CHARACTERIZATION OF MUSCLE TISSUE USING ULTRASOUND TIME-DOMAIN CORRELATION TECHNIQUES

BY

LINDA MARIE NOSTWICK

B.S., University of Illinois, 1991

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1993

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

JULY 1992

WE HEREBY RECOMMEND THAT THE THESIS BY

LINDA MARIE NOSTWICK

ENTITLED CHARACTERIZATION OF MUSCLE TISSUE USING ULTRASOUND TIME-DOMAIN CORRELATION TECHNIQUES

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

W^m O'Brien
Director of Thesis Research

[illegible]
Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

O-517

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

GRADUATE COLLEGE DEPARTMENTAL FORMAT APPROVAL

THIS IS TO CERTIFY THAT THE FORMAT AND QUALITY OF PRESENTATION OF THE THESIS SUBMITTED BY LINDA MARIE NOSTWICK AS ONE OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE ARE ACCEPTABLE TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING.

Full Name of Department, Division or Unit

27 July 1992

Date of Approval

[illegible]

Departmental Representative

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor William D. O'Brien, Jr., for his encouragement, advice and patience during this project and for the funding. Thanks is also due to Professor Jan Novakofski for his insight into muscle biology.

I would also like to acknowledge Ilmar Hein for his support in all areas of this project, Bob Cicone for his help with the Macintosh and Nadine Smith for her calibration of the ATL and the resultant plots. Finally, I would like to thank my parents, Allan and Hildegard Nostwick, for their many years of support without which this thesis would not have been possible.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

The ultrasound time-domain correlation (UTDC) technique is used to determine changes in the location of structures within the ultrasound scanning field over a specific time frame by determining the maximum correlation coefficient from digitized RF echo signals obtained at different times. Previous research, using a modified pulse-echo ultrasound imaging system, has successfully utilized ultrasound time-domain correlation techniques to estimate blood flow velocity [1] - [5], the movement of tissue (blood) in the scanning field. The accepted process of rigor mortis [6] also involves movement of tissue structure, the contraction and expansion of the muscle fibers. Therefore, this technique was applied to digitized RF echo signals obtained from porcine and bovine tissue to determine when physical activity occurs in the post-mortem muscle tissues and when the activity ceases [7].

1.1 Pulse-Echo Ultrasound Imaging

As shown in Figure 1, pulse-echo imaging systems function by transmitting very short ultrasonic pulses from the transducer and receiving the echoes resulting from reflection of the pulses. The active element within the transducer is a piezoelectric ceramic element. Piezoelectric materials change shape in response to an electric stimulus, for example, a voltage spike, and will oscillate

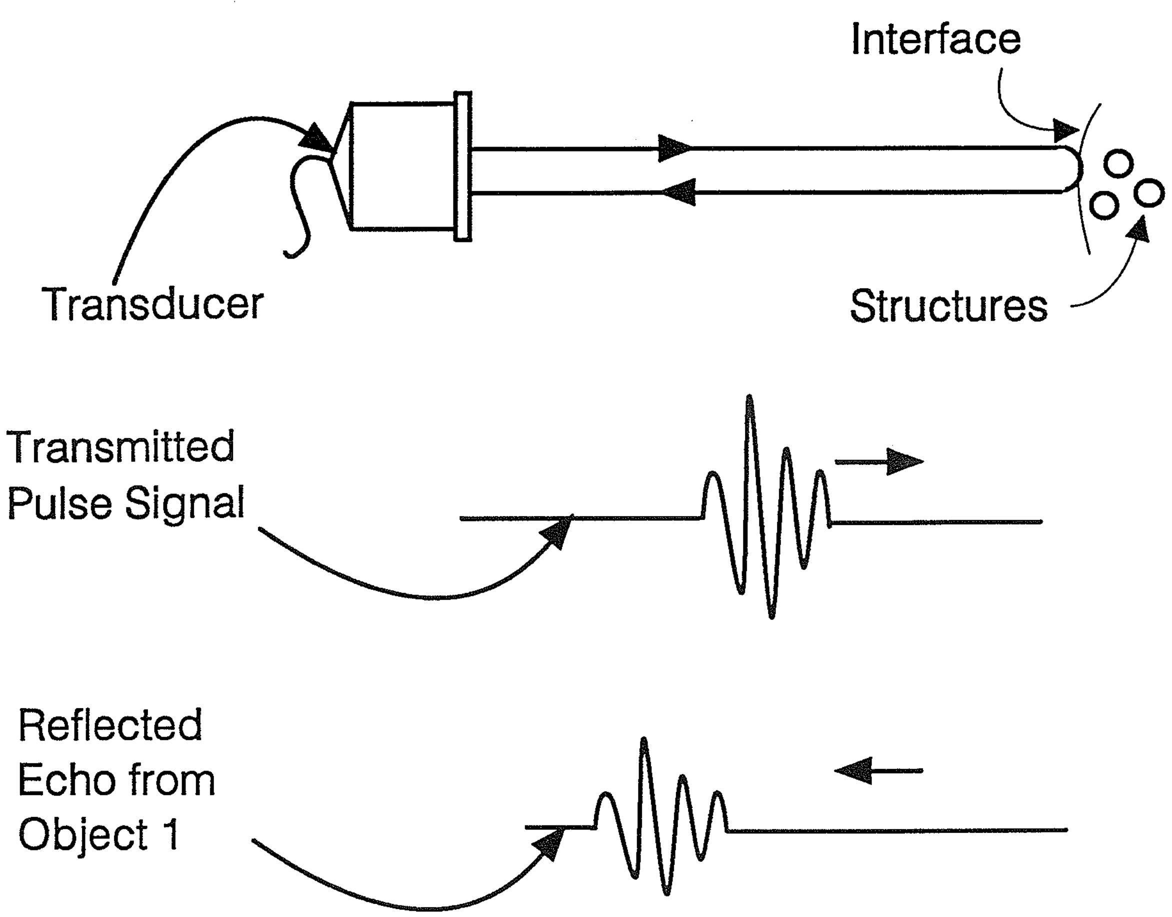


Figure 1. Example of the reflection at an interface of ultrasound pulses produced by a pulse-echo imaging system.

for a short period of time producing an ultrasound pulse of a specific frequency, dependent on the period of excitation. For example, at an operating frequency of 5 MHz the voltage spikes will be applied for half a period (0.1 ms), and will occur every 0.2 ms. A sequence of ultrasonic pulses produces an acoustic scan line.

A pulse-echo ultrasound imaging system consists of a piezoelectric transducer and a system to provide the necessary excitation. This particular system utilizes a focussed transducer with rotating piezoelectric elements that produce a scanning field similar in shape, in the axial direction, to the one shown in Chapter 2, Figure 6. The properties of the field vary with the distance from the transducer and can be characterized by calibrating the transducer to AIUM standards [8].

As the pulses come in contact with an interface between two media, some of the wave energy is reflected as shown in Figure 1. The shape of the reflected echo signal depends on structures within the material being interrogated that scatter the transmitted pulse signal. If two different materials are being imaged, for example, muscle and bone, the reflected echo signal from each will be different. Also, if the structures within the muscle move, as shown in Figure 2, perhaps through contraction and expansion of the muscle fibers, the backscatter properties of the muscle will change [9], [10] producing different reflected echo signals.

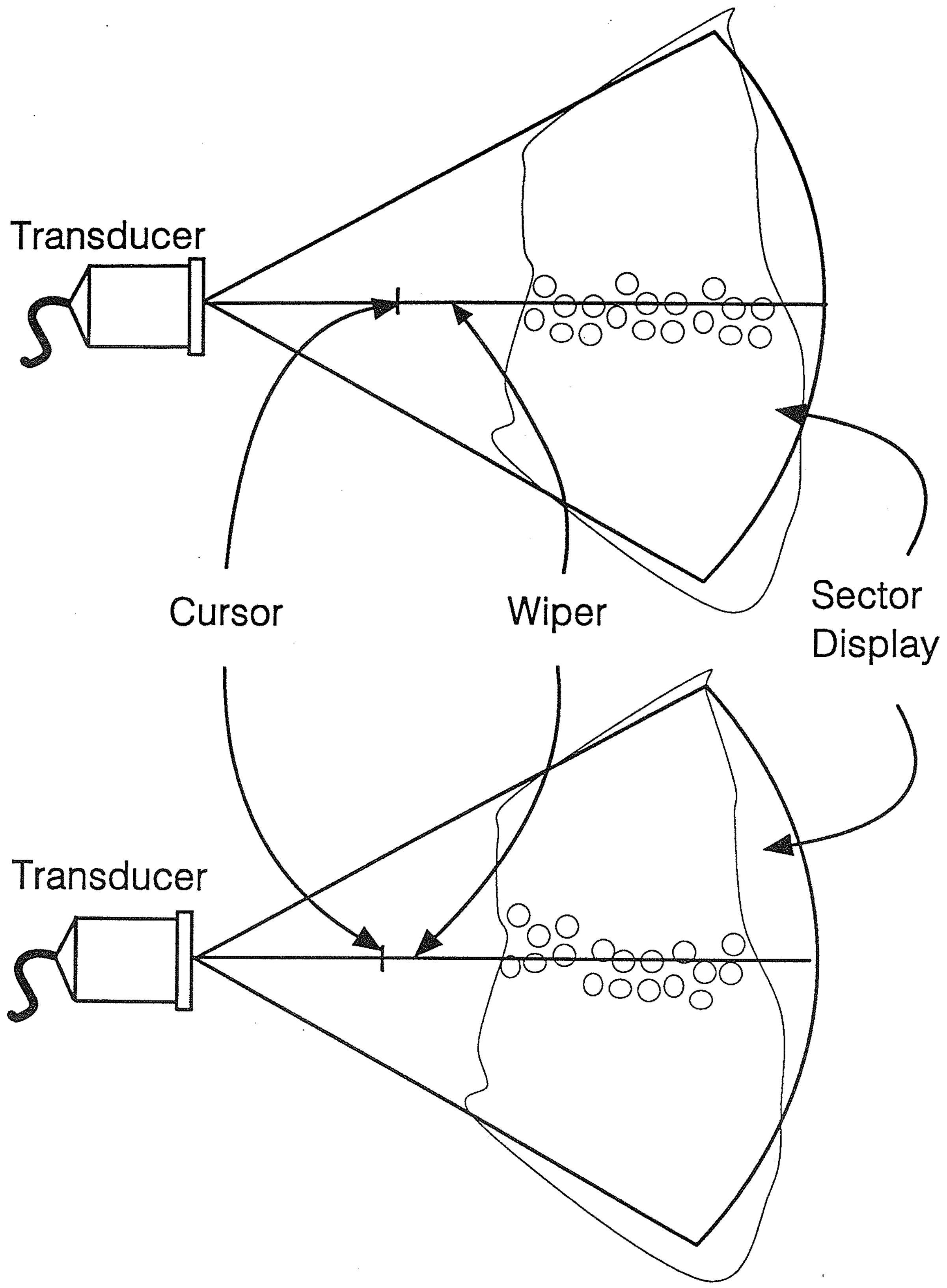


Figure 2. Example of object movement within the scanning field.

The reflected signal is an acoustic pressure wave. When the reflected signal is incident on the piezoelectric material, the acoustic pressures cause the material to vibrate producing measurable voltages. Thus, the reflected ultrasound signal is translated into a voltage signal corresponding to the amplitude of the reflected sound wave. The voltages are transmitted to the image processor of the system which produces a grayscale image to be displayed on a monitor. To process these reflected signals with a computer requires a digitized version of the signal. The voltages are accessed directly through the RF port for digitization by a 50 MHz analog-to-digital converter. These echo signals can then be stored on the hard drive of a PC for later correlation analysis.

1.2 Ultrasound Time-Domain Correlation Techniques

The UTDC technique involves the determination of a correlation coefficient which is calculated using echo signals acquired at different times. At the University of Illinois, a system similar to the one utilized for this research was developed for determining blood flow by Steve Foster [1] at the Bioacoustics Research Laboratory. The system was modified by Paul Embree [2], [5] and later by Ilmar Hein [3], [4], resulting in the system in Chapter 2, Figure 4.

The reflected ultrasound pulse will vary in shape depending on the material being imaged and the geometry and placement of the internal scattering structures. To obtain a quantitative description of this change, the correlation coefficient between two echo signals can be calculated. Values of the correlation coefficient vary between zero and one, where one indicates echo signals identical in shape and zero indicates total dissimilarity. If, during the time between acquisition of two echo signals, movement of the sample or the structures within occurs, the two echo signals will be shaped differently. For example, if a group of objects move as shown in Figure 2, the two reflected echo signals will be different in shape resulting in a low value for the correlation coefficient. Therefore the correlation coefficient can be used to determine if there are changes in the physical placement of the objects in the scanning field. If a group of objects move together within the imaging volume, as shown in Figure 3, the total distance the group moves can be calculated by determining the amount the second echo must be shifted with respect to the first echo to obtain the maximum correlation coefficient as shown by Ilmar Hein to determine blood flow [3].

Variations in the shape of the echo signals reflected from the same muscle tissue volume post-mortem are related to the changes occurring in the sample as a function of time post-mortem. Post-mortem variations in the acoustic parameters could be the result of rigor-related structural changes, muscle fiber movement or overall movement of the sample. The correlation coefficient is an indicator of the degree of structural change occurring in the sample volume

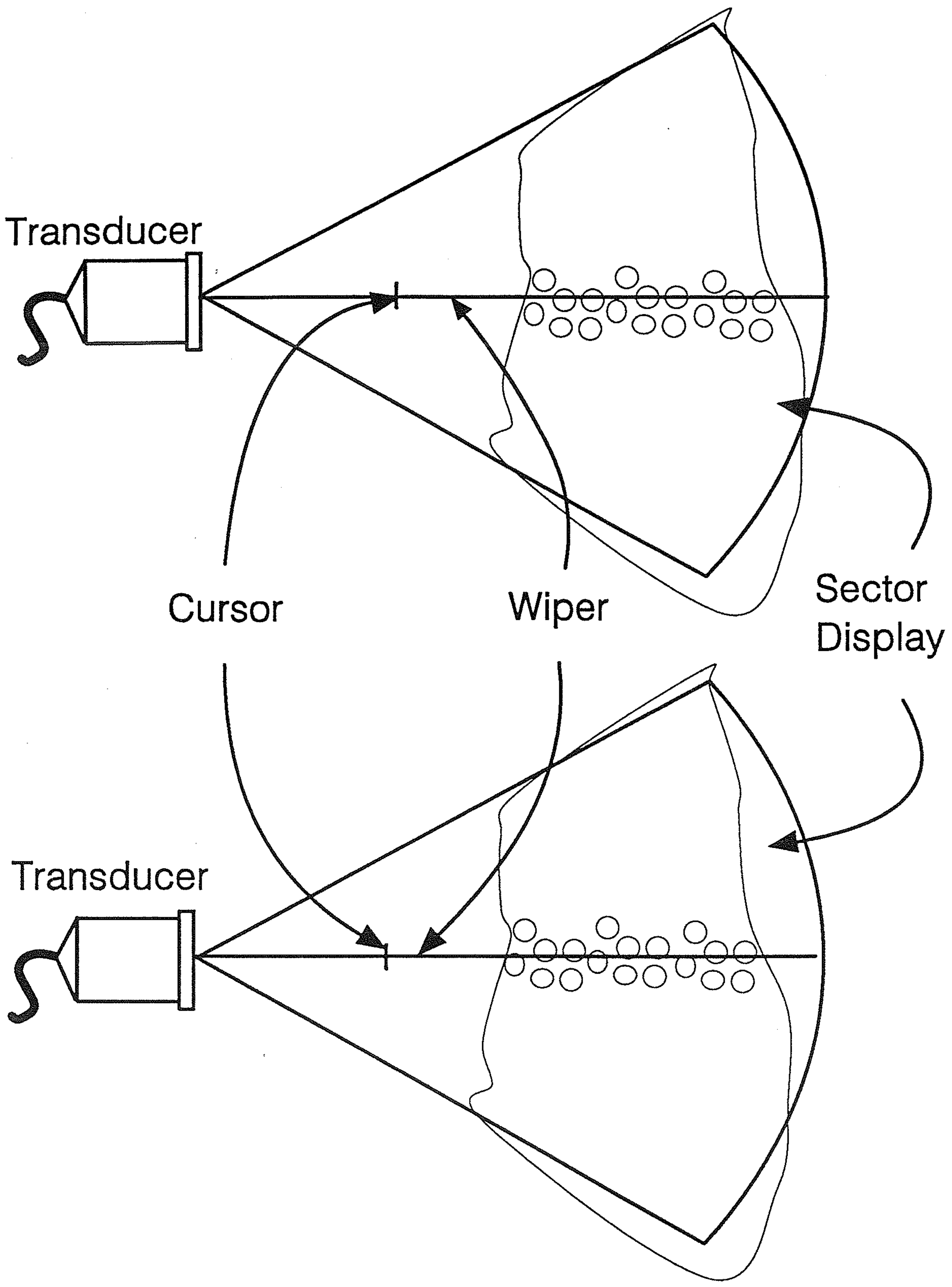


Figure 3. Example of movement of a group of objects within the scanning field.

during the time between two acquired echo signals. When the onset of rigor is complete, the structural changes occurring in post-mortem muscle cease, resulting in essentially identical echo signals. Therefore, the completion of rigor is indicated when the correlation coefficient stabilizes near unity.

1.3 Motivation for Characterizing Muscle Tissue

Traditional methods for grading beef are given by [9]. It may be possible to obtain a more consistent method through the use of ultrasound; however, first a time post-mortem when the muscle tissue is not changing must be determined. Although porcine muscle is not graded currently, porcine samples are easier to obtain than bovine samples and were used for the initial experiments. The results indicated that this method could be used to characterize post-mortem muscle behavior. Therefore, the same procedure was performed using a bovine sample, yielding preliminary results similar to the porcine as discussed in Chapter 4. The observed trends in the correlation coefficient can also be used to compare post-mortem muscle behavior between species and supplement the information on the processes of rigor mortis.

CHAPTER 2

MATERIALS, DATA ACQUISITION SYSTEM AND METHODS

This chapter describes the materials imaged, the data acquisition system and the methods utilized to obtain the echo signals to be correlated. The system, shown in Figure 4, is a modified version of the one used in [1] - [5] and consists of a water-filled tank, an ATL imaging system, a custom designed A/D, an interface board and a PC 486.

2.1 Materials

As a control, a water-filled cellulose sponge was imaged first. The sponge was shaped in a half-oval 18.8 cm long, 9.4 cm wide and 5.3 cm thick at the center. The sponge was allowed to soak in water for 24 h to remove all of the air before beginning data acquisition.

Boneless porcine longissimus muscle samples and boneless bovine longissimus muscle samples were then imaged. The porcine samples were approximately 17 cm long, 12 cm wide and 5.5 cm thick and the bovine samples were approximately 13 cm long, 15 cm wide and 7 cm thick. Both sample types were obtained from the Meat Science Laboratory within 30 min after slaughter. The muscle samples were vacuum packed in composite film Cryovac ® vacuum bags

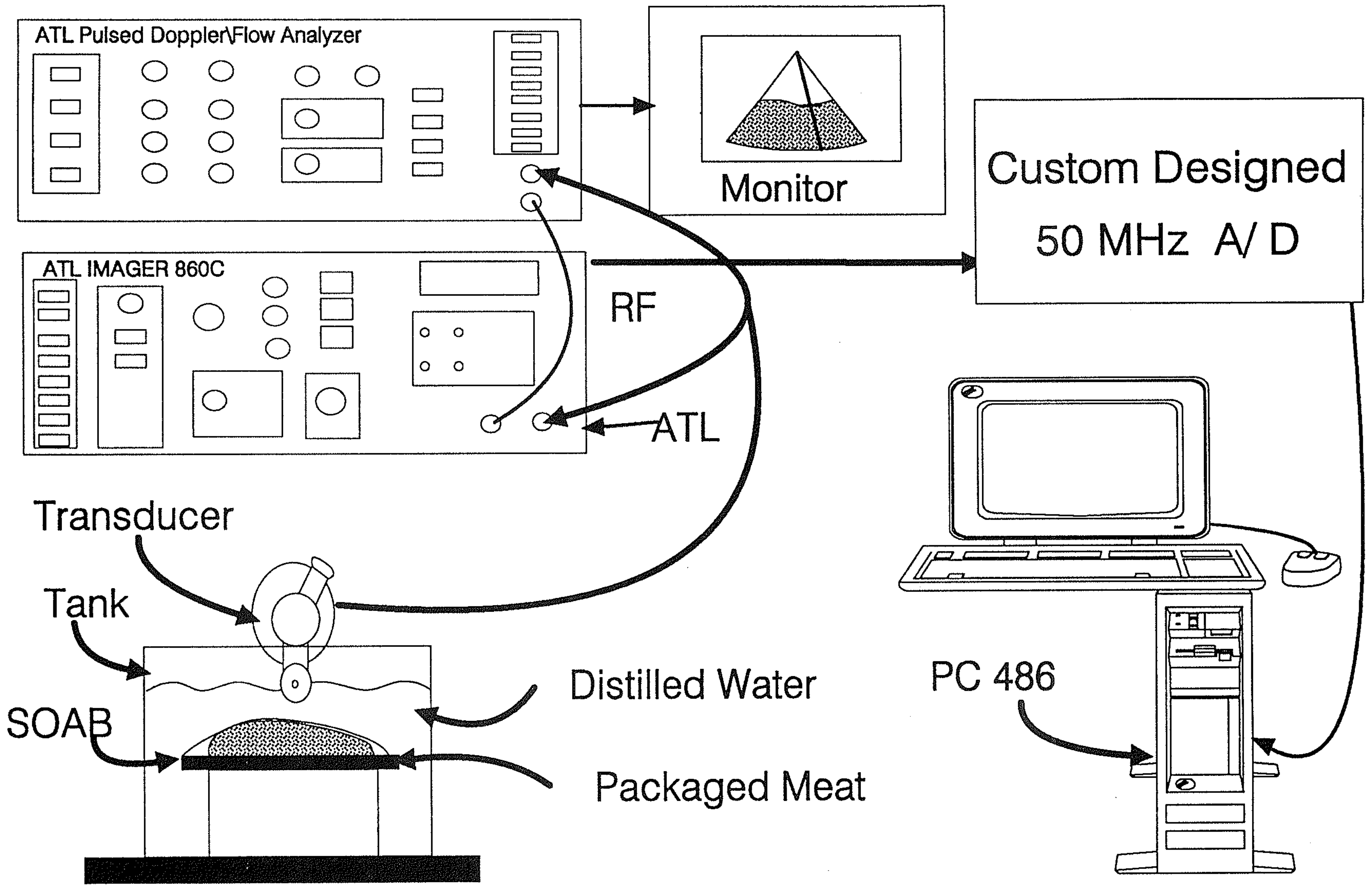


Figure 4. Block diagram of the data acquisition system.

and transported to the Bioacoustics Research laboratory in a room-temperature water bath (23°C).

2.2 Data Acquisition System

The data acquisition system is shown in Figure 4. The individual samples were placed on a piece of acoustic absorbing material, SOAB, to minimize reflections from the back surface. To reduce movement of the sample, the SOAB and sample were attached to a heavy rectangular transparent plastic plate, approximately 30 cm long and 3.5 cm thick. The cellulose sponge was secured by covering it and the SOAB with a piece of cheesecloth and clamping the edges of the cloth to the plastic plate. The muscle samples were secured by clamping the edges of the sealed vacuum bag around the SOAB and onto the plate. The samples were placed in a tank filled with distilled water, serving as an acoustic coupling agent.

The samples were imaged using a modified ATL system that served as both source of the transmitted ultrasound pulses and receiver of the reflected echoes. The system consists of an ATL Imager 860C and an ATL 724B multifrequency transducer operated at a center frequency of 4.5 MHz. Modification of the MK500 allowed for acquisition of the RF data following time gain compensation. Physically, data acquisition began at the position of the cursor (shown as the small horizontal black line on the monitor in Figure

4). A description of the triggering used to coordinate the transmitted pulses and the start of digitization of the reflected echoes is given by Ilmar Hein [3].

The digitization of the received echoes was performed by a 50 MHz analog-to-digital converter (A/D) [3]. The digitized echo signals were then transferred from the A/D board to an interface board and stored on the hard drive of the 486 PC for later correlation analysis. The PC 486 was used not only to store the digitized echo signals but also to control the echo length, the rate of data acquisition and the data transfer.

2.3 Data Acquisition Procedures

The sponge was placed in the tank 24 h before data acquisition was initiated. For the porcine and bovine muscle samples, data acquisition was initiated within 1 1/2 h after expiration of the animal. For all sample types, data acquisition was continued for 24 h. The ultrasound transducer was clamped above the tank as shown in Figure 4, with the head partially submerged in the water approximately 3 or 4 cm above the surface of the sample.

The MK500 was operated in M-mode which, through the use of a control pedal, can freeze the transducer along a selectable A-line (shown as the long black line on the monitor in Figure 4). The A-line was positioned near the center of the scan field and the cursor was positioned just inside the surface of the sample as shown in Figure 5. The volume being imaged was approximately cylindrical. To determine the dimensions, the scanning field of the transducer was characterized by calibrating the transducer to AIUM standards [8]. A description of the field parameters as well as the volume dimensions are shown in Figure 6.

The RF echoes were digitized at the rate of 1 every minute for the first 3.5 h and then 1 every 10 min from 3.5 to 24 h. It was desired to image a depth of 6 cm into the sample; therefore, digitization began at the cursor, near the sample surface, and assuming the speed of sound in the sample to be about 1560 m/s [11], was continued until a total of 4096 points were stored. The transmitted ultrasonic energy was adjusted to obtain the best dynamic range for the A/D converter, determined by using the PC 486 to display a plot of the digitized values vs. depth as shown in Figure 6. A description of the correlation analysis performed on the digitized echoes is given in Chapter 3.

2.4 Data Acquisition Software

One program had previously been written for the acquisition of the RF echoes for this system [3], using the Borland Turbo C Package. The program was originally executed using a Compaq 386 operating at 20 MHz. To increase the speed and allow for future additions to the experiment that would require greater memory

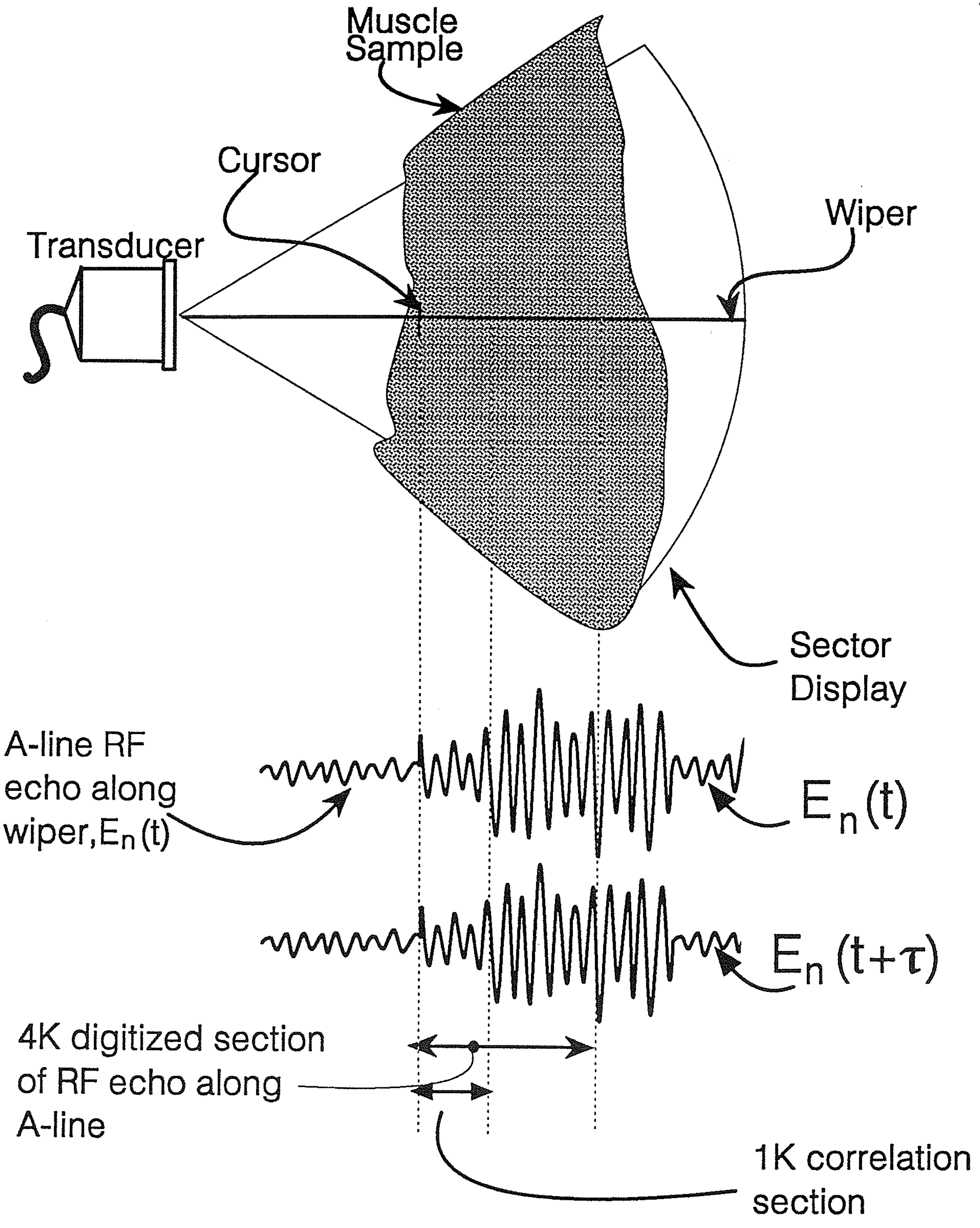
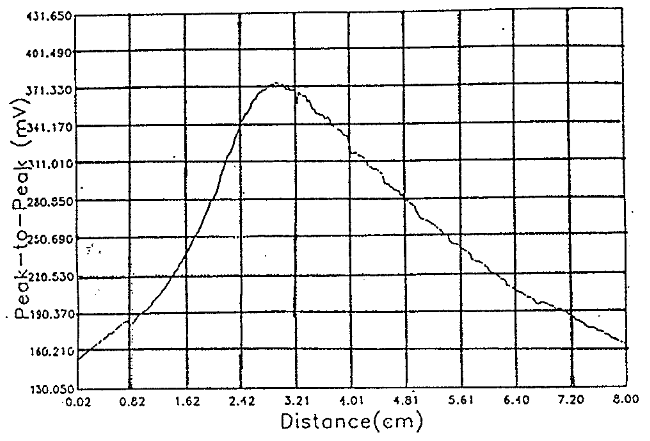
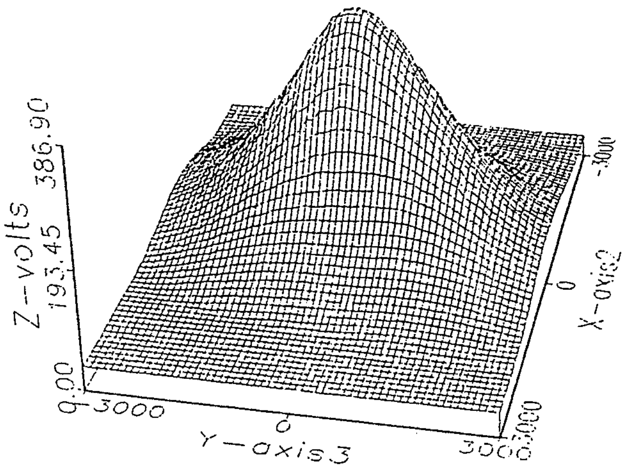


Figure 5. Ultrasound scanning field showing two digitized RF echo signals and the corresponding correlation range.

(a)



(b)



(c)

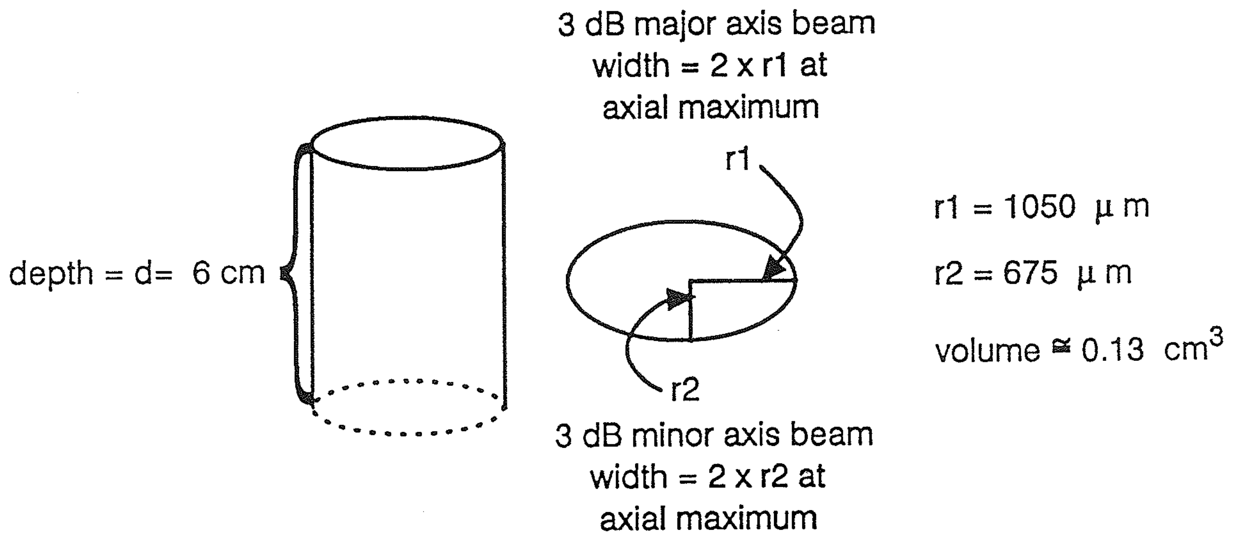

3 dB major axis beam width = 2 x r1 at axial maximum
r1
depth = d= 6 cm
r1 = 1050 µ m
r2 = 675 µ m
volume ≅ 0.13 cm³
r2
3 dB minor axis beam width = 2 x r2 at axial maximum

Figure 6. (a) Axial plot of voltage in the ultrasonic field of the transducer (b) plot of uniformity of ultrasound field at axial maximum in an area 6000 µm by 6000 µm (c) description of approximate meat volume imaged.

capabilities, a PC 486 replaced the Compaq. The program controlled the A/D and displayed a plot of the digitized echo signals on the PC monitor. Control of the length of the digitized echo and of the location in memory in which to store the echo signals were two features of this program. The program was modified for this application by the addition of the ability to choose the data acquisition period through the use of a menu. A listing of the program is given in Appendix A.

Chapter 3
RF ECHO SIGNAL ANALYSIS

This chapter presents the methods used for the analysis of the echo signals using the ultrasound time-domain correlation technique (UTDC), which involve the computation of the maximum correlation coefficient between two RF echo signals acquired at different times post-mortem separated by τ minutes. The average correlation coefficient for each τ and the standard deviation of the voltages in each individual echo signal were also calculated. To obtain the maximum value and isolate spatial shifts of the echo signals due to axial motion with respect to the fixed transducer, the second echo must be shifted with respect to the first echo until a maximum is found, as shown in Figure 7. The correlation coefficients from eight porcine data sets were averaged together, and the standard deviation of the averaged correlation coefficient was calculated.

3.1 Calculating the Maximum Correlation Coefficient

The echo signal analysis involved computing the correlation coefficients and ascertaining the shift at which the correlation coefficient was maximized. The correlation coefficients were computed according to [1] using the following equation:

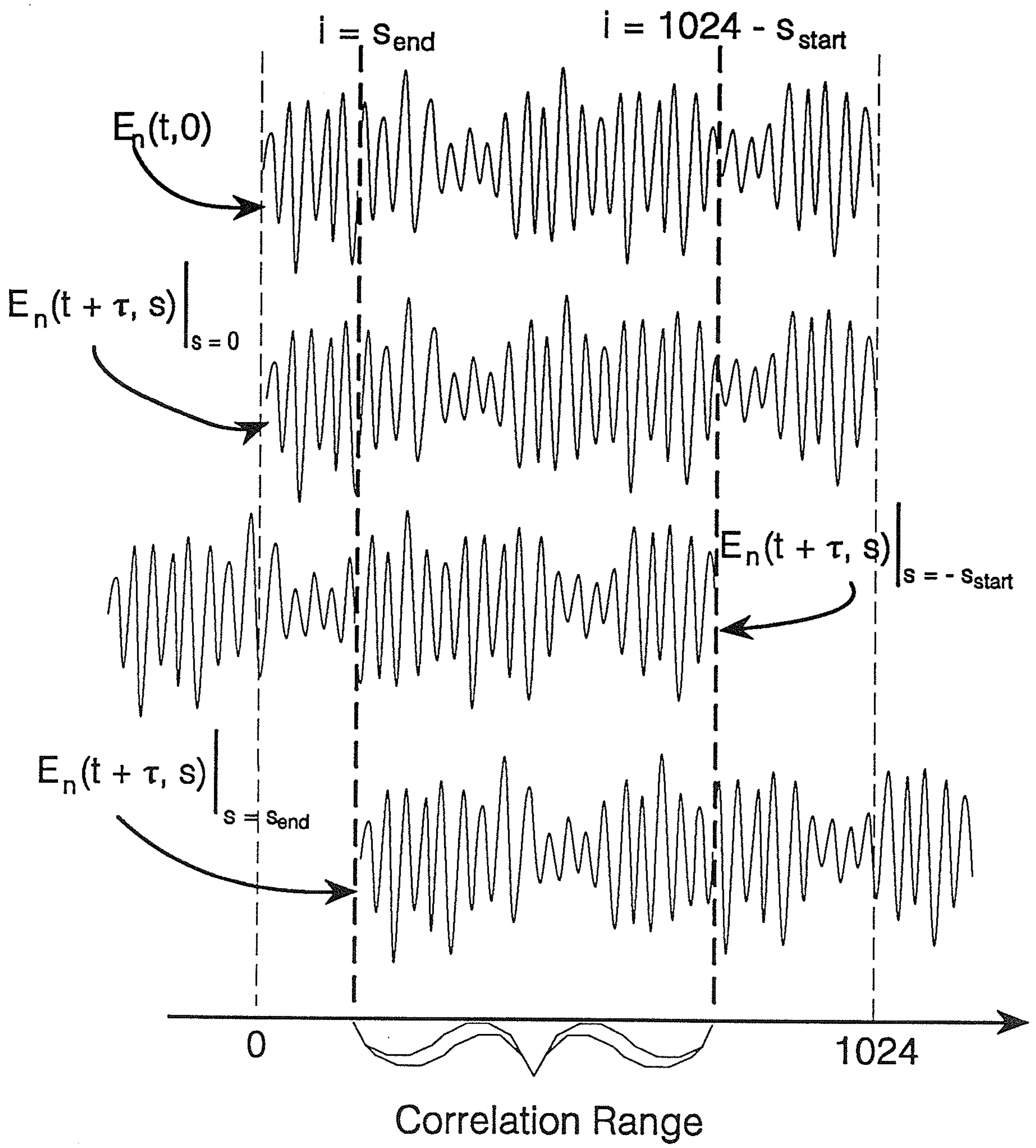

$i = s_{end}$
$i = 1024 - s_{start}$
$E_n(t,0)$
$E_n(t+\tau, s)\big|_{s=0}$
$E_n(t+\tau, s)\big|_{s=-s_{start}}$
$E_n(t+\tau, s)\big|_{s=s_{end}}$
0
1024
Correlation Range

Figure 7. Shifting of the digitized echo signals from $-s_{start}$ to s_{end} with respect to the first echo to calculate maximum correlation coefficient.

$$\rho_n(t, \tau, s) = \frac{\mathrm{Cov}[E_n(t,0), E_n(t+\tau,s)]}{\sqrt{\mathrm{Var}[E_n(t,0)] \times \mathrm{Var}[E_n(t+\tau,s)]}} \quad s_{start} < s < s_{end} \qquad (1)$$

where $E_n(t,0)$ and $E_n(t+\tau,s)$ are the digitized echo signals acquired at t min and at $t+\tau$ min, respectively, s is the shift in data points of the second echo signal with respect to the first echo, s_{start} is the maximum amount of data points that the second echo is shifted back with respect to the start of the echo, s_{end} is the amount of data points that the second echo signal (Figure 7) is shifted forward with respect to the start of the echo and $\rho_n(t,\tau,s)$ is the value of the correlation coefficient at time t, for echo spacing τ and shift s. Values for the spacing τ were 1, 5, 10, 30 and 60 min for the first 3.5 h post mortem, and 10, 50, 100, 300 and 600 min for the full 24 h. A shift of $E_n(t+\tau,s)$ by one data point corresponds to a time shift of 1/50 MHz = 0.02 µs, and, assuming the speed of sound to be approximately 1560 m/s, a spatial shift of (0.02 µs x 1560 m/s) = 0.031 mm. In practice, s_{start} and s_{end} were determined empirically such that the entire range of apparent motion of the tissue could be estimated. The range of shifts typically was between -60 to +30 data points with respect to the first echo. The maximum values of the correlation coefficient are found from

$$\rho_{nmax}(t,\tau,s_{max}(t,\tau)) = \max\,[\rho_n(t,\tau,s)] \qquad s_{start} < s < s_{end} \qquad (2)$$

where the maximum value of the correlation coefficient for the nth data set at time t for echo spacing τ is $\rho_{nmax}(t,\tau,s_{max}(t,\tau))$ and is found at the shift value defined as $s = s_{max}(t,\tau)$. For this analysis, it was desired to calculate the correlation coefficient over a depth of approximately 1.6 cm, therefore, assuming a speed of sound in the sample of about 1560 m/s [12], this corresponds to (3.2 cm x 50 MHz) / 1560 m/s $\approx$ 1024 points of the 4096 points stored. Thus, the total number of points used in the calculation (correlation range in Figure 7) is equal to 1024 - s_{start} - s_{end}.

3.2 Averaged Correlation Coefficient and Standard Deviation

The time averaged correlation coefficient for each value of τ is calculated according to

$$\rho_{nave}(\tau) = \frac{\sum_{t=t_{death}}^{t_{end}-\tau} \rho_{nmax}(t,\tau,s_{max}(t,\tau))}{N_\tau} \qquad (3)$$

where N_τ is the number of correlation coefficients calculated for a specific τ, t_{death} is the time after death that data acquisition began or, for the sponge, the time at the beginning of data acquisition, t_{end} is the time of the last echo stored, $\rho_{nmax}(t, \tau, s_{max}(t, \tau))$ is the value of the maximum correlation coefficient at time t for echo spacing τ at the shift $s_{max}(t, \tau)$ for the nth data set and $\rho_{nave}(\tau)$ is the average correlation coefficient for a specific τ for the nth data set. For an acquisition period of 1 min, t is incremented in 1 min steps, and, for a period of 10 min, t is incremented in 10 min steps. For a 24 h experiment, 144 digitized echo signals spaced 10 min apart are stored, totaling 1440 min; therefore, for τ = 100 min N_τ = (1440 -100)/10 = 134, whereas for τ=600 min, N_τ = 84.

The time averaged shift for each value of t is calculated according to

$$s_{nave}(\tau) = \frac{\sum_{t=t_{death}}^{t_{end}-\tau} s_{max}(t,\tau)}{N_\tau} \tag{4}$$

where N_τ is the number of correlation coefficients calculated for a specific τ, t_{death} is the time after death at which data acquisition began or the time of the start of data acquisition for the sponge, t_{end} is the time of the last echo stored, $s_{max}(t, \tau)$ is the shift for maximum correlation for time t, echo spacing τ for the nth data set, and $s_{nave}(\tau)$ is the average shift for maximum correlation for a specific τ for the nth data set.

The standard deviation of the voltages in the individual echo signals is calculated using

$$\sigma_n(t) = 3.9 \text{ mV/level} \times \sqrt{\sum_{p=0}^{N-1} \frac{[E_n(p,t)]^2}{N} - \frac{E_n^2(p,t)}{N^2}} \qquad (5)$$

where $E_n(p,t)$ is the pth data point of the echo at time t min post-mortem for the porcine and beef and t min since the beginning of data acquisition for the sponge of the nth data set, N is the total number of data points utilized (1024), and $\sigma_n(t)$ is the standard deviation of the echo signal acquired at t min post-mortem for the muscle samples and t min since the beginning of data acquisition for the sponge. To obtain the standard deviation in volts, a conversion factor of 3.9 mV/quantization level was used. The standard deviation is a measure of the variation in the values contained in the individual digitized echo signals. A larger value represents a greater spread in the magnitude of the digitized echo signal values. The standard deviation is related to variations of the scattering strength; therefore, changes in the standard deviation are representative of changes in the backscatter properties of the muscle.

3.3 Correlation Average of Eight Data Sets and the Standard Deviation

A total of eight correlation data sets were obtained from muscle samples of eight individual porcine animals with random fiber orientation with respect to the transducer. The eight correlation coefficient vs. time since death data sets were averaged according to

$$\rho_{Ave}(t,\tau) = \frac{\sum_{n=1}^{N_{SETS}} \rho_{nmax}(t,\tau,\ s_{max}(t,\tau))}{N_{SETS}} \tag{6}$$

where $\rho_{nmax}(t,\tau,\ s_{max}(t,\tau))$ is the maximum correlation coefficient for a specific echo spacing τ at a time t post-mortem from the nth data set, N_{SETS} is the number of individual correlation data sets being averaged for these experiments (N_{SETS} = 8), and $\rho_{Ave}(t,\tau)$ is the average of the eight correlation coefficient data sets for a specific echo spacing τ, a time t post-mortem.

The standard deviations of the averaged correlation coefficients for the specific τ's were determined using the expression

$$\sigma_{corr}(t,\tau) = \sqrt{\frac{\sum_{n=1}^{N_{SETS}} \rho^2{}_{nmax}(t,\tau,s_{max})}{N_{SETS}} - \frac{\sum_{n=1}^{N_{SETS}} \rho^2{}_{nmax}(t,\tau,s_{max})}{N^2{}_{SETS}}} \tag{7}$$

where $\rho_{nmax}(t,\tau,s_{max})$ is the maximum correlation coefficient for the nth data set at time t for echo spacing, τ is the number of data sets (8), and $\sigma_{corr}(t,\tau)$ is the standard deviation of the averaged correlation coefficient for a specific echo spacing τ and a time t post-mortem.

3.4 RF Echo Signal Analysis Software

All of the calculations were performed using programs written in C. One program, shown in Appendix B, calculates the correlation coefficient for all τ's, records the shift for the maximum correlation and calculates both the average shift and the average correlation. The program in Appendix C calculates the standard deviation. Both programs allow the user to start the calculations using an echo signal and a point in the data sets other than the first one obtained. This was done to allow for the possibility that the first echo and the first few points of each echo might be erroneous due to system initialization. Both programs are variations on ones originally written by Ilmar Hein. The original correlation program calculates the maximum correlation for all values of τ but stores the data only for certain values of τ. The original program is not shown due to its similarity to the one in Appendix B. The two-dimensional plots were created using the plotting program in Appendix D. To average the sets of data together the program in Appendix E reads all of the correlation file and performs both the averaging and the calculation of the standard deviation of the averaged correlation coefficient. The correlation data, the shift data and the averaged correlation coefficient were plotted using Spyglass Transform Version 2.0 for the Macintosh.

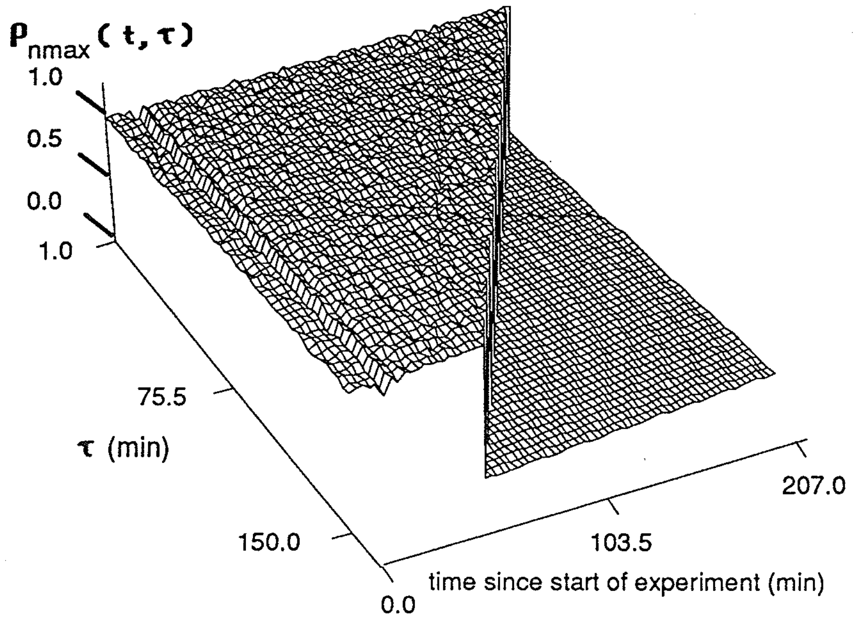
CHAPTER 4
RESULTS AND DISCUSSION

This chapter presents the results obtained from the RF echo signal analysis presented in the previous chapter and a discussion on the possible physical changes occurring in the meat sample yielding the results. The signal analysis was performed on RF echo signals from a water-filled sponge, porcine longissimus muscles and a beef longissimus muscle. In addition, eight sets of porcine correlation data were averaged together.

4.1 Sponge Results

Results from the analysis of the sponge are presented in Figure 8 through Figure 12. In Figure 8, the maximum correlation coefficient (Eqs. (1) and (2)) is plotted vs. time since the start of data acquisition (t) vs. the echo spacings (τ) for sampling periods of 1 min, (Figure 8(a)), and 10 min, (Figure 8(b)). In Figure 9 the maximum correlation coefficient vs. t for various τ's is plotted. Values for τ of 1, 5, 10, 30, 60, 50, 100, 300 and 600 min were used. For a τ of 10 min, for example, the correlation coefficient was calculated for the echo signals at 1 and 11 min, 2 and 12 min and so on. The shifts for maximum correlation vs. t vs. τ are presented for the 1 min sampling period in Figure 10(a) and for the 10 min sampling period in Figure 10(b). Individual plots at specific τ's are shown for the 1 min sampling period in Figure

(a)



(b)

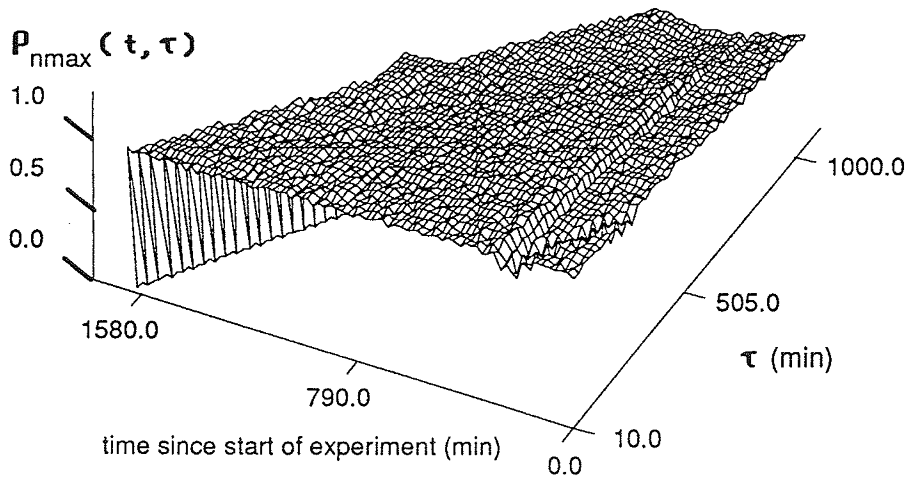


Figure 8. Plot of shift for maximum correlation vs. time since start of data acquisition vs. echo spacing (tau) for a sponge for (a) 1 min sampling period and (b) 10 min sampling period.

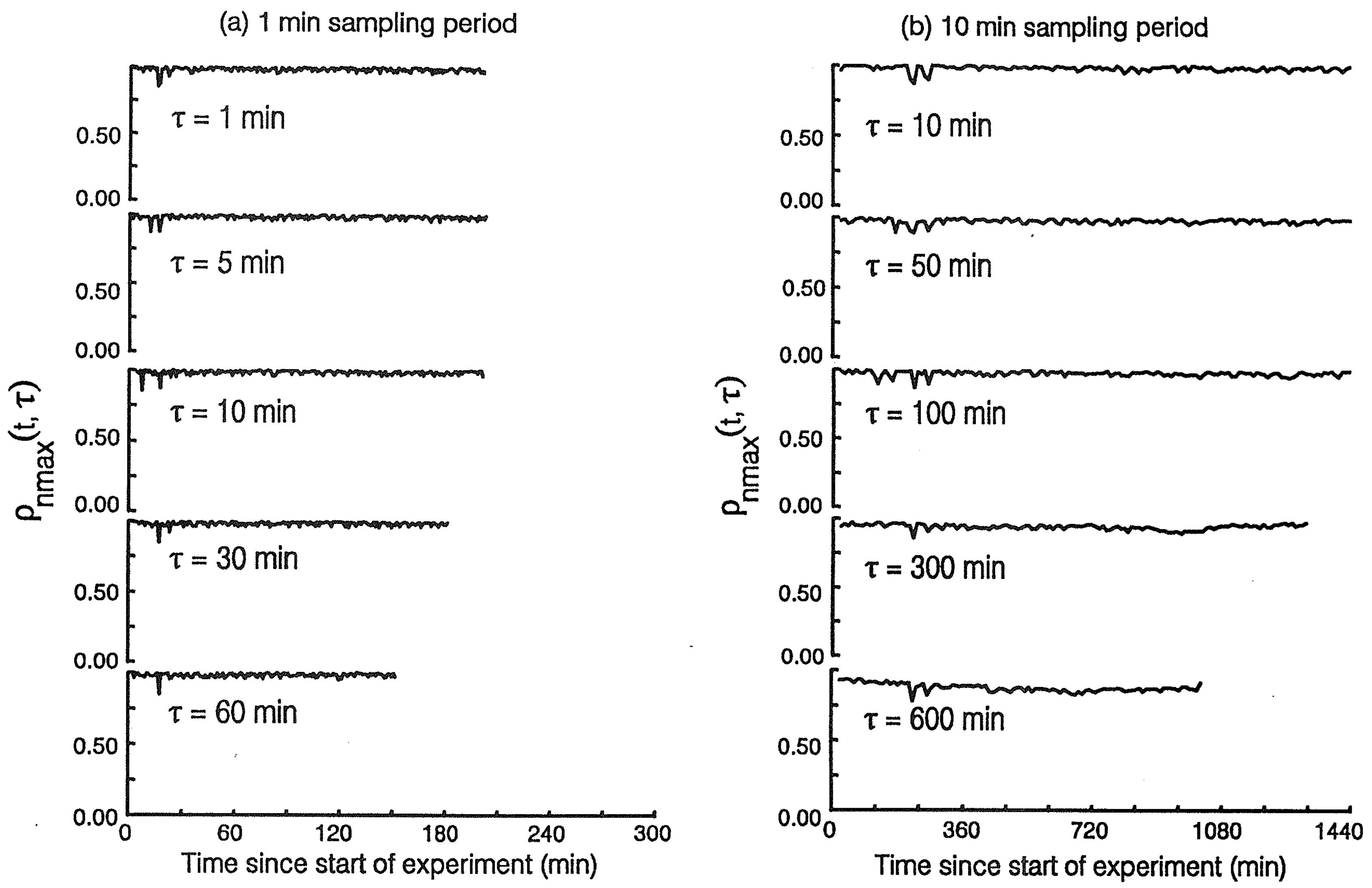
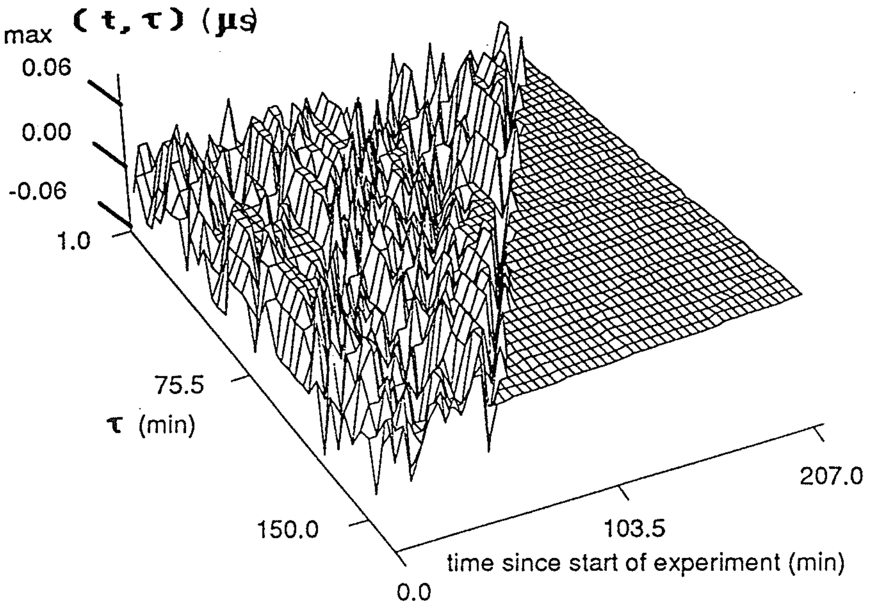


Figure 9. Maximum correlation coefficient for water-filled sponge for various τ vs. time since start of data acquisition for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

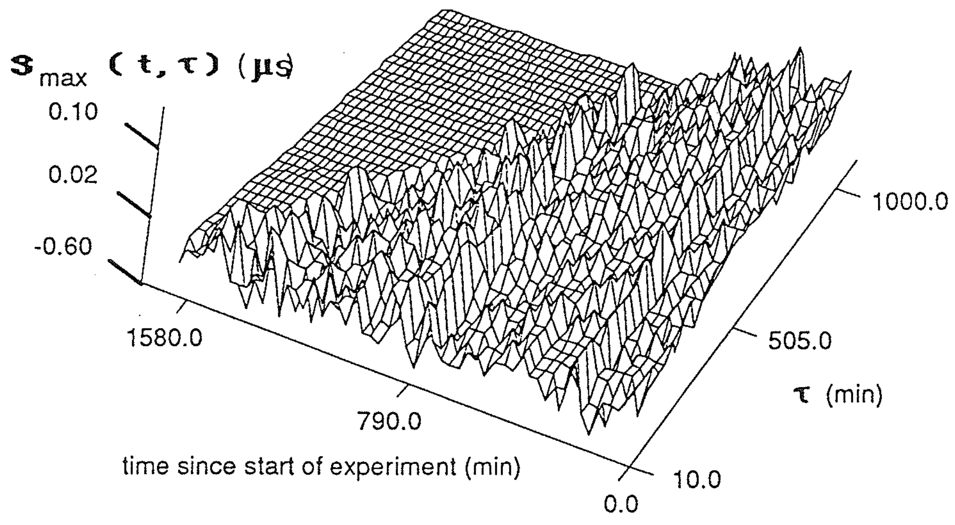


Figure 10. Shift for maximum correlation vs. time since beginning of data acquisition vs. tau for a sponge for (a) 1 min sampling period and (b) 10 min sampling period.

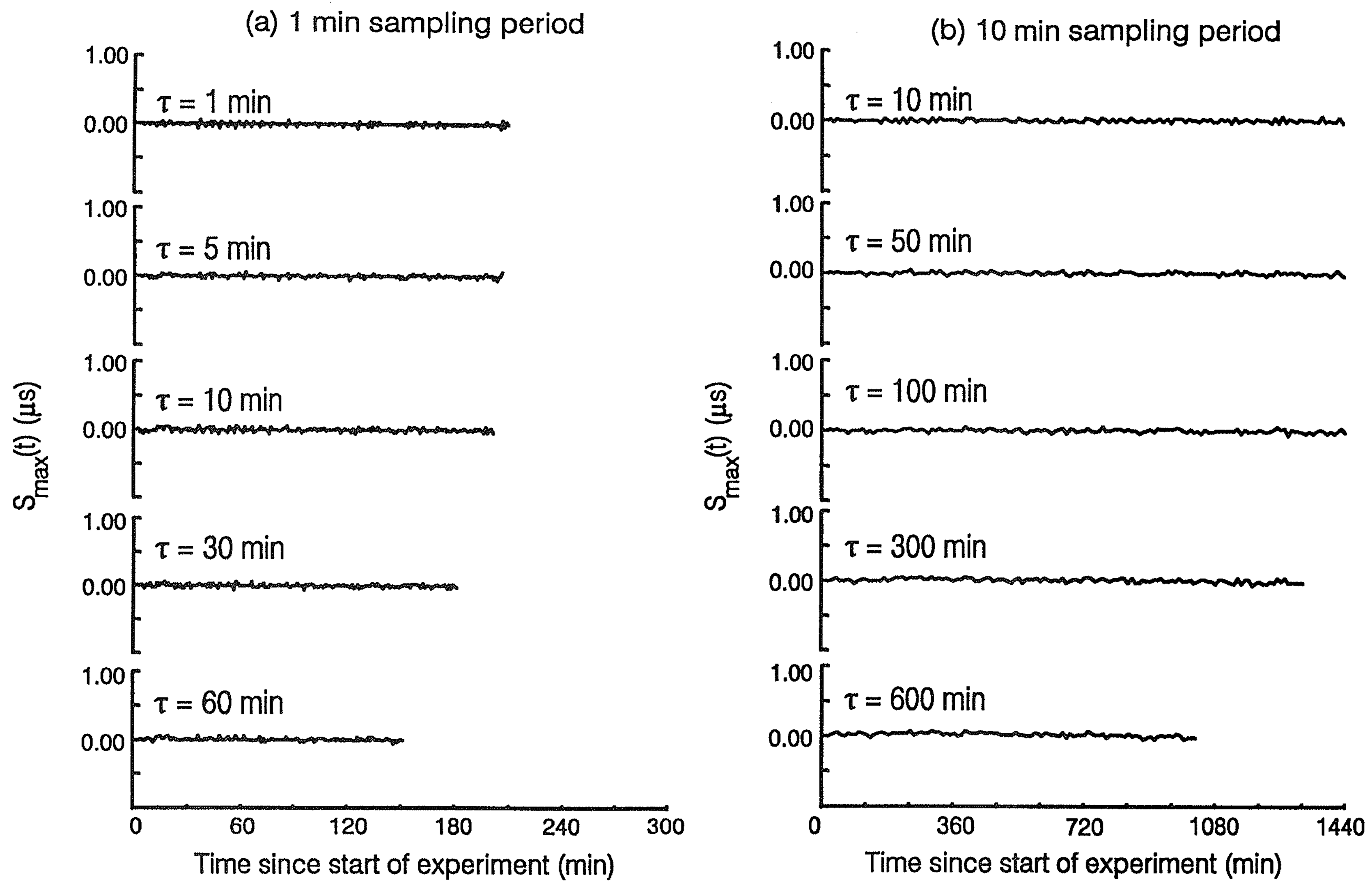


Figure 11. Shift for maximum correlation coefficient for a sponge for various τ vs. time since start of data acquisition for (a) 1 min sampling period and (b) 10 min sampling period.

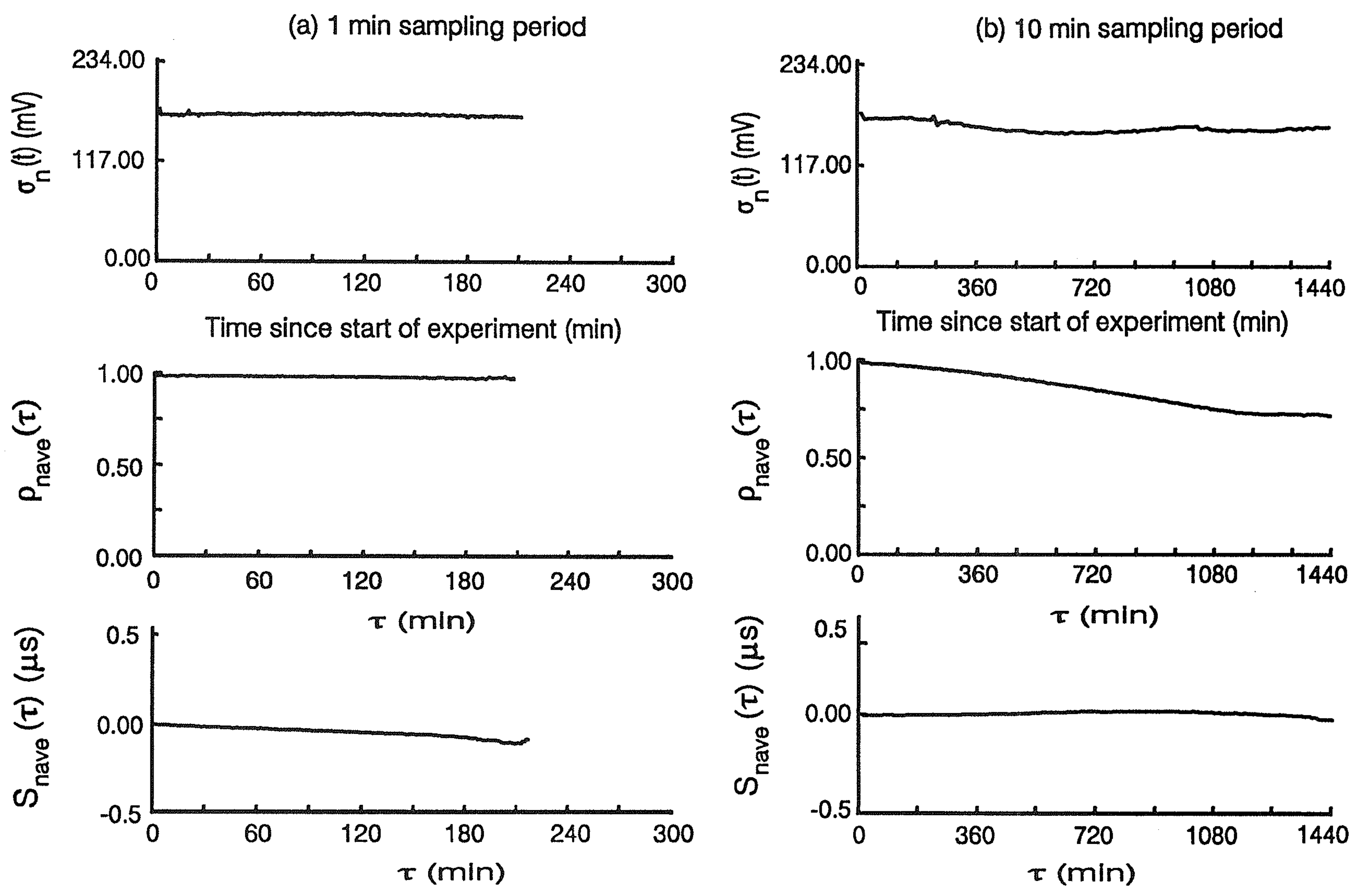


Figure 12. Standard Deviation vs. time since start of data acquisition, average correlation coefficient and average shift for a water-filled sponge vs. t for (a) 1 min sampling period and (b) 10 min sampling period.

11(a) and for the 10 min sampling period in Figure 11(b). The standard deviation of the individual echo signals ($\sigma_n(t)$) (Eq. (4)) vs. t and the average correlation coefficient ($\rho_{ave}(\tau)$) (Eq. (3)) vs. τ are plotted for both data acquisition spacings in Figure 12.

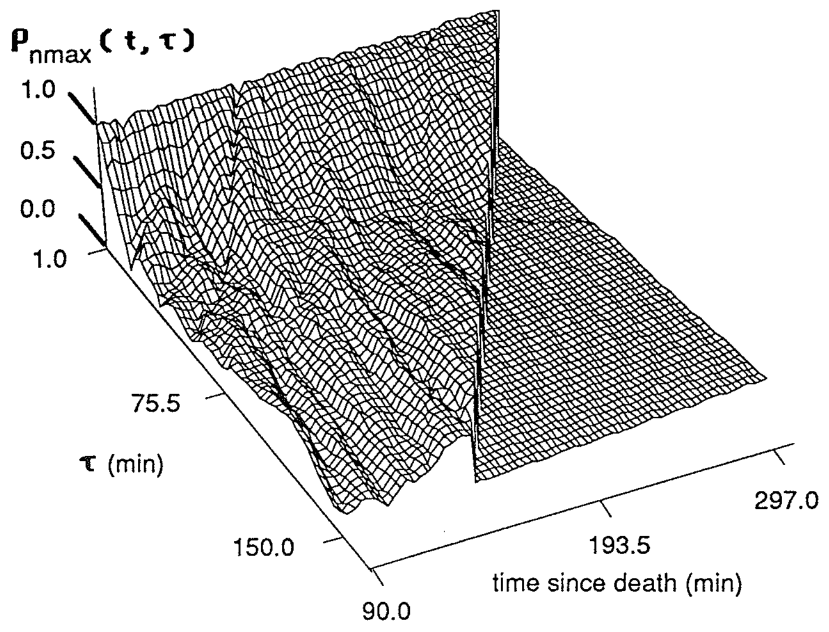
4.2 Porcine Longissimus Muscle Results

All of the results for the porcine longissimus muscle are presented in Figures 13 through 45. The plots of the maximum correlation coefficient ($\rho_n(t,\tau)$) vs. time (t) post-mortem vs. echo spacings (τ) for the 1 min sampling period are presented in Figures 13(a), 17(a), 21(a), 25(a), 29(a), 33(a), 37(a) and 41(a) and the 10 minute sampling period in Figures 13(b), 17(b), 21(b), 25(b), 29(b), 33(b), 37(b) and 41(b).

Individual plots of the maximum correlation coefficient at specific τ's are shown for the 1 min sampling period in Figures 14(a), 18(a), 22(a), 26(a), 30(a), 34(a), 38(a) and 42(a) and for the 10 min sampling period in Figures 14(b), 18(b), 22(b), 26(b), 30(b), 34(b), 38(b) and 42(b).

The shifts for maximum correlation are presented for the 1 min sampling period in Figures 15(a), 19(a), 23(a), 27(a), 31(a), 35(a), 39(a) and 43(a). For the 10 min sampling period the shifts are presented in in Figures 15(b), 19(b), 23(b), 27(b), 31(b),

(a)



(b)

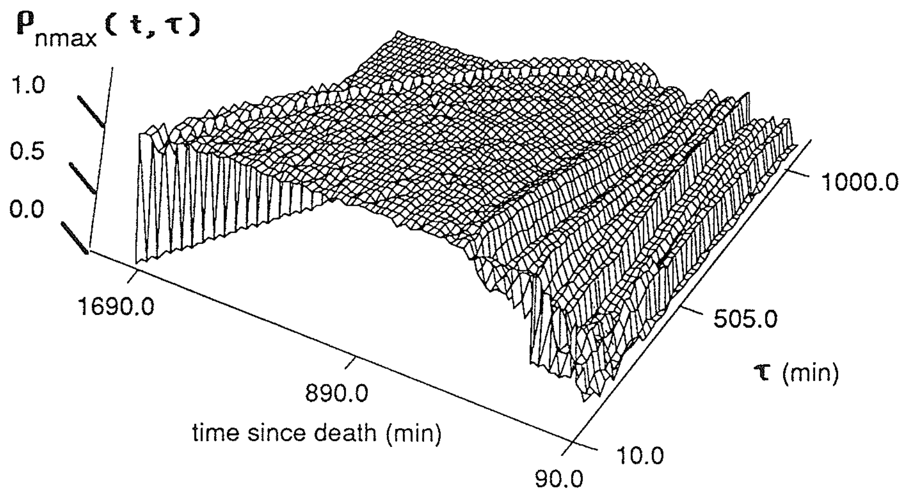


Figure 13. Plot of maximum correlation coefficient vs. time since death vs. tau for the porcine sample on Oct. 2, 1990 for (a) 1 min sampling period and (b) 10 min sampling period.

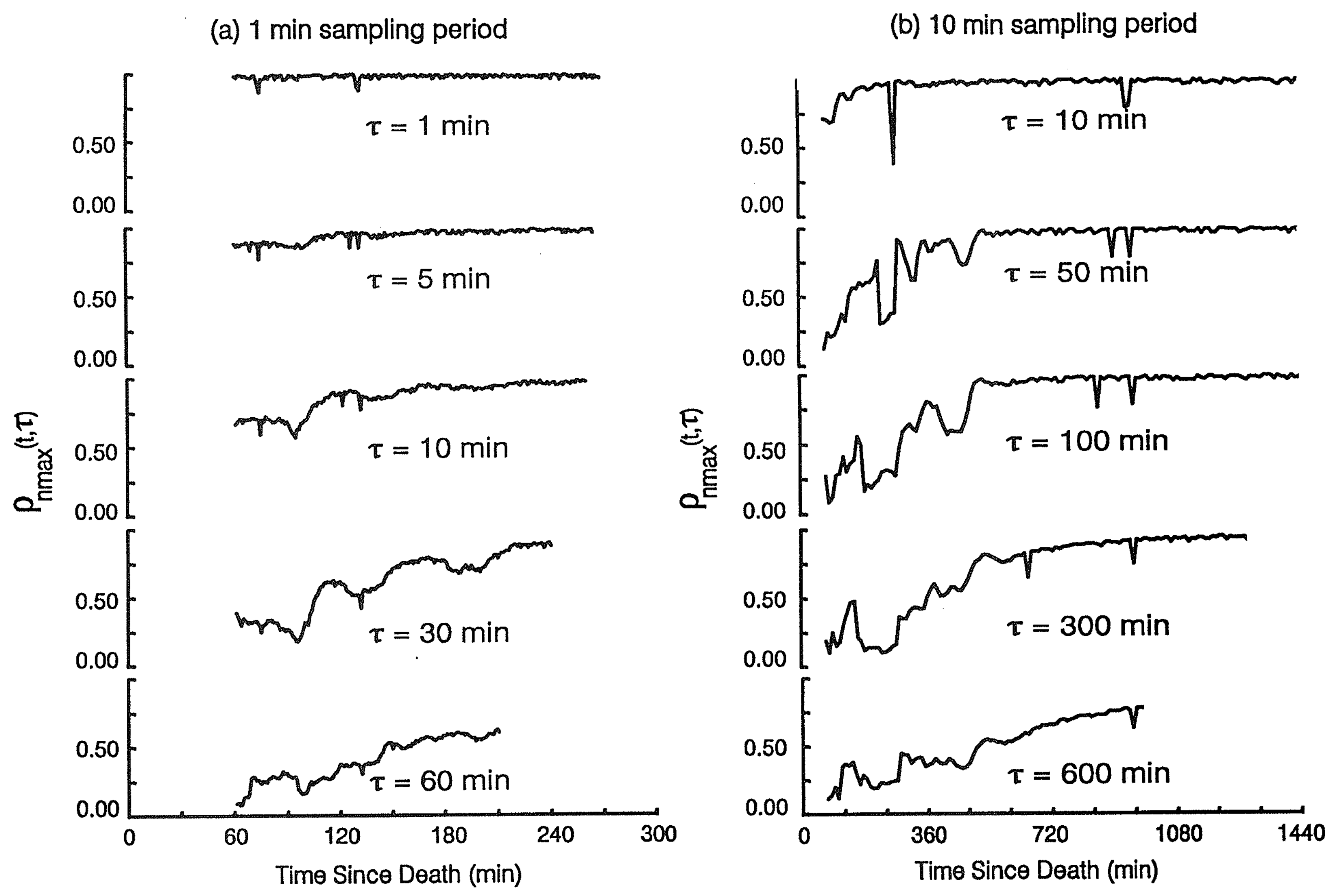
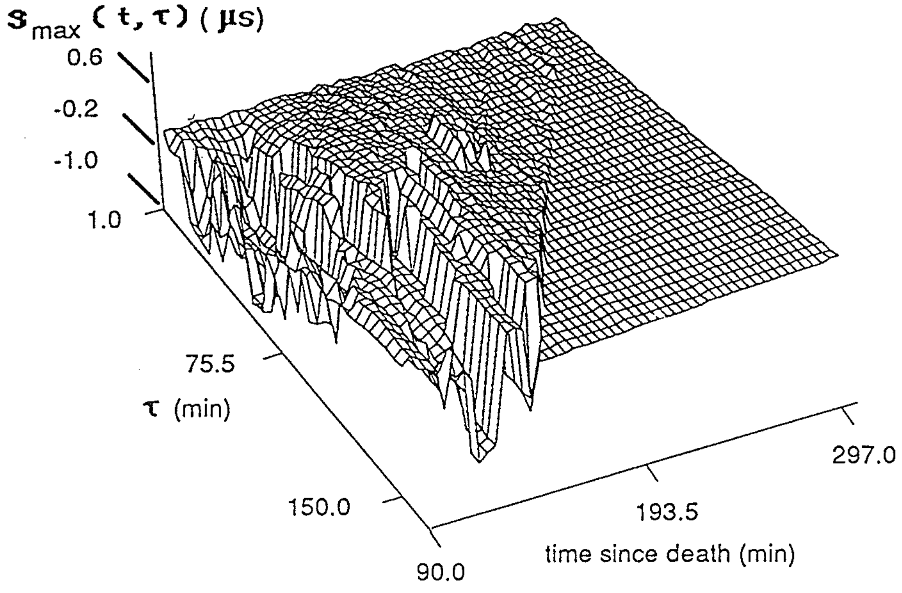


Figure 14. Maximum correlation coefficient for the porcine sample on Oct. 2, 1990 for various τ vs. time since death for (a) 1 min sampling periodand (b) 10 min sampling period.

(a)



(b)

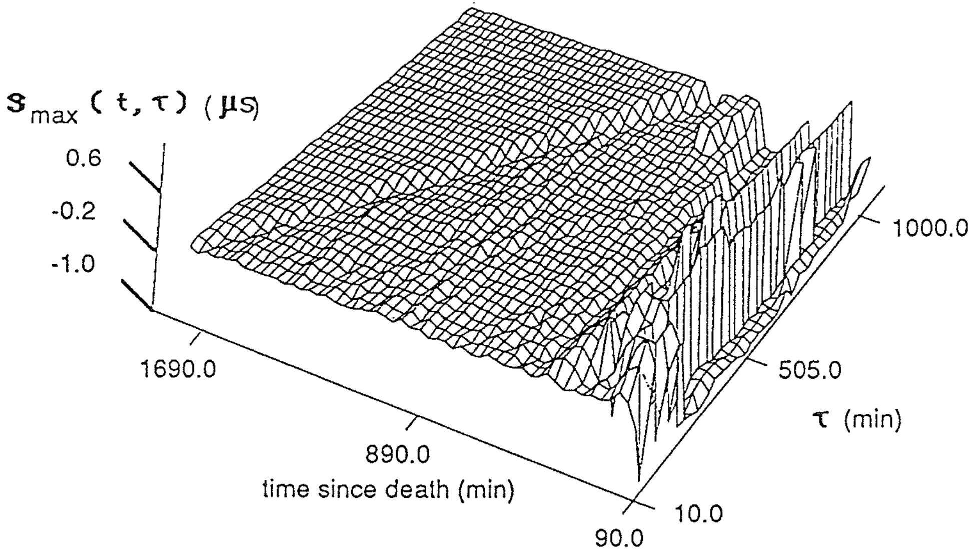


Figure 15. Shift for maximum correlation vs. time since death vs. tau for the porcine sample on Oct. 2, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

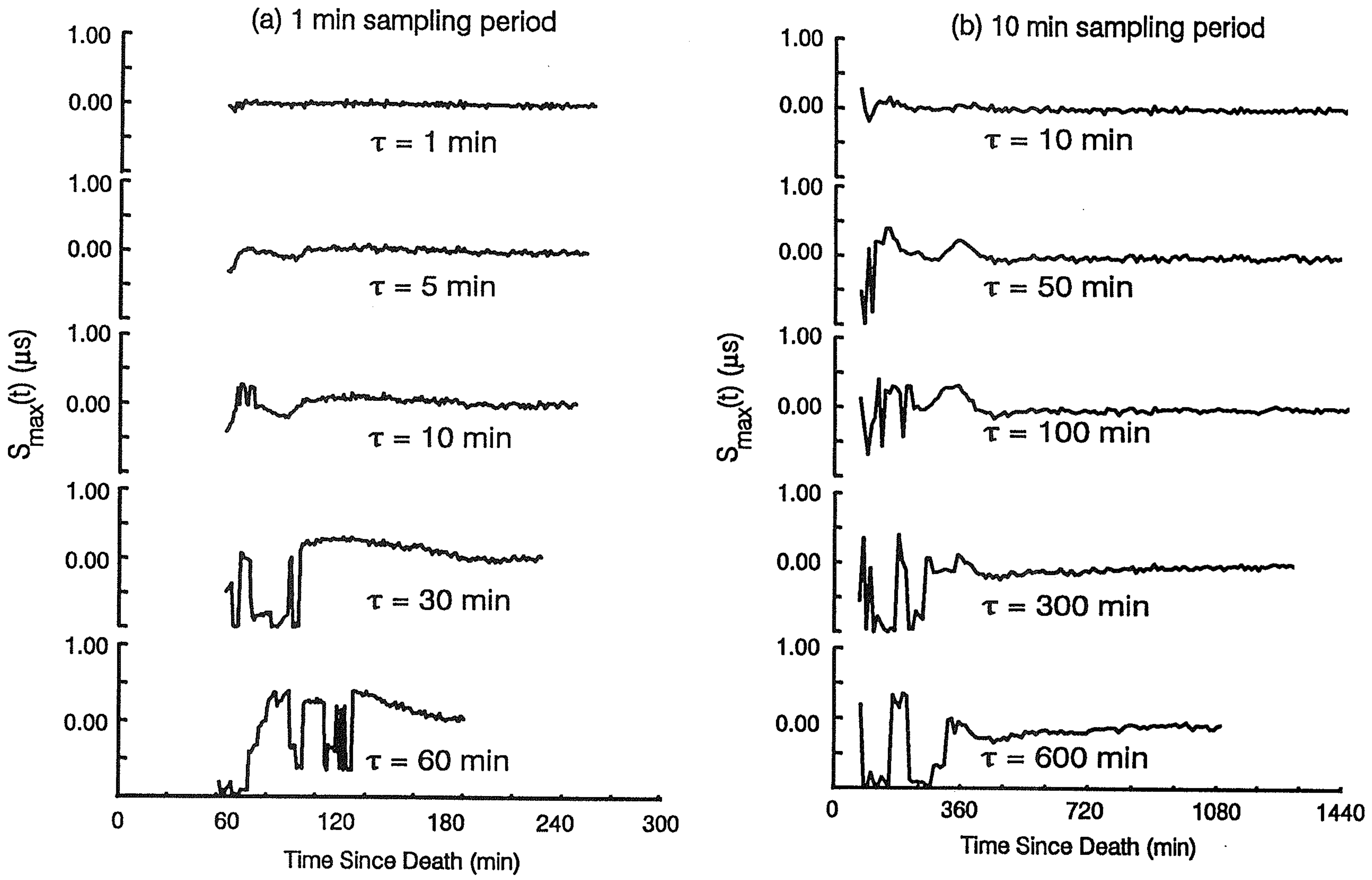
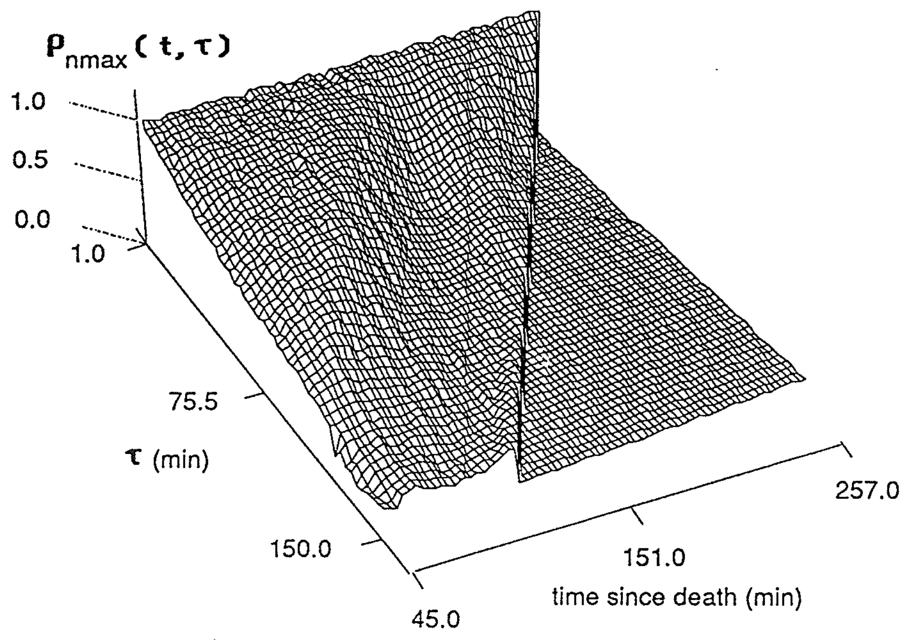


Figure 16. Shift for maximum correlation coefficient for the porcine sample on Oct 2, 1990 for various τ vs. time since death for (a) 1 min sampling periodand (b) 10 min sampling period.

(a)



(b)

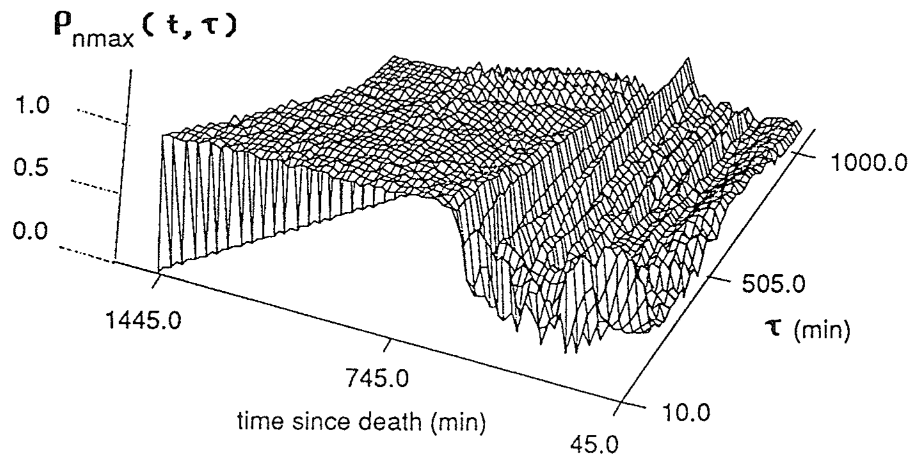


Figure 17. Plot of the maximum correlation coefficient vs. time from death vs. tau for the porcine sample on Dec. 10, 1990 for (a) 1 min sampling period and (b) 10 min sampling period.

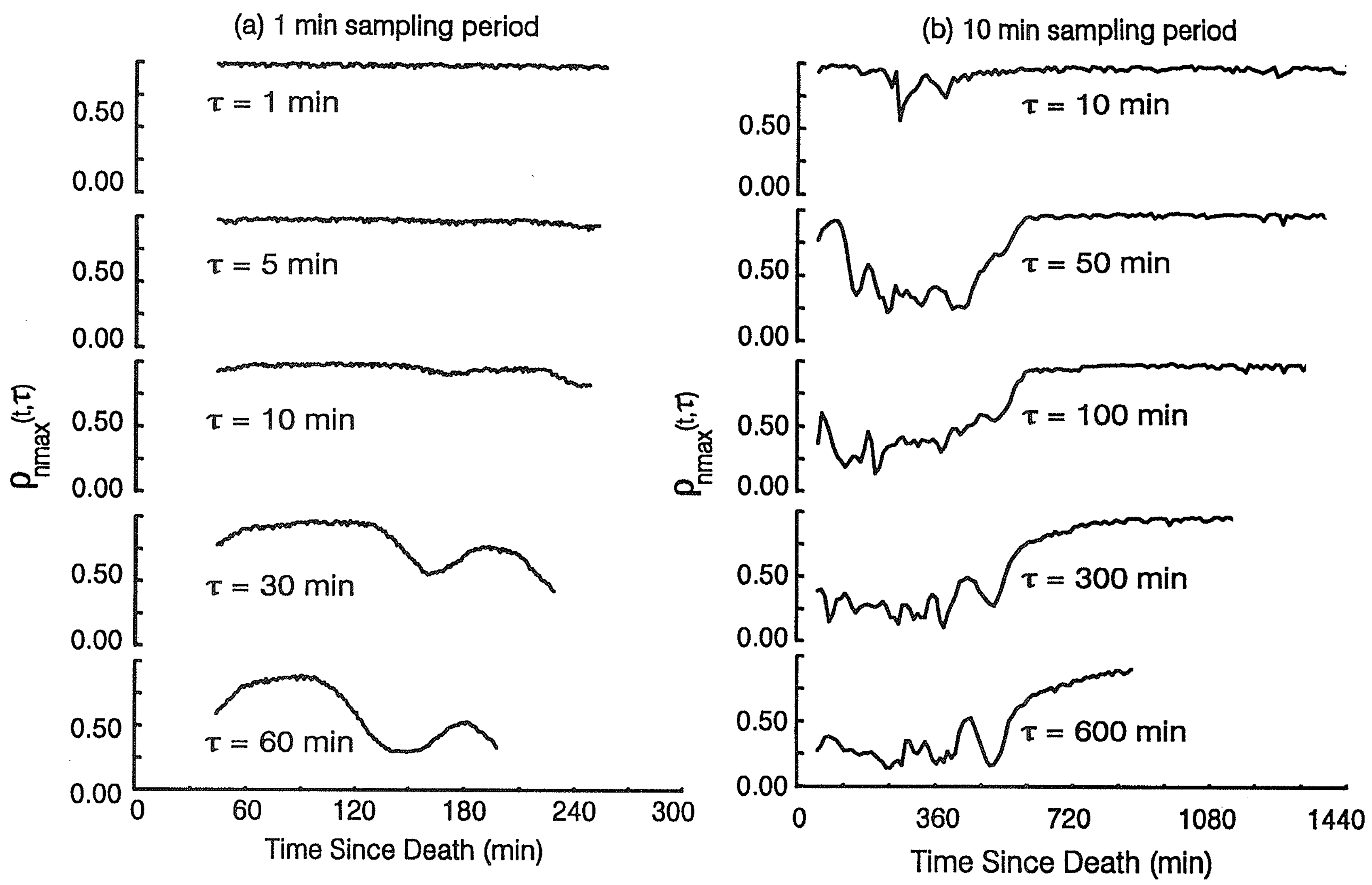
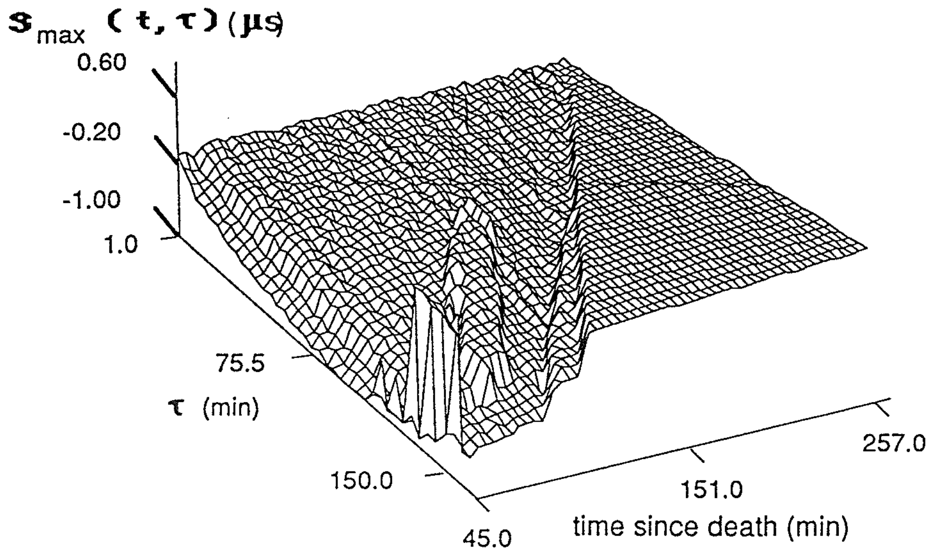


Figure 18. Maximum correlation coefficient for the porcine sample on Dec. 10, 1990 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

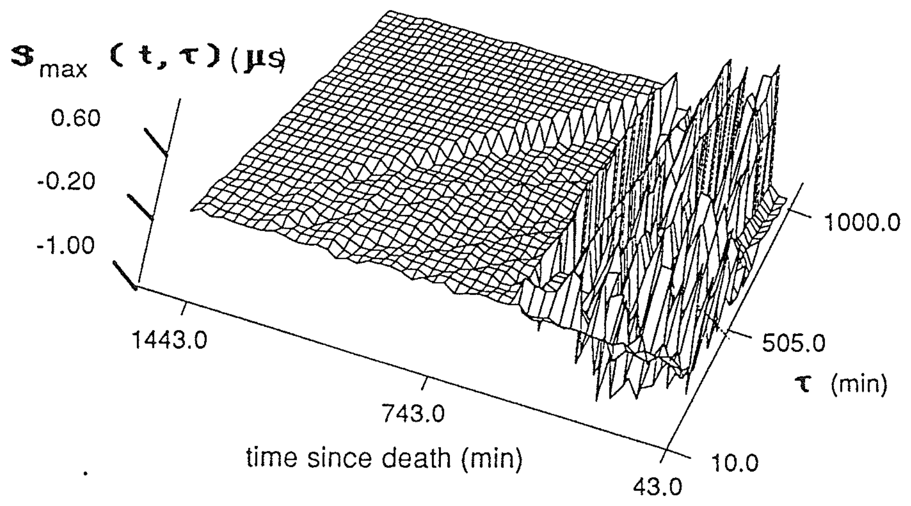


Figure 19. Shift for maximum correlation vs. time since death vs. tau for the porcine sample on Dec. 10, 1990 for (a) 1 min sampling period and (b) 10 min sampling period.

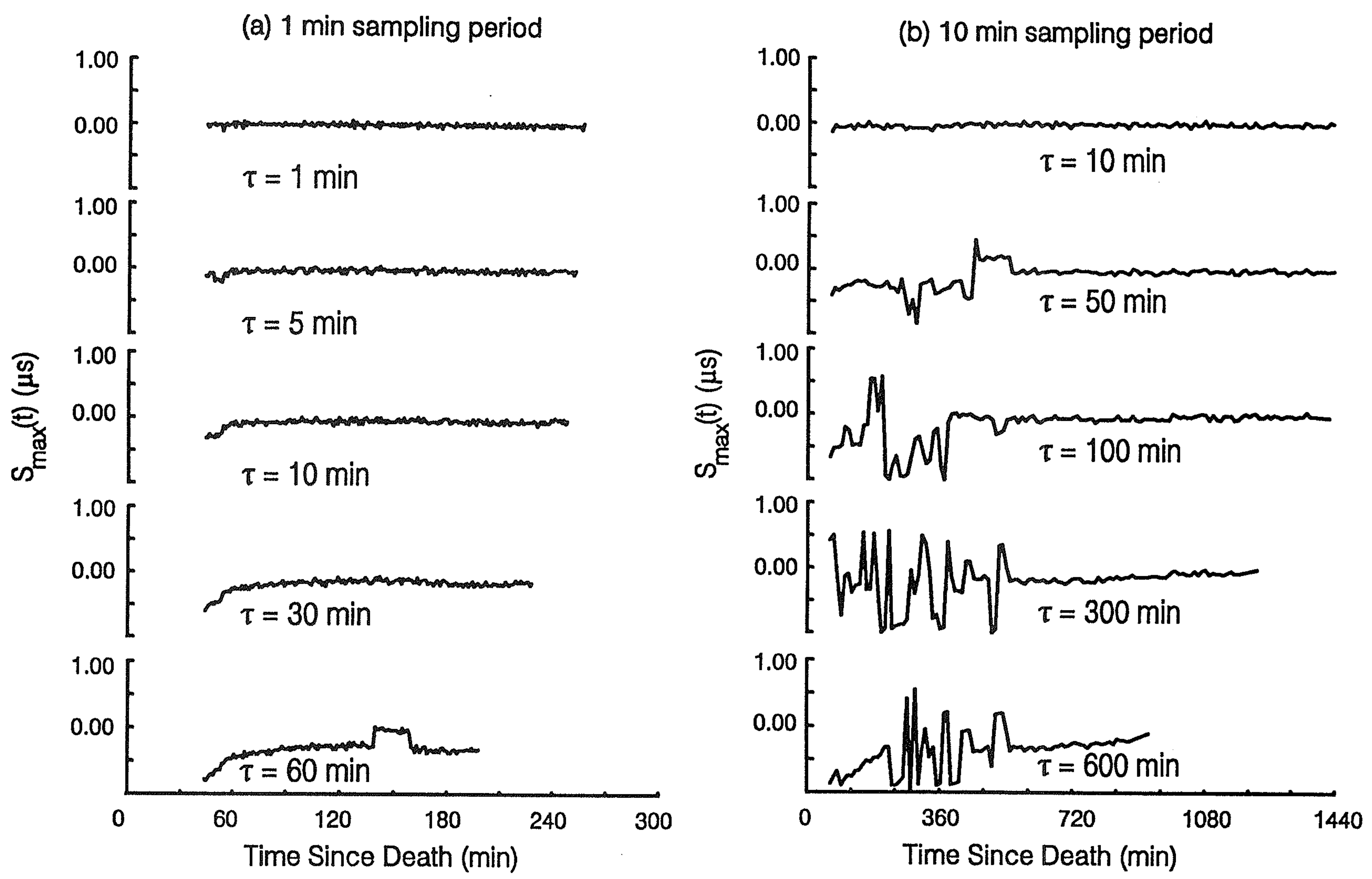
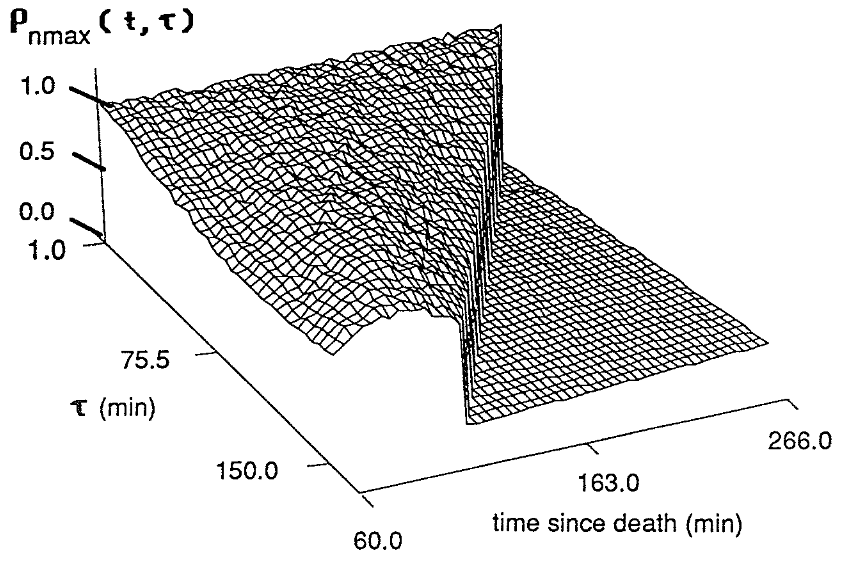


Figure 20. Shift for maximum correlation coefficient for the porcine sample on Dec. 10, 1990 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

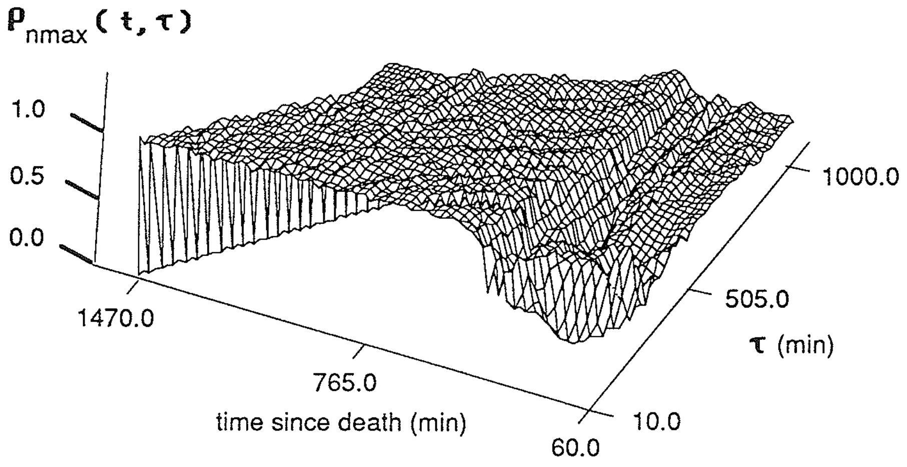


Figure 21. Plot of maximum correlation coefficient vs. time from death vs. tau for the porcine sample on June 11, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

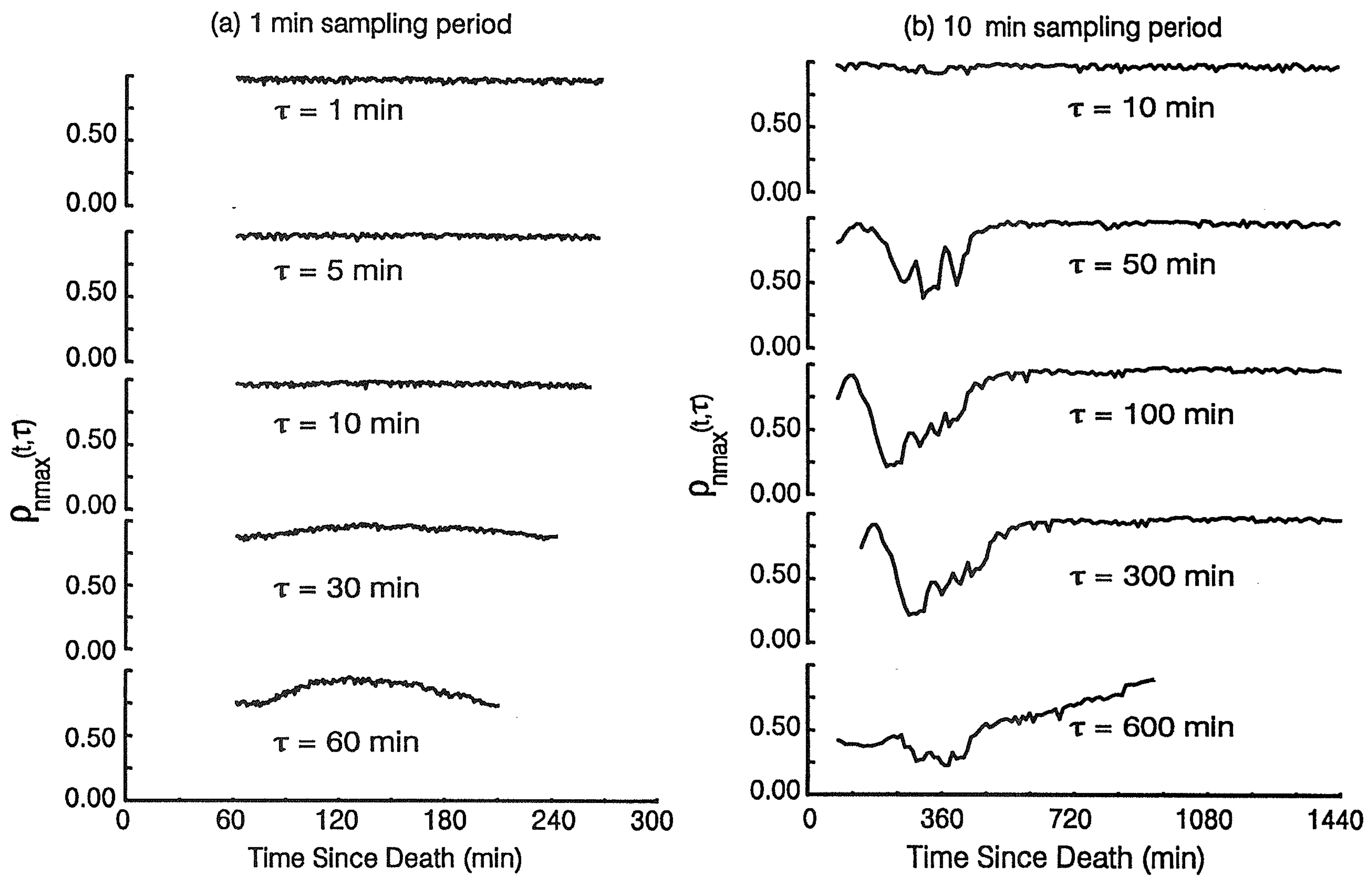
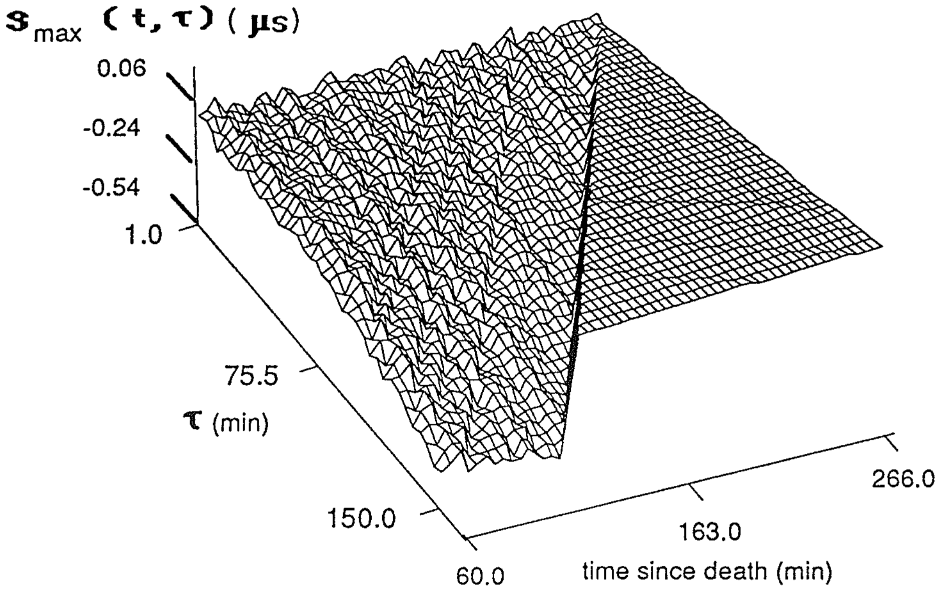


Figure 22. Maximum correlation coefficient for the porcine sample on June 11, 1991 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

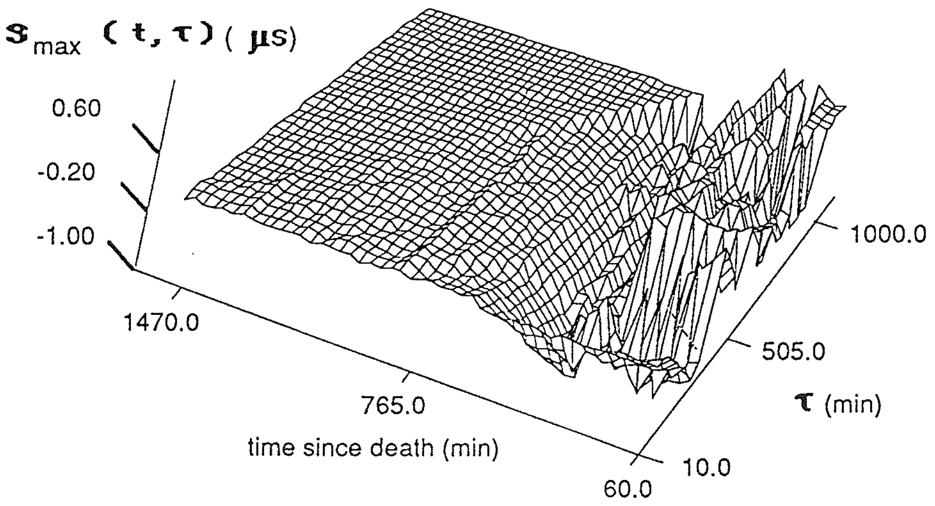


Figure 23. Shift for maximum correlation vs. time since death vs. tau for the porcine sample on June 11, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

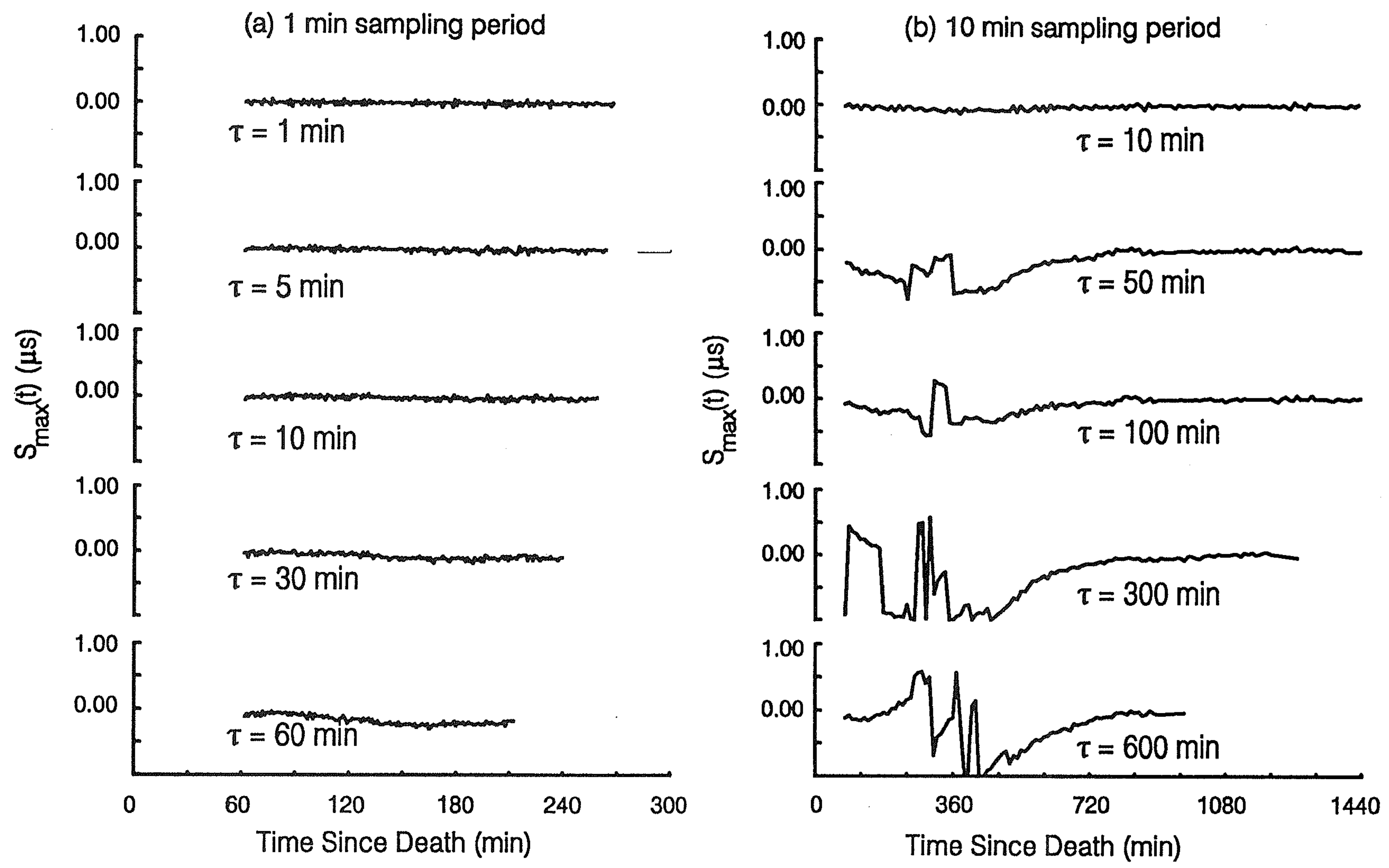
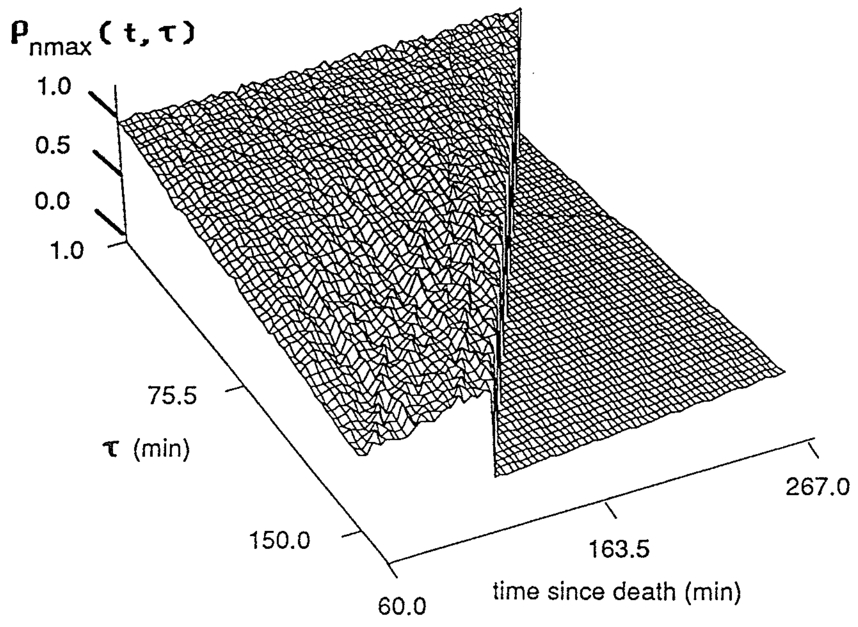


Figure 24. Shift for maximum correlation coefficient for the porcine sample on June 11, 1990 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

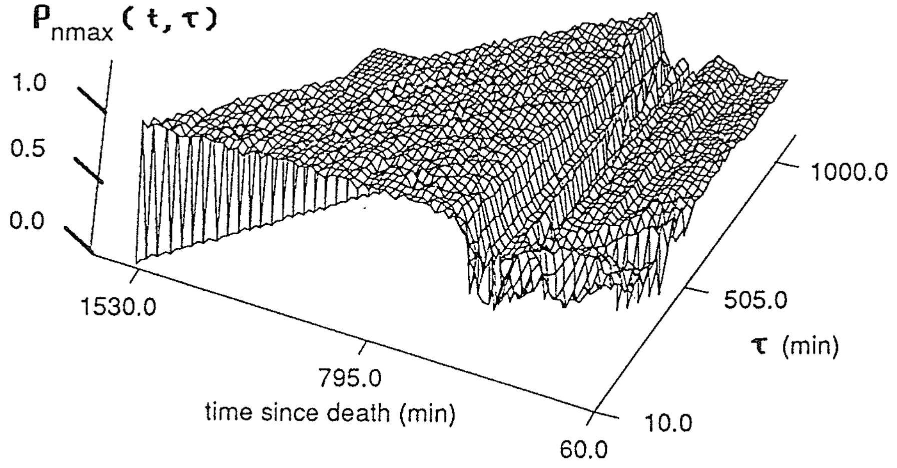


Figure 25. Plot of maximum correlation coefficient vs. time since death vs. tau for the porcine sample on July 9, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

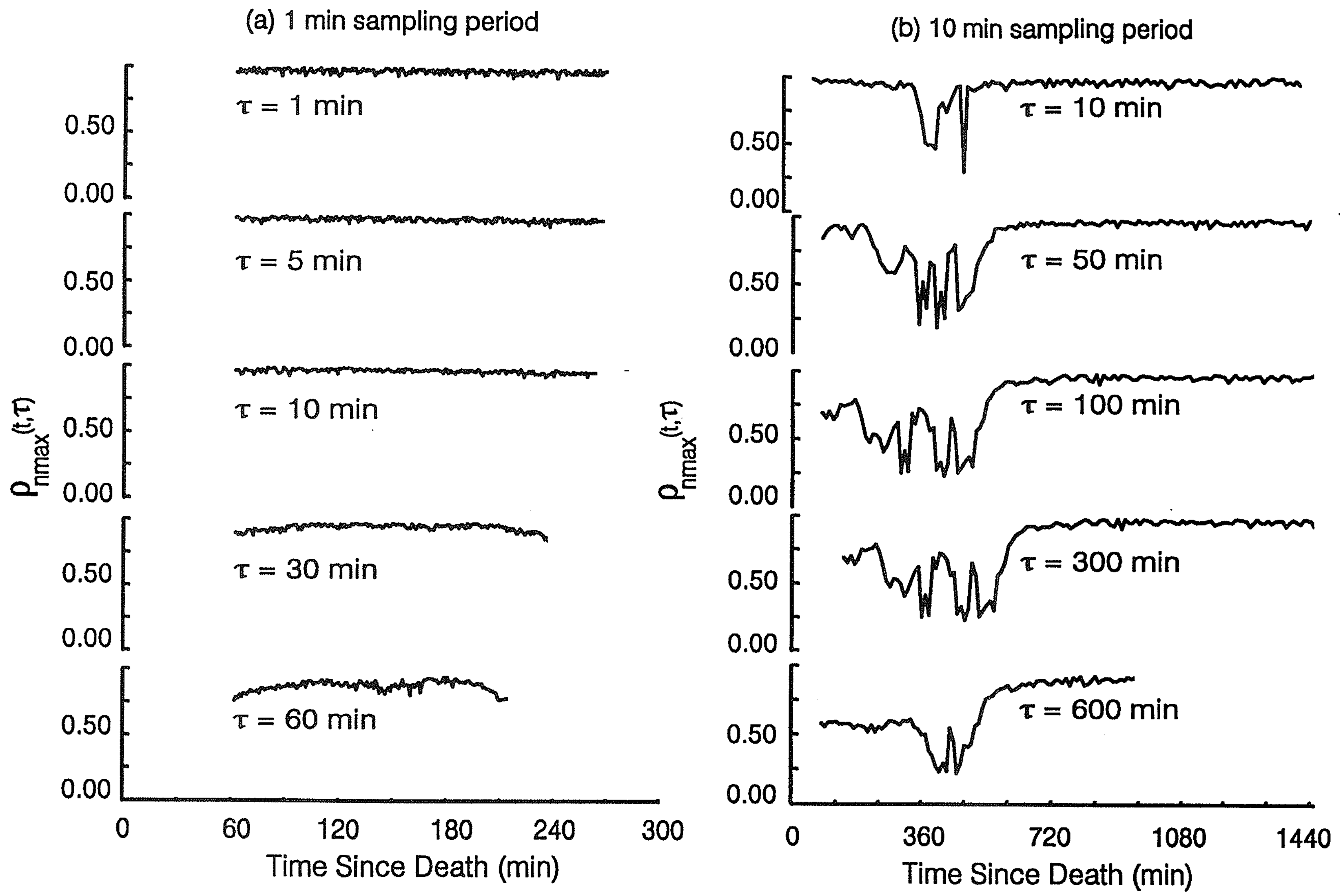
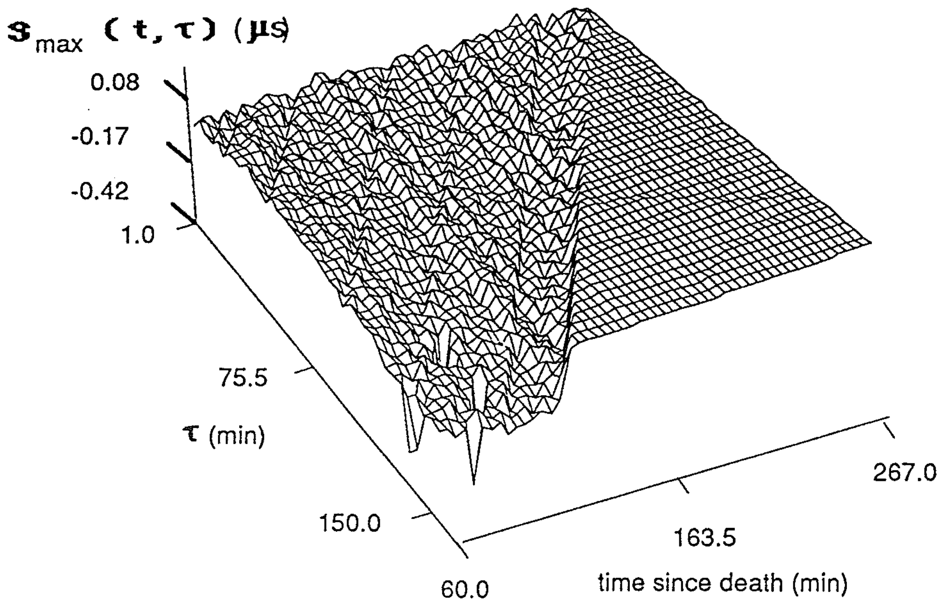


Figure 26. Maximum correlation coefficient for the porcine sample on July 9, 1991 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling periods

(a)



(b)

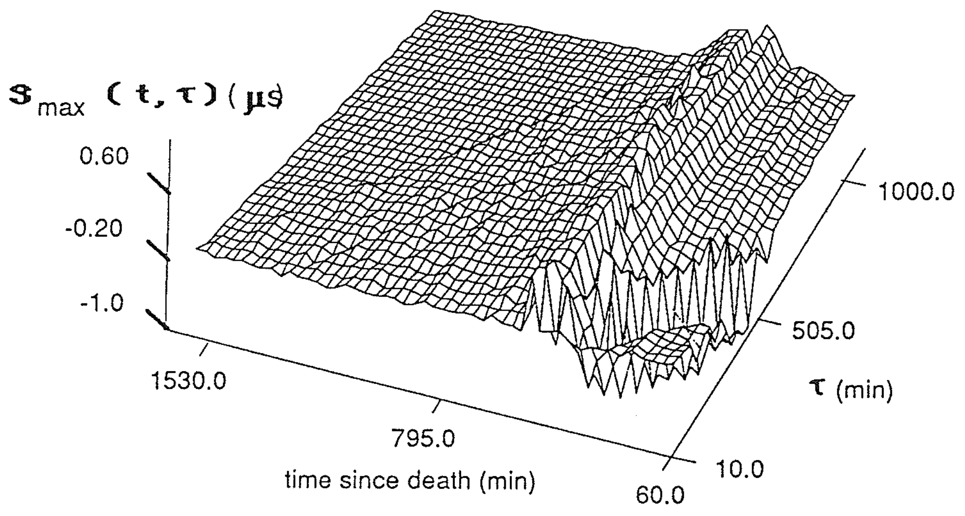


Figure 27. Shift for maximum correlation vs. time since death vs. tau for the porcine sample on July 9, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

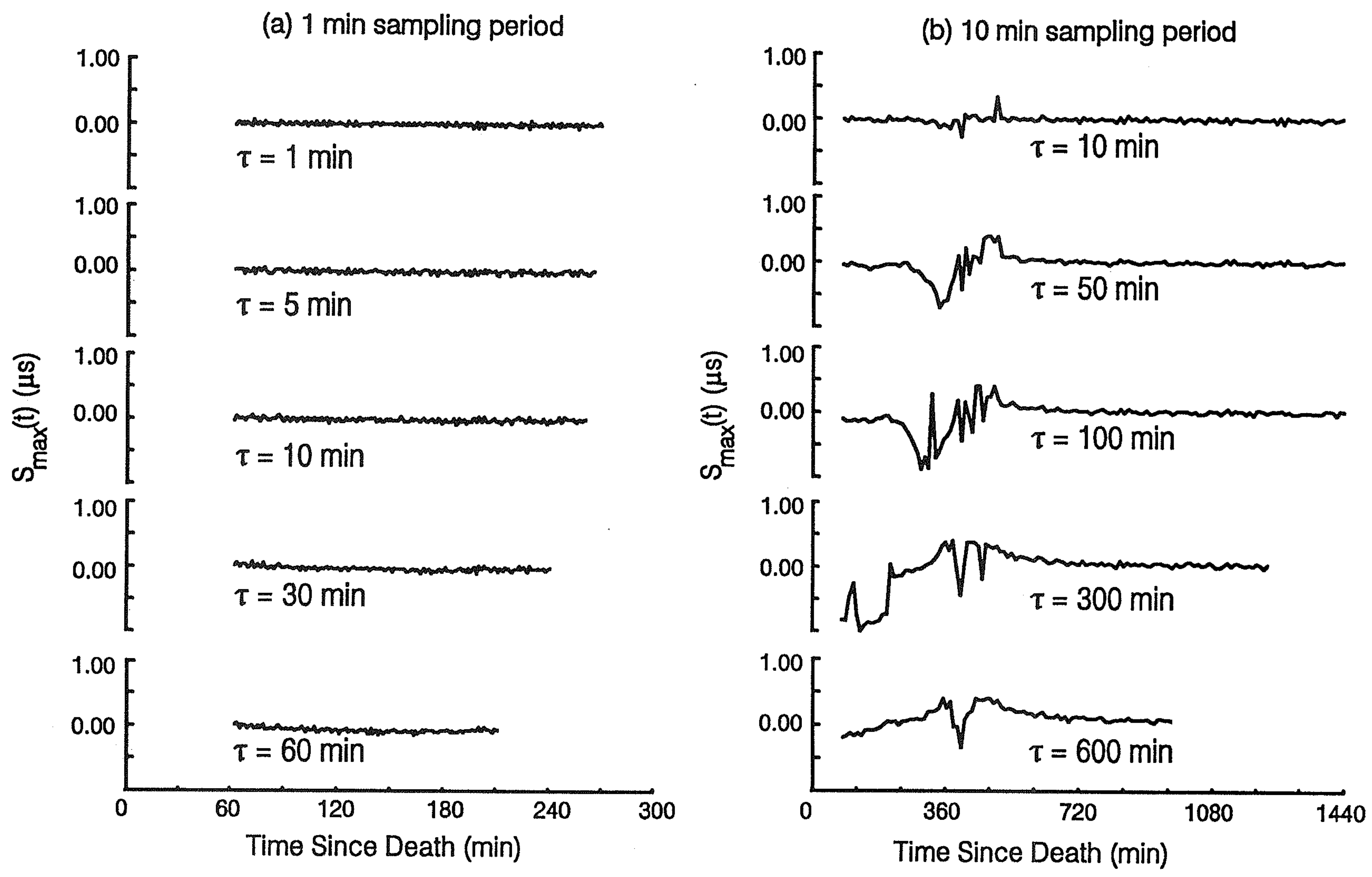
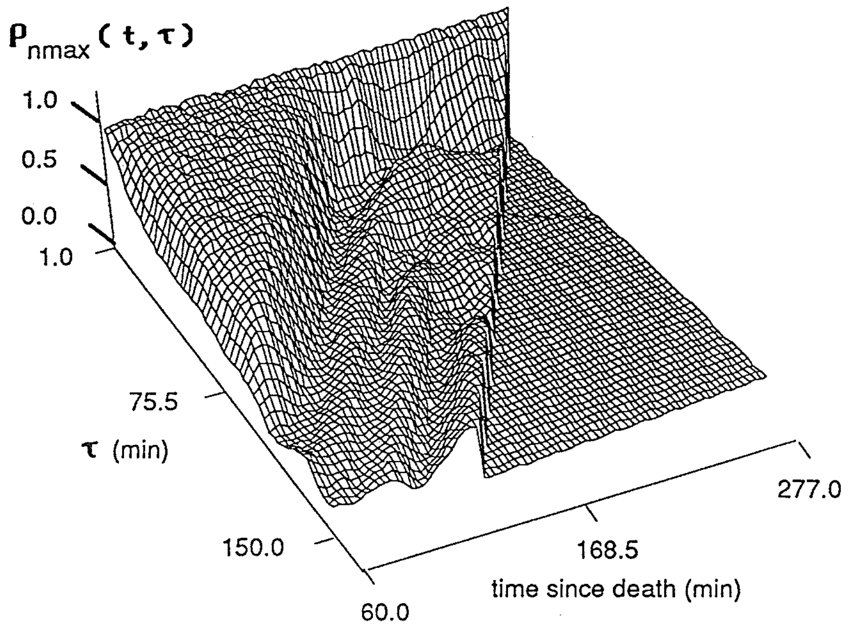


Figure 28. Shift for maximum correlation coefficient for the porcine sample on July 9, 1990 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

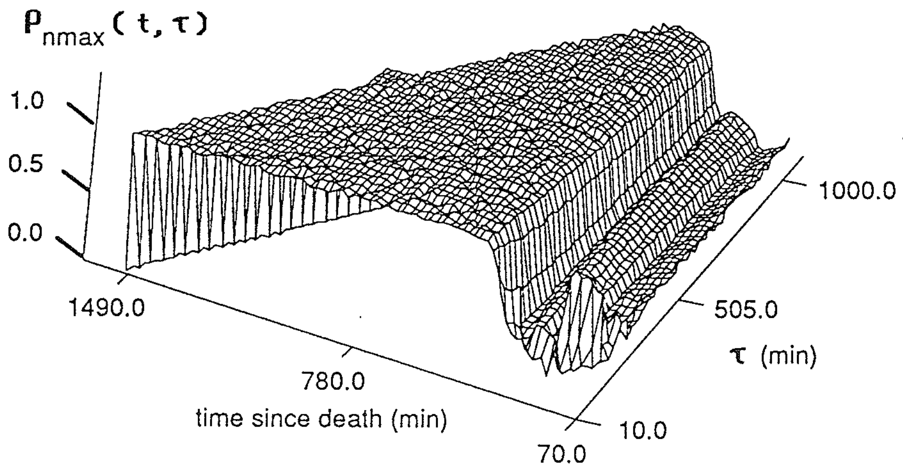


Figure 29. Plot of maximum correlation coefficient vs. time since death vs. tau for porcine sample on July 23, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

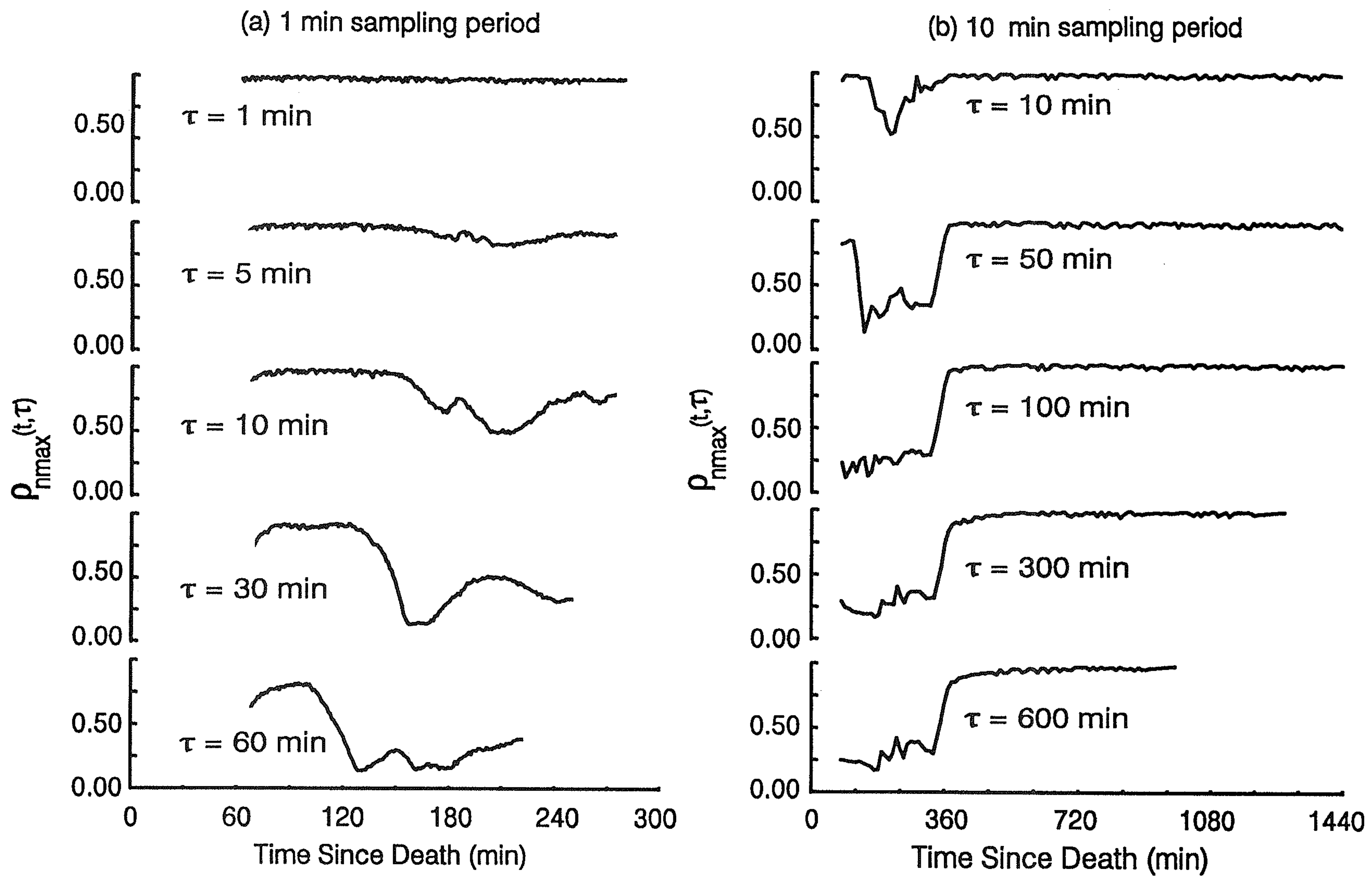
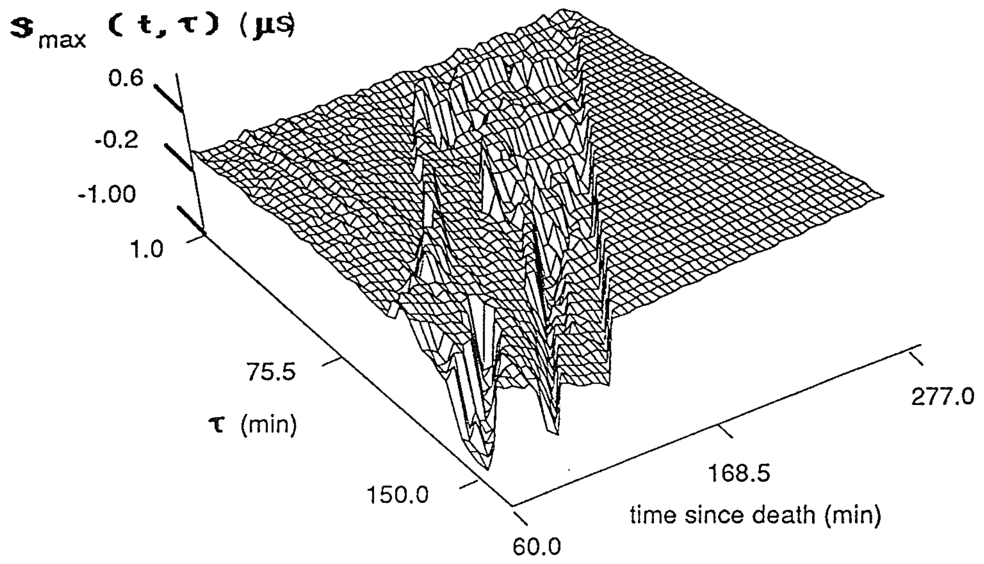


Figure 30. Maximum correlation coefficient for the porcine sample on July 23, 1991 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

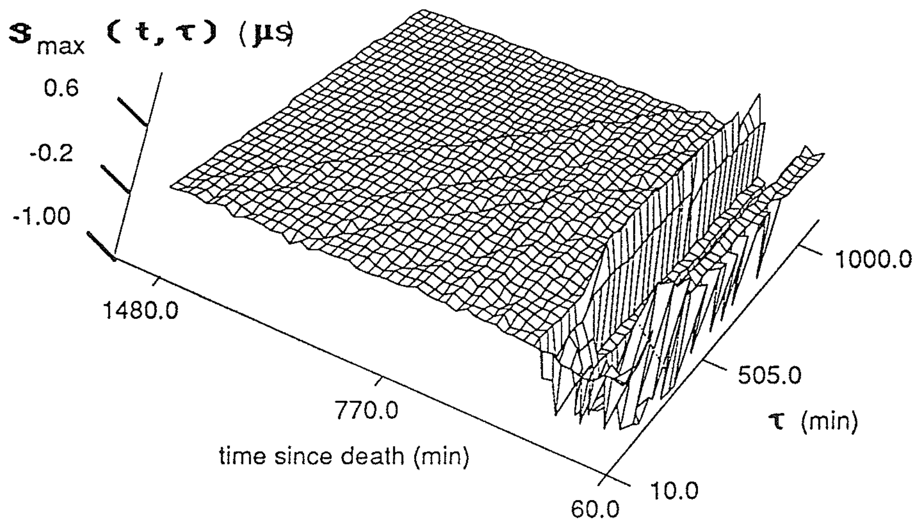


Figure 31. Shift for maximum correlation vs. time since death vs. tau for the porcine sample on July 23, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

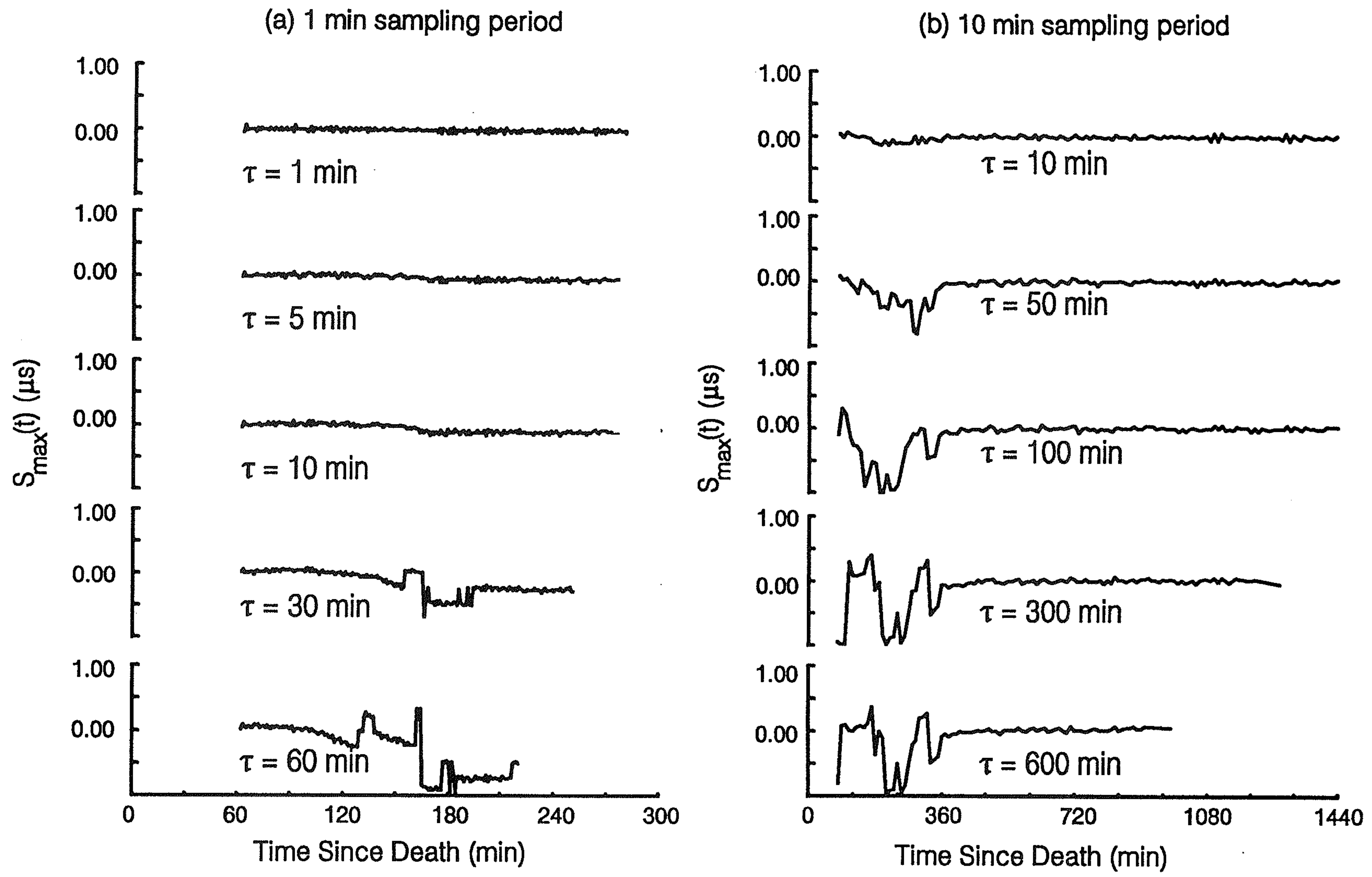
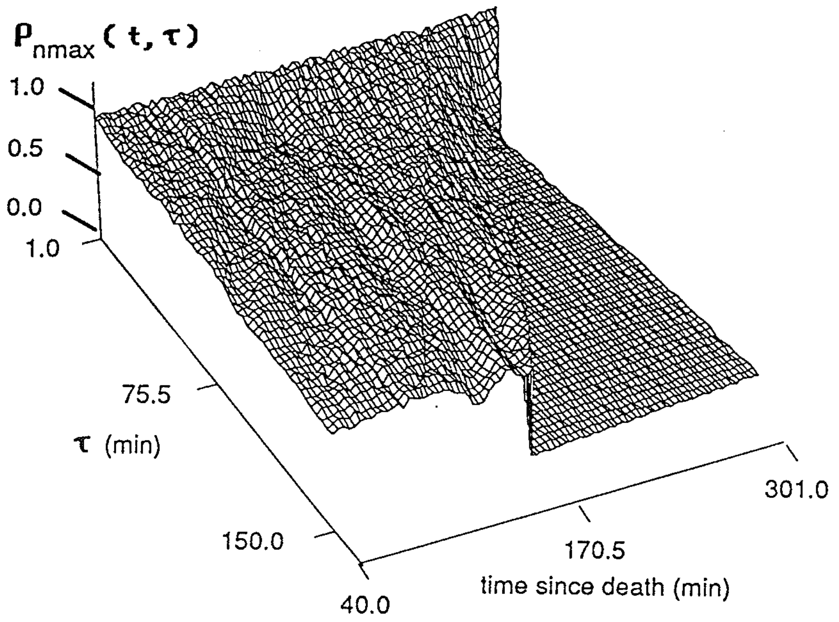


Figure 32. Shift for maximum correlation coefficient for the porcine sample on July 23, 1990 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

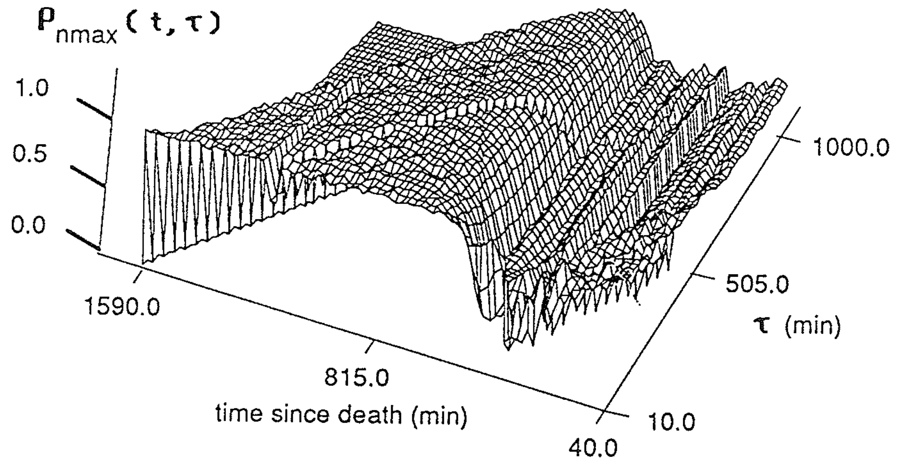


Figure 33. Plot of maximum correlation coefficient vs. time since death vs. tau for the porcine sample on July 30, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

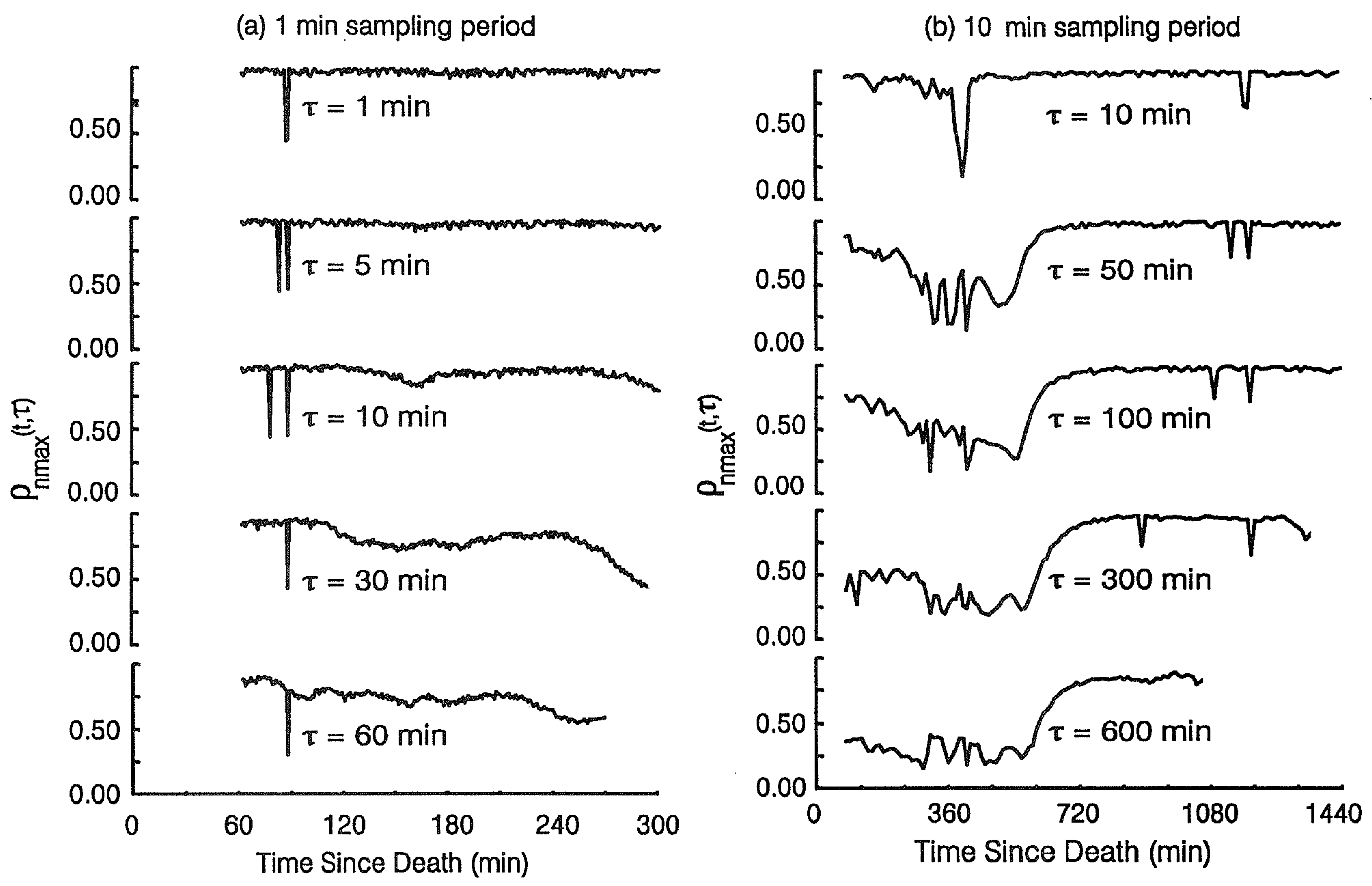
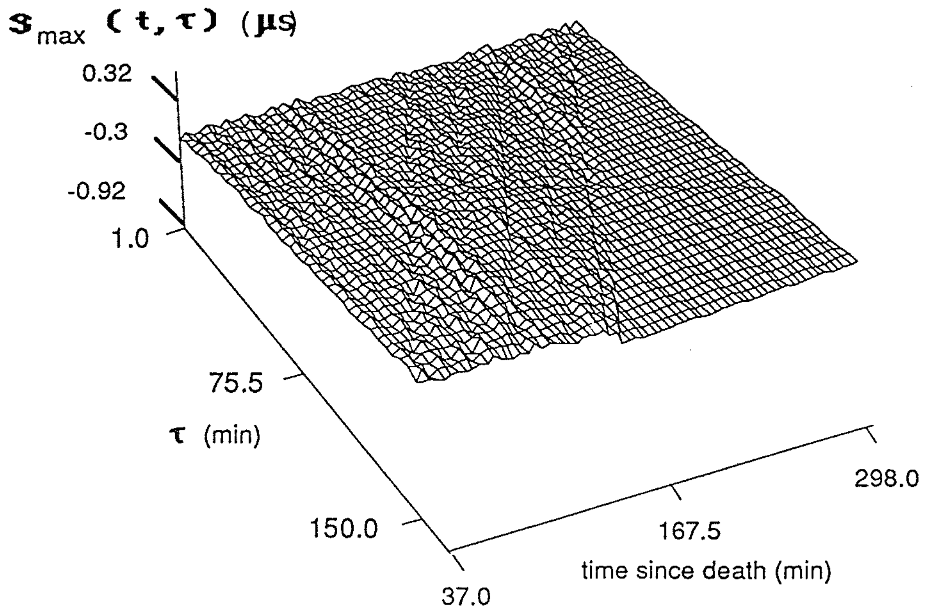


Figure 34. Maximum correlation coefficient for the porcine sample on July 30, 1991 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

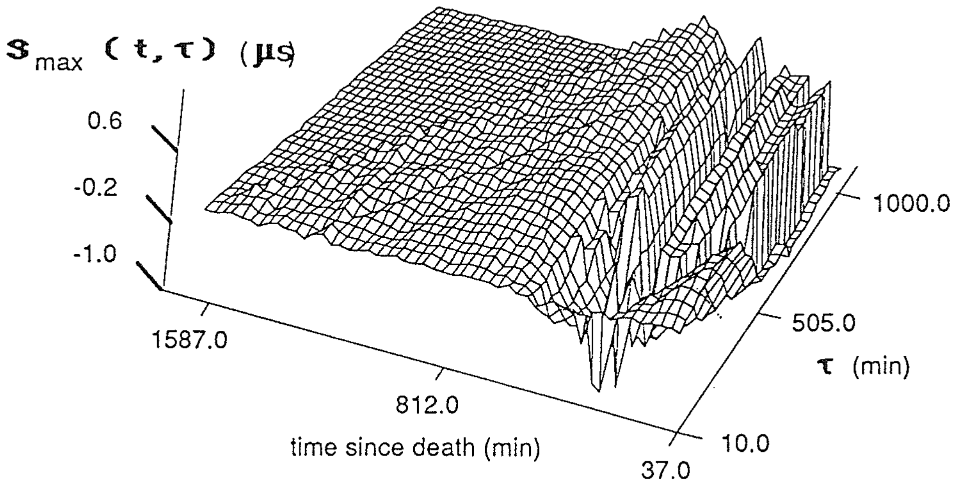


Figure 35. Shift for maximum correlation vs. time since death vs. tau for the porcine sample on July 30, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

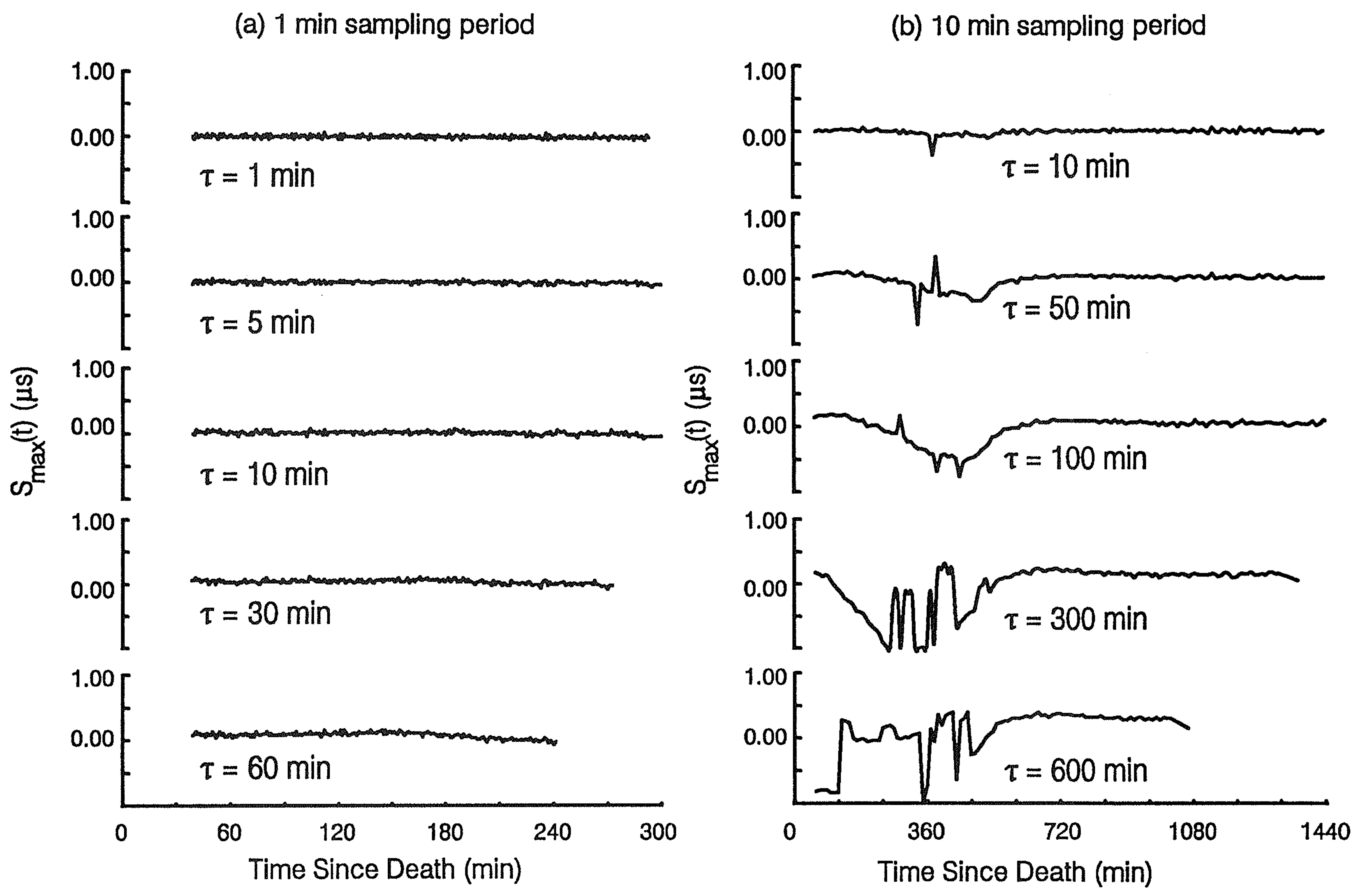
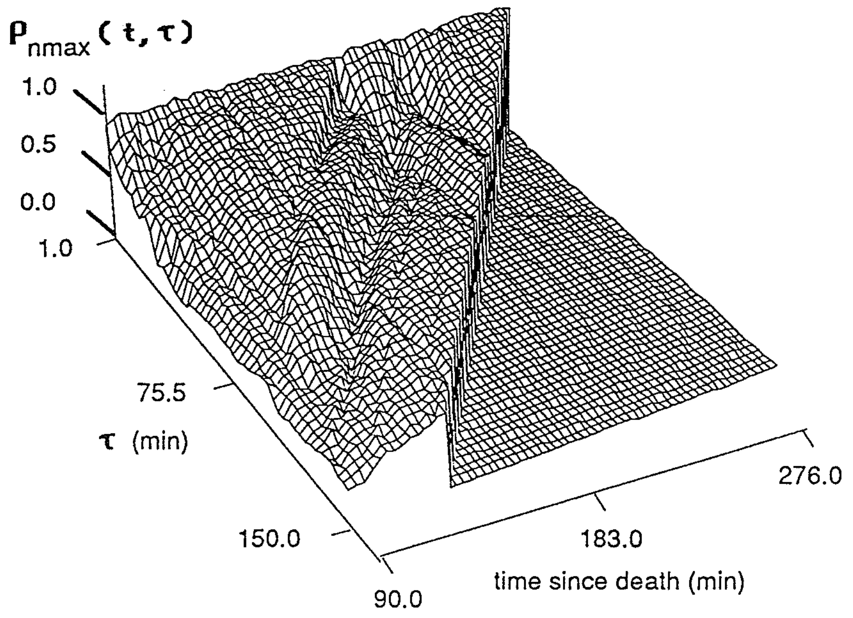


Figure 36. Shift for maximum correlation coefficient for the porcine sample on July 30, 1990 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min samplng period.

(a)



(b)

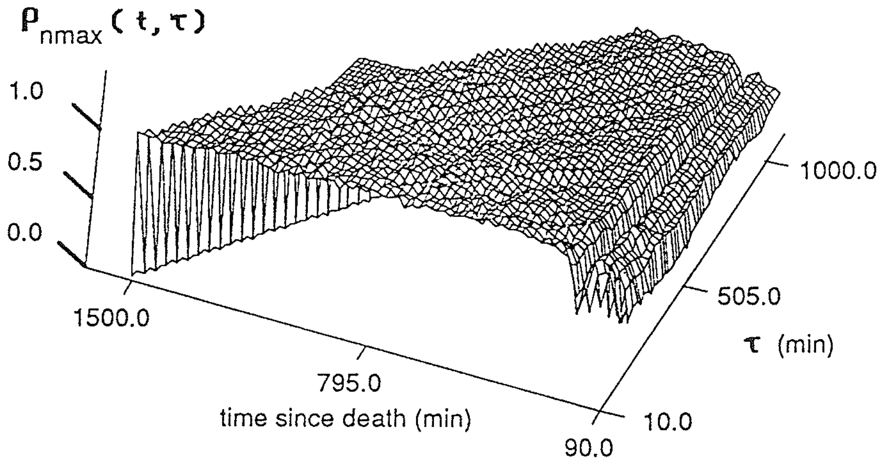


Figure 37. Plot of maximum correlation coefficient vs. time since death vs. tau for the porcine sample on Sept. 10, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

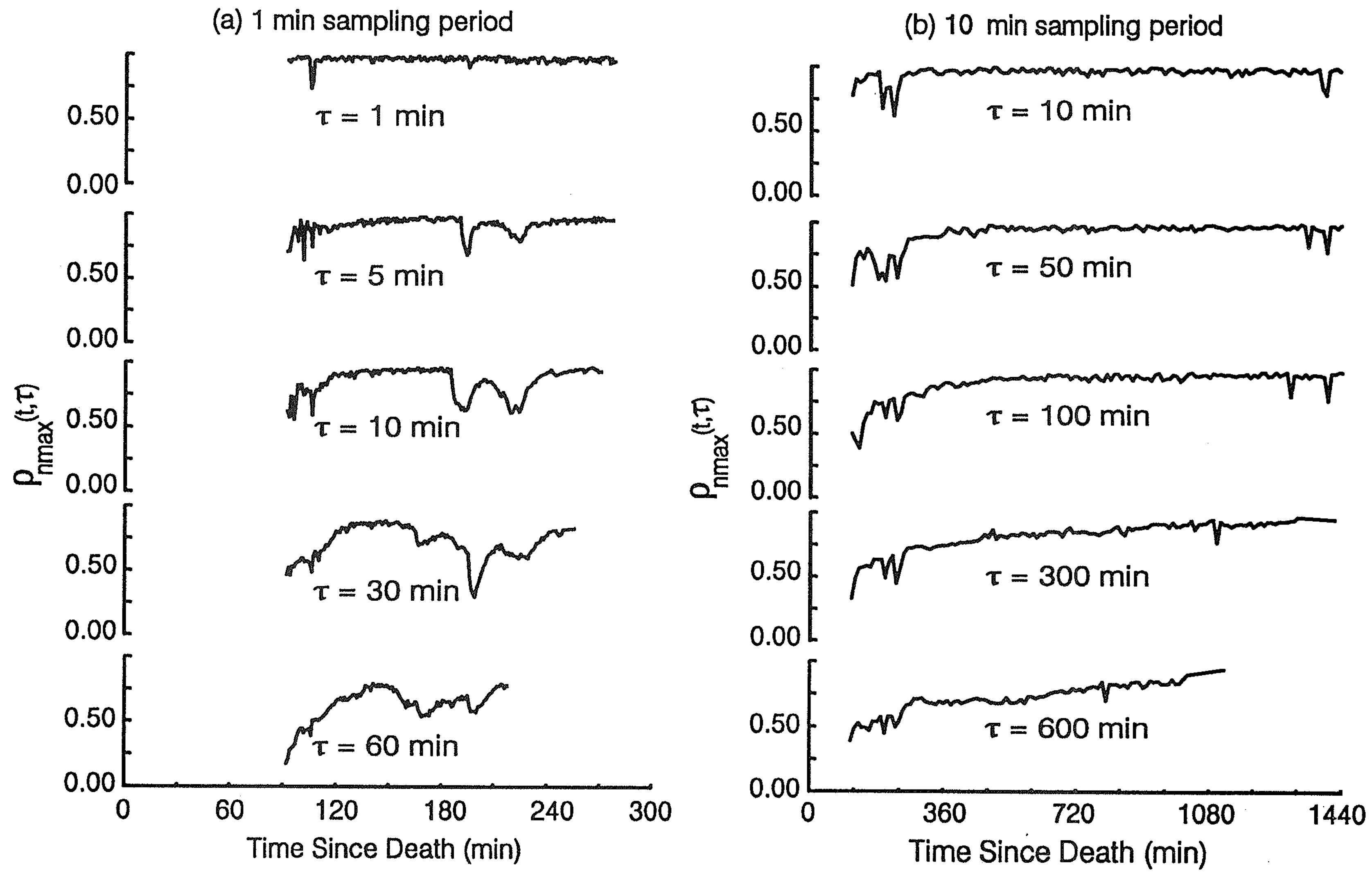
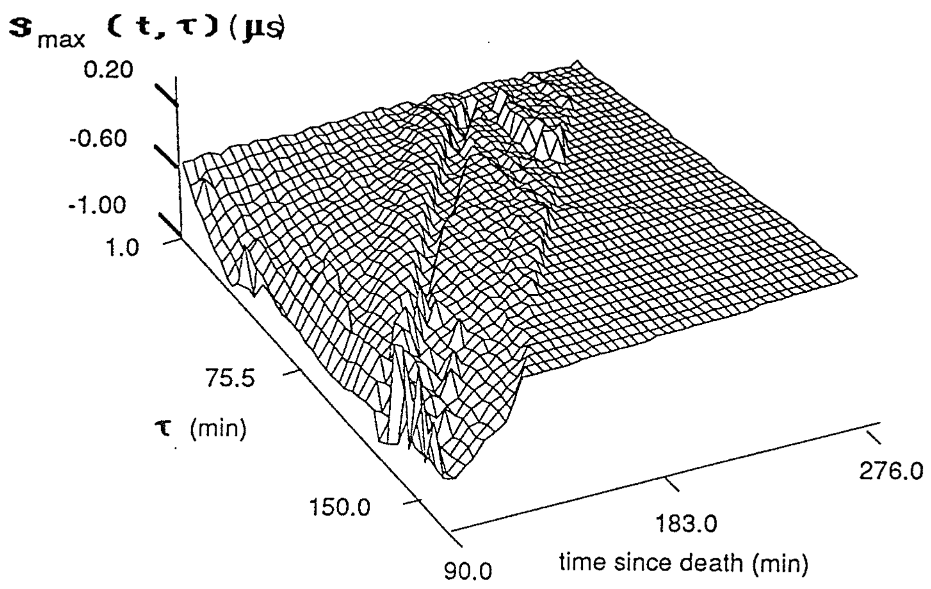


Figure 38. Maximum correlation coefficient for the porcine sample on Sept. 10, 1991 for various τ vs. time since death for (a) 1 min sampling periodand (b) 10 min sampling period.

(a)



(b)

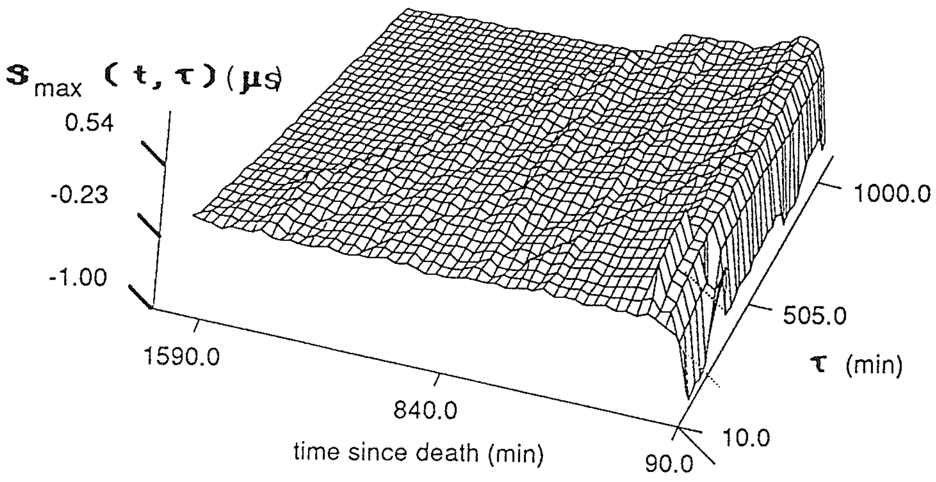


Figure 39. Shift for maximum correlation vs. time since death vs. tau for the porcine sample on Sept. 10, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

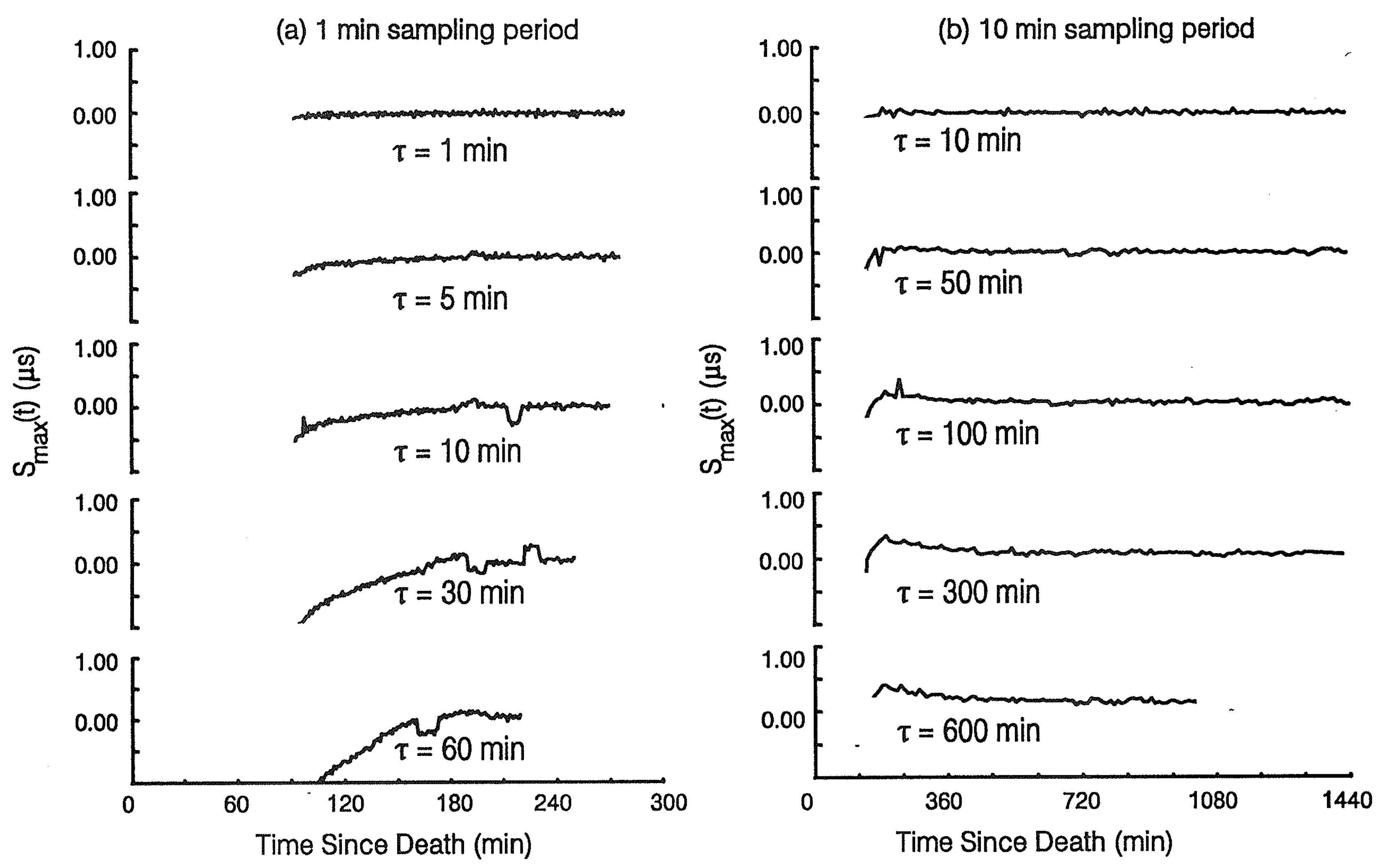
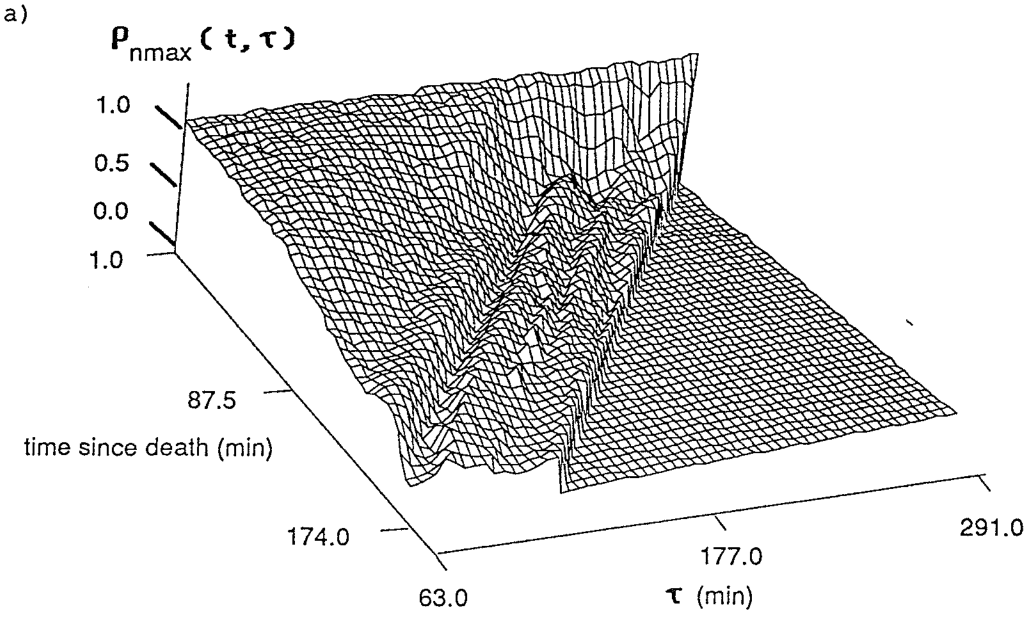


Figure 40. Shift for maximum correlation coefficient for the porcine sample on Sept. 10, 1990 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling periods.

(a)



(b)

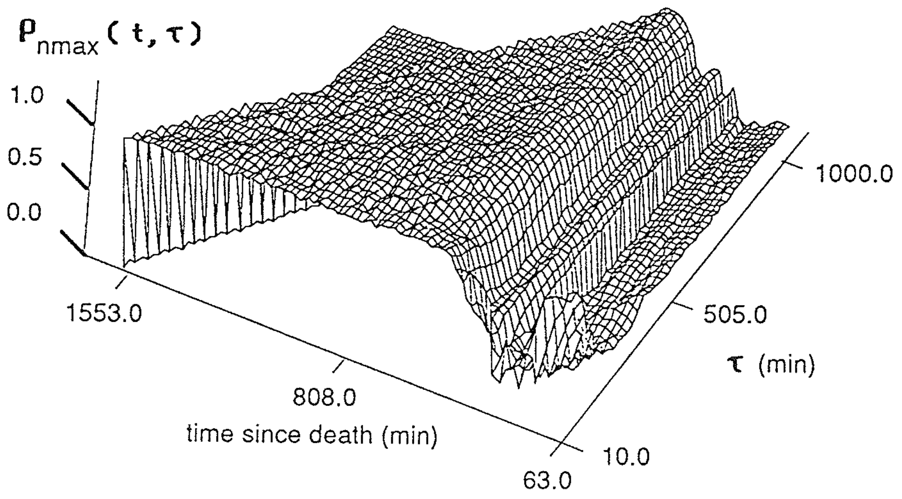


Figure 41. Plot of maximum correlation coefficient vs. time since death vs. tau for the porcine sample on Oct. 29, 1991 for (a) 1 min sampling period and (b) 10 min sampling period.

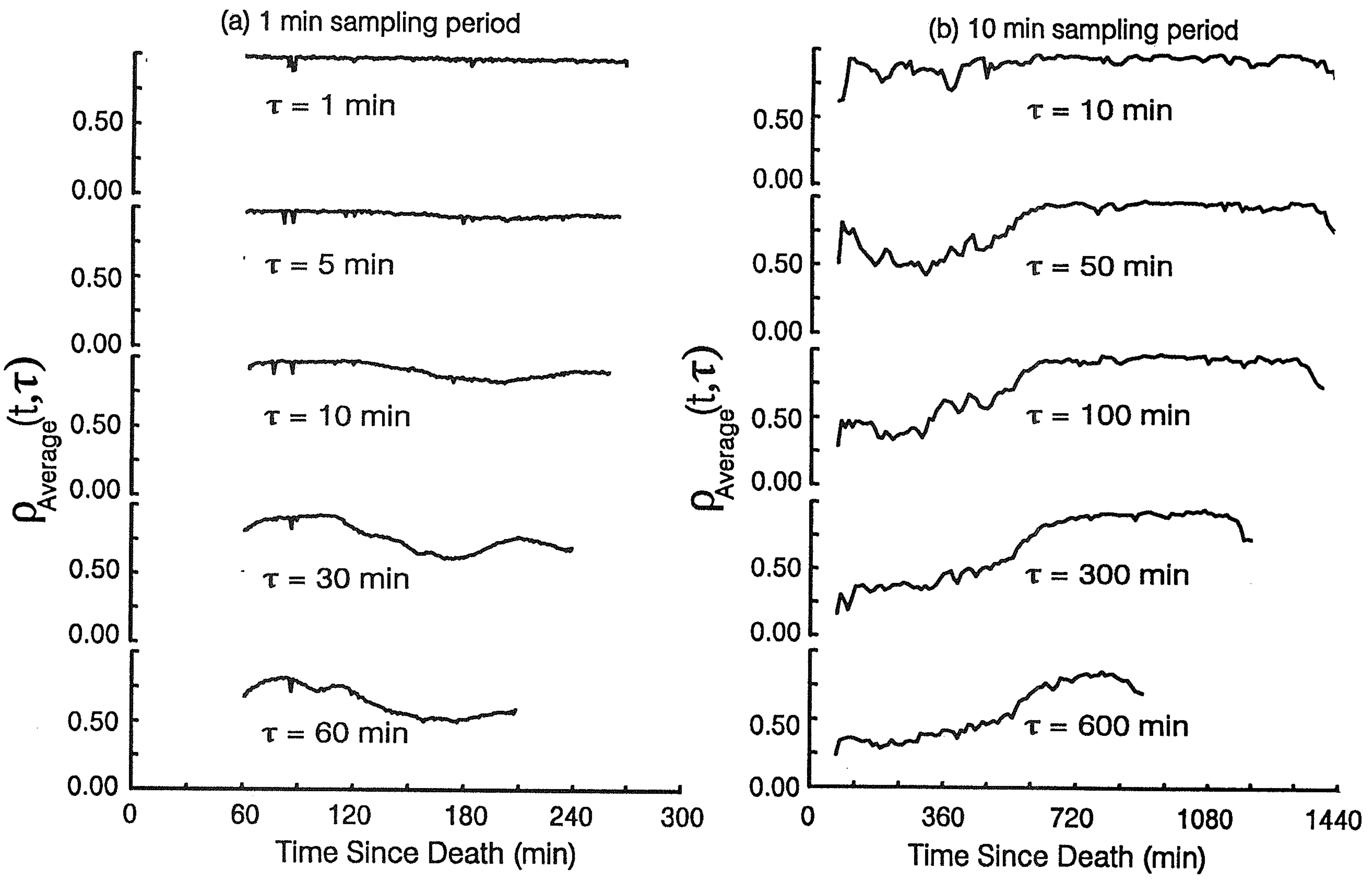
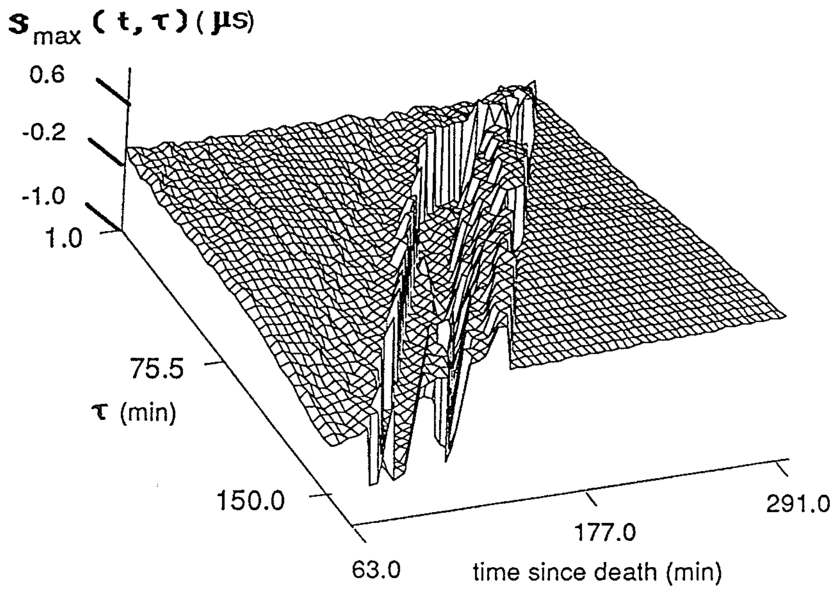


Figure 47. Average of eight correlation coefficient porcine data sets for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

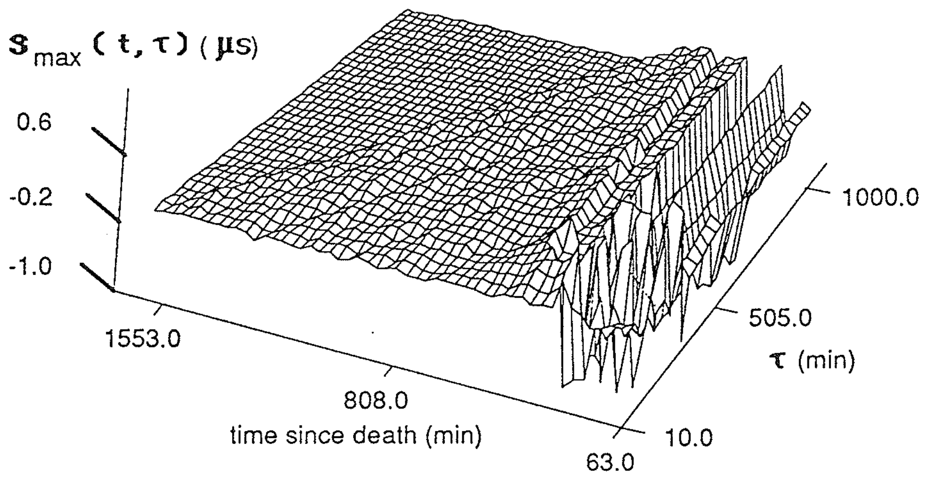


Figure 43. Shift for maximum correlation vs. time since death vs. tau for the porcine sample on Oct. 29, 1991 for (a) 1 min sampling period (b) 10 min sampling period.

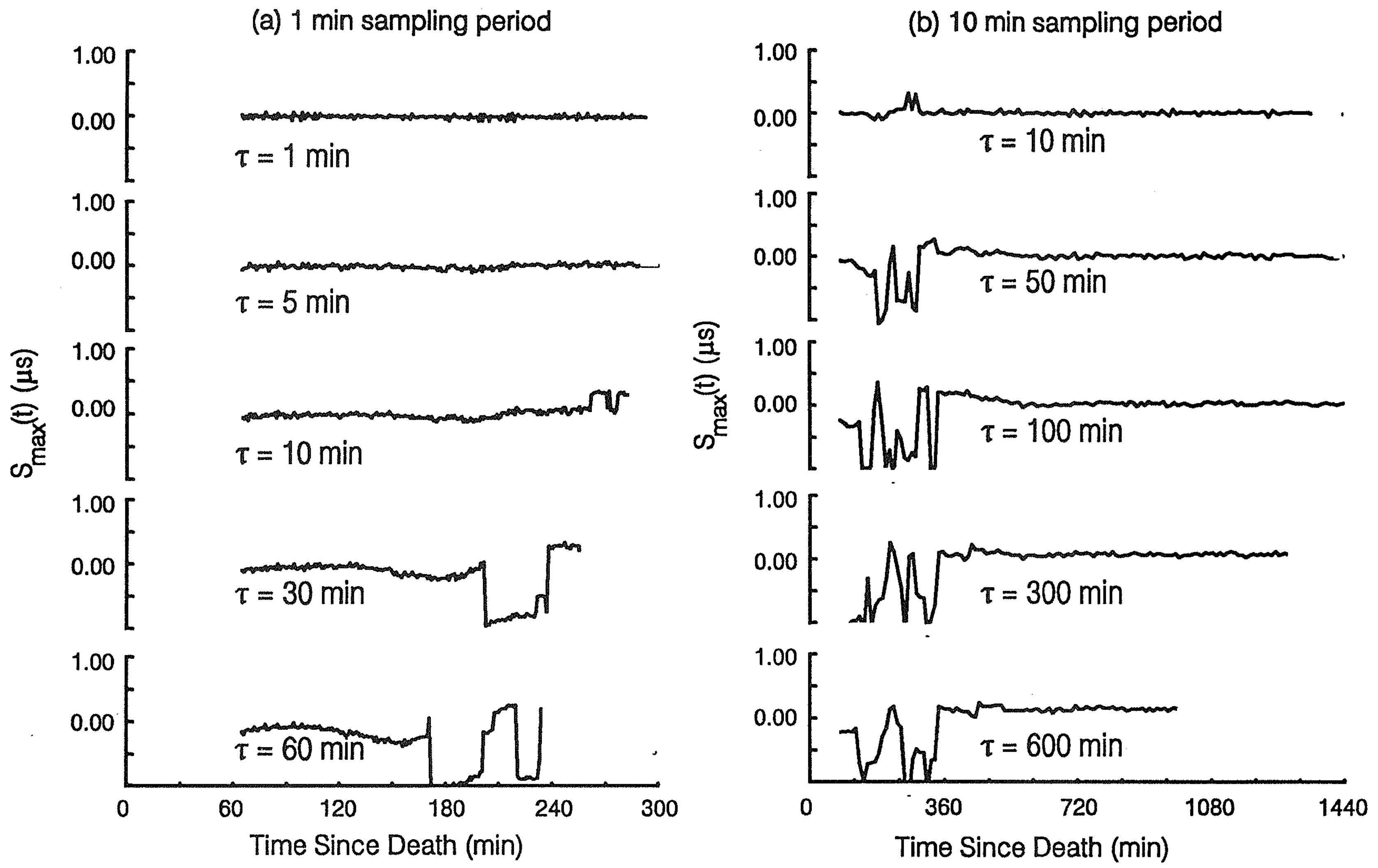


Figure 44. Shift for maximum correlation coefficient for the porcine sample on Oct. 29, 1990 for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

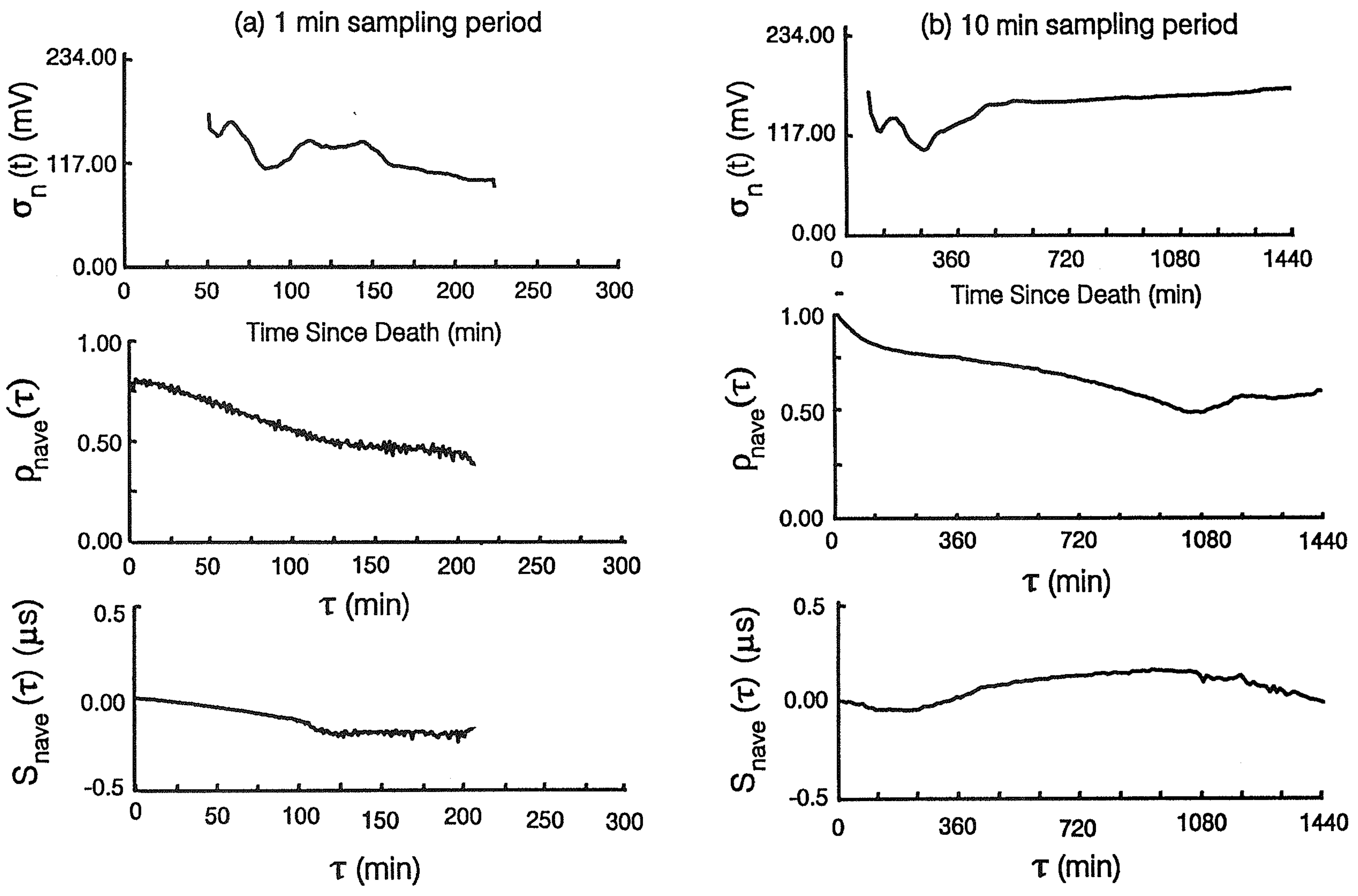


Figure 45. Standard Deviation vs. time since death, average correlation coefficient and average shift vs. τ for the porcine sample on Oct. 2, 1990 for (a) 1 min sampling period and (b) 10 min sampling period.

35(b), 39(b) and 43(b). Individual plots of the shift at specific τ's are shown for 1 min sampling period in Figures 16(a), 20(a), 24(a), 28(a), 32(a), 36(a), 40(a) and 44(a) and for 10 min sampling period in Figures 16(b), 20(b), 24(b), 28(b), Figure 32(b), 36(b), 40(b) and 44(b). In Figure 45, the plots of the standard deviation ($\sigma_n(t)$) of the echo signals (Eq. (4)) vs. t and the average correlation coefficient ($\rho_{ave}(\tau)$) (Eq. (3)) vs. τ are presented for one of the porcine data sets.

Eight sets of data were obtained from eight individual swine and the averaged correlation coefficients, ($\rho_{Average}(t,\tau)$) (Eq. (6)) were plotted vs. t vs. τ for the 1 min sampling period (Figure 46(a)), and for the 10 min sampling period (Figure 46(b)). Individual plots of the averaged correlation coefficient vs. t for various τ's were plotted for the 1 min sampling period (Figure 47(a)) and for the 10 min sampling period (Figure 47(b)). The standard deviations (Eq. (7)) of the correlation coefficients ($\sigma_{corr}(t,t)$) vs. t vs. τ were plotted for the 1 min sampling period (Figure 48(a)) and for the 10 min sampling period (Figure 48(b)). Individual plots of the shifts vs. time from death for various τ's were plotted for the 1 min spacings (Figure 47(a)) and for the 10 min spacings (Figure 47(b)).

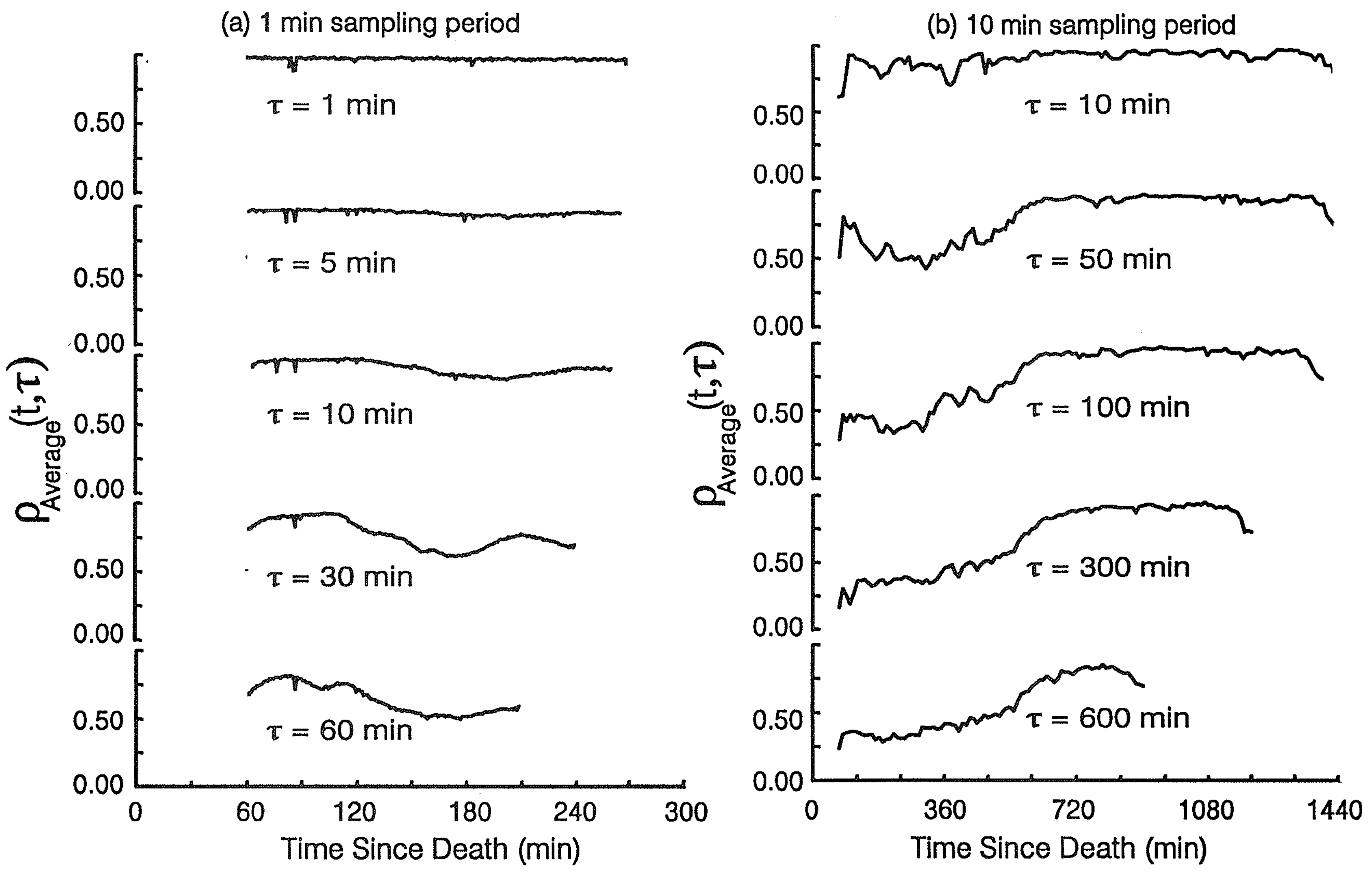
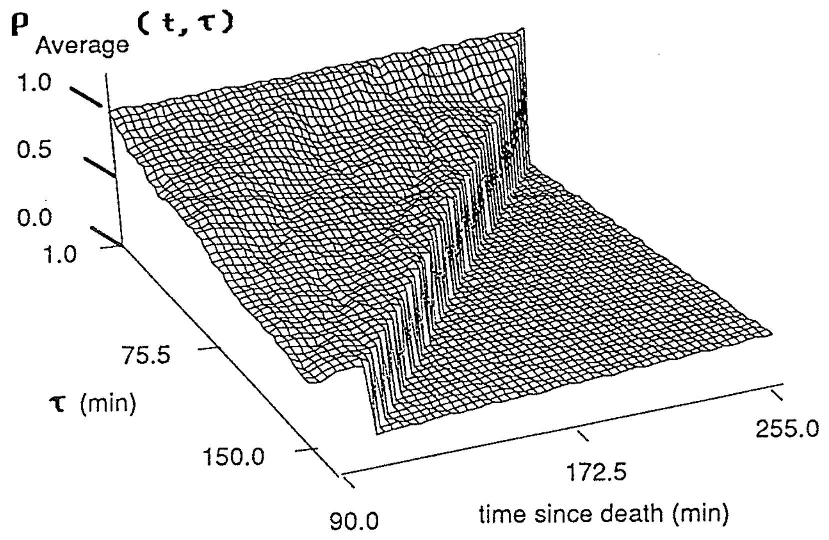


Figure 47. Average of eight correlation coefficient porcine data sets for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

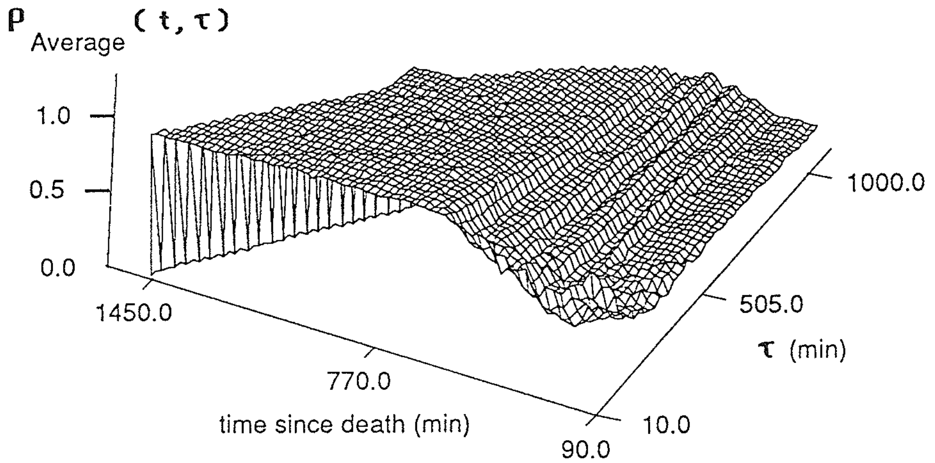
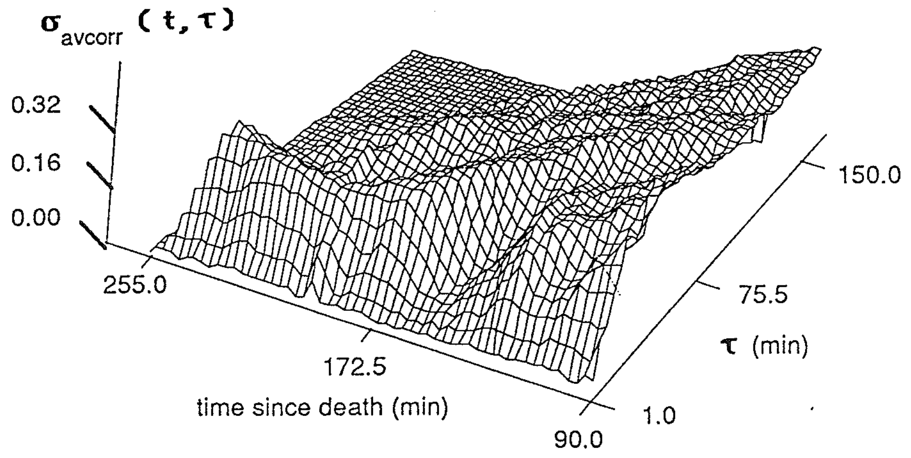


Figure 46. Plot of averaged correlation coefficient vs. time from death vs. tau for (a) 1 min sampling period and (b) 10 minute sampling period.

(a)



(b)

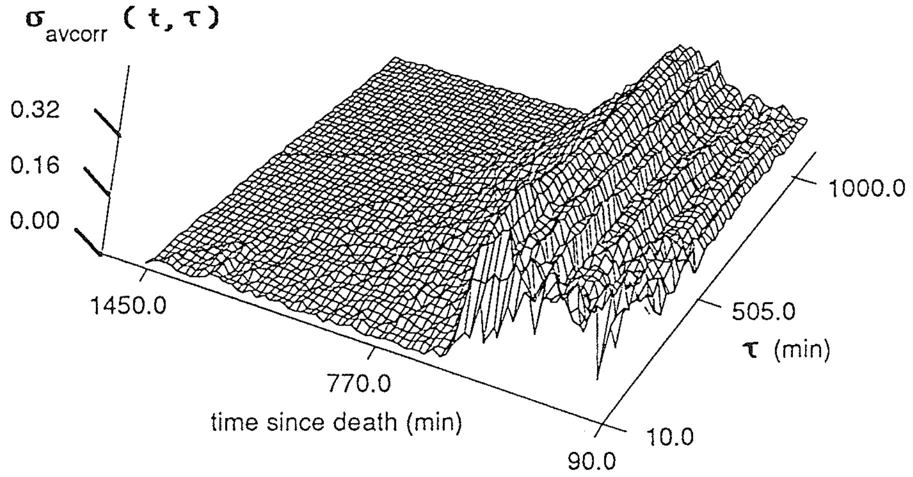


Figure 48. Plot of the standard deviation of the averaged correlation coefficient vs. time since death vs. tau for (a) 1 min sampling period and (b) 10 min sampling period.

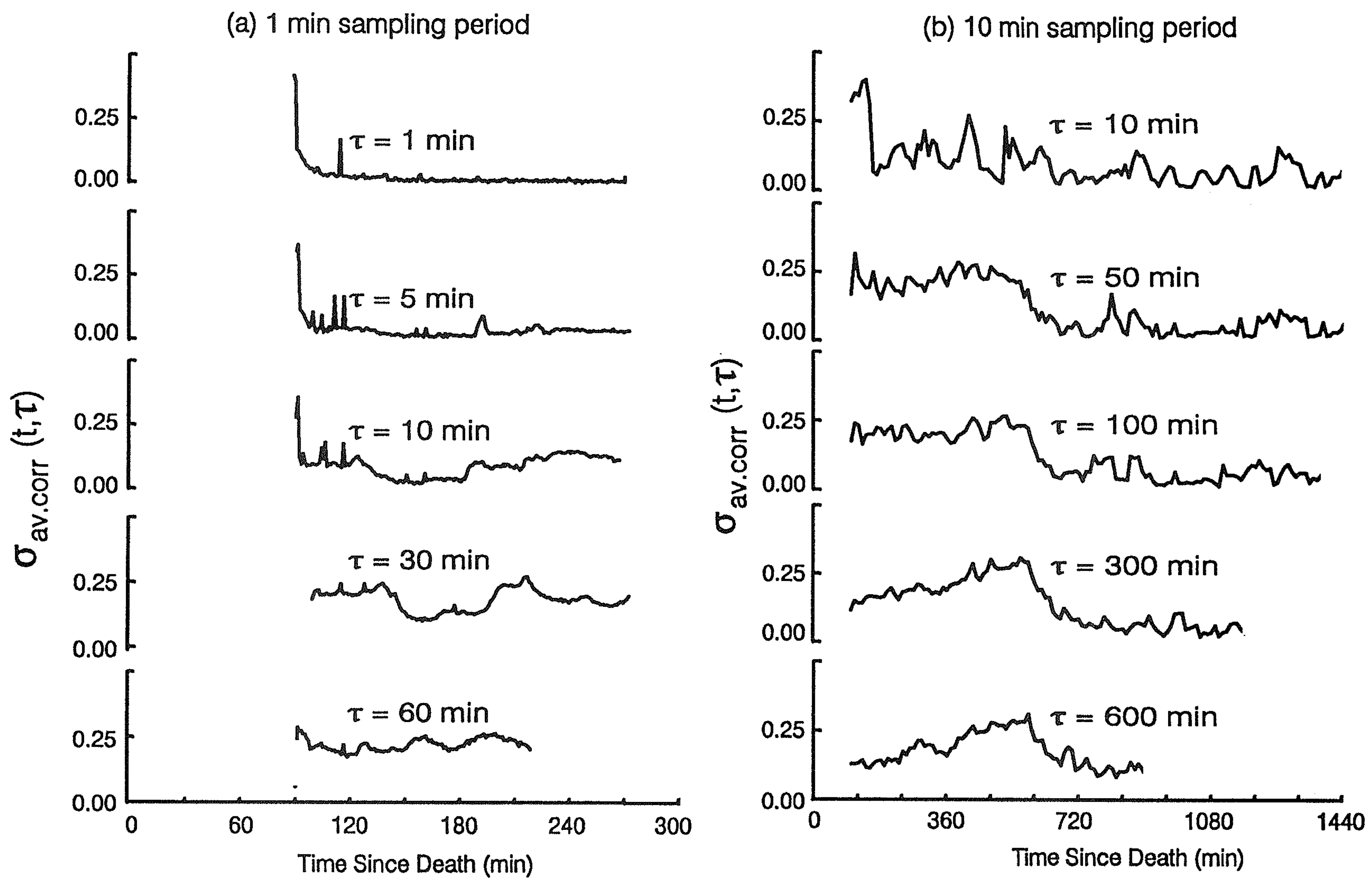


Figure 49. Standard deviation of the average correlation coefficient for eight porcine samples for various τ vs. time since death for (a) 1 min sampling periodand (b) 10 min sampling period.

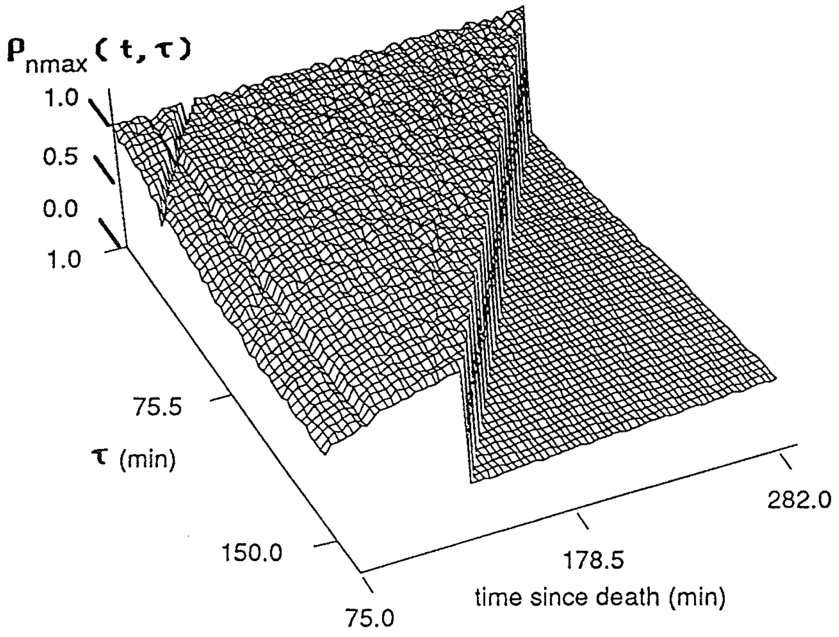
4.3 Bovine Longissimus Muscle Results

Preliminary results showed similar trends for the bovine muscle samples as well, shown in Figure 50 - Figure 53. The plot of the maximum correlation coefficient ($\rho_{nmax}(t)$) (Eqs. (1) and 2) vs. time (t) post mortem vs. τ for the 1 min sampling period is presented in Figure 50(a) and the 10 min sampling period in Figure 50(b). Individual plots of the maximum correlation coefficient at specific values of τ are shown for the 1 min sampling period in Figure 51(a) and for 10 minute sampling period in Figure 51(b). The shifts for the maximum correlation coefficient vs. t vs. τ are presented for the 1 min sampling period in Figure 52(a) and for the 10 min sampling period in Figure 52(b). Individual plots of the shift at specific values of τ are shown for the 1 min sampling period in Figure 53(a) and for the 10 min sampling period in Figure 53(b). In Figure 54, the plots of the standard deviation of the echo signals ($\sigma_n(t)$) (Eq. (4)) vs. time since death and the average correlation coefficient ($\rho_{nave}(\tau)$) (Eq. (3)) vs. τ are presented.

4.4 Discussion

The correlation coefficient gives an index for the similarity in shape between two echo signals. A low value indicates two echo signals that have very different shapes. Since a water-filled sponge is inert, it should exhibit very little change in the echo

(a)



(b)

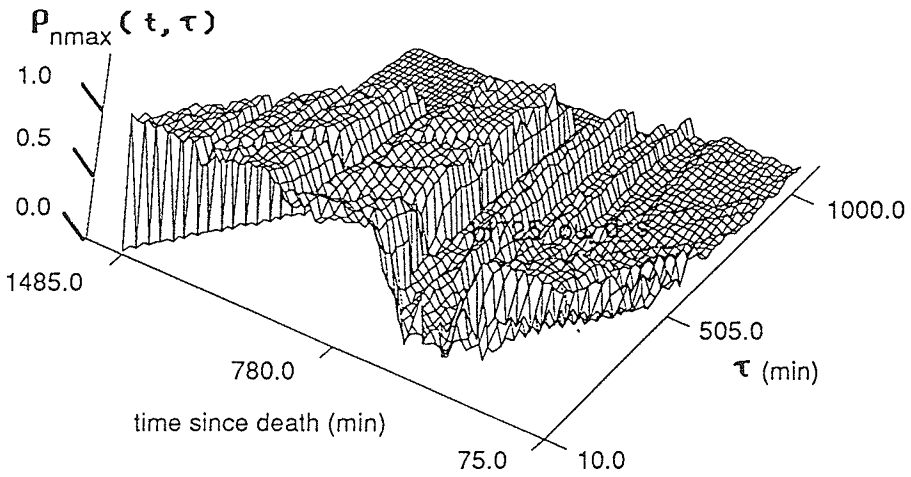


Figure 50. Plot of maximum correlation coefficient vs. time since death vs. tau for a bovine sample for (a) 1 min sampling period and (b) 10 min sampling period.

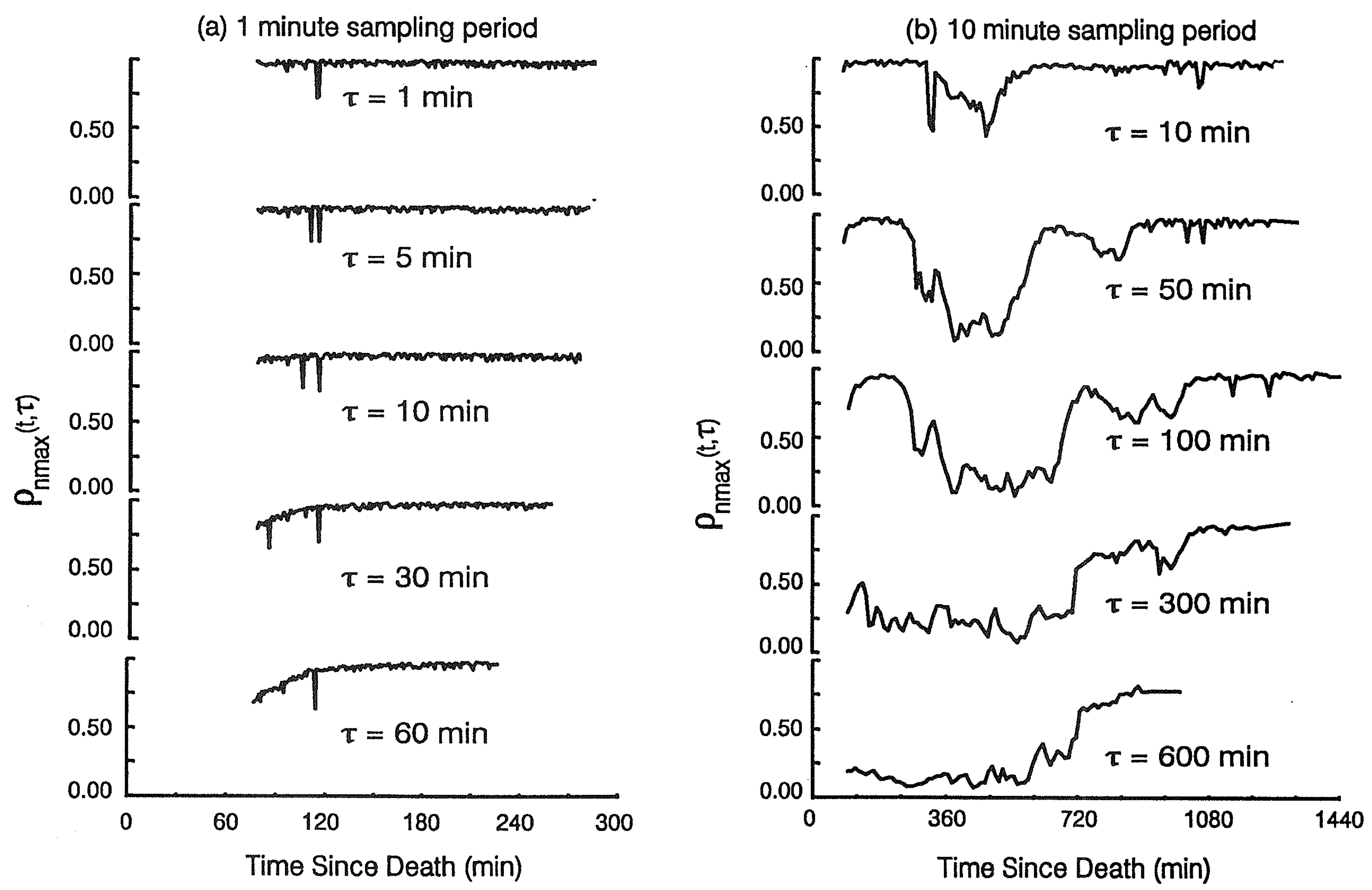
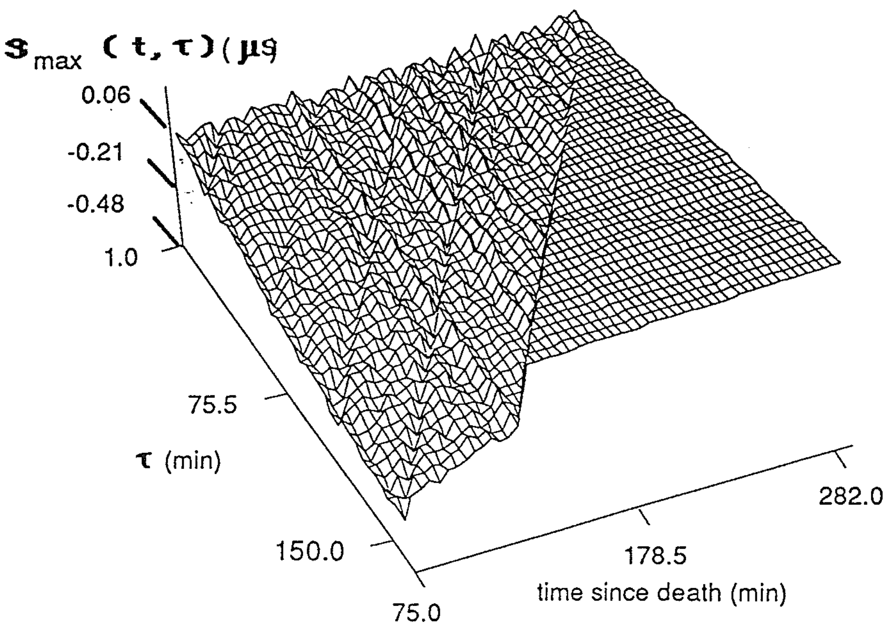


Figure 51. Maximum correlation coefficient for the bovine sample for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling period.

(a)



(b)

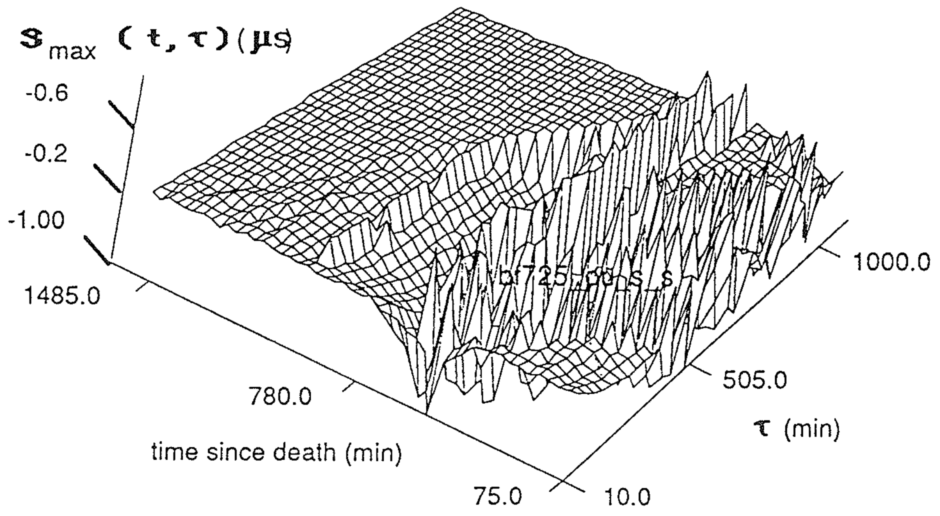


Figure 52. Plot of the shift for maximum correlation vs. time since death vs. tau for a beef sample for (a) 1 min sampling period and (b) 10 min sampling period.

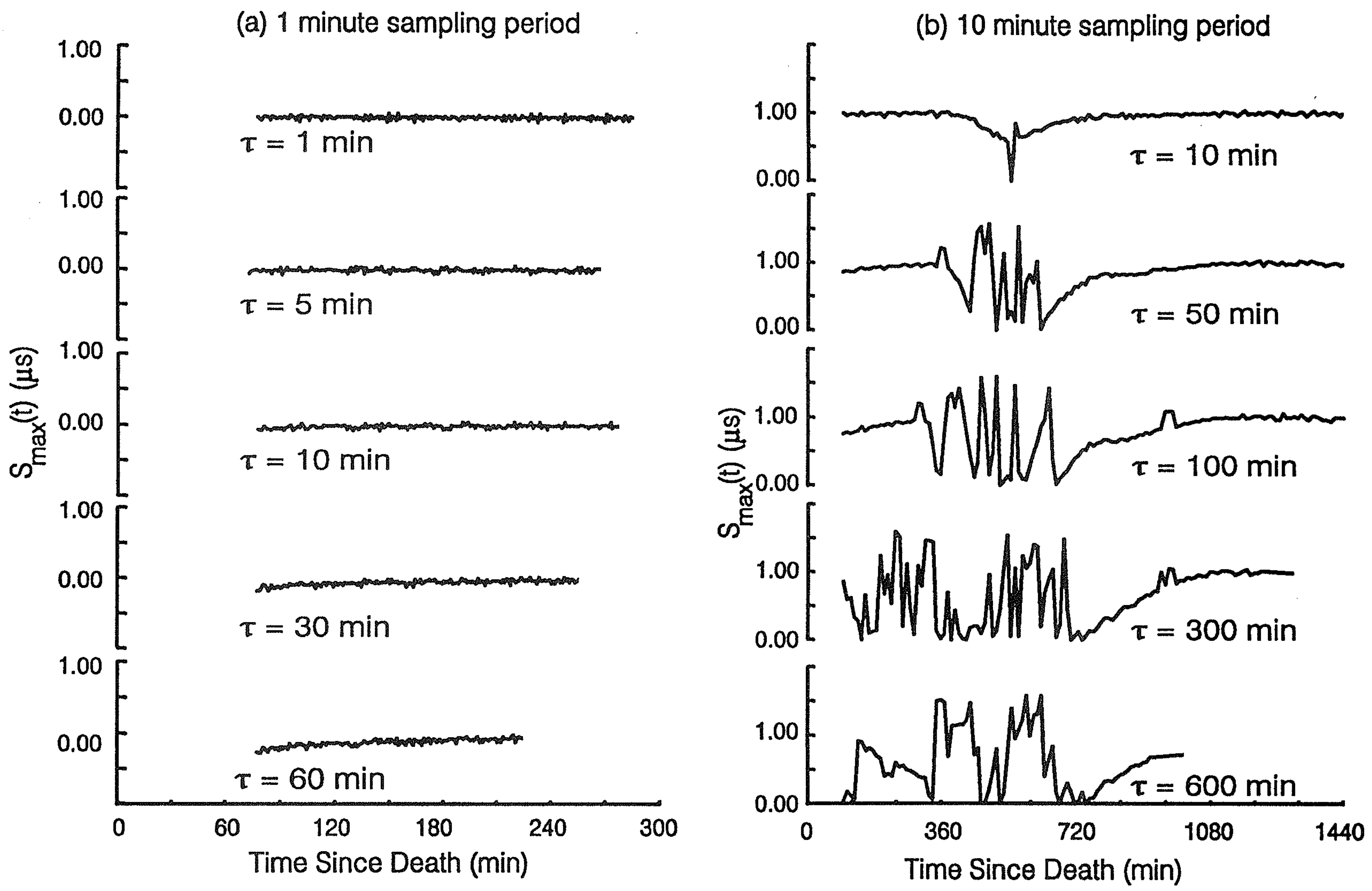


Figure 53. Shift for maximum correlation coefficient for a bovine sample for various τ vs. time since death for (a) 1 min sampling period and (b) 10 min sampling periods.

signal shape over time. Therefore, analyzing the echo signals from a water-filled sponge allows evaluation of the stability of the data acquisition system and identification of any system artifacts. The results yielded correlation coefficients (Eq. (1)) of near unity for all of the τ values as shown in Figure 8. For the larger values of τ, for example, 300 min and 600 min, a comparison of echo signals obtained yielded only slightly lower values as shown in Figures 8 and 9. Therefore, no abnormalities of the system were detected. The standard deviation (Eq. (3)) for the sponge remained near 175 mV for all time, and the average correlation (Eq. (2)) remained near unity with a slight decrease for very high τ values, also indicated no abnormalities of the system (Figure 10). Some noise in the system was indicated by the oscillation of the correlation coefficient, the slight decrease in the correlation coefficient for high values of τ and the decrease in the average correlation coefficient at high τ values.

For all of the porcine samples, the correlation coefficient (Eq. (1)) for a τ of 1 min, remains near unity for all time. For larger τ values the starting correlation coefficient is smaller, near zero for a τ as small as 60 min (Figures 13(a) and 14(a)). The shape of the correlation curve for the 1 min sampling period varies from animal to animal partially due to differences in the time after death at which data acquisition began; however, a definite trend can be seen for larger values of τ in Figures 17(a), 18(a), 21(a), 22(a), 29(a), 30(a), 37(a), 38(a), 41(a) and 42(a).

The correlation coefficient starts out high, near 0.75 at 60 min post-mortem and then decreases to values as low as 0.1 at 120 - 180 min post-mortem and remains low (Figure 30(a) τ = 60). In Figures 25(a), 26(a), 33(a), 34(a), 37(a) and 38(a) the correlation coefficient does not decrease as much as the other samples, but some decrease can still be observed.

For the 10 min sampling period a trend can also be observed, especially at values of τ greater than 50 min. The correlation coefficient starts out low for the first 8 h increasing to near unity by 10 h, Figures 13(b), 14(b), 17(b). 18(b), 21(b), 22(b), 25(b), 26(b), 29(b), 30(b), 33(b), 34(b), 41(b) and 42(b). From the start of data acquisition to 10 h post-mortem, the correlation coefficient oscillates between 0.1 and 0.75 at various times post-mortem depending on the the animal. In Figures 37(b) and 38(b) the correlation coefficient does not start as low as for the other samples but does increase to near unity at approximately 10 h. The averaged correlation coefficient (Eq. (5)) starts near unity for all 1 min spacings, decreasing to 0.50 at 2 h post-mortem, for τ of 30 and 60 min, and to 0.75 for a τ of 10 min (Figures 46(a) and 47(a)). For the 10 min intervals, the values start at 0.5 for τ of 50 and 100 min and near 0.25 for τ of 300 and 600 min increasing near unity at approximately 10 hours (Figures 46(b) and 47(b)).

For the bovine sample, for the 1 min sampling period, the correlation coefficient remains near unity for all values of τ as shown in Figures 50(a) and 50(b), indicating that the bovine muscle sample is not changing as rapidly as the porcine. For the 10 min sampling period the correlation coefficient oscillates around 0.25 before 9 h post-mortem and then increases to near unity at 12 h (Figures 50(b) and 51(b)). For large values of τ, the correlation coefficient increases to near 0.9 instead of unity due to excessive noise in the system caused by the analog-to-digital converter.

Since the echo signals were acquired at the same location in the image, the low correlation coefficient can be interpreted in several ways, one of which is that some sort of movement of or within the muscle has occurred. Since the sample itself is not moving in the tank nor is the water circulating, an explanation for low coefficients seen in the first ten hours, can be attributed to the physical changes occurring in the tissue post-mortem. For the first few hours after death the nerve cells slowly die releasing ATP to the system, and movement of the fibers occurs (contraction and expansion), resulting in the oscillatory behavior of the correlation coefficients between 0.1 and 0.5. As the ATP is depleted, the Na pump ceases to operate and Na leaks into the cells causing the fibers to contract. After a period of time, the majority of the fibers can no longer move, (rigor mortis is complete), yielding identical RF echo signals and correlation coefficients near unity. The correlation coefficients in both the individual porcine samples and the averaged values of the porcine samples were near unity at 10 hours post-mortem indicating the completion of rigor.

Trends similar to those seen for the correlation coefficient are seen in the shifts. The shift oscillates from 1.0 μs to -1.0 μs during the first 10 h for all of the porcine samples and then approaches zero by 10 hours (Figures 16, 17, 19, 20, 23, 24, 27, 28, 31, 32, 36, 37, 43 and 44). In Figures 39 and 40 the shift values do not oscillate as much as most of the other samples but they do approach zero at 10 h post-mortem. The shift values for the bovine samples also oscillated between 1.0 μs and -1.0 μs until approximately 11 hours at which point the values approached zero. The shift can also be interpreted as axial movement, with respect to the transducer, of the structures within the muscle [3].

The standard deviation for the porcine data began near 175 mV, decreased to near 117 mV by 270 min post-mortem and returned to 175 mV by 600 min (Figure 45). This is different from the sponge results (Figure 12) and indicates that some variation in the backscattering properties of the muscle occurred between the beginning of the experiment and 600 min. The average correlation for the porcine data (Figure 45) starts near unity for small τ values and decreases to 0.50 by 1080 min at which point were it then increases slightly. The average shift begins near zero and changes only slightly for large values of τ. All of the porcine samples and the bovine sample showed almost identical results for the standard deviation, the average correlation and the average shift and therefore, only one set of these plots is presented.

CHAPTER 5

CONCLUSIONS

The results of this research indicate that the calculation of the maximum correlation coefficient as a function of time can be used to clearly determine when rigor related contraction and expansion of the muscle fibers is occurring and when all activity stops, i.e., when rigor mortis is complete. The end of activity occurred at approximately 10 h post-mortem for both the porcine and bovine samples. In addition, the time frame over which these changes are occurring can be inferred from the τ spacing. For small τ's such as 1 min or 10 min, the correlation coefficient remained near unity and the shift remained near zero, indicating that little change occurred in the muscle from one minute to the next. In contrast, τ's of 60 min resulted in lower coefficients and large shift values before 10 h post-mortem, indicating that some change occurred in the muscle sample in a one hour time frame.

The correlation coefficient can be used to determine whether or not motion has occurred in the ultrasound beam; however, the value gives no indication of the direction of motion. The shift values give additional information by giving a quantitative measure of movement in the axial direction, with respect to the transducer. If the correlation coefficient is near unity then zero shift indicates no motion, and non-zero shift indicates motion mostly in the axial direction. The scatterers will also move out of the ultrasound beam, laterally and/or transversely, resulting in lower correlation coefficients than for only axial motion. If the motion is mostly lateral and/or transverse, the shift values will not yield any useful information. When motion is occurring in all directions, then the correlation coefficient will have low values and the shift will be non-zero. For this research, the correlation coefficient is an indicator of non-axial motion and the shift is an indicator of axial motion.

Thus, ultrasound correlation analysis can be a useful tool in determining when the completion of rigor mortis occurs, as well as any time post-mortem when little change is occurring. Knowing when the muscle is stable will help determine a time post-mortem for consistent grading of animals using ultrasound. Currently porcine muscle is not graded but the procedure can be applied to beef. Future research will include further ultrasound correlation analysis performed on bovine samples as well as correlating these results with other indices such as pH and temperature.

REFERENCES

[1] S.G. Foster, "A pulsed ultrasonic flowmeter employing time domain methods," Ph.D dissertation, Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1985.

[2] P.M. Embree, "The accurate ultrasonic measurement of the volume flow of blood by time domain correlation," Ph.D dissertation, Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1986.

[3] I.A. Hein, "Measurements of volumetric blood flow using ultrasound time-domain correlation," Ph.D dissertation, Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1991.

[4] I.A. Hein and W.D. O'Brien Jr., "Volumetric measurement of pulsatile flow via ultrasound time-domain correlation," *Journal of Cardiovascular Technology*, vol. 8, no. 4, pp. 339-348, 1989.

[5] P.M. Embree, "Volumetric blood flow via time-domain correlation: experimental verification," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 37, no. 4, pp. 176-189, May 1990.

[6] J.F. Price and B.S. Schweigert, *The Science of Meat and Meat Products*. Westport, Connecticut: Food & Nutrition Press, INC., 1978.

[7] I.A. Hein, J. Novakofski, L. Nostwick and W.D. O'Brien, Jr., "Application of ultrasound time-domain correlation to the ageing of porcine muscle for the first twenty-four hours post-mortem," 19th International Symposium on Acoustical Imaging Proceedings.

[8] AIUM Acousstic Output Measurement and Labeling Standard for Diaagnostic Ultrasound Equipment, August 1988.

[9} R.M. Glueck, J.G. Mottley, B.E. Sobel, J.G. Miller and J.E. Perez, "Changes in ultrasonic attenuation and backscatter of muscle with the state of contraction,"*Ultrasound in Medicine and Biology*. vol. 11, no. 4, pp. 605-610, July/August 1985.

[10] H. Rijsterborgh, F. Mastik, C.T. Lancee, F.W. Van Der Steen, L.M.A. Sassen, P.D. Verdouw, J. Roelandt and N. Bom. "Ultrasonic myocardial bacscatter and myocardial wall thickness in animal experiment," *Ultrasound in Medicine and Biology*, vol. 16, no. 1, pp. 29-36, 1990.

[11] USDA Food Safety and Quality Service, Official United States Standards for Grades of Carcass Beef, USDA, 1989.

[12] Kinsler, L.R, A.R. Frey, A.B. Coppens and J.V. Sanders, *Fundamentals of Acoustics*. New York: John Wiley & Sons, 1982.

APPENDIX A

MEAT ULTRASOUND DATA ACQUISITION PROGRAM

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <graphics.h>
#include <time.h>
#include <math.h>
#include <time.h>

#include <dos.h>

/*****************************************************************************************************/
/*                                                                                                   */
/*                          Meat Ultrasound Data Acquisition Program                                 */
/*                                                                                                   */
/*                                         11-29-90                                                  */
/*                                                                                                   */
/*                     written by Ilmar Hein modified by Linda Nostwick                              */
/*****************************************************************************************************/


/* The following are PC port addresses that communicate with RNS_UDAS */


#define RESET                  0x300
#define COMP_AQUIRE_DATA       0x302  /* Tells RNS_UDAS to start an aquire data cycle*/
#define WRITE_ECHO_LENGTH      0x304   /* Echo length divided by 4 stored here*/
#define WRITE_NUM_ECHOES       0x306   /* number of consecutive echoes to acquire is*/
                                       /*stored here*/
#define CLEAR_ND_LATCH       0x308    /* Clears the new data latch */
#define LOAD_START_ADDRESS     0x30a  /* loads start address of memory where echoes*/
                                       /*are to be placed */
#define READ_ADDRESS_COUNTER 0x30a /* Read the current value of the memory address*/
                                       /*counter*/
#define RD_TRANS_POS             0x30c   /* Read the current transducer position.  Bits 0-6
                            /* are the position of the wiper, bits 7 is the New Data latch, and*/
                          /* and bits 8-15 are the wiper location.                 */

#define STP gotoxy          /* Alias for Set Text Position */
#define CLS cleardevice  /* Clears the screen              */

#define MAX_CLIP 479/* Maximum and minimum y-values for the VGA display, which is*/
#define MIN_CLIP 0       /* 640 x 480.  Numbers above and below will be clipped to these values */
#define X_AXIS   350     /* VGA y-value where x=0 axis is placed                           */

/* time_t ltime; */
/*   used in do_corr */

int  effective_pr_period  =  1;

float  flow[100];

char  key,
      datafile [15],
       aquire_time[15],
      txt[80],
      choice,
       save_name[25],
      name[25] = "NOT SAVING"
     ;


         int
              bank = 1,           /* determins which memory bank is used in echo display */
             old_flow[100],     /* used to erase previous flow trace                     */
            set_av = 1,
               save_echo_length = 1024,  /* Default for 1K echoes = 1.5 cm depth */
             ad_echo_length=1024,         /* echo length written to A/D          */
              mem_number_echoes = 64,     /* number echoes stored in memory      */
               display_add = 0,           /* location in memory to start displaying/storing */
            base_echo = 0,
             max_delta=10,
            res_fac = 1,
             range_inc = 40,
              acquire_length,
            xmax,
            y_pt,
               num_disk_echoes = 1,   /* number of RF echoes to store to disk */
             number_echoes,
             sav_flag = -1,       /* save RF flag, -1=no 1=yes */
               store_flag = -1,      /* store shift -1 = don't store, 1 = store */
              load_flag = -1,       /* load RF from disk to correlator */
             echo_spacing_time = 0,
            min =0,
            sec=0
           ;

unsigned  int  c,
           d,
           dsa,
            xoffset,
            yoffset,
             ekos_to_plot = 1,
           top8
          ;

unsigned  int  value,
            values [1024]
         ;

unsigned long int address,
                 displayed_echo=1,
                 display_echoes = 262144,
              msa
              ;
long zzz,
     old_time,
     new_time,
     chk_time
    ;

    float
         pass=0,
         maxflow,
         maxvel,
         xscale,
          vel_scale = 1.5 ,       /* no angle correction */
          yscale = 70.0,
          error_count=0
        ;
  int
      g_driver,
     g_mode,
      g_error
    ;


    long t;
     char *ptr;


   FILE
        *fptr
       ;

main()

{

void plot_flow();
void plot_echo();
void save_echoes();
void load_echoes();

if( (fptr = fopen("meat-acq.sys","r") ) ==NULL )
  printf("\n No initialization file MEAT-ACQ.SYS.  Hope that's OK.\n" );

else
 {
    fscanf(fptr,"%s %d", txt, &ad_echo_length);
    fscanf(fptr,"%s %d", txt, &mem_number_echoes);
    fscanf(fptr,"%s %d", txt, &save_echo_length);
    fscanf(fptr,"%s %d", txt, &num_disk_echoes);
    fscanf(fptr,"%s %d", txt, &display_add);

    fclose(fptr);

    printf("\nad_echo_length      = %d",ad_echo_length);
     printf("\nmem_number_echoes  = %d",mem_number_echoes);
     printf("\nsave_echo_length    = %d",save_echo_length);
     printf("\nnum_disk_echoes     = %d",num_disk_echoes);
    printf("\ndisplay_add         = %d",display_add);

  }


outport(0x310,15);

outport(0x310, 0 );                     /* initialize */
outport(RESET, 0 );
outport( WRITE_NUM_ECHOES, mem_number_echoes+1 );
outport( WRITE_ECHO_LENGTH, ad_echo_length/4-1 );
outport(0x310,15);

   detectgraph(&g_driver, &g_mode);
   initgraph(&g_driver, &g_mode, "");
   g_error = graphresult();
  if(g_error < 0)
   {
       printf("initgraph error: %s.\n", grapherrormsg(g_error));
     exit(1);
   }

     display_echoes = 1024 * (long)ekos_to_plot;


while(1)
 {
    initgraph(&g_driver, &g_mode, "");
   closegraph();
   textmode(3);
   printf("***** MEAT Ultrasound Echo Collection System *****\n\n");

   printf("\n   [1] Write to address              [6] Write Echo length latch (%d [%d])",ad_echo_length,
     ad_echo_length/4-1);
   printf("\n   [2] Read from an address        [7] Write to # echoes latch (%d)", mem_number_echoes);
   printf("\n   [3] Write infinite loop          [8] Read Address Ctr");
   printf("\n   [4] Write to an address & read    [9] Rd TRans Pos");
   printf("\n   [5] Write address ctr & read       [a] Acquire Data ( Comp AQD ) ");
  printf("\n  [i] Initialize a/d                   [c] Clear New Data Latch");
printf("\n   [d] display RF echoes                    [z] test echo length/ # echoes");
printf("\n   [m] write & read from Memory         [q] Quit ");
printf("\n   [f] plot Flow vs distance               [b] choose mem Bank (%d)", bank);
printf("\n   [s] Sets to average (%2d)                 [y] base echo (%d) ", set_av, base_echo );
printf("\n   [r] Resolution factor    (%2d)            [w] Delta Max (%2d)", res_fac, max_delta );
printf("\n   [e] Echoes to plot (%2d)          [p] Set effective PR period (%2d)", ekos_to_plot,
    effective_pr_period);
printf("\n   [S] Save RF echo (%s)              [L] Load RF (%2d)", name, load_flag);


printf("\n   [N] # RF echoes to save (%3d)                          ", num_disk_echoes );
 printf("\n    [D] Display start address (%d)", display_add);

printf("\n   [g] sets to average = 20   delta max = 1   # echoes = 255");
 printf("\n    [h] Set time between echoes (%d min)",echo_spacing_time);
 printf("\n    [E] save Echo length (%d)\n",save_echo_length);

 printf("\n\nWhaddaya wanna do? ");

 fflush(stdin);
choice = getch();
switch( choice )
   {

      case 'D':
             printf("\nEnter display_add " );
            scanf("%d",&display_add);
         break;

      case 'E':
             printf("\nOk, what do you want the save echo length to be, hmmm?");
            scanf("%d",&save_echo_length);
         break;

      case 'g':
         max_delta = 1 ;
         set_av    = 20;
           outport(0x310, 0);
         outport(RESET, 0 );
          outport( WRITE_NUM_ECHOES, 255 );
           outport(0x310,15);
         break;

      case 'w':
            printf("\nEnter Delta Max (1-10) " );
           scanf("%d",&max_delta);
          if( max_delta < 1 || max_delta > 20 )
         {
                 printf("\n Requested input must fall between 1 and 10.\n");
            res_fac=1;
         }
         break;
```

```
case 'p':
      printf("\nEffective PR Period (1-10): " );
       scanf("%d",&effective_pr_period);
       if( effective_pr_period < 1 || effective_pr_period >10)
  {
          printf("\n Damn it!  It's gotta be between  1 and 10.\n");
        effective_pr_period = 1;
  }
  break;


case 'N':
      printf("\nNumber RF echoes to save (1-255): " );
      scanf("%d",&num_disk_echoes);
      if( num_disk_echoes < 1 || num_disk_echoes > 255)
  {
          printf("\n No way, ... has to be between  1 and 255.\n");
       num_disk_echoes = 1;
  }
  break;

case 'r':
      printf("\nEnter Resolution Factor (1, 2, or 3): " );
    scanf("%d",&res_fac);
   if( res_fac < 1 || res_fac > 3 )
  {
           printf("\n Requested input must fall between 1 and 3.\n");
     res_fac=1;
  }
  break;

case 'y':
     printf("\nEnter Base Echo: " );
     scanf("%d",&base_echo);
   if( base_echo < 1 || base_echo > 200 )
  {
          printf("\n Hey, between 1 and 200 please!!\n");
     base_echo = 0;
  }
  break;

case 's':
    printf("\nEnter Sets to average ( 1 <= S <= 24 ): " );
    scanf("%d",&set_av);

  if( set_av < 1 || set_av > 24 )
  {
       printf("\n Hey man, BETWEEN 1 and 24!!\n");
    set_av = 1;
  }
```

```
      number_echoes = set_av * 10 + 15;
      outport(0x310, 0 );
      outport( WRITE_NUM_ECHOES, number_echoes );
      outport(0x310,15);
    break;



case 'b':
    bank = bank * -1;
    break;

case 'S':
    sav_flag = sav_flag * -1;
    if(sav_flag == -1)
    {
       fclose(fptr);
       strcpy(name, "NOT SAVING");
    }
      if(sav_flag == -1) break;

key = 0;
    fflush(stdin);
    if(echo_spacing_time == 0)
  {
        printf("\nHey!!! Time between capturing echoes = 0!\n");
      printf("You sure you want to go on (A to abort)?");
     key = getch();
  }
      printf("%c",key);

     if(key == 'a' || key == 'A')
     {
       puts("ABORTING");
       sav_flag = -1;
        strcpy(name, "NOT SAVING");
     }

    if(sav_flag == 1)
    {
         printf("\n Enter filename:");
        scanf("%s",save_name);

       if( ( fptr = fopen( save_name, "a" ) ) == NULL )
      {
            printf("\nCan't open %s!\n",save_name);
         exit(0);
      }
       strcpy(name, "SAVING: ");
        strcat(name,save_name);
    }
  break;

case 'L':
    load_flag = load_flag * -1;
   if( load_flag == 1 )
   {
        puts("Loading RF data into correlator...");
     load_echoes();
   }
  break;

case 'e':
       printf("Enter echoes to be displayed:");
      scanf("%d",&ekos_to_plot);
       display_echoes = 1024 * (long)ekos_to_plot;
       printf("%lu\n", display_echoes );
   break;


case '1':

        printf("\n\n Enter address (hex)   value : ");
     fflush(stdin);
      scanf("%x %x", &address, &value);
       printf("address= %x ", address);
                  printf("value = %x\n", value );
      outport( address, value );
    break;

case '2':

        printf("\n\nEnter address to read:   ");
     fflush(stdin);
      scanf("%x", &address);
      value = inport(address);
        printf("Address %x contains %x", address, value);
    break;

case '3':
        printf("\n\nEnter address (hex)    value: ");
     fflush(stdin);
      scanf("%x %x", &address, &value );
     while( !kbhit() )
          outport(address, value);
    break;

case '4':
        printf("\n\nEnter address(hex) value: ");
     fflush(stdin);
      scanf("%x %x",&address, &value );
      outport(address, value);
     value = inport(address);
```

```
            printf("\nAddress = %x  value = %x  %d", address, value, value);
          fflush(stdin);
        break;

    case '5':

          printf("Enter value to write: ");
         fflush(stdin);
         scanf("%x", &value);
         outport(LOAD_START_ADDRESS, value);
       value = 0;
        value = inport(READ_ADDRESS_COUNTER);
           printf("\nValue returned = %xh  %d", value,value);
       break;

   case '6':
          printf("\Enter value to write to Echo Length Ctr : ");
         fflush(stdin);
          scanf("%d", &ad_echo_length);
         outport(0x310, 0);
          outport( WRITE_ECHO_LENGTH, ad_echo_length/4-1 );
         outport( 0x310, 5 );
       break;

  case '7':
          printf("\nEnter value to write to # Echoes latch (desired # echoes/4 - 1): ");
        fflush(stdin);
         scanf("%d", &mem_number_echoes);
        outport(0x310, 0);
        outport( WRITE_NUM_ECHOES, mem_number_echoes + 1 );
        outport(0x310, 5);
      break;


 case 'a':
     outport(COMP_AQUIRE_DATA, 0 );
    break;

 case '8':
    value = inport( READ_ADDRESS_COUNTER );
       printf("\nAddress Counter = %xh %d ",value,value );
   break;

 case '9':
  while( !kbhit() )
 {
   value = inp (RD_TRANS_POS);
      printf("%x\n", value &0x0080);
 }
 getch();
 break;
```

```
case 'c':
   outport(CLEAR_ND_LATCH, 0 );
  break;

 case 'i':
      outport(0x310, 0 );

     outport(RESET, 0 );
       outport( WRITE_NUM_ECHOES, mem_number_echoes );
        outport( WRITE_ECHO_LENGTH, ad_echo_length/4-1 );
       outport(0x310,15);

    break;

 case 'm' :
   while (1)
   {
        printf("\nEnter value to write: ");
      scanf("%ld", &zzz);
      if( zzz > 255 )break;


     for( c=0; c<640; c++ )
       {
           outport(0x310, 15 );
           outport(0x300,c);
           outport(0x302,c);
           outport(0x304,0);
         zzz = (long)c;
           if(zzz >255) zzz = (long)(c - 256);
           else if( zzz > 511 ) zzz = (long)(c - 512);

           outport(0x306, zzz);
           outport(0x308, zzz);
           outport(0x312,0);
       }
      outport( 0x310, 5 );
      outport( 0x300, 0 );
      outport( 0x302, 0 );
      outport( 0x304,  0 );
      zzz = inport( 0x30a );
         printf("\nValue returned = %lx \n",zzz );
   }
      outport(0x310,0);
    break;

case 'q':
     textmode(3);
    exit(0);
    break;
```

```
        case 'd':

            plot_echo();
            break;

        case 'f':

            plot_flow();
            break;

        case 'h':

                printf("\nHow many minutes(integer values between 0 and 60)??");
              scanf("%d",&echo_spacing_time);
            if (min<0 || min>60)
             {
                  printf("\n Value must be between 0 and 60.\n");
                echo_spacing_time=1;
             }
             sec=echo_spacing_time*60;
            break;

                /*------------------- Case z ----------------------*/
      /*                                  */
        /* Test routine to check if Echo Length counter and */
        /* Number Echoes counter are couning correctly.     */
       /* Assumes a value of 255 (Echo Length of 256) has */
        /* been written into the Echo Length counter and a  */
        /* value of 65 (64 echoes) has been written to the  */
         /* Number Echoes counter. After acquiring data, the */
       /* address counter should contain 0x4000.          */
      /*                                  */

       case 'z':
               acquire_length = (ad_echo_length/4) * mem_number_echoes;
           outport(0x310,0);
          while( !kbhit() )
            {
               outport( LOAD_START_ADDRESS, 0);
               outport( CLEAR_ND_LATCH, 0 );
               outport( COMP_AQUIRE_DATA, 0 );
                while( ( inport(RD_TRANS_POS) & 0x0080 ) == 0 ) ;
               value = inport(RD_TRANS_POS);

              pass++;

               value = inport(READ_ADDRESS_COUNTER);
                 if(value != acquire_length)
              {
                 error_count++;
                     printf("\n**pass = %f errors = %f length = %x read %x", pass,error_count,
acquire_length,value);
```

```
              }

                   printf("\npass = %f    error_count = %f  value = %x",pass, error_count, value);
             }
            getch();
              outport(0x310,15);
            break;

          default:
            ;
      }
  }
 }

/************************************* plot_flow *******************************/

void  plot_flow()
{

   initgraph(&g_driver,  &g_mode,  "");

  yscale = 46.67;
  STP(8,1);
  strcpy(txt," VERT = 1.5 cm/sec / div");
   printf("%s",txt);
  STP(2,1);
   strcpy(txt,"7.5");
   printf("%s",  txt);

  for(c=0; c<100; c++) old_flow[c]=0;                /* initialize buffer used in erasing the flow trace
*/

  xscale = 25;
  xmax = 23;


  while(1)
      {

       key = '0';
       if( load_flag == -1 )
     {
         while( (value = inport(READ_ADDRESS_COUNTER) ) != 0 )  /* The clock is reset on the next*/
                                                                 /*depth gate pulse  */
          {
              outport(0x310, 0 );
              outport(LOAD_START_ADDRESS, 0);              /* from the ATL. The computer must wait*/
                                                             /*until that  */
              STP(55,4);                             /* until that pulse arrives. This is done by writing*/
                 sprintf(txt,"START Address = %4x",value ); /* to the address counter... when the value*/
               printf("%s",txt);                   /* has been written in , we know that the clock has*/
            STP(55,5);                         /* been started.                    */
          strcpy(txt,"END Adress = waiting  ");
          sprintf("%s",txt );

        if( kbhit() )
          if( ( key = getch() ) == 'q' ) return;
      }

  STP(55,5);
    strcpy(txt,"END Address = loaded ");
   printf("%s",txt );

  STP(55,4);
    sprintf(txt,"START Address = %4x",value );
   printf("%s",txt);

  outport(CLEAR_ND_LATCH, 0 );                /* Initilialize New Data latch to 0 */

   outport(COMP_AQUIRE_DATA, 0 );
   while(  ( inport( RD_TRANS_POS ) & 0x0080 ) == 1 )  /* Wait until New Data latch is cleared
                                                        /*before continuing */
        if( kbhit() )
           if( ( key = getch() ) == 'q' ) return;

  STP(55,3);
   strcpy(txt,"AQUIRING DATA");
   printf("%s",txt);
  while(  ( inport( RD_TRANS_POS ) & 0x0080 ) == 0 )  /* Wait until New Data latch = 1,*/
                                                       /*indicating acquire */
       if( kbhit() )                      /* data cycle is complete  */
          if( ( key = getch() ) == 'q' ) return;

  value = inport( READ_ADDRESS_COUNTER );
  STP(55,5);
   sprintf(txt,"END Address = %4x   ",value );
   printf("%s",txt);
}


  STP(40,25);
    sprintf(txt,"EPRP = %2d  sets = %2d  delta = %2d", effective_pr_period, set_av, max_delta);
   printf("%s",txt);

  STP(55,3);
    strcpy(txt,"Correlating   ");
   printf("%s",txt);

   outport( 0x310, 5 );          /* pass control of RNS_UDAS bus from A/D to correlator */

  time(&t);                  /* print current time to screen */
   ptr = asctime(localtime(&t));
  *(ptr + 20) = 0;
    strcpy(aquire_time, ptr+11);
```

```
 STP(70,1);
   printf( "%s", aquire_time );

 if( load_flag == 1 )
  {
   STP(55,3);
      sprintf(txt,"Stored Data");
     printf("%s",txt);
  }

/* corr( set_av , res_fac);        do correlations */

 STP(55,3);
 strcpy(txt,"              ");
   printf("%s",txt);

 maxflow = 0.0;

  if( store_flag==1 )
 {
       fprintf(fptr, "\n%s\n", aquire_time );
 }

 for(c=0; c<=xmax; c++)
   {
     if( flow[c] > maxflow) maxflow = flow[c];
         if( store_flag == 1 ) fprintf(fptr,"%f %f \n",(float)c * .6/(float)res_fac, flow[c] );
   }

/*    maxvel = 100 * (maxflow *  20e-9 * 1500.0 )/(  2.0 * .001 * .707)   max vel in cm/sec*/
       maxvel = (vel_scale / effective_pr_period) * maxflow;

 STP( 43,1 );
   sprintf(txt,"Max Vel = %.2f", maxvel );
   printf("%s", txt);

 setcolor(0);                                  /* erase previous trace */
  moveto( 0,(int) old_flow[0] );
 for(c=1; c<=xmax; c++)
   {
       lineto( (int)((float)c * xscale),  old_flow [c] );
   }

setcolor(LIGHTGREEN);

 moveto(0,350);       /* horizontal axis */
  lineto(600,350);
STP(1,24);
if( res_fac == 1)
strcpy(txt,"    1    2    3    4    5    6    7    8    9   10   11   12   13   14   ");
if( res_fac == 2)
strcpy(txt,"        1        2        3        4        5        6           ");
```

```
if( res_fac == 3)
strcpy(txt,"                1                 2                 3            ");

  printf("%s",txt);
STP(25,25);
   strcpy(txt,"Distance  (mm)");
  printf("%s",txt);


for(c=0; c<15; c++)
  {
       moveto( (int)( (float)c*41.67 ), 345 );
        lineto( (int)( (float)c*41.67 ), 355 );
  }

moveto(0,0);      /* vertical axis */
lineto(0,350);

for( c=0; c<=350; c=c+70 )
{
   moveto(0,c);
    lineto(5,c);
}

    if( key == '0' ) if( kbhit() ) key = getch();

  if( key !='0' )
  {
    STP(1,25);
       sprintf(txt,"%c", key);
      printf("%s",txt);

    STP(8,1);

    if( key == '4' )
    {
       yscale = 35.00;
         strcpy(txt, " VERT = 2 cm/sec / div ");
         printf("%s",txt);

      STP(2,1);
         strcpy(txt,"10 ");
         printf("%s",txt);
    }

    if( key == '3' )
    {
       yscale = 46.67;
         strcpy(txt, " VERT = 1.5 cm/sec / div ");
         printf("%s",txt);

      STP(2,1);
      strcpy(txt,"7.5");
     printf("%s",txt);
}

if( key == '2' )
{
   yscale = 87.5;
     strcpy(txt," VERT =  .8 cm/sec / div ");
     printf("%s",txt);
  STP(2,1);
      strcpy(txt,"4.0");
     printf("%s",txt);
}

if( key == '1' )
{
   yscale = 350.0;
    printf(" VERT =  .2 cm/sec / div ");
  STP(2,1);
     strcpy(txt,"1.0");
    printf("%s", txt );
}

          /*------------------------ save  data  to  disk -------------------*/

if( key == 'r')
{
    store_flag = store_flag * -1;
   if( store_flag == 1 )
  {

     time(&t);
        ptr = asctime(localtime(&t));

    STP(  10, 10 );
         strcpy(txt,"Enter tape counter value from VCR: ");
      printf("%s", txt);
     gets( datafile );
       strcat(datafile, ".shf" );
      if(  ( fptr = fopen( datafile, "a") )== NULL )
    {
      STP( 1, 25);
           printf("Couldn't open datafile!");
       exit(0);
    }
         fprintf(fptr," %s   %d    %s\n", datafile, res_fac, ptr );
    STP(  10, 10 );
    strcpy(txt,"                                  ");
      printf("%s",txt);
    STP(74,25);
     strcpy(txt,"RECORD");
      printf("%s",txt);
  }
  else
  {
     STP(74,25);
      strcpy(txt,"       ");
       printf("%s",txt);
     fcloseall();
  }
 }

                /*------------------------------------------------------------*/
 if(key =='c' && store_flag == 1 )
{
 STP(3,5);
     sprintf(txt, "Enter Comment:");
    printf("%s", txt );
 STP(3,6);
  gets(txt);
      fprintf(fptr,"\n%s\n", txt);
  strcpy(txt,"                  ");
 STP(3,5);
 strcpy(txt,"                                                        ");
    printf("%s",txt);
 STP(3,6);
    printf("%s", txt);
}

 if( key == 'p' )
 {
      effective_pr_period++;
        if(effective_pr_period > 10 ) effective_pr_period = 1;
     outport(0x310, 0);
   outport( WRITE_NUM_ECHOES, 255 );
     outport(0x310, 5);
 }

 if( key == 'P' )
 {
      effective_pr_period--;
        if(effective_pr_period < 1 ) effective_pr_period = 10;
     outport(0x310, 0);
   outport( WRITE_NUM_ECHOES, 255 );
     outport(0x310, 5);
 }

 if( key == 'd' )
 {
   max_delta++;
    if(max_delta > 10 ) max_delta = 1;
 }
 if( key == 'D' )
 {
        max_delta--;
        if(max_delta < 1 ) max_delta = 10;
    }
    if( key == 's' )
    {
      set_av++;
      if(set_av > 24 ) set_av = 1;
       number_echoes = set_av * 10 + 15;
       outport(0x310, 0 );
       outport( WRITE_NUM_ECHOES, number_echoes );
       outport(0x310,15);
    }

    if( key == 'S' )
    {
      set_av--;
      if(set_av < 1 ) set_av = 24;
       number_echoes = set_av * 10 + 15;
       outport(0x310, 0 );
       outport( WRITE_NUM_ECHOES, number_echoes );
       outport(0x310,15);
    }

    if( key == 'A' )
    {
      max_delta = 1 ;
      set_av    = 20;
       number_echoes = set_av * 10 + 15;
       outport(0x310, 0 );
       outport( WRITE_NUM_ECHOES, number_echoes );
       outport(0x310,15);
    }
    if( key == 'B' )
    {
      max_delta = 10 ;
      set_av    = 1;
       number_echoes = set_av * 10 + 15;
       outport(0x310, 0 );
       outport( WRITE_NUM_ECHOES, number_echoes );
       outport(0x310,15);
    }

    if( key == 'C' )
    {
      max_delta = 10;
      set_av    = 3;
       number_echoes = set_av * 10 + 15;
       outport(0x310, 0 );
       outport( WRITE_NUM_ECHOES, number_echoes );
       outport(0x310,15);
    }
```

```
            if(key == 'q') { fcloseall(); return; }

          STP(1,25);
            strcpy(txt," ");
            printf("%s",txt);
          key = '0';
        }

        setcolor(YELLOW);
          y_pt = (int)(350. - vel_scale * yscale * flow [c]);
        moveto( 0, y_pt );
        old_flow[0] = y_pt;

       for(c=1; c<=xmax; c++)                          /* Plot the flow values           */
         {
             y_pt = (int)(350. - vel_scale * yscale * flow [c]);
            if( y_pt > 639 ) y_pt = 479;                  /* Clip points beyond current scale */
           if( y_pt < 0   ) y_pt = 0;
             lineto( (int)( (float)c * xscale ), y_pt );
           old_flow[c] = y_pt;
         }

       if( sav_flag==1 )
       {
         STP(1,25);
          sprintf(txt,"Hit s to save RF for this plot                    ");
            printf("%s",txt);
       }

        if(sav_flag==1 || load_flag ==1 )
       {
          while ( !kbhit() );
         key = getch();
       }
        if(sav_flag == 1 && key == 's' )save_echoes();
         if(key=='q')return;

      }

}

/***************************************************************************/

void  plot_echo()
{
    initgraph(&g_driver,  &g_mode,  "");
    time(&old_time);
   displayed_echo  =  1;

   for(c=0;  c<1024;  c++)  values[c]=0;
```

```
while(1)
  {
   while( ( value = inport(READ_ADDRESS_COUNTER) ) != 0 )

       {
          outport(0x310, 0 );
         outport(LOAD_START_ADDRESS, 0);
        STP(55,18);
           sprintf(txt,"START Address = %4x",value );
          printf("%s",txt);
        STP(55,19);
          strcpy(txt,"END Address    = waiting" );
          printf("%s",txt);
          if( kbhit() ) { getch(); return; }
       }

        STP(55,19);
          strcpy(txt,"END    Address = dtizing" );
          printf("%s",txt);

        STP(55,18);
           sprintf(txt,"START Address = %4x",value );
          printf("%s", txt);

         outport(CLEAR_ND_LATCH, 0 );

         outport(COMP_AQUIRE_DATA, 0 );
         while(  ( inport( RD_TRANS_POS ) & 0x0080 ) == 1 )
           {
                if( kbhit() ) { getch(); return; }
           }

        STP(55,17);
          strcpy(txt,"AQUIRING DATA");
          printf("%s", txt);

         while(  ( inport( RD_TRANS_POS ) & 0x0080 ) == 0 )
           {
                if( kbhit() ){ getch(); return; }
           }

          if( kbhit() ) { getch(); return; }

        value = inport( READ_ADDRESS_COUNTER );
        STP(55,19);

          sprintf(txt, "END    Address = %4x    ",value );
          printf("%s",txt);

        STP(55,17);
        strcpy(txt, "              ");
```

```
printf("%s",txt);


 if(sec !=0)
 {
     chk_time=old_time+sec;
    time(&new_time);
    while(1)                    /*loop until the next sampling period*/
    {
      time(&new_time);
       if (new_time == chk_time)
        break;
         ptr = asctime(localtime(&new_time));
      *(ptr + 20) = 0;
         strcpy(aquire_time, ptr + 11);
      STP(70,1);
         printf("%s",aquire_time);
        if(kbhit()) {getch();return;}
    }
    old_time= new_time;
 }


   xoffset = 0;   /* erase previous trace */
 yoffset = 0;

  setcolor(0);
  moveto( 0, values[0] + yoffset);

  for( c=1; c<1024; c++)
  {
    if( c==640 )
    {
      xoffset = 640;
      yoffset = 224;
        moveto(0, values[640] + yoffset);
    }
    else
       lineto( c-xoffset, values [c] + yoffset);
  }




    if(sav_flag==1) fprintf(fptr, "0\n%s\n",aquire_time);

outport( 0x310, 5 );

for(msa = 0; msa <  display_echoes; msa = msa + 1024 )
{

   if( kbhit() ) { getch(); return; }
```

```
    for( c=display_add; c<display_add+save_echo_length; c++)
 {

      if(display_echoes==1024) msa=1024;
    address = msa+(long)c;

      outport( 0x300, address & 0x00000fff );
      outport( 0x302, address & 0x00000fff );
     top8 = (int)( address / 4096 );
    outport( 0x304,  top8 + 256 * top8 );

   if( bank == -1 )
       value = inport( 0x30a ) & 0x00FF ;
  else
   {
        value = inport( 0x30a ) & 0xFF00 ;
     value = value >> 8;
   }
       if(sav_flag==1) fprintf(fptr,"%d\n",value);

    if( c-display_add < 1024 )
      values [c-display_add] = value;
 }

STP(55,24);
  sprintf(txt, "Add  = %7lx", msa);
 printf("%s", txt);

STP(70,24);
   sprintf(txt,"DA = %d",display_add);
 printf("%s", txt);

STP(55,25);
  sprintf(txt, "Echo = %5u", msa/1024 );
 printf("%s", txt);

STP(55,23);
    sprintf(txt,"%3u %3u %3u %3u", values[8], values[9], values[10], values[11] );
 printf("%s", txt);


xoffset = 0;
yoffset = 0;
 setcolor(14);
 moveto( 0, values[0] + yoffset);

for( c=1; c<1024; c++)
 {
  if( c==640 )
   {
     xoffset = 640;
```

```
                yoffset = 224;
                  moveto(0, values[640] + yoffset);
               }
              else
                 lineto( c-xoffset, values [c] + yoffset);
             }


             STP(1,1);
                  sprintf(txt,"%ld",displayed_echo);
               printf("%s",txt);
               displayed_echo++;

           }
   }
 }

/************************** save_echoes ***************************/
/*                                                                */
/*     Saves     1K echoes to the file rl-time.ech                */
/*                                                                */
/******************************************************************/

void  save_echoes( )

{
  int
      e,
       echo_n
      ;

  long
      base_add
     ;

  FILE *fecho
      ;

  if( ( fecho = fopen( name, "a" ) ) == NULL )
   {
       printf("\nCan't  open  %s!\n",name);
     exit(0);
   }


        for(echo_n = 0; echo_n < num_disk_echoes; echo_n++)
         {
               outport(0x310, 5);
               base_add = 1024 * (long)echo_n;
               printf("%d ",echo_n);
                fprintf( fecho, "%d\n", echo_n);
```

```
            time(&t);
              ptr = asctime(localtime(&t));
            *(ptr + 20) = 0;
                strcpy(aquire_time, ptr+11);
                 fprintf(fecho,"%s\n",aquire_time);

                for( c=display_add; c < save_echo_length + display_add; c++)     /* Read in 1st echo */
             {

                  address = base_add+(long)c;

                    outport( 0x300, address & 0x00000fff );
                    outport( 0x302, address & 0x00000fff );
                  top8 = (int)( address / 4096 );
                  outport( 0x304,  top8 + 256 * top8 );

               if( bank == -1 )
                     value = inport( 0x30a ) & 0x00FF ;
              else
                {
                      value = inport( 0x30a ) & 0xFF00 ;
                  value = value >> 8;
                }
                 fprintf( fecho, "%d\n", value );
             }
        }
fcloseall();
printf("\n");
}

/******************************** load_echoes *********************/
/*                                                                */
/*               Subtracts out stationary echo components         */
/*                                                                */
/******************************************************************/

void  load_echoes( )

{
  int
      e,
       echo_1 [1024],
      echo_n = 0
      ;

  long
      base_add = 0
     ;

  FILE *fecho
      ;
```

```
if( ( fecho = fopen( "rl-time.ech", "r" ) ) == NULL )
 {
    puts("Can't open rl-time.ech!\n");
   exit(0);
 }

   outport(0x310, 15);

  while( echo_n != 255 )
    {
        fscanf( fecho, "%d", &echo_n);
        printf("%d ",echo_n);
        base_add = 1024 * (long) echo_n;

       for(c=0; c<1024; c++)
           fscanf(fecho, "%d", &echo_1[c]);

           for(c = 0; c < 256; c++)
           {
              e = echo_1[4*c] + 256 * echo_1 [4*c+1];
               outport(0x306, e );
              e = echo_1[4*c+2] + 256 * echo_1 [4*c+3];
               outport(0x308, e );

               address = base_add + 4 * (long)c;

                 outport( 0x300, address & 0x00000fff );
                 outport( 0x302, address & 0x00000fff );
               top8 = (int)( address / 4096 );
               outport( 0x304,  top8 + 256 * top8 );
               outport(0x312,0);
           }

    }
}
```

APPENDIX B

ECHO SIGNAL CORRELATION PROGRAM

```
/*************************************************************************************************************/
/*                                                                                                         */
/*                              MEAT ECHO CORRELATION PROGRAM                                              */
/*                                                                                                         */
/*                    Correlates echo signals from n to number of echoes                                   */
/*                                                                                                         */
/*                     Generates files                                                                     */
/*                                                                                                         */
/*                   .cor            average correlation vs tau                                            */
/*                   .sav            average shift vs tau                                                  */
/*                   .d              cor coef vs time for all taus                                         */
/*                   .s              shift vs time for all taus                                            */
/*                                                                                                         */
/*************************************************************************************************************/

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

#define  XSIZE  175
#define  YSIZE  250
#define  ECHO_LENGTH  1024

char  *echoes
   ;

void  main()
{

void  go();

echoes     = (char  *)malloc(942080*sizeof(char));
        if(echoes == NULL ){ puts("Couldn't malloc"); exit(0); }

go(echoes);
}


void  go(fr_1)

signed  char  fr_1[]  [1024]
   ;
{

 char
    key
   ;

int
   x,
   c,
   y,
   i,
   ro,
    delta,
      start_shift,         /*starting value for shift in data points*/
     end_shift,            /*ending value for shift in data points*/
    npts,
    offset,
     data_pt,
      start_pt,            /*data point to start correlations*/
     end_pt,               /*data point to end correlations*/
     cor_cntr,
     echo_spacing,         /*sampling period in minutes*/
     start_echo,           /*echo signal to begin correlations*/
     echo_length = 1024,
    echo_cntr = 0
   ;

float
     r,                     /* correlation coefficient */
     sum_r,                 /* sum of cor coeffs, used in calculating average   */
     rmax,                  /* maximum value of corr coefficient      */
       sum_squared_1,
       sum_squared_2,
      sum_offset=0,
     s_average,             /* Average shift for a delta     */
     r_average,             /* Average correlation for a delta */
     zro,
      sum_1_x_2
     ;

char
      input_file  [20],
       output_file [20],
     outfiled     [20],
     outfiles     [20],
     file_name    [20],
     echo_time    [10]
    ;

 FILE
      *fptr
      *fptrs,
      *fptrd
     ;

          if( ( fptr = fopen( "meat.sys", "r") )== NULL )
```

```
        {
            printf("corstuf: Couldn't open meat.sys!");
          exit(0);
        }

  fscanf(fptr,"%d  %d  %d", &start_echo, &start_pt, &end_pt);
  fscanf(fptr,"%s  %d",file_name, &echo_length);
  fscanf(fptr,"%d", &echo_spacing );
   fscanf(fptr,"%d  %d",&start_shift, &end_shift);
 fclose( fptr );

  printf("Start echo      = %d\n", start_echo);
  printf("Start pt        = %d\n", start_pt);
 printf("End    pt       = %d\n", end_pt);
 printf(" Filename       = %s\n", file_name );
  printf(" Echo_length    = %d\n", echo_length );
  printf(" Echo_spacing = %d\n", echo_spacing);
   printf(" Start shift = %d\n", start_shift);
  printf(" End_shift     = %d\n", end_shift);

   strcpy(input_file, file_name);

   strcat(input_file, ".ech");

  printf("\n File = %s    Echo length = %d \n", input_file, echo_length);

   printf("\n\nReading %s\n", input_file);
 if( ( fptr = fopen( input_file, "r") )== NULL )
  {
      printf("Couldn't open %s!", input_file);
    exit(0);
   }

   while( (fscanf(fptr,"%s",&echo_time) ) != EOF )                /*reading echo file*/
    {
     fscanf(fptr,"%s",&echo_time);            /*reading in time of first echo*/
     printf("%d  %s\n", echo_cntr, echo_time);

     for(y=0; y<ECHO_LENGTH; y++) /*reading echo signal*/
        {
         fscanf(fptr,"%d", &data_pt );
         fr_1 [echo_cntr] [y] = (char)(data_pt-128);
        }

     if(echo_length == 4096)
         {
              /* skip over rest of echo */
          for(y=0; y< 3072; y++) fscanf(fptr,"%d",&data_pt);
         }

              echo_cntr++;  /*count the number of echo signals in the file*/
         }
      puts("Awesome! End of file encoutered!  Ready to Analyze!!");
       printf("Total of %d echoes read in!\n\n", echo_cntr );


fclose(fptr);

      strcpy(outfiled,file_name);
      strcpy(outfiles,file_name);
     strcat(outfiled,".d");
     strcat(outfiles,".s");

   if(  ( fptrd = fopen( outfiled, "w") )== NULL )
  {
        printf("Couldn't open %s!", outfiled);
     exit(0);
  }

   if(  ( fptrs = fopen( outfiles, "w") )== NULL )
  {
        printf("Couldn't open %s!", outfiles);
     exit(0);
  }

 for( delta=1; delta<echo_cntr-1; delta++ )
{
  sum_r=0.0;
  sum_offset=0;

      printf("\n Writing to file %s\n", outfiled);
      printf("\n Writing to file %s\n", outfiles);

  npts=0;
      for(cor_cntr = start_echo; cor_cntr < echo_cntr - delta; cor_cntr++ )
 {
   rmax=-1.0;
        for(offset=start_shift; offset<=end_shift; offset++)
  {

     sum_1_x_2      = 0.0;
      sum_squared_2 = 0.0;
      sum_squared_1 = 0.0;

       for(c=start_pt; c<end_pt; c++)
           sum_squared_1 += (float) fr_1 [cor_cntr] [c] * (float) fr_1 [cor_cntr] [c];

         for(c=start_pt; c<end_pt; c++)
              sum_squared_2+=(float)fr_1[cor_cntr+delta][c+offset] *
                 (float)fr_1[cor_cntr+delta][c+offset];


        sum_1_x_2 = 0.0;
```

```
            /********************calculating corr. coeff. at shift offset************************/

                for(c=start_pt; c<end_pt; c++)
                        sum_1_x_2 += (float) fr_1 [cor_cntr] [c] * (float) fr_1 [cor_cntr+delta] [c+offset];
                      r = sum_1_x_2 / ( (float) sqrt( (double)sum_squared_1 * (double)sum_squared_2 ) );
              if(r >= rmax)
              {
                rmax=r;
                ro=offset;
              }
            } /* offset */

            printf(" D = %d  Correlation %d - %d = %f  %d\n", delta, cor_cntr, cor_cntr+delta,rmax, ro);

                    fprintf(fptrd, "%f ", rmax);
                    fprintf(fptrs, "%f ", (float)ro);

              npts++;

           } /* cor_cntr */

/**zero padding of the correlation and shift files to have proper format for Macintosh Spyglass**/
        zro = 0;
         if(delta > 2)
        {
            for(i=1;i<delta;i++)
           {
               fprintf(fptrd,"%f ",zro);
               fprintf(fptrs,"%f ",zro);
           }
        }
         if(delta == 2)
        {
              fprintf(fptrd,"%f ",zro);
              fprintf(fptrs,"%f ",zro);
        }

           fprintf( fptrd,"\n");
           fprintf( fptrs,"\n");

        }  /* delta */


       puts("Whew... that was hard! But I'm done now.");
      exit(0);
}
```

APPENDIX C

STANDARD DEVIATION AND AVERAGING PROGRAM

```
/*****************************************************************************************************/
/*                                                                                                 */
/*                        STANDARD DEVIATION AND AVERAGING PROGRAM                                 */
/*                                                                                                 */
/*                          Adds all the data points from an echo and takes the average            */
/*                        Calculates average power in echoes also.                                 */
/*                                                                                                 */
/*                        generates the following files:                                           */
/*                                                                                                 */
/*                             .avg  average value of echo                                         */
/*                            .sts                                                                 */
/*                               .std  standard deviation of echo                                  */
/*                                                                                                 */
/*****************************************************************************************************/


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


#define XSIZE 175
#define YSIZE 250
#define ECHO_LENGTH 1024



main()


{

char
    fr_1 [300][1024]
   ;

int
   x,
   c,
   y,
    data_pt,
    start_pt,           /*point in each echo to start calculations with*/
    end_pt,             /*point in each echo to end calculations with*/
    start_echo,         /*echo to begin calculations with*/
    echo_length,        /*length of each digitized echo signal*/
    echo_spacing,        /*sampling period in minutes*/
    echo_cntr = 0,
    yoffset,
    xoffset,
    echo,
    average_num
   ;

float
     r,
       sum_average=0.0,  /*average value of echo for a particular  τ*/
        sum_squares=0.0,
      variance,          /*variance*/
       std_dev           /*standard  deviation*/
     ;

char
     file_name    [20],
      input_file  [20],
     outfile_avg [20],
      outfile_std [20],
     echo_time    [10]
    ;

 FILE
     *f_avg,
      *f_std,
      *fptr
     ;

            if(  ( fptr = fopen( "/mnt/meat/meat.sys", "r") )== NULL ) /*open file containing */
                                                                        /*parameters  for  calculations*/
          {
                printf("Couldn't open /mnt/meat/meat.sys!");
            exit(0);
          }

fscanf(fptr,"%d  %d  %d", &start_echo, &start_pt, &end_pt);

fscanf(fptr,"%s  %d",file_name,  &echo_length);
fscanf(fptr,"%d",  &echo_spacing  );

fclose(  fptr  );

printf("Start  echo      = %d\n",  start_echo);
printf("Start  pt        = %d\n",  start_pt);
printf("End    pt        = %d\n",  end_pt);
printf(" Filename        = %s\n",  file_name );
printf(" Echo_length     = %d\n",  echo_length );
printf(" Echo_spacing  =  %d\n",  echo_spacing);
printf("\n  echo_length  =  %d\n",echo_length);

strcpy(input_file,  file_name);
strcpy(output_file,  file_name);
strcpy(outfile_avg,  file_name);
strcpy(outfile_std,  file_name);

strcat(input_file,  ".ech");
strcat(output_file,  ".sts");
strcat(outfile_avg,  ".avg");
strcat(outfile_std,  ".std");


           printf("\n\nReading %s\n", input_file);       /*open file containing echo signal*/
         if(  ( fptr = fopen( input_file, "r") )== NULL )
        {
            printf("Couldn't open %s!", input_file);
          exit(0);
        }


           while( (fscanf(fptr,"%s",&echo_time) ) != EOF )  /*read echo signal until end of file*/
           {
               printf("%s\n",echo_time);                    /*read in time echo signal acquired*/
               fscanf(fptr,"%s",&echo_time);
             echo_cntr++;                                   /*count the total number of echo   */
                                                            /*signals in file*/
                printf("%d %s\n", echo_cntr, echo_time);

            for(y=0; y<ECHO_LENGTH; y++)      /*read in 1024 data points and store in array*/
             {
                 fscanf(fptr,"%d", &data_pt );
                   fr_1 [echo_cntr] [y] = (char)(data_pt -128);
             }

                if(echo_length == 4096)
                     for(y=0; y< 3072; y++) fscanf(fptr,"%d",&data_pt);   /* skip over rest of echo */
           }

           puts("Awesome! End of file encoutered!  Ready to Analyze!!");
            printf("Total of %d echoes read in!\n\n", echo_cntr );

         fclose(fptr);

         if(  ( f_avg = fopen( outfile_avg, "w") )== NULL )
        {
            printf("Couldn't open %s!", outfile_avg);
          exit(0);
        }

         if(  ( f_std = fopen( outfile_std, "w") )== NULL )
        {
```

```
        printf("Couldn't open %s!", outfile_std);
      exit(0);
    }
       fprintf(f_avg, "%d\n%d\n",echo_spacing, echo_cntr); /*print to both the st. deviation and*/
       fprintf(f_std, "%d\n%d\n",echo_spacing, echo_cntr); /*the ·average file, the sampling */
                                                            /*period and the total # of echo signals*/
     puts("calculating average and st dev of echoes.");

     for(average_num = 1;average_num <= echo_cntr;average_num++)
    {
      sum_average=0.0;
       sum_squares=0.0;

         for(c=start_pt; c<end_pt; c++)  /*sum up the values of the echo signal and the square of*/
                                         /*the data point values*/
     {
          sum_average += (float) fr_1 [average_num] [c];
             sum_squares +=(((float)fr_1 [average_num] [c])*((float)fr_1 [average_num] [c]));
     }

          sum_squares = sum_squares/(float)(end_pt - start_pt);
         sum_average = sum_average/(float)(end_pt - start_pt);  /*calculation of average*/

        variance= (sum_squares - (sum_average * sum_average));
       std_dev = (float)sqrt( (double)variance );      /*calculation of the standard deviaiton*/

         fprintf(f_avg,"% 6.2f % 6.2f\n", (float)average_num * (float)echo_spacing, sum_average );
         fprintf(f_std,"% 6.2f % 6.2f\n", (float)average_num * (float)echo_spacing, std_dev );

    }

     fclose(f_avg);
      fclose(f_std);

      puts("Whew... that was hard! But I'm done now.");
    exit(0);
}
```

APPENDIX D

PLOTTING OF CORRELATION ANALYSIS DATA

```
/*************************************************************************************************************/
/*                                                                                                          */
/*  Plotting program for all of the meat data analysis programs. This version                               */
/*  AUTOMATICALLY generates gem files to be imported into the designer program                              */
/*  and must be linked with grgem.lib.  grgem.lib has modified the routines                                 */
/*  from Graphic 4.1 to automatically do this. The original Graphic 4.1                                     */
/*  routines were bplt.c, tplt.c, and prntset.c and the modified ones are                                   */
/*  bplta.c, tplta.c and prntseta.c.  The gem filename is stored in the                                     */
/*  variable gem_file, and prntseta.c reads this instead of prompting                                       */
/*  for a filename. The gem file extensions are listed in the menu page with                                */
/* the square brackets.                                                                                     */
/*                               BY ILMAR HEIN                                                              */
/*************************************************************************************************************/

#include  "graph.h"
#include  "stdio.h"
#include  "stdlib.h"
#include  "process.h"

#define CLS "\x1B[2J"



 float
      x_data[1024],
     y[1024],

     xmin [4],
     xste [4],
     xmax [4],

     ymin [9],
     yste [9],
     ymax [9],

     xmin_p,
     xste_p,
     xmax_p,

     ymin_p,
     yste_p,
     ymax_p

    ;

    int
```

```
      echo_spacing = 1,        /*sampling  period*/
      t_f_death = 0
     ;

    int
       echo_spacing_time,
      delta,                   /*tau*/
       err_flag=0,
     cc,
     i,
      npts,
       numpoints,
      fntc,
      sym
     ;

     char fnts;
     char  fnt[6] = "\312";
    char
      a,
      c,
      d,
      e,
      f,
      h,
      p,
      t,
      s,
      x,
       command[30],
         list_files[30],
       choice,
        old_choice,
       key,
       fnme[15],
        gem_file[15],
        data_file [15],
        plot_file [15],
       xnme[35],
       ynme[35],
        dummy[100],
        title[70]
      ;

      FILE *fptr;

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

strcpy(gem_file,"test.gem");
```

```
if(argc !=2)
    strcpy( fnme, "-----" );
  else strcpy (fnme,argv[1]);

choice = 'a';
a = '*';

  if( (fptr = fopen("axes.sys","r" )) == NULL )
 {
    printf("\n\n CAN'T FIND AXES.SYS! HELP ME!\n");
   exit(0);
 }

   puts("Reading axes scaling info...\n");

      fgets(dummy,100,fptr);
     fscanf( fptr, "%s",dummy );            /* x-axis origin data */
     printf("%s: ",dummy );
   for( cc = 0; cc < 4; cc++ )
    {
        fscanf( fptr, "%f",&xmin[cc] );
        printf("%4.2f ", xmin[cc] );
    }
   puts(" ");

     fscanf( fptr, "%s",dummy );            /* x-axis step data */
     printf("%s: ",dummy);
   for( cc = 0; cc < 4; cc++ )
    {
        fscanf( fptr, "%f",&xste[cc] );
        printf("%4.2f ",xste[cc] );
    }
    puts(" ");

     fscanf( fptr, "%s",dummy );            /* x-axis maximum data */
     printf("%s: ",dummy);
   for( cc = 0; cc < 4; cc++ )
    {
        fscanf( fptr, "%f",&xmax[cc] );
        printf("%4.2f ",xmax[cc] );
    }
      printf("\n\n");


    for(cc=0; cc<9; cc++)
         fscanf( fptr, "%s",dummy );            /* y-axis minimum data */
      fscanf( fptr, "%s",dummy );           /* y-axis minimum data */
      printf("%s: ",dummy);
    for( cc = 0; cc < 9; cc++ )
     {
         fscanf( fptr, "%f",&ymin[cc] );
```

```
            printf("%4.2f ",ymin[cc]);
        }
        puts(" ");

         fscanf( fptr, "%s",dummy );           /* y-axis minimum data */
         printf("%s: ",dummy);
        for( cc = 0; cc < 9; cc++ )
         {
            fscanf( fptr, "%f",&yste[cc] );
            printf("%4.2f ",yste[cc] );
         }
         puts(" ");

         fscanf( fptr, "%s",dummy );           /* y-axis maximum data */
         printf("%s: ",dummy);
        for( cc = 0; cc < 9; cc++ )
         {
            fscanf( fptr, "%f",&ymax[cc] );
            printf("%4.2f ",ymax[cc] );
         }
          printf("\n\n");
         fclose( fptr );

while  (1)
   {
     printf(CLS);
   printf("************** MEAT PLOTALL Version Auto-GEM 1.1 *************************\n");
     printf("\n%c [a] Averages of echoes              vs time  (.avg) [.g1 ]"     ,a);
     printf("\n%c [c] average Correlation             vs delta (.cor) [.g2 ]"     ,c);
      printf("\n%c [d] correlations  (delta = 1 5 10 30 60) vs time  (.d* ) [.gda - .gdf]" ,d);
     printf("\n%c [e] avErage shift of max corr        vs delta (.sav) [.g4 ]"     ,e);
     printf("\n%c [f] average Frequency               vs time  (.fav) [.g5 ]"     ,f);
      printf("\n%c [h] sHift of corr (delta = 1 5 10 30 60) vs time  (.s*)  [.gsa - .gsf]" ,h);
     printf("\n%c [p] average Power                  vs time  (.apw) [.g7 ]"     ,p);
     printf("\n%c [t] aTtenuation                    vs time  (.a*)  [.gaa - .gae]" ,t);
     printf("\n%c [s] Standard Dev                   vs time  (.std) [.g9 ]"     ,s);
     printf("\n%c [x] maX frequency                  vs time  (.fmx) [.g10]"     ,x);
   printf("\n\n***********************************************************\n\n");

       printf("Other  options:\n");
     printf("\n             [l] Listing of files");
     printf("\n              [n] New file (currently = %s)", fnme);
     printf("\n              [z] Time from death = %d min",t_f_death );
     printf("\n             [q] Quit" );

      printf("\n\n~~~ Talk to me  --->");

    a = c = d = e = f = h = p = t = s = x = ' ';
     err_flag = 0;

     old_choice = choice;
      fflush(stdin);
```

```
choice = getch();
 if( choice == 13 ) choice = old_choice;
switch( choice )
  {

    case 'a':
          a = '*';

           ymin_p = ymin[0];
           yste_p = yste[0];
           ymax_p = ymax[0];

            strcpy( gem_file,  fnme   );
             strcpy( data_file, fnme   );
             strcpy( plot_file, fnme   );
             strcat( data_file, ".avg" );    /*set up to open the average file and the*/
             strcat( plot_file, ".pav" );    /*corresponding plot and GEM file*/
            strcat( gem_file,  ".g1"  );

              strcpy( xnme, "Time from death (min)");
             strcpy( ynme, "Average ( x 3.9 mv) ");
            strcpy( title, fnme );
             strcat( title, ": Ave value of echo");

            read_and_plot();

          break;


    case 'c':
          c = '*';

           ymin_p = ymin[1];
           yste_p = yste[1];
           ymax_p = ymax[1];

            strcpy(gem_file,   fnme   );
             strcpy( data_file, fnme   );
             strcpy( plot_file, fnme   );
             strcat( data_file, ".cor" );    /*set up to open the average correlation file*/
             strcat( plot_file, ".pac" );    /*and the corresponding plot and GEM file*/
            strcat( gem_file, ".g2" );

             strcpy( xnme, "Delta (min)");
              strcpy( ynme, "Average Correlation" );
            strcpy( title, fnme );
             strcat( title, ": Ave Delta Corr");

            read_and_plot();

          break;
```

```
case 'd':
      d = '*';

        ymin_p = ymin[2];
        yste_p = yste[2];
        ymax_p = ymax[2];

           strcpy( xnme, "Time since death (min) ");
          strcpy( ynme, "Correlation Coeff" );



        for( c=0; c<5; c++ )
        {
             strcpy( gem_file , fnme   );
             strcpy( data_file, fnme   );
             strcpy( plot_file, fnme   );

          if( c == 0 )                /*several files need to be opened for the correlation*/
          {                           /*coefficient plots. There is a different file for each*/
                  strcat( data_file, ".d1" ); /*tau (1,5,10,30,60 min. or */
                                          /*10, 50, 100, 300, 600 min)*/
               strcat( plot_file, ".x1" );
              strcat( gem_file,  ".gda");
          }


          if( c == 1 )
          {
               strcat( data_file, ".d5");
               strcat( plot_file, ".x5");
              strcat( gem_file,  ".gdb");
          }

          if( c == 2 )
          {
               strcat( data_file, ".d10");
               strcat( plot_file, ".x10");
              strcat( gem_file,  ".gdc");
          }

          if( c == 3 )
          {
               strcat( data_file, ".d30");
               strcat( plot_file, ".x30");
              strcat( gem_file,  ".gdd");
          }

          if( c == 4 )
          {
            strcat( data_file, ".d60");
            strcat( plot_file, ".x60");
           strcat( gem_file,  ".gde");
        }


          strcpy( xnme, "Time from death (min) ");
          strcpy( ynme, "Correlation Coeff" );
          strcpy( title, data_file );
         strcat( title, ": Cor Coeff vs Time");

        read_and_plot();
         if(err_flag==1)break;
        if( kbhit() ) break;
      }


      break;


case 'e':
           printf("\nAverage shift of maximum correlation\n");
      e = '*';

       ymin_p = ymin[3];
       yste_p = yste[3];
      ymax_p = ymax[3];

        strcpy( gem_file, fnme    );
         strcpy( data_file, fnme   );
         strcpy( plot_file, fnme   );
          strcat( data_file, ".sav" );
          strcat( plot_file, ".psv" );
        strcat( gem_file,  ".g4"  );

          strcpy( xnme, "Delta (min)");
           strcpy( ynme, "Ave shift (x 20 nsec)" );
         strcpy( title, fnme );
            strcat( title, ": Ave shift for max cor");

        read_and_plot();

      break;


case 'f':
          printf("\nAverage Frequency\n");
      f = '*';

       ymin_p = ymin[4];
       yste_p = yste[4];
      ymax_p = ymax[4];
```

```
            strcpy( gem_file, fnme    );
            strcpy( data_file, fnme   );
            strcpy( plot_file, fnme   );
            strcat( data_file, ".fav" );
            strcat( plot_file, ".pfv" );
            strcat( gem_file,  ".g5"  );

            strcpy( xnme, "Time from death (min)");
            strcpy( ynme, "Ave Freq (MHz)" );
            strcpy( title, fnme );
            strcat( title, ": Ave Freq");

            read_and_plot();

          break;



    case 'h':
          h = '*';

          ymin_p = ymin[5];
          yste_p = yste[5];
          ymax_p = ymax[5];

            strcpy( xnme, "Time since death (min) ");
            strcpy( ynme, "Shift (x 20 nsec)" );

          for( cc=0; cc<5; cc++ )
          {
              strcpy( gem_file,  fnme   );
              strcpy( data_file, fnme   );
              strcpy( plot_file, fnme   );

            if( cc == 0 )
            {
                strcat( data_file, ".s1");
                strcat( plot_file, ".z1");
                strcat( gem_file,  ".gsa");
            }


            if( cc == 1 )
            {
                strcat( data_file, ".s5");
                strcat( plot_file, ".z5");
                strcat( gem_file,  ".gsb");
            }


            if( cc == 2 )
```

```
                {
                        strcat( data_file, ".s10");
                        strcat( plot_file, ".z10");
                        strcat( gem_file,  ".gsc");
                }

                if( cc == 3 )
                {
                        strcat( data_file, ".s30");
                        strcat( plot_file, ".d30");
                        strcat( gem_file,  ".gsd");
                }

                if( cc == 4 )
                {
                        strcat( data_file, ".s60");
                        strcat( plot_file, ".z60");
                        strcat( gem_file,  ".gse");
                }



                    strcpy( title, data_file );
                    strcat( title, ": Shift vs time");

                  read_and_plot();
                  if(err_flag == 1)break;
            }

          break;


case 'p':
        p = '*';

          ymin_p = ymin[6];
          yste_p = yste[6];
          ymax_p = ymax[6];

            strcpy( gem_file,  fnme   );
            strcpy( data_file, fnme   );
            strcpy( plot_file, fnme   );
            strcat( data_file, ".apw" );
            strcat( plot_file, ".pvp" );
            strcat( gem_file,  ".g7" );

            strcpy( xnme, "Time from Death (min)");
            strcpy( ynme, "Ave Power (dBm)" );
            strcpy( title, fnme );
            strcat( title, ": Ave power in echo");

            read_and_plot();
        break;

    case 't':
        t = '*';
            printf("\nAttenuation\n");
        ymin_p = ymin[7];
        yste_p = yste[7];
        ymax_p = ymax[7];

            strcpy( xnme, "Time since death (min) ");
            strcpy( ynme, "Attenuation (dB/cm)" );

        for( cc=0; cc<5; cc++ )
        {
            strcpy( gem_file,  fnme   );
             strcpy( data_file, fnme   );
             strcpy( plot_file, fnme   );

          if( cc == 0 )
          {
                strcat( data_file, ".a3");
                strcat( plot_file, ".w3");
               strcat( gem_file,  ".gaa");
          }

          if( cc == 1 )
          {
                strcat( data_file, ".a4");
                strcat( plot_file, ".w4");
               strcat( gem_file,  ".gab");
          }

          if( cc == 2 )
          {
                strcat( data_file, ".a5");
                strcat( plot_file, ".w5");
               strcat( gem_file,  ".gac");
          }

          if( cc == 3 )
          {
                strcat( data_file, ".a6");
                strcat( plot_file, ".w6");
               strcat( gem_file,  ".gad");
          }

          if( cc == 4 )
          {
                strcat( data_file, ".a7");
                strcat( plot_file, ".w7");
               strcat( gem_file,  ".gae");
            }

              strcpy( title, data_file );
              strcat( title, ": Atten vs time");

             read_and_plot();
             if( err_flag == 1 )break;
          }

        break;

case 's':
        s = '*';

         ymin_p = ymin[8];
         yste_p = yste[8];
         ymax_p = ymax[8];

          strcpy( gem_file,  fnme   );
          strcpy( data_file, fnme   );
          strcpy( plot_file, fnme   );
           strcat( data_file, ".std" );     /*set up to open the average standard deviation file*/
           strcat( plot_file, ".psd" );     /*and the corresponding plot and GEM file*/
          strcat( gem_file,  ".g9" );

            strcpy( xnme, "Time from death (min)");
          strcpy( ynme, "Std Dev ( x 3.9 mv)");
          strcpy( title, fnme );
           strcat( title, ": Ave value of echo");

          read_and_plot();

        break;


case 'x':
              printf("\nMaximum Frequency\n");
        x = '*';

         ymin_p = ymin[4];
         yste_p = yste[4];
         ymax_p = ymax[4];

          strcpy( gem_file,  fnme   );
          strcpy( data_file, fnme   );
          strcpy( plot_file, fnme   );
           strcat( data_file, ".fmx" );
           strcat( plot_file, ".pfm" );
          strcat( gem_file,  ".g10" );

            strcpy( xnme, "Time from death (min)");
```

```
                strcpy( ynme, "Max Freq (MHz)" );
                strcpy( title, fnme );
                strcat( title, ": Max Freq");

               read_and_plot();

             break;




     case 'l':

             printf(CLS);
               strcpy(command,"dir ");
                printf("\n\n DIR of what files (. for all)  >");
               scanf("%s", list_files);
                strcat(command, list_files);
               strcat(command," /p");
               printf("%s\n",command);
              if( system(command) != 0 )
             {
                   printf("Ouch! Unable to execute %s\n",command);
                    printf("Perhaps you should call a human for help.\n");
             }
                 puts("Gently press a key to continue");
             key = getche();
             break;

     case 'n':
                printf("\n\n Yo! Enter new filename!  >");
              scanf("%s", fnme);
             break;

     case 'z':
                  printf("\n\n How long since the animal expired?  >");
               scanf("%d", &t_f_death);
             break;

     case 'q':
             printf(CLS);
                printf("\n\nCiao babe\n");
             exit(0);
            break;

 default:
     printf(CLS);
          puts("A funky symbol that I can't understand has just been input.");
          puts("Press anything to continue, and I'll forget it ever happened...");
     key = getche();
   ;
```

```
        } /* switch */

      } /* while(1) */

}


read_and_plot()
{

        printf("\nOpening %s \n",data_file);
        if( (fptr=fopen( data_file, "r"))==NULL)

      {
         err_flag=1;
         fclose(fptr);
            printf("\nCan't find %s!!!\n", data_file);
          puts("Hit [q] to quit or any other key to go on!");
         key = getche();
           printf("\n\n A wise choice.");

         if( key == 'q' || key == 'Q' ) exit(0);
         else return;
      }


         printf("\nPlotting to file %s\n",plot_file);


         fscanf(fptr,"%d",&echo_spacing_time);
        fscanf(fptr,"%d",&npts);
        printf("\n Echo spacing time = %d    npts = %d\n",echo_spacing_time, npts );

              /*--------------------------------set  x-axes  scales  ------------------------*/
        if( echo_spacing_time == 1)
       {
           if( c == '*' || e == '*' )     /* avg correlation or avg shift */
          {
             xmin_p = xmin[2] ;
             xste_p = xste[2] ;
             xmax_p = xmax[2] ;
          }

          else
          {
             xmin_p = xmin[0] ;
             xste_p = xste[0] ;
              xmax_p = xmax[0] + t_f_death ;
          }
```

```
}

 else if( echo_spacing_time == 10 )
{
     if( c == '*' || e == '*' )      /* avg correlation or avg shift */
    {
       xmin_p = xmin[3] ;
       xste_p = xste[3] ;
       xmax_p = xmax[3] ;
    }

    else
    {
       xmin_p = xmin[1] ;
       xste_p = xste[1] ;
        xmax_p = xmax[1] + t_f_death ;
    }
}


        /*--------------------------------  read  in  the  data  ------------------------------*/
 for(i=0; i<npts; i++)
 {
       fscanf(fptr,"%f %f",&x_data[i], &y[i]);
   if( c == '*' || e == '*' )
    ;
    else
      x_data[i] += (float)t_f_death;
 }

 fclose(fptr);

  printf ("xmin = %f   ymin = %f\n",xmin_p, ymin_p);
  printf ("xste = %f   yste = %f\n",xste_p, yste_p);
  printf ("xmax = %f   ymax = %f\n",xmax_p, ymax_p);


         /*--------------------------------  do  the  plot  ----------------------------------*/
   bgnplot(1,'g',plot_file);
  startplot(0);
        font(4,"microg.fnt",'\310',"triplex.fnt",'\311',"simplex.fnt",'\312'
         ,"duplex.fnt",'\313');

 page(9.0,6.855);
 area2d(8.0,5.5);
 upright(1);

sym = 0;
 xname(fnt);
 color(9);
    graf("%-1.0f", xmin_p, xste_p, xmax_p, "%-4.2f", ymin_p, yste_p, ymax_p );
```

```
        color(14);

         heading(title);
        color(9);

        yname(ynme);
        xname(xnme);

        color(14);

          curve(x_data, y, npts, sym);

        endplot();
        stopplot();
}
```

APPENDIX E

PROGRAM TO AVERAGE CORRELATION DATA SETS

```
/*****************************************************************************************************/
/*                    PROGRAM TO AVERAGE CORRELATION DATA SETS                                     */
/*                                                                                                 */
/*              Reads in correlation files and time after death that data acquisition began        */
/*              Truncates data so that averaging starts with the data points at the longest time   */
/*              after death and stops averaging when it reaches the end of the shortest file       */
/*                                                                                                 */
/*            Averaged file stored in *.avg                                                        */
/*              Standard deviation of averaged correlation - *.std                                 */
/*****************************************************************************************************/


#include <stdio.h>
#include <io.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>

int filenumber = 2,     /*number of files to be averaged*/
    delta = 1,          /*echo sampling period in minutes*/
     echo_counter[15],
     echo_counter2[15],
     end_of_average,
     tim_death[30],     /*time after death data acquisition began*/
     long_death,
     short_echo,
    tim,
     fil_length,
     g_driver,
    g_mode,
     g_error,
   a = 0,
   b = 0,
   c = 0,
   d = 0,
   j = 0,
   i;

FILE *fptr,
     *fpts,
     *fptd;

char filename[30][15],
      inputfile[15],
      tempfile[30][15],
      temp_outfil1[15],
        temp_outfil2[15],
      temp,
      choice,
      key,
        outputfil[15];


float  data,
       zro,
        sum_cor,
        avcor[500],
        avcor2[500],
         standard_dev,      /*standard deviation of averaged correlation*/
       sum = 0,
       average = 0.0,
        average2[500],
        square,
         square_2[500],
          data_array[300][15];

void  av_corr();


main()
{
     detectgraph(&g_driver, &g_mode);
     initgraph(&g_driver, &g_mode, "");
     g_error = graphresult();
   if (g_error < 0)
     {
           printf("Graphing Error:%s\n",grapherrormsg(g_error));
       exit(1);
     }
   closegraph();

   while(1)
  {
       initgraph(&g_driver, &g_mode, "");
      textmode(3);
      cleardevice();
      closegraph();

       printf("\n (a) Number of files = %d",filenumber);
       printf("\n (b) Enter file names with extension");
       printf("\n (o) Output file name (%s)", outputfil);
       printf("\n (s) Echo spacings. (%d)", delta);
       printf("\n (c) Start averaging");
       printf("\n (q) Quit");

       printf("\n\nWhat's up doc?");

     choice = getch();
```

```
switch(choice)
{
   case 'a':        /*user enters number of files to be averaged*/
         printf("\nEnter the number of files you want to average ");
        scanf("%d",&filenumber);
     break;

   case 'b':        /*user enters the name of the correlation files to be averaged*/
      for(i=1;i <= filenumber; i++)
    {
            printf("\nEnter filename %d ",i);
           scanf("%s",inputfile);
            strcpy(filename[i],inputfile);

           if((fptr = fopen(filename[i],"r")) == NULL)
        {
                printf("\n Couldn't open file %s",filename[i]);
              printf("\n Type A to abort");
            key = getch();

            if(key == 'a' || key == 'A')
          {
             exit(0);
             break;
          }
        }
          fclose(fptr);

            printf("\n Enter time from death ");    /*enter time of death*/
          scanf("%d", &tim_death[i]);

          if (i == 1) long_death = tim_death[i];
          if (tim_death[i] > long_death)
        {
             long_death = tim_death[i];
        }
   }
    break;

   case 'o':
         printf("\n Enter output filename with 6 or less characters ");
        scanf("%s",outputfil);
         strcpy(temp_outfil1,outputfil);
         strcpy(temp_outfil2,outputfil);
        strcat(temp_outfil1,".cor");
        strcat(temp_outfil2,".std");
     break;

   case 's':
       printf("\n Enter echo spacing ");
      scanf("%d",&delta);
    break;
```

```
        case 'c':
          av_corr();
          break;

        case 'q':
          exit(0);
          break;
    }
  }
}

/**************************************************************************/
/*              Averaging of Correlation Coefficients                    */
/**************************************************************************/

void  av_corr()
{
      initgraph(&g_driver, &g_mode, "");
     textmode(3);
     cleardevice();
     closegraph();


     if((fpts = fopen(temp_outfil1,"w")) == NULL)
   {
         printf("\n Couldn't open %s",temp_outfil1);
       printf("\n Press a");
      key = getch();
      if(key == 'a' || key == 'A')
     {
         exit(0);
     }
   }

     if((fptd = fopen(temp_outfil2,"w")) == NULL)
   {
         printf("\n Couldn't open %s",temp_outfil2);
       printf("\n Press a");
      key = getch();
      if(key == 'a' || key == 'A')
     {
     }
   }

    for(i=1;i <= filenumber; i++)              /*reads in the first set of data to determine the */
   {                                           /*amount of data points to be averaged             */

        tim = tim_death[i];
          strcpy(tempfile[i],filename[i]);

         if((fptr = fopen(tempfile[i],"r")) == NULL)
    {
           printf("\n Couldn't open %s",tempfile[i]);
         printf("\n Press a");
       key = getch();
       if(key == 'a' || key == 'A')
      {
         exit(0);
         break;
      }
    }

     echo_counter[i] = 0;
      if( kbhit()) exit(0);

     while(1)
    {
          fscanf(fptr,"%c",&temp);              /*reading in data*/
         if(temp == '\n') break;
         if( kbhit()) exit(0);

        echo_counter[i]++;                      /*counting number of data points*/
    }

     if(tim < long_death)                       /*subtracts off differences in length caused by*/
                                                /*differences in time from death              */
            echo_counter2[i] = echo_counter[i]/9 - ((long_death - tim)/delta);
       else echo_counter2[i] = echo_counter[i]/9;

        printf("\necho length = %d",echo_counter2[i]);

     if(i == 1) short_echo = echo_counter2[i];

     if(echo_counter2[i] < short_echo)
        short_echo = echo_counter2[i];

     if(i == 1) end_of_average = (echo_counter[i]/9);

      if((echo_counter[i]/9) < end_of_average)
         end_of_average = echo_counter[i]/9;


     fclose(fptr);
}

b = 0;

  while(b < end_of_average)                     /*read in data files */
{
      if( kbhit()) exit(0);
        printf("\nPerforming average on delta = %d",((b+1)*delta));

     for(i=1;i <= filenumber; i++)
```

```
{
      tim = tim_death[i];
        strcpy(tempfile[i],filename[i]);
       if( kbhit()) exit(0);

       if((fptr = fopen(tempfile[i],"r")) == NULL)
    {
             printf("\n Couldn't open %s",tempfile[i]);
           printf("\n Press a");
         key = getch();
          if(key == 'a' || key == 'A')
        {
             exit(0);
            break;
        }
    }

    j = 0;
    a = 0;
     if(b >= 25 && b <= 35)
         printf("a = %d, b = %d, echo_counter = %d \n",a,b,((b+1) * (echo_counter[i]/9)));

       while(a < ((b+1) * (echo_counter[i]/9)))
    {
           if( kbhit()) exit(0);
            fscanf(fptr,"%f",&data);

           if(a >= (b * (echo_counter[i]/9)))
        {
              if(tim >= long_death)           /*only save data after the longest time after death*/
           {
                data_array[j][i] = data;
               if(b >= 25 && b <= 35)
                    printf("set %d data %f",i,data_array[j][i]);
              j++;
           }
             tim = tim + delta;
        }
        a++;
       }/*end of echo length while loop*/
  fclose(fptr);
 }/*end of for loop*/

 for(c = 0; c < short_echo - b; c++)
{
       if( kbhit()) exit(0);
     square = 0.0;
     sum_cor = 0;
       for(d = 1; d <= filenumber;d++)
    {
           sum_cor += data_array[c][d];
             square +=((data_array[c][d]) * (data_array[c][d]));
        }
            square_2[c] = square / (float)filenumber;
           avcor[c] = sum_cor / (float)filenumber;  /*calculate the average*/
         average2[c] = (avcor[c] * avcor [c]);
         sum = sum + avcor[c];

           fprintf(fpts,"%f ",avcor[c]);

          standard_dev = 0.0;
          if(square_2[c] >= average2[c])       /*calculate the standard deviation*/
        {
               standard_dev = ((float) sqrt(square_2[c] - average2[c]));
        }
            else standard_dev = ((float) sqrt(average2[c] - square_2[c]));

            fprintf(fptd,"%f ",standard_dev);
     }

     if(b > 1)                             /*zero pad files for plotting program*/
     {
        for(j = 0;j < b; j++)
        {
              fprintf(fpts,"%f ",zro);
              fprintf(fptd,"%f ",zro);
        }
     }

     if(b == 1)
     {
          fprintf(fpts,"%f ",zro);
          fprintf(fptd,"%f ",zro);
     }

        fprintf(fpts,"\n");
        fprintf(fptd,"\n");

     b++;
    }/*end of file length while loop*/

   fclose(fptd);
   fclose(fpts);
}/*end of av_corr*/
```