASE_DAS16+10, 2); /* Enable 100kHz clock to ctr 0 */
    Mode = 10;
    Params[0] = 2;
    mscl_dasg(&Mode, Params, &Flag); /* Setup div-by-N config ctr 0 */
    Mode = 11;
    Params[0] = 1333; /* Set divisor:  freq = 100,000/1333 = 75.01875 Hz */
        /* use divisor-1 because clock pulses low for one count */
            /*  once count reaches terminal before resetting */
    mscl_dasg(&Mode, Params, &Flag); /* Setup div-by-n config ctr 0 */
}

int read_counter0(void){
    int    Mode, Params[16], Flag;
    long foo;
    Mode = 12;
    Params[0] = 1;  /* Latch before Read */
    mscl_dasg(&Mode, Params, &Flag); /* Read Counter 0 */
        if (Params[1] > 32767) Params[1] = Params[1]-65537;
    return Params[1];
}
```

APPENDIX C: calcon11.for Program

```
program calcon11

c This program is designed to move the poles of the theoretical frequency response function of the wave
c gage to best fit a frequency response magnitude as measured by calibrating the wave gage. In other
c words, the theoretical shape is going to be fit to the measured data. The program then calculates the
c calibration constant for each channel.

c   step1.dat - step4.dat contain four step responses of the wave gage
c   avdata.out contains the average of experimentally measured H(w)
c   oldom.out contains the original theoretical frequency response
c   newom.out contains the modified (fitted) theo. frequency response

C The program also lists any channels that have excessive noise levels. These channels are also not used
c in further calculations. The data for faulty channels is interpolated from the two surrounding channels
c in the following program, autoprp*.for. If two or more adjacent channels are faulty, the calculations
c are stopped and the user is informed that the hardware situation needs to be fixed before further
c calculations can accurately be obtained.

C The step height for channel 0 is calculated from data collected on
C February 14th. Please see my notes for clarification.

      implicit none
      real w(1024), pi, a(3), yfit, homega(64,1024)
      real wstep1, tstep, max(64), sig(1024), max4
      real test(512), dyda(3), temp(400,64), have(512)
      real peak(64), peak1(64), maxv(64), max3(64)
      real covar(3,3), alpha(3,3), chisq, alamda, chiset
      real mean, mean1, shgt, const(64), w1(2048)
      real std, std1, meanp, peak2(64), spec(8,64,1024)
      real specstep(64,512), speczero(64,512), peak3(64)
      real homega3(512), max1, hd(64), hd1(64),hout(64), sf
      integer topctof, botctof
      integer i, ma, numsamp,n,j,k, ia(3), step, chan
      integer badchan(64), lowstop, highstop, numgoodch
      integer subzero, nch
      complex c(1024), homega2(2048)

      external funcs

      open(10,file='step1.dat',status='old')
      open(11,file='step2.dat',status='old')
      open(12,file='step3.dat',status='old')
      open(13,file='step4.dat',status='old')
      open(14,file='zero1.dat',status='old')
      open(15,file='zero2.dat',status='old')
      open(16,file='zero3.dat',status='old')
      open(17,file='zero4.dat',status='old')
      open (100, file='newom.out',status='unknown')
      open(110,file='avdata.out',status='unknown')
```

```
      open(120,file='oldom.out',status='unknown')
      open(30,file='const.cal',status='unknown')
      open(200,file='stats.out',status='unknown')
      open(220,file='foo.out',status='unknown')
      open(230,file='step1.cal',status='unknown')
      open(240,file='step2.cal',status='unknown')
      open(250,file='step.cal',status='unknown')
      open(500,file='have.out',status='unknown')

      write(6,'(a,$)') ' Enter the calibration sampling freq. (Hz) : '
      read(6,*) sf
      write(200,*) ' Calibration sampling rate = ',sf
      write(6,'(a,$)') ' Enter the number of channels sampled : '
      read(6,*) nch
      write(200,*) ' # channels sampled = ',nch
      write(200,*) ' '

      pi = 3.141593
C **** stepheight in centimeters for channel 0 ****
      shgt = 1.0*2.54
C number of samples collected from each channel on wave gage
      numsamp = 250
C frequency step in radians from ft of actual step response
      wstep1 = sf/1024.*2*pi
C data acquisition timestep in seconds
      tstep = 1./sf

      write(6,'(a,$)') ' Enter channel you want to look at: '
      read(6,*) chan
      chan = chan + 1
      subzero = 1

      do 7 i=1,nch
        badchan(i) = 0
        max3(i) = 0.0
        maxv(i) = 0.0
        max(i) = 0.0
        peak(i) = 0.0
        peak1(i) = 0.0
        peak2(i) = 0.0
        peak3(i) = 0.0
 7    continue
      do 8 i=1,512
        have(i) = 0.0
 8    continue
      numgoodch = 0

C  Print introductory stuff...
        print *
        write(6,*) 'The following channel(s) is(are) suspected of '
        write(6,*) 'having ground faults or being excessively '
        write(6,*) 'noisy.  The channels listed will not be included'
        write(6,*) 'in any further calculations.'
        print *
```

```
      write(6,*) 'Channel(s) '
      write(200,*) ' '
      write(200,*) 'The following channel(s) is(are) suspected of '
      write(200,*) 'having ground faults or being excessively '
      write(200,*) 'noisy.  The channels listed will not be included'
      write(200,*) 'in any further calculations.'
      write(200,*) ' '
      write(200,*) 'Channel(s) '

C Read in input data files from wave gage
    do 150 n=10,17
      do 30 i=1,numsamp
        read(n,*,end=31) (temp(i,j), j=1,nch)
30    continue
31    continue

c *****
c - determine if step data is faulty
c - if not then:
c  - take fourier transform of each channel of data in the data
c    files (will need to zero pad because only have 225 pts)
c  - bandlimit and square them
c  - calculate the area underneath them
c  - average the areas for each channel over the four data sets
c    divide this area by the area calculated above to get the
c    calibration constant
*****
      do 100 j=1,nch
        do 40 i=1,1024
          if (i.le.numsamp) then
            c(i) = temp(i,j)
          else
            c(i) = (0.0,0.0)
          endif
40       continue

c Measure maximum value of voltage input
        if (n.le.13) then
          do 42 i=1,numsamp
         if (abs(temp(i,j)).gt.maxv(j)) maxv(j) = abs(temp(i,j))
42          continue
        endif

C  Check for ground fault.
c  If maximum voltage from step input does not exceed 4 volts, or
c  is 10 volts (ie - off-scale all the time) the
c  channel likely has a ground fault.  Assume that any faults that
c  occur will occur in all data files, so only need to look at first
c  data set.
        if (n.eq.10) then
          if (maxv(j).lt.(3.0).or.maxv(j).ge.(9.99)) then
            if (j.eq.1) then
              write(6,*) 'Channel 0 is faulty - calcs stopped'
              write(200,*) 'Channel 0 is faulty - calcs stopped'
```

```
            goto 999
          endif
          write(6,'(i4,a)') j-1,' not receiving signal '
          write(200,'(i4,a)') j-1,' not receiving signal '
          badchan(j) = 1
          goto 100
        else
          numgoodch = numgoodch + 1
        endif
      endif

c Take fourier transform of step and zero data.
      c(1024) = c(1)
      call FFTJM(c,10,1)

c Multiply transform by omega to get impulse response. It also needs to be multiplied by the number of
c data points and the sampling frequency to get the correct magnitude as described in FFTJM.FOR  Also,
c save spectral data for each data set.
      do 43 k=1,1024
        c(k) = c(k)*(k/1024./tstep*2.*pi)*(1024.*tstep)
        spec(n-9,j,k) = cabs(c(k)*c(k))
43      continue

c Homega will be the averaged magnitude of the frequency response first by averaging all the channels
c together and then by averaging over the four data set. The approximate areas under each freq. resp.
c are normalized so that channels with much larger or much smaller amplitude frequency responses are
c not artificially weighted more or less than any other channel. Also, find locations of peak of each
c channel to insure that they are all at about the same place.
      if (n.le.13) then
        if (badchan(j).ne.1) then
          peak1(j) = 0.0
          do 45 k=1,400
            if (peak1(j).lt.cabs(c(k))) then
              peak1(j) = cabs(c(k))
              peak2(j) = k/1024./tstep
            endif
45          continue
          endif
        endif
100   continue

c Sum peak frequencies over all data sets
      if (n.le.13) then
        do 105 i=1,nch
          if (badchan(i).ne.1) then
            peak(i) = peak(i) + peak2(i)
            peak3(i) = peak3(i) + peak1(i)
          endif
105       continue
      endif
150 continue

c Generate homega vector for each channel
c  - first average spectra (both step and zero) over the four data sets
```

```
c  - subtract zero specta from step spectra
c  - take square root to get homega for each channel
      do 170 i=1,nch
        if (badchan(i).ne.1) then
          do 165 j=1,512
            do 160 k=1,4
              specstep(i,j)=specstep(i,j)+spec(k,i,j)
 160        continue
            do 163 k=5,8
              speczero(i,j) = speczero(i,j) + spec(k,i,j)
 163        continue
 165      continue
        endif
 170  continue

      do 180 i=1,nch
        if (badchan(i).ne.1) then
          do 175 j=1,512
            specstep(i,j) = specstep(i,j)/4.
            speczero(i,j) = speczero(i,j)/4.
            if (subzero.eq.1) then
              if (specstep(i,j).gt.speczero(i,j)) then
                homega(i,j) = sqrt(specstep(i,j)-speczero(i,j))
              else
                homega(i,j) = 0.0
              endif
            else
              homega(i,j) = sqrt(specstep(i,j))
            endif
            if (homega(i,j).gt.max(i)) max(i) = homega(i,j)
 175      continue
        endif
 180  continue

c Generate frequency vector.
      do 190 i=1,512
        w(i) = i/1024./tstep*2.*pi
        write(110,*) w(i)/2./pi, homega(chan,i)
 190  continue

      write(6,*) ' '
      write(6,*) 'Done reading and calculating step data'
      write(200,*) 'Done reading and calculating step data'
      print *

c  The following loop calculates a best fit for each of the channels.  It prints the values of the best fit
c  parameters to an output file called params.out.  The magnitude of the theoretical transfer functions are
c  stored in a data file called newom.out.  The real and imaginary parts of the transfer functions are stored
c  in data files called step1.cal and step2.cal, respectively.
      do 300 k=1,nch
      if (badchan(k).ne.1) then

      do 205 i=1,512
        homega3(i) = homega(k,i)
```

```
      have(i) = have(i) + homega3(i)
 205  continue

c  Values for the capacitor(s) and the time constant for the
c  RMS-DC converter low-pass filter
      ma = 3
      a(1) = 0.22
      a(2) = 0.015
      a(3) = 1.0

c Find max of the theoretical frequency response
      max1 = 0.0
      do 207 i=1,512
        call funcs(w(i),a,yfit,dyda,ma)
        if (yfit.gt.max1)  max1 = yfit
 207  continue

c  Evaluate transfer function for initial conditions of a()
      do 208 i=1,512
        call funcs(w(i),a,yfit,dyda,ma)
        if (k.eq.chan) then
          write(120,*) w(i)/2/pi,yfit/max1*max(k)
        endif
 208  continue

      a(3) = max(k)/max1

c  Call Marquardt's Method subroutine which performs a nonlinear parameter estimation for the values of
c  a( ) (capacitors).

c  Set standard deviation of data set (ie. set weights for fitting function - the smaller sigma is the higher
c  the weight at that point)  From 0 to 2 Hz fit has medium importance, from 2 to 12 Hz fit has most
c  importance, from 12 to 37.5 Hz fit has medium importance.

      do 210 i=1,512
        if (i.lt.(1024/sf*12.)) then
          sig(i) = 0.1
        else
          sig(i) = 1.0
        endif
 210  continue
      ia(1) = 1
      ia(2) = 1

c  Initialize mimization subroutine
      alamda = -1.0
      call mrqmin(w,homega3,sig,512,a,ia,ma,covar,alpha,
     +            3,chisq,funcs,alamda)
      chiset = chisq

c  Call mimization subroutine
      step = 0
      do 260 i=1,30
        call mrqmin(w,homega3,sig,512,a,ia,ma,covar,alpha,
```

```
     +          3,chisq,funcs,alamda)
        if ((chiset-chisq).lt.0.0000001.and.step.gt.2) then
          goto 261
        else
          step = step + 1
          chiset = chisq
        endif
 260  continue
 261  continue
write(200,*)  k-1,a(1), a(2), a(3)

c  Generate best fit transfer function using best fit values of a()
      do 280 i=1,512
        call funcs(w(i),a,yfit,dyda,ma)
        test(i) = yfit
        if (yfit.gt.max3(k)) max3(k) = yfit
        if (k.eq.chan) then
          write(100,*) w(i)/2./pi, test(i)
        endif
 280  continue

c  Generate real and imaginary parts of transfer function using best fit values of a(). Write output to
c  step1.cal and step2.cal. Create new frequency vector which makes calculations in autoprp*.for easier.
      do 285 i=1,2048
        w1(i) = i/2049.*16.*2.*pi
 285  continue

       call toth(w1,a,homega2)
         write(230,*) k-1, (real(homega2(i)/max3(k)), i=1,2048)
         write(240,*) k-1, (aimag(homega2(i)/max3(k)), i=1,2048)

c  Calculate square area under normalized theoretical frequency resp. Cutoff frequencies correspond to
c  approximately 2 Hz and 10 Hz (limited by resolution of FFT of step data set)
      lowstop = nint(2.*1024.*tstep)
      highstop = nint(10.*1024.*tstep)
      hd1(k) = 0.
      do 299 j=lowstop,(highstop-1)
        hd1(k)=hd1(k)+((test(j)/max3(k))**2+(test(j+1)/max3(k))**2)/2.
 299   continue
      hd(k) = hd1(k) * wstep1
      write(200,*) '          A^2 under fitted norm. |H(w)| = ',hd(k)
      endif
 300  continue

C ** Calculate average homega vector
        do 305 i=1,512
          have(i) = have(i)/numgoodch
          write(500,*) have(i)
 305     continue

c  Values for the capacitor(s) and the time constant for the
c  RMS-DC converter low-pass filter
      ma = 3
      a(1) = 0.22
```

```
      a(2) = 0.015
      a(3) = max(nch)/max1

c  Call Marquardt's Method subroutine which performs a nonlinear
c  parameter estimation for the values of a() (capacitors).

      do 310 i=1,512
        if (i.lt.(1024/sf*12.)) then
          sig(i) = 0.1
        else
          sig(i) = 1.0
        endif
  310 continue
      ia(1) = 1
      ia(2) = 1

c  Initialize mimization subroutine
      alamda = -1.0
      call mrqmin(w,have,sig,512,a,ia,ma,covar,alpha,
     +            3,chisq,funcs,alamda)
      chiset = chisq

c  Call mimization subroutine
      step = 0
      do 360 i=1,30
        call mrqmin(w,have,sig,512,a,ia,ma,covar,alpha,
     +            3,chisq,funcs,alamda)
        if ((chiset-chisq).lt.0.00001.and.step.gt.2) then
          goto 361
        else
          step = step + 1
          chiset = chisq
        endif
  360 continue
  361 continue
      max4 = 0.
      do 362 i=1,512
        if (have(i).gt.max4) max4 = have(i)
  362 continue
      write(200,*) ' have ',a(1), a(2), a(3)

c  Generate real and imaginary parts of transfer function using best fit values of a(). Write output to
c  step1.cal and step2.cal. Create new frequency vector which makes calculations in autoprp*.for
c  easier. (normalize them with max4)

      do 385 i=1,2048
        w1(i) = i/2048.*16.*2.*pi
  385 continue
       call toth(w1,a,homega2)
       do 390 i=1,2048
         write(250,*) real(homega2(i))/max4,aimag(homega2(i))/max4
  390   continue

c  Calculate square area under step responses (between 2 and 10 Hz)  [need to mulitply fft by n and by
```

```
c  sampling interval to get correct result (see fftjm.for program)]
      botctof = 1024./sf*2.
      topctof = 1024./sf*10.
      do 404 i=1,nch
        if (badchan(i).ne.1) then
          hout(i) = 0.0
          do 402 j=botctof,topctof-1
            hout(i) = hout(i) + homega(i,j)*homega(i,j)
 402      continue
          hout(i) = hout(i)*wstep1
          write(200,*) i-1, hout(i), hd(i)
        endif
 404  continue

C  Calculate calibration constants
      do 405 i=1,nch
        if (badchan(i).ne.1) then
          const(i) = sqrt(hout(i)/hd(i))/shgt
        else
          const(i) = 0.0
        endif
 405  continue

C Find mean of each channel
      mean = 0.
      mean1 = 0.
      do 410 i=1,nch
        if (badchan(i).ne.1) then
          mean1 = mean1 + const(i)
        endif
 410  continue
      mean = mean1/numgoodch
      write(30, '(f9.3)') mean
      meanp = 0.0
      do 420 i=1,nch
        write(30,'(i5, f10.4)') i-1, const(i)
        if (badchan(i).ne.1) then
          write(220, *) peak(i)/4., peak3(i)/4.
          meanp = meanp + peak(i)/4.
        endif
 420  continue
      meanp = meanp/numgoodch

c  Calculate standard deviation of peak locations
      std = 0.
      std1 = 0.
      do 425 i=1,nch
        if (badchan(i).ne.1) then
          std1 = std1 + (peak(i)/4.-meanp)**2
        endif
 425  continue
      std = sqrt(std1/numgoodch)

      write (6,430) 'The number of channels used = ',numgoodch
```

```
      write (6,440) 'The mean of the cal. constants = ',mean
      write (6,'(1x,a,f10.4)') 'Channel 0 cal. const. = ',const(1)
      write(6,450) 'The mean and std of the peaks occur at (Hz): ',
     + meanp, std
      write (200,430) 'The number of channels used = ',numgoodch
      write (200,440) 'The mean of the cal. constants = ',mean
      write (200,'(1x,a,f10.4)') 'Channel 0 cal. const. = ',const(1)
      write(200,450) 'The mean and std of the peaks occur at (Hz): ',
     + meanp, std
  430 format (1x,a,i4)
  440 format (1x,a,f9.3)
  450 format (1x,a,2f9.3)

  999 continue
      end

c ********************************************************
c  The following subroutine evaluates the real and imaginary parts
c  of the theoretical transfer function which will be used by
c  autoprp*.for program which turns the voltage wave height output
c  from the gage into centimeter wave height output.
c********************************************************

c  This subrouting calculates the value of the real and imaginary
c  parts of the theoretical frequency response function (as a function
c  of the capacitors in the filters).
c       c = capacitor value in microfarads (from high pass filters)
c       t = time constant for RMS-DC converter 'filter'
c       mag = scaling factor
c       homega = complex array containing frequency response

      subroutine toth(omega,a,homega)

      implicit none
      real w,rlp,ra,rb,rhp1,rhp2,glp,ghp1,ghp2,s
      real k, omega(2048), a(3)
      complex clp,chp1,chp2,clp2,cj,homega(2048)
      integer i

c  Resistor and capacitor values and constants
c  See notes for locations of components (pg. 214 and 195) from
c  Theory and Design of Linear Active Networks by Natarajan

      cj = (0.0,1.0)
c resistor values
      rlp = 75000.
      ra = 18200.
      rb = 30100.
      rhp1 = 121000.
      rhp2 = 100000.
      glp = 1/rlp
      ghp1 = 1/rhp1
      ghp2 = 1/rhp2
      s = 1/(a(1)*0.000001)
```

```
      k = 1 + ra/rb
      do 900 i = 1,2048
        w = omega(i)
        clp=k*glp*glp*s*s/((cj*w)**2+cj*w*(s*2*glp+s*glp*(1-k))+
     +       glp*glp*s*s)
        chp1=k*(cj*w)**2/((cj*w)**2+cj*w*(s*2*ghp1+s*ghp1*
     +       (1-k))+ghp1*ghp1*s*s)
        chp2=k*(cj*w)**2/((cj*w)**2+cj*w*(s*2*ghp2+s*ghp2*
     +       (1-k))+ghp2*ghp2*s*s)
        clp2 = 1/(1+cj*w*a(2))
        homega(i) = a(3)*clp*chp1*chp2*clp2
  900 continue
      return
      end
```

APPENDIX D: autop1.for Program

```
program autop1

c This file reads in a data file of voltage values which is as large as 64 colums x 4096 rows in size, (128
c seconds of data sampled at 32 Hz) calculates the mean value and variance of each column, subtracts out
c the column means from each column of data. It also incorporates the previously calculated calibration
c constants and uses them to change the input values from volts to cm using transfer function relations.
c Two output files are written. One contains a list of all the channels, thier mean value and variance, the
c other contains the data in proper format to be used as input to sw_auto.c which calculates the
c autocorrelation of the data set.
c This program also interpolates a height value for the channels that were determined to be bad due to
c noise or other problems during the calibration constant calculations.

      implicit complex (c)
      real var, mean, meancal, cal, realh, imagh
      real var3, mean3, sum3, energyav
      real new, direct, pfreq, numbot1, numtop1
      real fmax, fmax1, fmax2
      integer badchan, channum, rlgth, newzero, useaveh
      integer chan, nch
      character fname1*18
      dimension mean(4097), data(4097,64), out(4097,64), sum(4097)
      dimension var(64), cal(64), badchan(64)
      dimension cfreq(64,4097), c(4097), realh(64,4097)
      dimension imagh(64,4097), direct(4097), new(4097,64)
      dimension var3(64), sum3(4097), mean3(4097)
      dimension energy(64), c1(64,4097), fmax1(64), fmax2(64)

c Get input data file name
      write (*, '(a,$)')' Type input data file name  '
      read (*, '(a)') fname1
      write(*,*)
      write (*, '(a,$)')' Type # of channels sampled  '
      read (*,*) nch
      write(*,*)
      write (*,'(a,$)')' Type data sampling frequency  '
      read (*,*) pfreq
      write(*,*)
      useaveh = 2
      numbot1 = nint(2.0*4097./pfreq)
      numtop1 = nint(10.0*4097./pfreq)

c Open calibration constants file and read
      open (19, file='const.cal', status='old')
      read (19,*) meancal
      do 20 ii=1,nch
        read (19, *) channum, cal(ii)
        if (cal(ii).eq.(0.0)) then
          badchan(ii) = 1
          cal(ii) = meancal
```

```
      else
        badchan(ii) = 0
      endif
20  continue
    close(19)
    write(6,*) 'Done reading const.cal'

c Open files with real and imaginary parts of H(w)
    open (40, file='step1.cal', status='old')
    open (50, file='step2.cal', status='old')
    open (500, file='foo.cal', status='unknown')
c length of theoretical freqency response
    rlgth = 2049

c Read in theoretical frequency response function
c Mirror it so it can be used in Jerry's FFTJM.for program
    do 30 i=1,nch
      if (badchan(i).ne.1.and.useaveh.eq.2) then
        read(40,*,end=25) chan, (realh(chan+1,j), j=2,rlgth)
        write(500,*) chan,realh(chan+1,rlgth)
        read(50,*,end=25) chan, (imagh(chan+1,j), j=2,rlgth)
        write(500,*) chan,imagh(chan+1,rlgth)
      else
        write (6,*) 'Do not want to be here!!!!', i-1
        open (60, file='step.cal', status='old')
        do 27 j=2,rlgth
          read(60,*,end=28) realh(i,j), imagh(i,j)
27      continue
28      continue
        close(60)
25    endif

      do 29 j=1,rlgth
        if (j.eq.1) then
          cfreq(i,j) = (0.0,0.0)
        else
          cfreq(i,j) = cmplx(realh(i,j),imagh(i,j))
          if (j.ne.1.or.j.ne.rlgth) then
            cfreq(i,4098-j) = conjg(cfreq(i,j))
          endif
        endif
29    continue

c Set nyquist frequency value to real magnitude
      cfreq(i,2049) = cabs(cfreq(i,2049))

30  continue
    close(30)
    write(6,*) 'Done reading step.cal'


    do 50 i=1,4097
      sum(i)=0.0
      sum3(i) = 0.0
```

```
 50  continue


c Read data file and calculate various sums to help calculate variance and means.
      open (20, file=fname1, status='old')
      do 100 i=1,4096
        read (20, *,end = 105) direct(i), (data(i,k), k=1,nch)
        direct(i) = direct(i) - 30.0
        do 60 j=1,nch

c Interpolate between bad channels and kill if three adjacent channels are bad.  Note that channel numbers
c  in program are actually one greater than channel numbers on the gage (i.e. they run from 1-64 vs. from
c  0-63)
c Note that some of the conditionals below are redundant, but it seemed clearer to me to leave them in
c large repeatable blocks as apposed to fewer lines but way too many nested loops... if you don't like it,
c TOUGH!!! *smile*
            if (badchan(j).eq.1) then
c Channel 1 wraparound problem
            if (j.eq.1) then
              if ((badchan(1)+badchan(2)).eq.2) then
                data(i,1) = data(i,nch)*2./3.+data(i,3)/3.
c Channel 2 will be taken care of later in the program
              elseif ((badchan(1)+badchan(nch)).eq.2) then
                data(i,1) = data(i,nch-1)/3.+data(i,2)*2./3.
c Channel 64 will be taken care of later in the program
            elseif (badchan(1)+badchan(nch)+badchan(nch-1).eq.3.or.
     +              badchan(1)+badchan(2)+badchan(nch).eq.3.or.
     +              badchan(1)+badchan(2)+badchan(3).eq.3) then
              write(*,*) 'Three adjacent chans are not working'
              write(*,*) 'Program stopped'
              write(*,*) 'The truth is out there... '
              goto 999
            else
              data(i,1) = (data(i,nch) + data(i,2))/2.
            endif
c Channel 64 wraparound problem
            elseif (j.eq.nch) then
              if ((badchan(nch)+badchan(1)).eq.2) then
                data(i,nch) = data(i,nch-1)*2./3.+data(i,2)/3.
c Channel 1 should have been taken care of above
              elseif ((badchan(nch)+badchan(nch-1)).eq.2) then
                data(i,nch) = data(i,nch-2)/3.+data(i,1)*2./3.
c Channel 63 will be taken care of later in the program
          elseif (badchan(nch)+badchan(nch-1)+badchan(nch-2).eq.3.or.
     +            badchan(nch)+badchan(nch-1)+badchan(1).eq.3.or.
     +            badchan(nch)+badchan(1)+badchan(2).eq.3) then
              write(*,*) 'Three adjacent chans are not working'
              write(*,*) 'Program stopped'
              write(*,*) 'The truth is out there... '
              goto 999
            else
              data(i,nch) = (data(i,nch-1) + data(i,1))/2.
            endif
c Channel 2 wraparound problem
```

```
          elseif (j.eq.2) then
            if ((badchan(2)+badchan(3)).eq.2) then
              data(i,2) = data(i,1)*2./3.+data(i,4)/3.
c Channel 3 will be taken care of later in the program
            elseif ((badchan(2)+badchan(1)).eq.2) then
              data(i,2) = data(i,nch)/3.+data(i,3)*2./3.
c Channel 1 should have been taken care of above
            elseif (badchan(2)+badchan(3)+badchan(4).eq.3.or.
     +            badchan(2)+badchan(1)+badchan(3).eq.3.or.
     +            badchan(2)+badchan(1)+badchan(nch).eq.3) then
              write(*,*) 'Three adjacent chans are not working'
              write(*,*) 'Program stopped'
              write(*,*) 'The truth is out there... '
              goto 999
            else
              data(i,2) = (data(i,1) + data(i,3))/2.
            endif
c Channel 63 wraparound problem
          elseif (j.eq.nch-1) then
            if ((badchan(nch-1)+badchan(nch)).eq.2) then
              data(i,nch-1) = data(i,nch-2)*2./3.+data(i,1)/3.
c Channel 64 should have been taken care of above
            elseif ((badchan(nch-1)+badchan(nch-2)).eq.2) then
           data(i,nch-1) = data(i,nch-3)/3.+data(i,nch)*2./3.
c Channel 62 will be taken care of later in the program
          elseif (badchan(nch-1)+badchan(nch)+badchan(1).eq.3.or.
     +          badchan(nch-1)+badchan(nch-2)+badchan(nch).eq.3.or.
     +     badchan(nch-1)+badchan(nch-2)+badchan(nch-3).eq.3) then
            write(*,*) 'Three adjacent chans are not working'
              write(*,*) 'Program stopped'
              write(*,*) 'The truth is out there... '
              goto 999
            else
              data(i,nch-1) = (data(i,nch) + data(i,nch-2))/2.
            endif
c Deal with rest of channels
          else
            if ((badchan(j)+badchan(j+1)).eq.2) then
              data(i,j)=data(i,j-1)*2./3.+data(i,j+2)/3.
c Other channel will be taken care of later in iterations
            elseif ((badchan(j)+badchan(j-1)).eq.2) then
              data(i,j)=data(i,j-2)/3.+data(i,j+1)*2./3.
c Other channel will be taken care of in later iterations
            elseif (badchan(j)+badchan(j+1)+badchan(j+2).
     +            eq.3.or.badchan(j)+badchan(j-1)+badchan(j+1).
     +            eq.3.or.badchan(j)+badchan(j-1)+badchan(j-2).
     +            eq.3) then
              write(*,*) 'Two adjacent chans are not working'
              write(*,*) 'Program stopped'
              write(*,*) 'The truth is out there... '
              goto 999
            else
              data(i,j) = (data(i,j-1) + data(i,j+1))/2.
            endif
```

```
          endif
        endif
        sum3(j) = sum3(j) + data(i,j)
60      continue
100   continue
105   continue
    write(6,*) 'Done reading data file ',fname1
    close (20)

c Incorporate calibration constant
      open (102,file='foo2.dat',status='unknown')
      open (103,file='foo3.dat',status='unknown')
      open (104,file='foo4.dat',status='unknown')

    energyav = 0.0
    do 142 j=1,nch
      energy(j) = 0.0
      do 130 i=1,4096
         c(i)=data(i,j)
130   continue

      call FFTJM(c,12,1)
c I know that the first coefficient (the DC) has to be zero, so.. (or if it isn't, I'm going to force it to be..)
      c1(j,1) = 0.
c I'm going to bandpass filter this data because I can't physically resolve anything with a wavelength
c shorter than about 2 cm. This corresponds to a wavenumber of 3.0 cm/rad and a frequency of
c 70.73 rad/s or 11.3 Hz. Thus, I'm going to bandpass filter between 2 and 10 Hz.
c Calculate the energy (area under spectrum) between 2 and 10 Hz for each channel. This should be the
c same for every channel. Find the the average of this energy and then find the ratio of each channel's
c energy to the average. Use this ratio along with the calibration constant to 'calibrate' the output.
      fmax1(j)=0.0
      fmax2(j)=0.0
      do 140 k=2,4096
        if (k.gt.numbot1.and.k.lt.numtop1.or.k.gt.(4096-(numtop1-2)).
     +       and.k.lt.(4096-(numbot1-2))) then
          if (cfreq(j,k).eq.(0.0)) then
            write(6,*) 'chan = ', j-1,' data pt. = ',k
            goto 999
          endif
          c1(j,k)=c(k)/(cfreq(j,k)*cal(j))

c Calculate energy (don't need to include conversion factors because going to use a ratio of the energy to
c the average energy)
          energy(j) = energy(j) + cabs(c1(j,k)*c1(j,k))
          if(cabs(c1(j,k)).gt.fmax1(j)) then
            fmax1(j)=cabs(c1(j,k))
            fmax2(j)=k/4097.*pfreq
          endif
        else
          c1(j,k) = 0.0000000
        endif
140   continue
      energyav = energyav + energy(j)
142   continue
```

```
      energyav = energyav / nch

    open(600,file='foo2.dat',status='unknown')
    write(600,*) energyav
    write(*,*) ' '
    do 143 j=1,nch
      write(600,*) energyav/energy(j)
143 continue

    do 160 j=1,nch
      do 145 i=1,4096
        c(i) = c1(j,i)*sqrt(energyav/energy(j))
        if (j.eq.6) write(104,*) pfreq/4096*(i-1), c(i)
145   continue

      call FFTJM(c,12,-1)
      do 150 i=1,4096
        out(i,j) = real(c(i))
        sum(j) = sum(j) + out(i,j)
        if (j.eq.6) write(104,*) c(i)
150   continue
160 continue

c Renumber channels to take into account direction that gage is facing (i.e. - renumber so north is always
c associated with chan 0 where chan 0 is not fixed)

    do 195 i=1,4096
      newzero = anint((direct(i)/5.625))
      do 180 j=1,nch
        indx1 = newzero+j
        indx2 = newzero+j-nch
        if (indx1.le.nch) then
          new(i,j) = out(i,indx1)
        else
          new(i,j) = out(i,indx2)
        endif
180   continue
195 continue
    write(6,*) 'newzero = ',newzero


c Write adjusted data to file autoin.dat
    open(22, file='autoin.dat', status='unknown')
    do 200 k=1,4096
      write (22,'(64f12.5)') (new(k,j), j=1,nch)
200 continue
    close(22)

c Calculate means, variances (don't need to subtract out means, this is done in frequency space later)
    fmax=0.0
    do 220 j=1,nch
      mean(j) = sum(j)/4096.
      mean3(j) = sum3(j)/4096.
      var(j) = 0.
```

```
      var3(j) = 0.
      fmax = fmax + fmax2(j)
      do 210 k=1,4096
        var(j) = var(j) + (out(k,j) - mean(j))**2
        var3(j) = var3(j) + (data(k,j) - mean3(j))**2
210   continue
    var(j) = var(j) / 4096.
    var3(j) = var3(j) / 4096.
220 continue
    fmax = fmax/nch
    write(6,*) 'Average frequency of input sine wave is (Hz) ',fmax

c Write mean and variance of each channel
    open (23, file='check.dat',status='unknown')
    do 250 i=1,nch
      write (23, '(i5,4f15.4)') i-1,mean(i),var(i),mean3(i),var3(i)
250 continue
    close(23)

999 end
```

APPENDIX E: sw_auto.c Program

```
/**                SW_auto.C                          **/
/** Code to reconstruce the spectral energy of a seaway from   **/
/** wave height data points gathered on a circular harp        **/
/** Revision has check of autocorrelation                 **/
/** Average of autocorrelation is calculated beforehand        **/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

float ff[11], f[200][5];
float correl[70][70], xi[70][70], eta[70][70];
float avg_sum[70][70], avg_correl[70][70], sum_correl[70][70];

void main(int argc, char *argv[])
{
char ch, str[80], str2[22];
char *buffer;
FILE *f4, *f6, *f8;

int nc, np, num_pts, num_zero=55, l, m, n, o, p, q, r, scan, jout, sals;
int num_check, s_count=0,mass;
int ns, decimal, sign, jump;
int i, j, ii, jj, iii, jjj;
unsigned long num, count, num_calcs;
float dia, radius, angle, cord;
float rho=1.02, pi=3.14159;
float rmin= - 1, rmax=1.0, tor, tpi, pio2;
float d_gama, d_gamao2, d_nu, d_r, d_theta;
float x, y;
double d_ro2, dval, ten=10.0, nn;

/*************************************************************************/
/**  # angle segments=na, # k segments=nk, # wave component          **/
/**                CGS UNITS                                 **/
/**  rmin and rmax are log(base_10) of kmin and of kmax             **/
/*************************************************************************/

char fname[18];
printf("Enter the number of scans you want to evaluate:  ");
scanf("%d", &ns);
/** Open file for writing **/
    if((f6=fopen("autoin.dat","r"))==NULL)
        {
        printf("Cannot open file\n");
        exit(1);
        }
for(scan=1;scan<=ns;++scan)
```

```
    {
    nn = (double) scan;
    num_pts= 64;
    printf("Working on scan %d of %d\r",scan,ns);
    for(i=0;i<num_pts;i++){
            fscanf(f6,"%f",&cord);
            f[i][1]=cord;
        }

/** calculate the delta gama for every segment **/
    tpi=2.0*pi;
    pio2=pi/2;
    d_gama=tpi/num_pts;
    d_gamao2=d_gama/2.0;
    d_nu=tpi/num_pts;
    radius=10.0;                    /** Radius **/

            /*********************************************/
            /** Find AUTOCORRELATION of points on circle **/
            /*********************************************/

    for(i=0;i<num_pts;i++)
        {
        for(j=0;j<num_pts;j++)
            {
            correl[i][j]=f[i][1]*f[j][1];
            }
        }

    for(i=0;i<num_pts;i++)
        {
        for(j=0;j<i;j++)
            {
            cord = 2*radius*sin(abs(i-j) * d_gamao2);
            xi[i][j] = cord * cos(((num_pts - i) + j) * d_gamao2 + i * d_gama);
            eta[i][j] =cord * sin(((num_pts - i) + j) * d_gamao2 + i * d_gama);
            if(i<num_pts/2 && j<num_pts/2)
                {
                ii=j + num_pts/2;
                jj=i + num_pts/2;
                avg_correl[i][j] = (correl[i][j] + correl[ii][jj])/2;
                }
            else if(i<num_pts/2 && j>=num_pts/2)
                {
                ii=j - num_pts/2;
                jj=i + num_pts/2;
                avg_correl[i][j] = (correl[i][j] + correl[ii][jj])/2;
                }
            else if(i>=num_pts/2 && j<num_pts/2)
                {
                ii=j + num_pts/2;
                jj=i - num_pts/2;
                avg_correl[i][j] = (correl[i][j] + correl[ii][jj])/2;
                }
```

```
            else
                {
                ii=j - num_pts/2;
                jj=i - num_pts/2;
                avg_correl[i][j] = (correl[i][j] + correl[ii][jj])/2;
                }
            }

        for(j=i;j<num_pts;j++)
            {
            cord = 2*radius*sin(abs((i-j)) * d_gamao2);
            xi[i][j] = cord * cos((j - i) * d_gamao2 + i * d_gama);
            eta[i][j] =cord * sin((j - i) * d_gamao2 + i * d_gama);
            if(i<num_pts/2 && j<num_pts/2)
                {
                ii=j + num_pts/2;
                jj=i + num_pts/2;
                avg_correl[i][j] = (correl[i][j] + correl[ii][jj])/2;
                }
            else if(i<num_pts/2 && j>=num_pts/2)
                {
                ii=j - num_pts/2;
                jj=i + num_pts/2;
                avg_correl[i][j] = (correl[i][j] + correl[ii][jj])/2;
                }
            else if(i>=num_pts/2 && j<num_pts/2)
                {
                ii=j + num_pts/2;
                jj=i - num_pts/2;
                avg_correl[i][j] = (correl[i][j] + correl[ii][jj])/2;
                }
            else
                {
                ii=j - num_pts/2;
                jj=i - num_pts/2;
                avg_correl[i][j] = (correl[i][j] + correl[ii][jj])/2;
                }
            }
        xi[i][i] = 0;
        eta[i][i] = 0;

/** Calculate the 00, 11, 22, 33, 44, ... repeated points **/
        avg_correl[i][i]=0.0;
        for(j=0;j<num_pts;j++)
            {
            avg_correl[i][i]=(avg_correl[i][i] + correl[j][j]);
            }
        avg_correl[i][i]=avg_correl[i][i]/num_pts;
        for(j=0;j<num_pts;j++)
            {
            sum_correl[i][j]=(sum_correl[i][j] + avg_correl[i][j]);
            }
    }  /**** End of ns for loop  ***/
```

```
/****** Turn the sum of scans into average of scans ****/

}
for(i=0;i<num_pts;i++)
    {
    for(j=0;j<num_pts;j++)
        {
        sum_correl[i][j] = sum_correl[i][j]/(scan - 1);
        }
    }

/*************************************************************************/
/*** Print output file for the H.S. Olmez Solver***/
/*************************************************************************/

if((f8=fopen("foo.hso","w"))==NULL)
    {
    printf("Cannot open file\n");
    exit(1);
    }

/** Print the 0,0 case  ***/
fprintf(f8,"    %10.6f  %10.6f  %10.6f\n\n", xi[0][0], eta[0][0], sum_correl[0][0]);

for(i=0;i<num_pts;i++)
    {
    if(i<num_pts/2)
     {
      for(j=(i+num_pts/2);j<num_pts;j++)
        {
        fprintf(f8,"    %10.6f  %10.6f  %10.6f",  xi[i][j], eta[i][j], sum_correl[i][j]);
        }
      for(j=0;j<i;j++)
        {
         fprintf(f8,"    %10.6f  %10.6f  %10.6f",  xi[i][j], eta[i][j], sum_correl[i][j]);
        }
     }
    if(i>=num_pts/2)
     {
      for(j=(i-num_pts/2);j<i;j++)
        {
         fprintf(f8,"    %10.6f  %10.6f  %10.6f",  xi[i][j], eta[i][j], sum_correl[i][j]);
        }
     }

    fprintf(f8,"\n");
    }
    fclose(f8);

/*************************************************************************/
          /** Print work file for jmass  **/
/*************************************************************************/

if((f8=fopen("correl.mas","w"))==NULL)
```

```
    {
    printf("Cannot open file\n");
    exit(1);
    }

/** Print the 0,0 case  ***/

fprintf(f8,"    %10.6f  %10.6f  %10.6f\n", xi[0][0], eta[0][0], sum_correl[0][0]);

/** Print the rest of the cases **/

for(i=0;i<num_pts;i++)
    {
    if(i<num_pts/2)
     {
      for(j=(i+num_pts/2);j<num_pts;j++)
        {
        fprintf(f8,"    %10.6f  %10.6f  %10.6f\n",  xi[i][j], eta[i][j], sum_correl[i][j]);
        }
      for(j=0;j<i;j++)
        {
        fprintf(f8,"    %10.6f  %10.6f  %10.6f\n",  xi[i][j], eta[i][j], sum_correl[i][j]);
        }
     }

    if(i>=num_pts/2)
     {
      for(j=(i-num_pts/2);j<i;j++)
        {
        fprintf(f8,"    %10.6f  %10.6f  %10.6f\n",  xi[i][j], eta[i][j], sum_correl[i][j]);
        }
     }
    }
x=0.0;
y=0.0;

for(i=0;i<=num_zero;i++)
    {
    for(j=0;j<=num_zero;j++)
     {
     x=i*2*radius/num_zero;
     y=j*2*radius/num_zero;
     if((2*radius*2*radius) < (x*x + y*y))
        {
        fprintf(f8,"    %10.6f  %10.6f  %10.6f\n",  x, y, 0.0);
        fprintf(f8,"    %10.6f  %10.6f  %10.6f\n",  x, - y, 0.0);
        fprintf(f8,"    %10.6f  %10.6f  %10.6f\n",  - x, y, 0.0);
        fprintf(f8,"    %10.6f  %10.6f  %10.6f\n",  - x, - y, 0.0);
        }
     }
    }
    fclose(f8);
```

```
/*****************************************************************************/
/*** Print output file for tecplot ***/
/** File converts autocorrelation data into i x j matrix data for **/
/** TECPLOT graphs. The data is printed in the format PREPLOT requires **/
/** Data is converted by stretching the center point of the circle into a line **/
/** and using double points **/
/*****************************************************************************/

if((f4=fopen("correl.tec","w"))==NULL)
    {
    printf("Cannot open file\n");
    exit(1);
    }

fprintf(f4,"TITLE = \"Autocorrelation for %d Scans\"\n",scan-1);
fprintf(f4,"VARIABLES = X, Y, Autocorrelation\n");
fprintf(f4,"ZONE T =\"Zone One\", I=%.2d, J=%.2d, F=POINT\n", num_pts + 2, num_pts + 1);

j=0;
p=0;

/**  first set of numbers  **/

for(i=num_pts/2;i>0;i--)
    {
    if(i>j)
     {
      fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[i][j], eta[i][j], sum_correl[i][j]);
      fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[i][j], eta[i][j], sum_correl[i][j]);
      j++;
      p++;
     }
    else if(p=num_pts/4)
     {
      fprintf(f4,"    %10.6f  %10.6f  %10.6f\n", xi[0][0], eta[0][0], sum_correl[0][0]);
      fprintf(f4,"    %10.6f  %10.6f  %10.6f\n", xi[0][0], eta[0][0], sum_correl[0][0]);
      i=0;
     }
    }

  j=num_pts/4 - 1;        /** correct the j count **/
  for(i=0;i<=num_pts/2;i++)
    {
    if(i>num_pts/4)
     {
      fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[i][j], eta[i][j], sum_correl[i][j]);
      fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[i][j], eta[i][j], sum_correl[i][j]);
      j--;
     }
    }

/** second set  **/

ii=num_pts/2;
```

```
jj=0;
iii=num_pts/4;
jjj=num_pts/4;

p=0;
n=0;
o=0;

for(m=(num_pts - 1);m>0;m--)
    {
    if(p<num_pts/4)
        {
        j=jj;
        o=jjj;
        for(i=ii-1;p<num_pts/4;i--)
            {
            if(i<0 && j<0)
                {
                fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[num_pts + i][num_pts + j], eta[num_pts +
i][num_pts + j], sum_correl[num_pts + i][num_pts + j]);
                fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[num_pts + i][num_pts + j], eta[num_pts +
i][num_pts + j], sum_correl[num_pts + i][num_pts + j]);
                }
            else if(i>=0 && j<0)
                {
                fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[i][num_pts + j], eta[i][num_pts + j],
sum_correl[i][num_pts + j]);
                fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[i][num_pts + j], eta[i][num_pts + j],
sum_correl[i][num_pts + j]);
                }
            else if(i<0 && j>=0)
                {
                fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[num_pts + i][j], eta[num_pts + i][j],
sum_correl[num_pts + i][j]);
                fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[num_pts + i][j], eta[num_pts + i][j],
sum_correl[num_pts + i][j]);
                }
            else
                {
                fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[i][j], eta[i][j], sum_correl[i][j]);
                fprintf(f4,"    %10.6f  %10.6f  %10.6f\n",  xi[i][j], eta[i][j], sum_correl[i][j]);
                }
            j++;
            p++;
            }

        fprintf(f4,"    %10.6f  %10.6f  %10.6f\n", xi[0][0], eta[0][0], sum_correl[0][0]);
        fprintf(f4,"    %10.6f  %10.6f  %10.6f\n", xi[0][0], eta[0][0], sum_correl[0][0]);

        for(n=iii+1;r<num_pts/4;n++)
            {
            if(n<0 && o<0)
                {
```

```
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + n][num_pts + o], eta[num_pts +
n][num_pts + o], sum_correl[num_pts + n][num_pts + o]);
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + n][num_pts + o], eta[num_pts +
n][num_pts + o], sum_correl[num_pts + n][num_pts + o]);
                }
            else if(n>=0 && o<0)
                {
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[n][num_pts + o], eta[n][num_pts + o],
sum_correl[n][num_pts + o]);
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[n][num_pts + o], eta[n][num_pts + o],
sum_correl[n][num_pts + o]);
                }
            else if(n<0 && o>=0)
                {
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + n][o], eta[num_pts + n][o],
sum_correl[num_pts + n][o]);
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + n][o], eta[num_pts + n][o],
sum_correl[num_pts + n][o]);
                }
            else
                {
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[n][o], eta[n][o], sum_correl[n][o]);
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[n][o], eta[n][o], sum_correl[n][o]);
                }
            o--;
            r++;
            }
        jj--;
        iii++;
        }
    else if(p==num_pts/4)
        {
        j=jj;
        o=jjj;
        for(i=ii - 1;p<num_pts/2;i--)
            {
            if(i<0 && j<0)
                {
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + i][num_pts + j], eta[num_pts +
i][num_pts + j], sum_correl[num_pts + i][num_pts + j]);
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + i][num_pts + j], eta[num_pts +
i][num_pts + j], sum_correl[num_pts + i][num_pts + j]);
                }
            else if(i>=0 && j<0)
                {
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[i][num_pts + j], eta[i][num_pts + j],
sum_correl[i][num_pts + j]);
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[i][num_pts + j], eta[i][num_pts + j],
sum_correl[i][num_pts + j]);
                }
            else if(i<0 && j>=0)
                {
                fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + i][j], eta[num_pts + i][j],
sum_correl[num_pts + i][j]);
```

```
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + i][j], eta[num_pts + i][j],
sum_correl[num_pts + i][j]);
            }
        else
            {
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[i][j], eta[i][j], sum_correl[i][j]);
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[i][j], eta[i][j], sum_correl[i][j]);
            }
        j++;
        p++;
        }

    fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[0][0], eta[0][0], sum_correl[0][0]);
    fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[0][0], eta[0][0], sum_correl[0][0]);

    for(n=iii+1;r<num_pts/2;n++)
        {
        if(n<0 && o<0)
            {
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + n][num_pts + o], eta[num_pts +
n][num_pts + o], sum_correl[num_pts + n][num_pts + o]);
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + n][num_pts + o], eta[num_pts +
n][num_pts + o], sum_correl[num_pts + n][num_pts + o]);
            }
        else if(n>=0 && o<0)
            {
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[n][num_pts + o], eta[n][num_pts + o],
sum_correl[n][num_pts + o]);
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[n][num_pts + o], eta[n][num_pts + o],
sum_correl[n][num_pts + o]);
            }
        else if(n<0 && o>=0)
            {
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + n][o], eta[num_pts + n][o],
sum_correl[num_pts + n][o]);
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[num_pts + n][o], eta[num_pts + n][o],
sum_correl[num_pts + n][o]);
            }
        else
            {
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[n][o], eta[n][o], sum_correl[n][o]);
            fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[n][o], eta[n][o], sum_correl[n][o]);
            }
        o--;
        r++;
        }
    p=0;
    r=0;
    ii--;
    jjj++;
    }
}

q=0;
```

```
j=num_pts/2;
p=0;

/** last set of numbers **/

for(i=0;i<num_pts/2;i++)
    {
    if(i<j)
     {
      fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[i][j], eta[i][j], sum_correl[i][j]);
      fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[i][j], eta[i][j], sum_correl[i][j]);
      j--;
      p++;
     }
    else if(p=num_pts/4)
     {
      fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[0][0], eta[0][0], sum_correl[0][0]);
      fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[0][0], eta[0][0], sum_correl[0][0]);
      i=num_pts/2;
     }
    }

  j=num_pts/4 + 1;       /** correct the j count **/

  for(i=num_pts/4 - 1;i>=0;i--)
    {
    fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[i][j], eta[i][j], sum_correl[i][j]);
    fprintf(f4,"    %10.6f %10.6f %10.6f\n", xi[i][j], eta[i][j], sum_correl[i][j]);
    j++;
    }

fclose(f4);
}
```

APPENDIX F: dfsum13.for Program

```
      program dfsum13
      parameter(m=4893,n=17)

C  This program calculates the directional spectrum given the autocorrelation output from sw_auto*.c in
c  the file correl.mas.  It also recreates the autocorrelation which can be compared to the autocorrelation
c  as generated by the sw_auto*.c program.  I have also included a section which generates output of the
c  form necessary to create a dB by log scale of k with relation to wind direction so it can be compared to
c  data collected by H. Yuen as presented in ONR Ship Wake Detection Program Review Meeting
c  September 1991.
C
C*****************************************************************
C     n must be odd for correct count a(k,ip)
C*****************************************************************
C     2317=2049+268 (268 zeroes)
C     m : # of points on the surface
C     n : # of points for the Fourier sum

      dimension sol(n*(2*n-1)+3),delta(m)
      dimension a(m+2,(n*(2*n-1)+4)),skx(n),sky(2*n-1)
      dimension rn(m),x(m),y(m),r(m), ang(110)
      dimension spp(n,n*(2*n-1)),il(n*(2*n-1)+3)
      dimension aa(m+2,(n*(2*n-1)+4)), spout(50), ptspec(65)

      real lx, ly, newr(m), rr, kkx, kky, sol, spsum
      real den1, den2, kxy2, ptspec, skx, sky, deltang, ang
      real avgfrst, avg, angle, anginc, ptsum
      integer w, count, spout, numang, cnt

      data pi /3.141592654/

C     DIAMETER OF WIRE GAGE
      diameter = 20.

C     RADIUS OF AUTOCORRELATION SPACE
      radius = diameter

      lx=diameter
      ly=diameter

      open(10,file='correl.mas',status='old')
      open(20,file='z.tec',status='unknown')
      open(30,file='s.tec',status='unknown')
      open(60,file='check.out',status='unknown')
      open(90,file='anglspec.out',status='unknown')
      open(100,file='sk.tec',status='unknown')
      open(110,file='ptspec.out',status='unknown')
      open(500,file='logs.tec',status='unknown')

      do 1 i=1,m
```

```
      read(10,*) x(i),y(i),r(i)
      if(x(i).eq.0.0 .and. y(i).eq.0.0) r0=r(i)

C*************************************************************
C    convolve the auto correlation with a smoothing window
C*************************************************************
      wn =  (sqrt(x(i)*x(i) + y(i)*y(i)))/radius

      if(wn.le.1.0) then
        rn(i)=r(i)*(2./pi)*(acos(wn) - wn*sqrt(1. - (wn*wn)))
        if(wn.eq.(0.0)) write(6,*) 'a(0,0) = ',rn(i)
      else
        rn(i) = 0.0
      end if
 1    continue

     do 30 k=1,m
       a(k,1)=1.
       do 31 j=2,n
 31       a(k,j) = 2.*cos((j-1)*pi*y(k)/ly)
       ip=n
       do 20 i=1,n-1
         do 10 j=-(n-1),(n-1)
           ip=ip+1
           a(k,ip)=2.*cos((i)*pi*x(k)/lx)*cos((j)*pi*y(k)/ly)-
     &             2.*sin((i)*pi*x(k)/lx)*sin((j)*pi*y(k)/ly)
      aa(k,ip) = a(k,ip)
 10       continue
 20    continue
       a(k,ip+1)=rn(k)
 30  continue
       write(6,*)'ip =',ip

C  **Note:  ip = nsq = number of unknowns
     nsq=2*n*(n-1)+1
     write(*,*) '# of unknowns: ',nsq
     call glsqns(a,sol,il,m,ip,alpha,0.000000001,0.000000001)

C  Check matrix solver solution and compare with autocor. coefs.

     do 45 k=1,m
       newr(k) = 0.
       do 40 i=1,nsq
         newr(k) = newr(k) + aa(k,i)*sol(i)
 40    continue
         delta(k) = newr(k) - rn(k)
         write(60,*) newr(k), delta(k)
 45  continue

c*************************************************************
c    Regenerate the autocorrelation from fourier coefficients
c
c    For regeneration to work producing equally spaced point the original autocorrelation
c    file must be padded with zeros in the corners
```

```
c*****************************************************************

    write(20,*) 'TITLE = "Reconstructed Autocorrelation"'
    write(20,*) 'VARIABLES = X, Y, Autocorrelation'
    write(20,*) 'ZONE T = "Zone One", I=64, J=64, F=POINT'

    do 70 ix=1,64
     xn=-lx+((ix-0.5)*2.0*lx/64.0)
       do 65 jy=1,64
        ip=0
        sum=0.0
        yn=-ly+((jy-0.5)*2.0*ly/64.0)
        sum=sol(1)
         do 50 j=2,n
50          sum=sum+2.*sol(j)*cos((j-1)*pi*yn/ly)
         ip = n
         do 60 i=1,n-1
           do 55 j=-(n-1),(n-1)
            ip=ip+1
            sum=sum+(sol(ip)*
   &           (cos((i)*pi*xn/lx)*cos((j)*pi*yn/ly)-
   &            sin((i)*pi*xn/lx)*sin((j)*pi*yn/ly)))

55         continue
60       continue
        write(20,*) xn,yn,sum

65     continue
70   continue

c*****************************************************************
c Calculate the spectra from the solution matrix sol()
c*****************************************************************

    write(30,*) 'TITLE = "Spectral Energy"'
    write(30,*) 'VARIABLES = X, Y, S'
    write(30,*) 'ZONE T = "Zone One", I=33, J=17, F=POINT'

    write(100,*) 'TITLE = "Spectral Energy"'
    write(100,*) 'VARIABLES = X, Y, S*K^2'
    write(100,*) 'ZONE T = "Zone One", I=33, J=17, F=POINT'

    write(500,*) 'TITLE = "Spectral Energy"'
    write(500,*) 'VARIABLES = X, Y, log(S)'
    write(500,*) 'ZONE T = "Zone One", I=33, J=17, F=POINT'

    ip=0
    sum=0.0
    fac=pi*pi/(lx*ly)

    do 90 i=1,n
      do 80 j=1,(2*n-1)
        if (i.eq.1) then
          skx(i) = 0.
```

```
          if (j.lt.n) then
            sky(j) = (-n+j)*pi/ly
            ind = n-j+1
          else if (j.gt.n) then
            sky(j) = (-n+j)*pi/ly
            ind = j-n+1
          else
            sky(j) = 0.
            ind = 1
          endif
        else
          ind = (2*n-1)*(i-2) +j +n
          skx(i)=(i-1)*pi/lx
          sky(j)=(-(n-1)+j-1)*pi/ly
        endif
c  Double except on axes because plot only has half of total dir. spec.
        if (i.ne.1) then
          spp(i,j)=(2.0*sol(ind)/fac)
          if (spp(i,j).gt.(0.0)) then
            sum = sum+spp(i,j)
          endif
          sq=2.0*sol(ind)
        else
          spp(i,j)=sol(ind)/fac
          if (spp(i,j).gt.(0.0)) then
            sum=sum+spp(i,j)
          endif
          sq=sol(ind)
        endif
C  Calculate magnitude of k squared
        kxy2 = (skx(i)**2.)+(sky(j)**2.)
        write(30,*) skx(i),sky(j),sq/fac
          write(100,*) skx(i),sky(j),sq/fac*kxy2
          if ((sq/fac).gt.(0.00001)) then
            write(500,*) skx(i),sky(j),alog10(sq/fac)
          else
            sq = -8.0
            write(500,*) skx(i),sky(j), sq
          endif
 80     continue
 90   continue

c The actual integral under the spectrum will be the sum of the
c spectral components times the kx and ky spacing (=fac)
     write(*,*) 'r0, totsum, scaled sum', r0,sum,sum*fac

*****************************************************************
*       Calculate Spectrum at various kx and ky                 *
*****************************************************************

     write (6,*) 'Enter wind direction in degrees relative to N'
     write (6,*) 'where    -180ø < wind <= 180ø     :'
     read(6,*) w
```

```
c I'm dividing up angles such that I can easily average over 15 degrees
c surrounding the angles I'm interested in which are every 15 degrees
c starting at 0 and running through 165 degrees.

      deltang = 180./108.
      numang = 109
      ang(1) = 0
      do 153 i = 2,numang
        ang(i) = ang(i-1) + deltang
153   continue

      ptsum = 0.0
      do 180 k = 2,60
        ptspec(k) = 0.0
c rr is the k value of interest
        rr = k/20.
c count is a counter for the angle spectra output loop below
        count = 1
        avgfrst = 0.0
        avg = 0.0
        cnt = 0

c Calculate spectral sum for i=j=0 (see wave gage notes)
        do 175 j = 1,(numang-1)
          kkx = rr*sin(ang(j)*pi/180.)
          kky = rr*cos(ang(j)*pi/180.)
          if (abs(kkx).gt.(0.005).and.abs(kky).gt.(0.005)) then
           spsum = sol(1)/pi/pi*sin(-lx*kkx)*sin(-ly*kky)/(kkx*kky)
          elseif (abs(kkx).le.(0.005).and.abs(kky).gt.(0.005)) then
            spsum = sol(1)/pi/pi*(-lx)*sin(-ly*kky)/kky
          elseif (abs(kky).le.(0.005).and.abs(kkx).gt.(0.005)) then
            spsum = sol(1)/pi/pi*(-ly)*sin(-lx*kkx)/kkx
          else
            spsum = sol(1)/pi/pi*lx*ly
          endif

c Calculate spectral sum for i=0
          do 155 jj = 1,n-1
            if (abs(kkx).gt.(0.005)) then
              spsum = spsum + 2/pi/pi*sol(jj)*lx*ly*
     +         sin(-lx*kkx)*sin(jj*pi-ly*kky)/
     +         (-lx*kkx*(jj*pi-ly*kky))
            else
              spsum = spsum + 2/pi/pi*sol(jj)*lx*ly*
     +         sin(jj*pi-ly*kky)/(jj*pi-ly*kky)
            endif
155       continue
          ind = n

c Calculate spectral sum for all other 0<i<n-1 and -(n-1)<j<n-1
          do 165 ii = 1,n-1
            do 160 kk = -(n-1),n-1
              den1 = kk*pi-ly*kky
              den2 = ii*pi-lx*kkx
```

```
            ind = ind + 1
          if (abs(den1).le.(0.005).and.abs(den2).gt.(0.005)) then
              spsum=spsum+2/pi/pi*sol(ind)*lx*ly*
     +          sin(den2)/den2
        elseif (abs(den2).le.(0.005).and.abs(den1).gt.(0.005)) then
              spsum=spsum+2/pi/pi*sol(ind)*lx*ly*
     +          sin(den1)/den1
        elseif (abs(den1).le.(0.005).and.abs(den2).le.(0.005)) then
              spsum=spsum+2/pi/pi*sol(ind)*lx*ly
            else
              spsum=spsum+2/pi/pi*sol(ind)*lx*ly*
     +          sin(den2)*sin(den1)/(den2*den1)
            endif
  160       continue
  165     continue
```

C Write spectrum along specific angles to file taking into account the incident angle of the wind

```
        if (j.le.5.or.j.ge.105) then
          avgfrst = avgfrst + spsum
        else
          avg = avg + spsum
          cnt = cnt + 1
        endif

        if (cnt.eq.9) then
          if (avg.le.(0.0)) then
            spout(count) = -300
          else
            spout(count) = 10*alog10(avg/9./(10**4))
          endif
          count = count + 1
          avg = 0.0
          cnt = 0
        endif

        if (j.eq.(numang-1)) then
          if (avgfrst.le.(0.0)) then
            spout(count) = -300
          else
            spout(count) = 10*alog10(avgfrst/9./(10**4))
          endif
          count = count + 1
          cnt = 0
        endif


        ptspec(k) = ptspec(k) + spsum

  175   continue
```

c Calculate point spectra from (ptspec sum) * k * d-theta. Note that point spectra is the integral over the
c total spectrum because the spectral sum was doubled in the above calcs

```
      ptspec(k) = ptspec(k)*rr*(deltang/180.0*pi)
      write(110,*) rr, ptspec(k)
      ptsum = ptsum + ptspec(k)

      write(90,'(F6.2,29F10.4)') rr, (spout(ijk), ijk=1,12)

180  continue
     ptsum = ptsum/20
     write(6,*) 'Wave number spectral integral = ',ptsum

     anginc = 15.
     do 184 kk=1,12
       if (w.le.0.0) then
         angle = anginc*kk - (w + 180.)
         write(90,*) angle
       else
         angle = anginc*kk - w
         write(90,*) angle
       endif
184  continue

     stop
     end


C**********************End of Program****************************
C
C          Beginning of Matrix Solver Subroutine
C
C  Let matrix be of the form MX = B
C  A = coefficient matrix ([M:B])
C  X = solution matrix
C  IL = workspace matrix
C  N = number of equations
C  M = number of unknowns
C  ALPHA = sum of squares of errors
C  E1 & E2 = convergence limits
C
C **NOTE** :  Matricies in subroutine must be same dimensioned
C          the same as those in the main program. (obvious, I know)

     SUBROUTINE GLSQNS(A,X,IL,N,M,ALPHA,E1,E2)
     parameter(md=4893,nd=17)
     DIMENSION A(md+2,(nd*(2*nd-1)+4)),X(1),
    1 IL(1)
     write(*,*)
     MM=M+1
     LL=1
     DO 60 J=1,MM
60    IL(J)=0
     I=1
     DO 3 K=1,MM
       write (*,2) K,MM
2      format('+','Loop # ',i4,' of ',i4)
```

```
      II=I+1
      DO 4 J=II,N
       IF (ABS(A(J,K))-E1) 4,4,6
6      T1=SQRT((A(J,K))**2+(A(I,K))**2)
       S=A(J,K)/T1
       C=A(I,K)/T1
       DO 5 L=K,MM
        T2=C*A(I,L)+S*A(J,L)
        A(J,L)=-S*A(I,L)+C*A(J,L)
5      A(I,L)=T2
       LL=LL+1
4     CONTINUE
      IF (ABS(A(I,K))-E2) 3,3,8
8     IL(K)=I
      I=I+1
3    CONTINUE
     X(MM)=-1.0
     II=M
     DO 35 I=1,M
35    X(I)=0.0
     write(*,*) 'Creating solution vector'
     DO 30 J=1,M
      IF (IL(II)) 30,30,31
31     S=0.0
      LL=II+1
      I=IL(II)
      DO 32 K=LL,MM
32     S=S+A(I,K)*X(K)
      X(II)=-S/A(I,II)
30    II=II-1
     IF (IL(MM)) 50,51,50
51    ALPHA=0.0
     GO TO 52
50    I=IL(MM)
     ALPHA=A(I,MM)
52    RETURN
     END
```

APPENDIX G: swsim_n2.c Program

```
/**                SWSIM_n2.C                          **/
/** Circular wave scan feature added to SWSIM.C code obtained from Jerry Milgram, 16 Jan 93,  **/
/** simulates sea waves. Modified by Nicole Suoja in April 1996 to allow various k   **/
/** and angle values to be used.                        **/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

float brs(float qk, float theta, float us);
float filter(float om);

float  ds[20][40],kx[20][40],ky[20][40],wa[20][40];
float  kxp[20][40],kyp[20][40],rphase[20][40];
float  theta[40],k[26],outght[100][3],outsx[100],outsy[100];
float  om[20],t[514],x[514],y[514],dst[514],r[20];

float  dx[100], dy[100];
float  phi[100];

void main(int argc, char *argv[])
{

FILE *f3,*f6,*f7,*f8,*f9;

int na, nk, nc, np, ns, n, num_pts, i, j, jout;
int decimal, sign, dd;

float zeta, dia, radius, delta_ang, angle, cord;
float us=24.0, g=981.0, st=72.0, rho=1.02, pi=3.1415926535;
float rmin= - 1.510642358602, rmax= 0.79818, tor, tpi, pio2, dtheta, dr, b;
float length, startt, cs, sn, dt, dl, qln, phase, val;
float outghtsum, outghtave;
float gain, gain_db, hs, foo1, foo2;

double dro2, dval, ten=10.0, nn;

/*************************************************************************/
/**  # angle segments=na, # k segments=nk, # wave components nc=na*nk  **/
/**  us=friction velocity of the wind                    **/
/**  CGS UNITS                                           **/
/**  rmin and rmax are log(base_10) of kmin and of kmax          **/
/*************************************************************************/

char fname[18];
char *buffer;
                /** seed random # generator **/
```

```
srand((unsigned)time(NULL));


/**printf("Enter the Diameter of the Circle <cm> :  ");
    scanf("%f",&dia);  **/
dia = 20.0;

/**printf("Enter the Number of Wave Wires used on the Circle:  ");
    scanf("%d",&num_pts);  **/
num_pts = 64;

/**printf("Enter the Start Delay Time:  ");
    scanf("%f",&startt);  **/
startt = .1;

/**printf("Enter the Scan Time: ");
    scanf("%f",&st);  **/
st =  0.005;

/**printf("Enter the Number of K values: ");
    scanf("%d",&nk);  **/

/**printf("Enter the Number of Theta's: ");
    scanf("%d",&na);  **/

printf("Enter the Number of Scans you want written to output files:  ");
    scanf("%d",&ns);

if(ns==0)
    {
    printf("You Have Choosen 0 Output.");
    exit(1);
    }

    if((f9=fopen("ks.in","r"))==NULL)
        {
        printf("Cannot open file\n");
        exit(1);
        }

/** Read input file containing #k values and a listing of the k's and the number of angles desired **/

    fscanf(f9,"%d%d",&nk,&na);
    for(i=1;i<=nk;++i)
      {
      fscanf(f9,"%f",&foo1);
      k[i] = foo1;
      }
    fclose(f9);
    printf("nk = %d\n",nk);
    printf("na = %d\n",na);
```

```
            /** Calculation of circular wave measuring **/
            /** device characteristics              **/

nc=na*nk;
tor=st/rho;
tpi=2.0*pi;
pio2=pi/2.0;
dtheta=pi/na;
dr=(rmax - rmin)/nk;
dro2=dr/2.0;
b=pow(ten,dro2) - pow(ten,-dro2);
delta_ang=tpi/num_pts;
radius=(float)dia/2.0;
zeta=(delta_ang/2.0);
angle=zeta;
cord=2.0*radius*sin(zeta);

dt=st/(num_pts);

printf("\nDia =       %8.3f cm\nnum_pts =     %d\ndelta_ang =    %8.3f deg\nzeta =        %8.3f \ncord
=       %8.3f cm\n\n",dia,num_pts,(delta_ang/tpi)*360,(zeta/tpi)*360,cord);

x[1]=0.0;
y[1]=0.0;
phi[1] = 0.0;
t[1]=startt;
dx[0]=0.0;
dy[0]=0.0;

for(i=2;i<=num_pts;++i)
    {
    t[i]=dt*i + startt;
    dx[i]=cord*cos(angle);
    dy[i]=cord*sin(angle);
    phi[i]=angle;
    x[i]=x[i - 1] + cord*cos(angle);
    y[i]=y[i - 1] + cord*sin(angle);
    angle=angle + delta_ang;
    }

np=na + nk;

/*****************************************************************/
/*************** Calculate the K #'s and angles of attack *********/
/*****************************************************************/

      for(i=1;i<=na;++i)

        {
        theta[i]= - pio2 + 0.5*dtheta;
        printf("i= %d,  theta= %f  dtheta= %f\n",i,theta[i], dtheta);
        }

    if(nk== 1)
```

```
    {
      dval=g*k[1] + tor*k[1]*k[1]*k[1];
      om[1]=sqrt(dval);
      printf("K value = %10.8f  wavelength (cm) = %10.8f\n",
      k[1], (6.2831853071959/k[1]));
    }

  else
    {
     for(i=1;i<=nk;++i)
      {
      r[i]=rmin + 0.5*dr + (i - 1)*dr;
      dval=g*k[i] + tor*k[i]*k[i]*k[i];
      om[i]=sqrt(dval);
      printf("K value = %10.8f  wavelength (cm) = %10.8f\n", k[i],
         (6.2831853071959/k[i]));
      }
    }

printf("\n\n");

/*****************************************************************/
/************** begin generating output files ******************/
/*****************************************************************/

  if((f3=fopen("autoin.dat","w"))==NULL)
      {
      printf("Cannot open file\n");
      exit(1);
      }

  rphase[1][1] = 0.0;

for(n=1;n<=ns;++n)
  {
  nn = (float) n;
  dd=0;

  printf("Currently Calculating Number: %.2d of %.2d\r",n, ns);

  for(i=1;i<=nk;++i)
      {
       gain = 0.10*0.8;
      for(j=1;j<=na;++j)
          {
          kx[i][j]=k[i]*cos(theta[i]);
          ky[i][j]=k[i]*sin(theta[i]);
          kxp[i][j]=kx[i][j]*cos(phi[i]) + ky[i][j]*sin(phi[i]);
          kyp[i][j]= - kx[i][j]*sin(phi[i]) + ky[i][j]*cos(phi[i]);
          ds[i][j]=1.0;
          wa[i][j]=gain*sqrt(2.0*ds[i][j]*b*dtheta);
          rphase[i][j] = rphase[i][j]+om[i]/32.0;
          }
      }
```

```
    for(jout=1;jout<=num_pts;++jout)
        {
        outght[jout][1]=0.0;
        outsx[jout]=0.0;
        outsy[jout]=0.0;
        for(i=1;i<=nk;++i)
            {

            for(j=1;j<=na;++j)
                {
                phase=kx[i][j]*x[jout] + ky[i][j]*y[jout] - om[i]*t[jout]+rphase[i][j];
                outght[jout][1]=outght[jout][1] + wa[i][j]*cos(phase);
                outsx[jout]=outsy[jout] - wa[i][j]*kxp[i][j]*sin(phase);
                outsy[jout]=outsy[jout] - wa[i][j]*kyp[i][j]*sin(phase);
                }
            }
        }
            /** correct the outght to show measured height **/

    outghtsum=0.0;
    for(jout=1;jout<=num_pts;++jout)
        outghtsum = outghtsum + outght[jout][1];
        outghtavc=outghtsum/num_pts;
    for(jout=1;jout<=num_pts;++jout)
        outght[jout][2]=outght[jout][1] - outghtave;

      /*************************************************/
      /**** outght [jout][1] ==> uncorrected outght *****/
      /**** outght [jout][2] ==> corrected outght  *****/
      /*************************************************/

    for(jout=1;jout<=num_pts;++jout) fprintf(f3,"%12.8f\t",outght[jout][1]);
       fprintf(f3,"\n");
    }
if((f8=fopen("swsim.dat","w"))==NULL)
    {
    printf("Cannot open file\n");
    exit(1);
    }

    fprintf(f8,"\ntheta  \t %5.3f  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f
%5.3f", theta[1], theta[2], theta[3], theta[4], theta[5], theta[6], theta[7], theta[8], theta[9], theta[10]);
    fprintf(f8,"\n\t  K #");
    for(i=20;i<=nk;++i)
       fprintf(f8,"\n \t%8.6f  %8.6f  %8.6f  %8.6f  %8.6f  %8.6f  %8.6f  %8.6f  %8.6f  %8.6f
%8.6f", k[i], ds[i][1], ds[i][2], ds[i][3], ds[i][4], ds[i][5], ds[i][6], ds[i][7], ds[i][8], ds[i][9], ds[i][10]);

    fclose(f3);
    fclose(f8);
printf("\n\n");

}
```

APPENDIX H: Laser Slope Gage Test Setup and Calibration

The test setup in the wave tank included one laser slope gage located at the same longitudinal position in the wave tank as the center of the wave gage. It was a reflective laser slope gage which measured high frequency waves between about 4 and 15 Hz. Figure H.1 shows how the laser slope gage was located in the wave tank in comparison to the location of the circular wave gage. This setup was used for all of the resolution and accuracy tests described in Chapter 4.

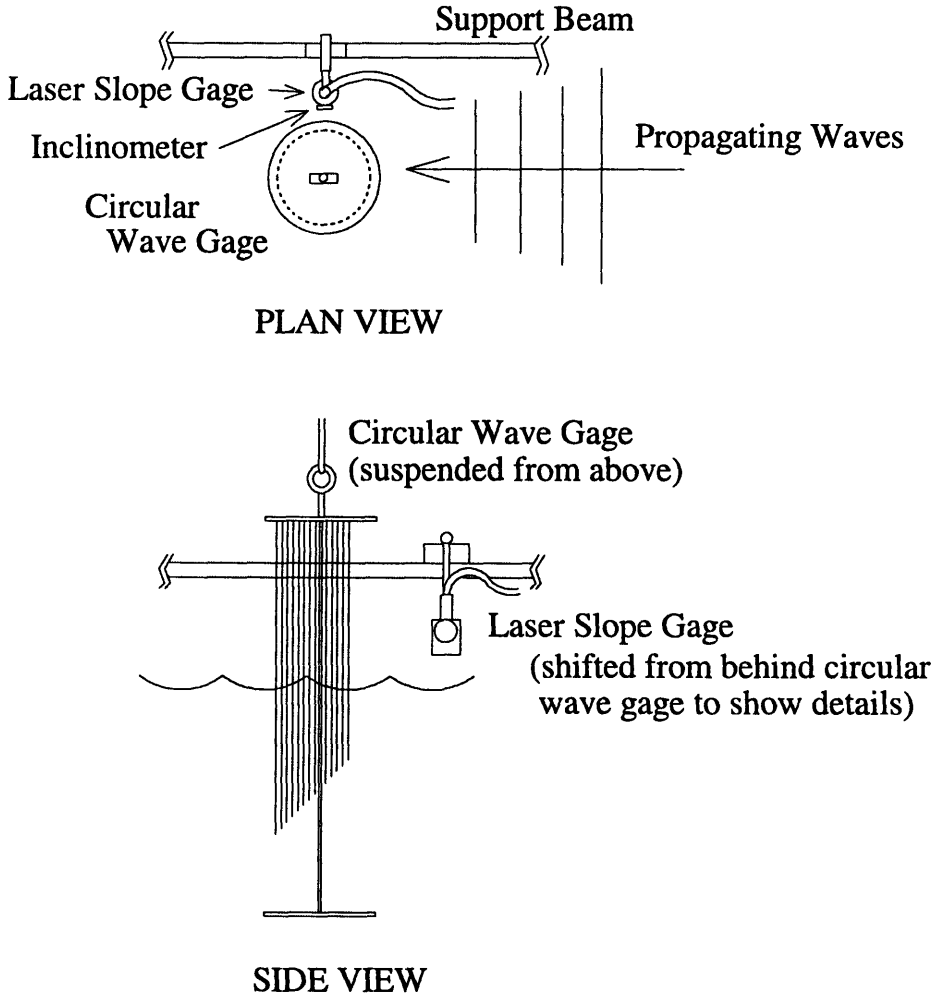

FIGURE H.1: Test Setup with Reflective Laser Slope Gage and Circular Wave Gage

The reflective laser slope gage generated a laser source above the water surface, reflected it off the surface, and measured it using a photo detector which hung above the surface. The laser source and the photo detector were part of the same physical unit. The detector generated an output voltage proportional to the location of the laser beam along its length.

The voltage versus time output signal was calibrated to produce a wave slope versus time output. The calibration of this instrument was obtained by rotating the laser-source / photo-detector unit and measuring the output voltage of the photo detector as a function of angle of rotation. The measured photo detector voltage was then multiplied by the slope of the calibration data which had units of angle of rotation per volt to give a final output of angle of wave slope. This could be done because the relationship between angle of rotation and photo detector voltage was approximately linear, as shown in a sample calibration plot in Figure H.2. The physics of this system are shown in Figure H.3.

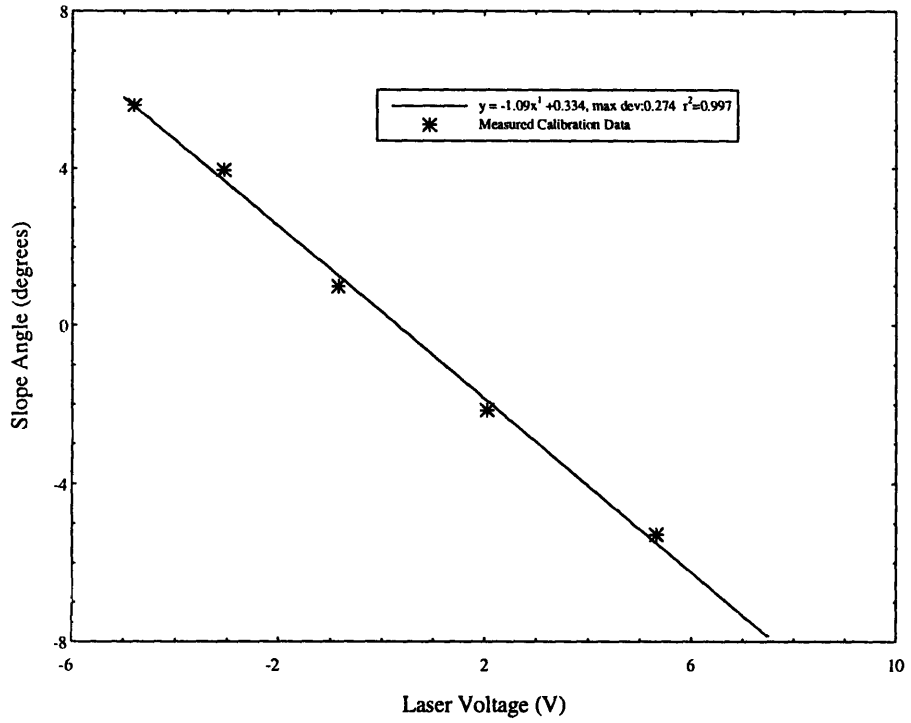

FIGURE H.2: Sample Calibration Plot for Reflective Laser Slope Gage

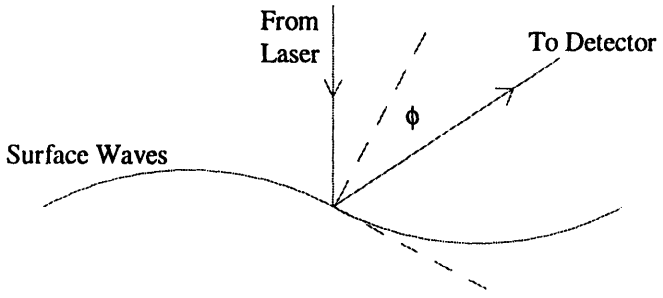

FIGURE H.3: Laser Slope Gage Physics

In order to obtain wave height information from the slope versus time data which was collected, one had to use the relationship that

$$\text{slope} = kA \tag{H.1}$$

where k is the wave number and A is the wave amplitude. The frequency spectrum measured by the laser slope gages was calculated, thus, by calculating the slope spectrum and then dividing it by the wave number squared (because the spectrum is a measure of amplitude squared, not just amplitude.)

APPENDIX I: Wave Gage Precision Test Data

Circular Wave Gage Data

An example of the time series wave height history as measured by the circular wave gage is shown in Figure I.1. The variance and resulting amplitude of each of the test data sets are shown in Table I.1.

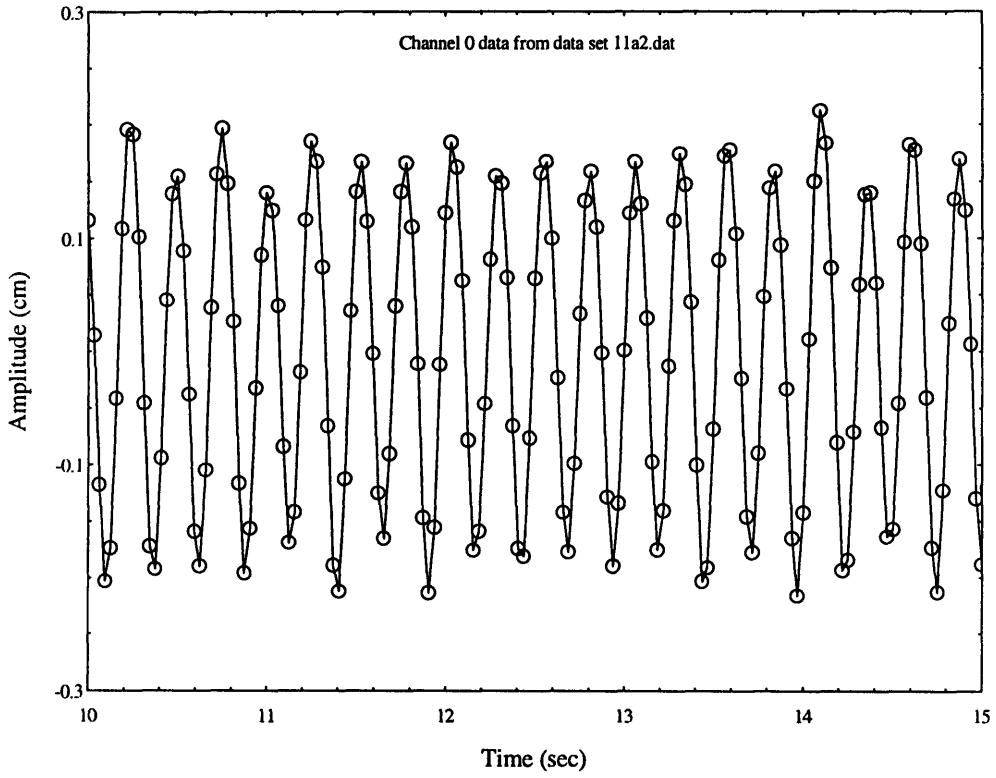

FIGURE I.1: Sample Time Series Wave Height Data

TABLE I.1: Average Amplitude and Variance for Circular Wave Gage

Test #	Ave. Variance (cm^2)	Ave. Amplitude (cm)
1	0.0332	0.258
2	0.0179	0.189
3	0.0083	0.129
4	0.0034	0.082
5	0.0004	0.028

Laser Slope Gage Data

The following lists show the output of the laser slope gage calibration and data processing program for the five tests which were conducted. The two variance measures are given because two different methods were used to calculate the variance in the wavesp.for program.

Using file 11a1las.dat (Test 1)

Maximum (unaliased) frequency is : 16.0000
Cutoff freqs for analysis are (Hz): 2.00000 10.0000
Calibration slope (mV/V): -67.8000
Y-intercept of calibration (mV): 29.0000
Laser calibration (mV/deg): 60.6000

VARIANCE [cm^2]		AMPLITUDE [cm.]
INT[S(f)df]	1/2SUM[a(i)^2]	
0.0349578	0.0349578	0.26442

Using file 11a2las.dat (Test 2)

Maximum (unaliased) frequency is : 16.0000
Cutoff freqs for analysis are (Hz): 2.00000 10.0000
Calibration slope (mV/V): -67.8000
Y-intercept of calibration (mV): 29.0000
Laser calibration (mV/deg): 60.6000

VARIANCE [cm^2]		AMPLITUDE [cm.]
INT[S(f)df]	1/2SUM[a(i)^2]	
0.0212506	0.0212506	0.20616

Using file 11a3las.dat (Test 3)

Maximum (unaliased) frequency is : 16.0000
Cutoff freqs for analysis are (Hz): 2.00000 10.0000
Calibration slope (mV/V): -67.8000
Y-intercept of calibration (mV): 29.0000
Laser calibration (mV/deg): 60.6000

VARIANCE [cm^2]		AMPLITUDE [cm.]
INT[S(f)df]	1/2SUM[a(i)^2]	
0.0119999	0.0120000	0.15492

Using file 11a4las.dat (Test 4)

Maximum (unaliased) frequency is : 16.0000
Cutoff freqs for analysis are (Hz): 2.00000 10.0000
Calibration slope (mV/V): -67.8000
Y-intercept of calibration (mV): 29.0000
Laser calibration (mV/deg): 60.6000

VARIANCE [cm^2]		AMPLITUDE [cm.]
INT[S(f)df]	1/2SUM[a(i)^2]	
0.0063701	0.0063701	0.11287

Using file 11a5las.dat (Test 5)

Maximum (unaliased) frequency is : 16.0000
Cutoff freqs for analysis are (Hz): 2.00000 10.0000
Calibration slope (mV/V): -67.8000
Y-intercept of calibration (mV): 29.0000
Laser calibration (mV/deg): 60.6000

VARIANCE [cm^2]		AMPLITUDE [cm.]
INT[S(f)df]	1/2SUM[a(i)^2]	
0.0004263	0.0004263	0.02920