NASA Contractor Report 172311

THE VERRUN AND VERNAL SOFTWARE SYSTEMS FOR STEADY-STATE VISUAL EVOKED RESPONSE EXPERIMENTATION

William H. Levison (Bolt Beranek and Newman)
Greg L. Zacharias (Charles River Analytics)

BOLT BERANEK AND NEWMAN INC.
Cambridge, Massachusetts 02238

Contract NAS1-16982
March 1984

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This report summarizes the work performed for NASA Langley Research Center under Contract No. NAS1-16982 by Bolt Beranek and Newman Inc. (BBN). Dr. Greg L. Zacharias was the initial Principal Investigator for BBN and was responsible for development and implementation of the VERRUN and VERNAL software systems at BBN and at LRC. Upon Dr. Zacharias' departure from BBN, Dr. William H. Levison became Principal Investigator and assumed responsibility for completion of the program documentation and final technical report. Ms. Regis Donovan and Mr. Adrian Ho served as programmers for BBN. Dr. Alan Pope served as Technical Monitor for NASA.

SUMMARY

Two digital computer programs have been developed for use in experiments involving steady-state visual evoked response (VER): VERRUN, whose primary functions are to generate a sum-of-sines (SOS) stimulus and to digitize and store electro-cortical responses; and VERNAL, which provides both time- and frequency-domain metrics of the evoked response. These programs have been coded in FORTRAN for operation on the Digital Equipment Corporation PDP-11/34, using the RSX-11 Operating System, and the PDP-11/23, using the RT-11 Operating System. Users' and programmers' guides to these programs are provided, and guidelines for model analysis of VER data are suggested.

1. INTRODUCTION

Considerable effort has been devoted in recent years to the development of reliable metrics for pilot workload. Assessment of workload (more generally, operator cognitive state), would allow the identification of workload "bottlenecks", provide useful data for the evaluation of the crew/system interface and, in general, provide information necessary for maintaining task workload within desired limits throughout a given mission. Reliable measures of workload could also be useful in assessing the state of operator training in situations where objective measures of man/machine system performance alone are inadequate.

Numerous efforts have been undertaken to develop reliable metrics of pilot workload, including subjective estimates, primary and secondary task measures, and physiologic measures. Exploration of physiologic measures has been motivated by the desire to obtain one or more measures that are non-interfering with the primary task mission, and are not likely to be biased by the operator's preference for a given man/machine interface or by his unwillingness to admit that a particular task is difficult.

Cortical evoked response -- electrical potentials recorded from the scalp obtained in response to a visual or auditory stimulus -- is being explored as a workload metric. The bulk of such efforts has dealt with the transient response to a pulse-like stimulus. Typically, responses to multiple stimuli

are averaged on a point-by-point basis so that the specific response to the test stimulus can be extracted from the background electro-cortical activity.

Research has also been conducted with steady-state visual stimuli. In this arrangement, the amplitude of a stimulus light source is driven by an electrical signal consisting of one or more sinusoids; and the recorded scalp potentials are subsequently analyzed to quantify sinusoidal response components at the specific frequencies contained in the stimulus. Use of steady-state inputs of this sort allows the application of systems analysis techniques that have received widespread success in the characterization of non-biological electrical and mechanical dynamically-responding systems.

This report contains descriptions of two digital computer programs intended for use in experiments involving steady-state visual evoked response (VER): VERRUN, whose primary functions are to generate a sum-of-sines (SOS) stimulus and to digitize and store electro-cortical responses; and VERNAL, which provides both time- and frequency-domain metrics of the evoked response. These programs have been coded in FORTRAN for operation on the Digital Equipment Corporation PDP-11/34, using the RSX-11 operating system, and the PDP-11/23, using the RT-11 operating system.

The report is organized as follows. In the remainder of this introductory section we present some preliminary data that suggest the feasibility of a VER-based workload metric. Chapter

2 provides a theoretical background regarding sum-of-sines input generation and frequency-response analysis via fast-Fourier transform techniques. Guidelines for performing model analysis on the frequency-response data are also provided.

Chapter 3 provides a user's guide to the VERRUN runtime program. Major functions of this program are summarized, and instructions for generating and operating VERRUN are given. Chapter 4, similarly structured, provides a user's guide to the VERNAL analysis program.

A set of five appendices contains information of interest to the programmer. Appendices A and B describe the VERRUN and VERNAL main programs, respectively, along with the major FORTRAN subprograms used by these programs. Major FORTRAN subprograms used by both main programs are described in Appendix C.

Appendix D contains descriptions of FORTRAN input/output library routines, and assembly-language routines are described in Appendix E.

Figure 1 presents some (very) preliminary data[1] that suggest the feasibility of a VER-based workload metric. Frequency-response metrics from a single subject are shown for three experimental conditions: (a) SOS visual stimulus only; (b)

---

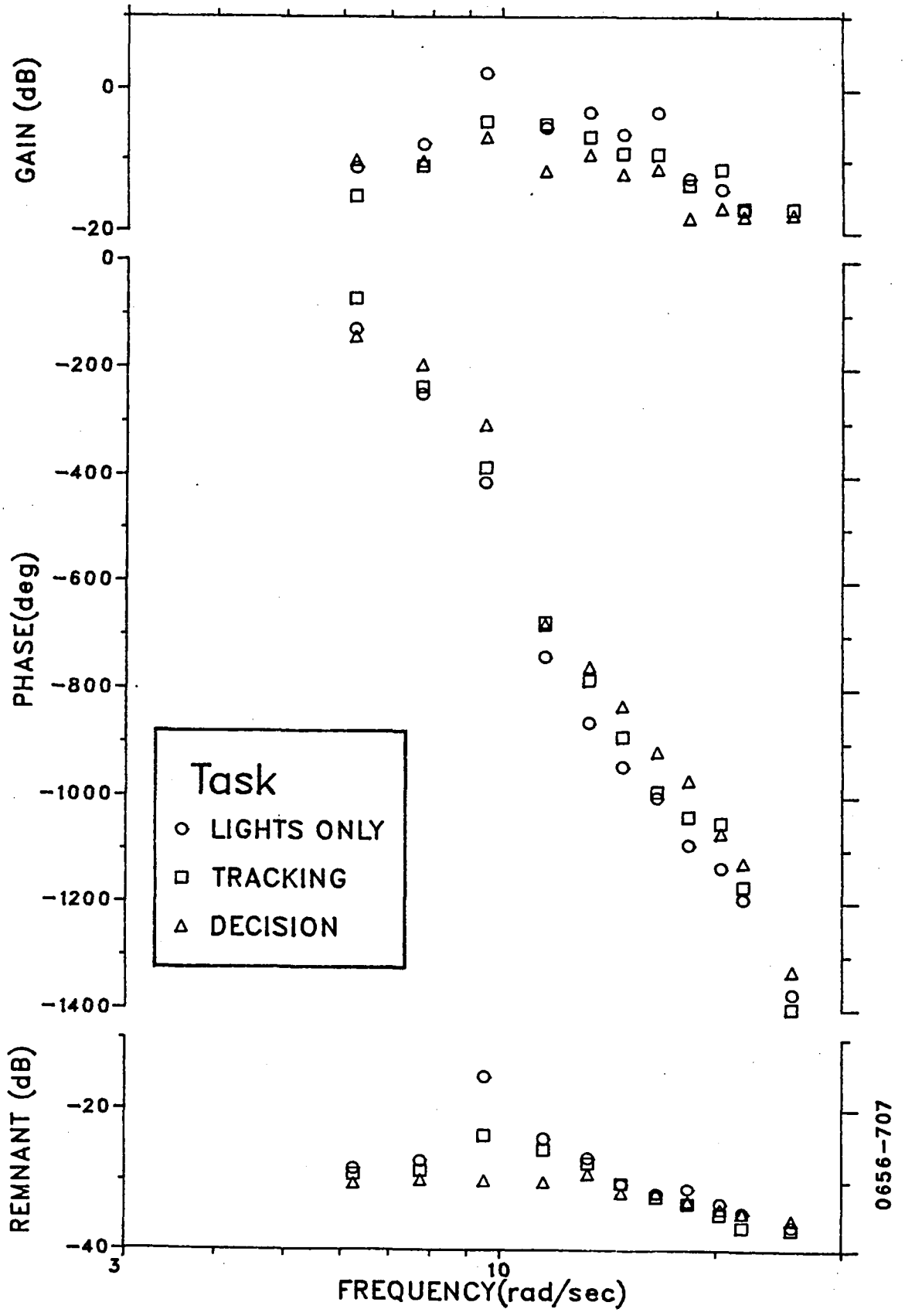
[1] Provided by Mr. Andrew M. Junker of the Air Force Aerospace Medical Research Laboratory.

SOS visual stimulus plus manual tracking task, and (c) SOS visual stimulus plus a laboratory-type decision-making task. (The particular metrics shown -- gain, phase, and remnant -- are described in Chapter 2 for the benefit of readers unfamiliar with control systems analysis).

Task-related effects are greatest at the stimulus frequency of 9.5 Hz, which is within the normal range of the EEG alpha component. A consistent progression from lights-only, to tracking, to decision making is observed at this frequency: (a) a decrease in the describing-function "gain" (amplitude ratio), (b) a decrease in the phase lag, and (c) a decrease in the remnant. These results are consistent with the expectation that, as the subject is required to attend to a motor or cognitive task (and thus attend less to the visual stimulus), the overall strength of the VER should be reduced. These results are thus consistent with results that have been obtained with the transient VER, in which certain response components (especially the P300 component) diminish in amplitude as external task loading is imposed.

This trend also suggests that the tracking task provides a lower workload than the decision task, or, more precisely, that the combined tasks of attending to the VER stimulus and making decisions draw more heavily upon common "resource pools" than do the combined tasks of attending to the lights and manual tracking.

Because of the small data base reflected in Figure 1 (1


GAIN (dB)
0
-20
PHASE(deg)
0
-200
-400
-600
-800
-1000
-1200
-1400
Task
LIGHTS ONLY
TRACKING
DECISION
REMNANT (dB)
-20
-40
3
10
FREQUENCY(rad/sec)
0656-707

subject, 2 trials/condition), we cannot assess the statistical significance of the apparent task-related changes in response, nor can we perform a reliable model analysis of these data. Nevertheless, these results are sufficiently encouraging to warrant further development and testing of the VER-based workload metric.

2. METHODOLOGY

The VERRUN and VERNAL computer programs are tools to facilitate application, to the study of visual evoked response, of an experimental methodology that has been successfully applied over the years to the study of manual control behavior. The use of sum-of-sines (SOS) inputs has been driven by efforts to construct linear models of the controller's response behavior. To construct these models, it is necessary to distinguish between (a) the portion of the controller's response linearly correlated with the external input, and (b) the "noisy" portion of the response not linearly correlated with the input. In the jargon of manual control, the noisy response component is known as "remnant", as it contains the portion of the response power that remains when we remove the response component that can be accounted for by a linear response strategy having time-invariant parameters. In this report we shall apply the term "remnant" to the portion of the VER not linearly related to the stimulus. The SOS input capitalizes on the property of a linear system that a continuous sinewave input will yield, in the steady-state, a sinewave output having the same frequency. Because of the property of superposition, a sum of sinewaves input will yield a steady-state sum of sinewaves output having identical frequency composition. The SOS input, then, enhances post-experiment analysis in the following ways:

a. Response power at non-input frequencies is by definition remnant, as input-correlated power can occur

only at input frequencies. Thus, it is relatively easy to distinguish remnant from input-correlated response components.

b. By concentrating input power at a few selected frequencies, signal/noise ratios (i.e., ratio of input-correlated to remnant-related power) can be enhanced, thereby increasing the reliability of performance metrics based on input-correlated response components.

Three topics are discussed individually in the remainder of this chapter: (a) generation of the SOS input, (b) quasi-linear analysis of systems driven by the SOS input, and (c) guidelines for linear model analysis. The VERRUN and VERNAL programs reflect implementations of the techniques discussed under items (a) and (b), respectively.

2.1 Generation of Sum-of-Sines Inputs

The VERRUN program is intended to allow modulation of a visual stimulus intensity by a sum-of-sines electrical signal of the form:

$$I(t) = \sum_{j=1}^{N_c} a_j \sin(\omega_j t + \phi_j) \qquad (1)$$

which is a summation over N_c sinewaves, where the jth wave has associated with it an amplitude a_j, a relative phase ϕ_j, and a frequency ω_j, where

$$\omega_j = h_j \omega_o \qquad (j=1,\ldots,N_c) \qquad (2)$$

where, in turn, h_j is the associated integer harmonic multiplier, and ω_o is the "base frequency". By choosing a desired period T_o

for the SOS signal, so that I(t) repeats itself every T_o seconds (i.e., $I(t)=I(t+T_o)$), then the base frequency will be specified by:

$$\omega_o = 2\pi/T_o \quad \text{rad/sec} \tag{3}$$

The harmonics, amplitudes, and phases are generally free parameters which can be chosen to "shape" the SOS signal as required. By choosing the harmonics (h_j) as positive integers, we can ensure, for each sinewave component of the signal, that an integral number of cycles appear in one period of the stimulus I(t). The amplitudes (a_j) can then be chosen to distribute the stimulus power over frequency in a manner deemed appropriate for the measurement situation. Finally, the phases (ϕ_j) can be changed to vary the temporal pattern of the signal I(t), while leaving unchanged its power spectral characteristics.

The SOS stimulus generation is done digitally, with one time sample of the signal generated every T_S seconds (the sample period). Thus, the kth sample is given by:

$$I_k \equiv I(t=kT_s) \qquad (k=0,1,2,\ldots) \tag{4}$$

where k ranges from zero up to some upper limit determined by the overall run time. The I_k values can be computed in an efficient manner if we choose to quantize the allowable choices of the SOS phases ϕ_j, according to

$$\phi_j = p_j\phi_o \qquad (j=1,\ldots,N_c) \tag{5}$$

where p_j is an integer "phase multiplier" (analogous to the harmonic multiplier for the frequency) and ϕ_o is a "base phase" given by

$$\phi_o = \omega_o T_s \tag{6}$$

where ω_o is the base frequency and T_S is the sample period. Direct substitution of (2) through (6) into the continuous-time version of the SOS equation (1) then yields the following sampled-time version:

$$I_k = \sum_{j=1}^{N_C} a_j \sin [\phi_o(kh_j+p_j)] \tag{7}$$

which conveniently defines the SOS signal at the kth sample instant.

Although this relation can be used to directly compute the SOS signal at each sample time, computational efficiency can be gained by use of an intermediate sinusoidal "look-up" table. This can be created by first recognizing that the SOS period T_o and the sample period T_S must be related by N_o, the number of samples in one period of the signal, according to:

$$T_o = N_o T_s \tag{8}$$

This then allows us to reexpress the base phase as follows:

$$\phi_o = \left(\frac{2\pi}{T_o}\right) T_s = 2\pi/N_o \tag{9}$$

which, in turn, allows us to define the following tabular sinusoidal function S_n:

$$S_n \equiv \sin(n\phi_o) = \sin\left[2\pi\left(\frac{n}{N_o}\right)\right] \qquad (10)$$

where n is the table index, which ranges from 1 to N_o, the total length (period) of the table. The sampled-time version of (7) may then be expressed as:

$$I_k = \sum_{j=1}^{N_c} a_j S_{n_{j,k}} \qquad (k = 0,1,2,\ldots) \qquad (11)$$

where the table index $n_{j,k}$ is given by

$$n_{j,k} = kh_j + p_j \qquad (j=1,\ldots,N_c) \qquad (12)$$

With a new computation each kth sample, this reduces to the following "incremental" form:

$$n_{j,k} = n_{j,k-1} + h_j \qquad (j=1,\ldots,N_c) \qquad (13)$$

with $n_{j,o}$ equalling p_j. Additional (storage) efficiencies are obtained by use of a "quarter-wave" lookup table for S_n, and a simple logic for determining index quadrant.

As noted above, three quantities must be defined for each of the N_c sinewave components in order to generate a sample waveform: the harmonic index, the amplitude, and the initial relative phasing. The harmonic indices are usually selected to span the frequency range of interest -- often, the range over which the system is expected to exhibit significant response. Component phase indices are typically selected randomly so that the stimulus has the appearance of a random process. Note that a selection of, say, zero for all component phase indices would

provide a highly-structured input that may well induce a response different from that to a random-appearing input.

There are a number of bases for selecting component amplitudes. One may select SOS amplitudes to approximate the power distribution of some underlying theoretical power spectral density function. This approach is often adopted in manual control studies to facilitate certain types of post-experiment model analysis or to reflect a linear representative of some real-world disturbance (e.g., a wind gust). Alternatively, one may simply assign the same amplitude to all components; this approach is commonly adopted when one does not have a basis for "shaping" the input spectrum. Still another approach is to "pre-whiten" the input: that is, attempt to compensate for the filtering effects of the system to yield an SOS response having a uniform set of amplitudes.

At present, VER applications seem to be employing components of like amplitudes. Nevertheless, here we describe a procedure for constructing an SOS input to approximate a known spectral density function, since the VERNAL analysis program allows for approximate reconstruction of a theoretical power spectral density function.

Figure 2 shows a sketch of a continuous power spectral density function, approximated by a sum of four sinusoids.

Frequency is divided into "windows" defined by the geometric[2] midpoints of adjacent SOS frequencies. The power contained in a given SOS component is the integral of the theoretical power spectral density function within the corresponding window, as indicated in the figure. Thus:

$$\left.\begin{aligned} \omega_o^- &= [\omega_j \cdot \omega_{j-1}]^{1/2} \\ \omega_j^+ &= [\omega_j \cdot \omega_{j+1}]^{1/2} \end{aligned}\right\} \tag{14}$$

$$a_j = \left[2\int_{\omega_j^-}^{\omega_j^+} \Phi(\omega)\, d\omega\right]^{1/2} \tag{15}$$

where $\Phi(\omega)$ is the continuous power spectral density function.

The first and last SOS components are special cases. For the first component, the minimum frequency is set to Ø; for the last component, the maximum frequency is computed as

$$\omega^+ = \omega_j^2/\omega_{j-1} \tag{16}$$

2.2 Quasi-linear Analysis

The primary function of program VERNAL is to allow a quasi-linear analysis of the VER. Certain frequency-response

---

[2] The use of geometric, rather than arithmetic, means stems from the tradition in manual control experimentation to locate input frequencies at approximately equal logarithmic intervals.

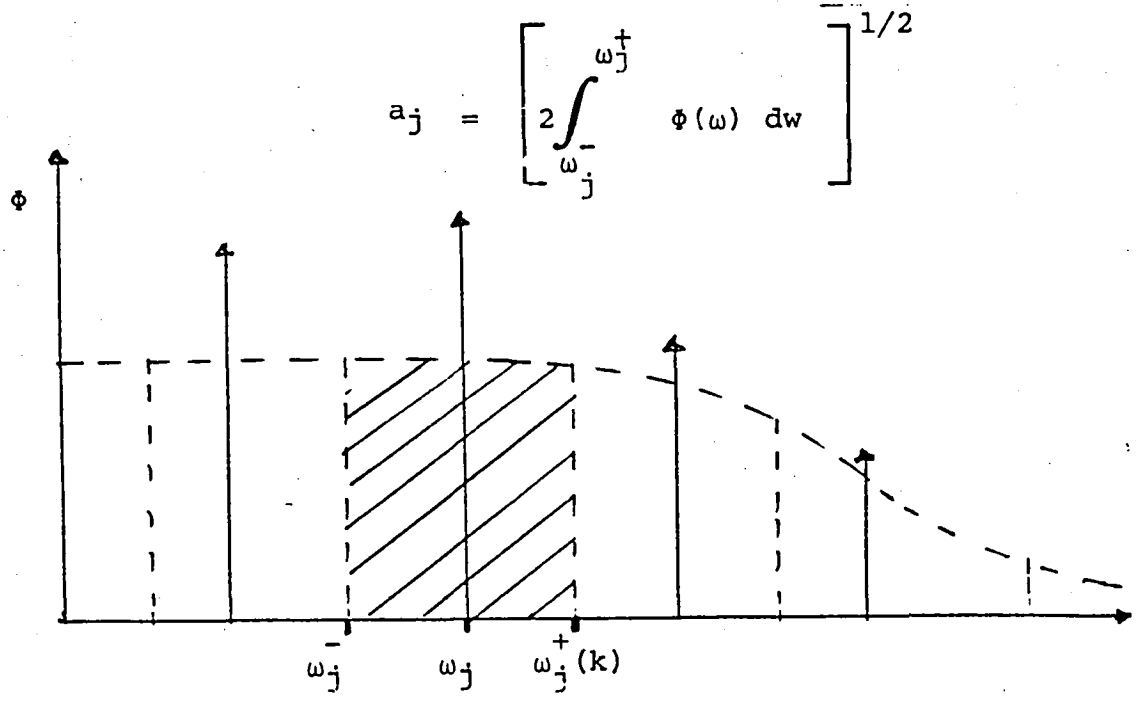


FIG. 2. COMPUTATION OF SUM-OF-SINES COMPONENT ALTITUDE

metrics are computed to facilitate linear modeling of the relationship between the "system input" (the visual stimulus) and the "system output" (evoked electro-cortical response). Other frequency-response metrics are computed to allow characterization of the portion of the evoked response that cannot be characterized by a time-invariant linear transformation of the stimulus.

Standard time-domain statistics of mean, standard deviation, and rms are also computed. If we let $x(k)$ $(k=1,\ldots,N_o)$ be the sampled time history of some variable of interest, these statistics are computed as follows:

$$\text{mean} = \frac{1}{N_o} \sum_{k=1}^{N_o} x(k)$$

$$\text{rms} = \left[\frac{1}{N_o} \sum_{k=1}^{N_o} x^2(k) \right]^{1/2} \qquad (17)$$

$$\text{standard deviation} = [(\text{rms})^2 - (\text{mean})^2]^{1/2}$$

Two frequency-response metrics are of primary interest: the "describing function", which relates the evoked response to the visual stimulus; and the spectrum of the evoked response. For this discussion we define the describing function empirically as the Fourier transform of the response divided by the Fourier transform of the stimulus, measured at input frequencies only. Because the Fourier transforms are complex quantities, each describing function estimate may be characterized by its magnitude (which we call the "gain" or "amplitude ratio") and by its relative phase shift. Describing-function measures are often used in a model-matching procedure to derive analytic representations of system input/output characteristics.

The power spectrum is partitioned into input-correlated and remnant components. The remnant is of particular interest, as it serves two functions:

1. It provides a test of the reliability of the describing function estimates.
2. In the case of the VER experiment, it provides an indication of the background electro-cortical activity not linearly related to the visual stimulus.[3]

Frequency-response analysis requires Fourier transforms of the desired response signal (or "channel") and the visual stimulus. VERNAL, along with many other programs that perform this type of analysis, uses a "fast-Fourier transform" (FFT) to perform this operation efficiently. The particular algorithm used in VERNAL requires that the time-history sample length contain $N_o = 2^n$ points, where n is a postive integer. The experimental run length is typically longer (N_r points = T_r seconds) to allow the system to reach steady-state. The interval used for analysis, then, is of length N_o and usually begins a number of sample points beyond the start of the run.

In order that the FFT of a sum-of-sines time history contain significant response at SOS frequencies only (i.e., no "side

---

[3] Even though remnant is, by definition, not linearly correlated with the external stimulus, there may be a functional relationship between the two. One of the questions for VER research is, in fact, to determine whether or not such a functional relationship exists.

bands"), the sample period N_o used in constructing the SOS input signal must be the same as the number of samples processed by the FFT routine. (The VERNAL and VERRUN programs are configured so that N_o is the same for both.)

Assume that the FFT routine processes the sampled time history x(k) (k=i,...,N_o+i), where "i" is the "start point" for analysis, and returns the Fourier transform X(k), (k=1,...,N_o/2). (The FFT returns independent transforms for N/2 frequencies only.) Since the transform is a complex quantity, the transform X(k) may be considered to be two vectors XR(k) and XI(k) containing the real and imaginary parts, respectively. Each frequency index "k" represents a frequency "bin" of $2\pi/T_o$. Thus, the bin width of each FFT result is identical to the base frequency ω_o used in constructing the stimulus SOS.

Once the FFT's have been computed for the signals of interest, we can then proceed to estimate spectral and describing function quantities as discussed below.

2.2.1 Computation of Signal Spectra

The signal spectrum is defined at each FFT index as

$$P(k) = [XR(k)^2 + XI(k)^2]/2 \qquad (18)$$

Because the signal being analyzed is an SOS input, or the response to an SOS input, partitioning the spectrum into input-correlated and remnant components is relatively

straightforward. By definition, all power estimates at indices not corresponding to SOS frequencies constitute the "remnant power" and, to a first approximation, all power estimates at input frequencies constitute the "correlated power". Thus:

$$\text{remnant power} = P(k),\ k \neq h_j$$

$$\text{correlated power} = C(k),\ k = h_j$$

where h_j is the jth SOS frequency, and N_c is the number of sinusoidal components in the SOS input, as defined earlier.

The fractional remnant power -- the fraction of total signal power contained at non-input frequencies -- is often of interest. This computation is performed as follows:

$$\text{TOTPOW} = \sum_{k=1}^{N/2} P(k)$$

$$\text{CORPOW} = \sum_{j=1}^{N_c} C_j \qquad (19)$$

$$\text{FRREM} = (\text{TOTPOW}-\text{CORPOW})/\text{TOTPOW}$$

where TOTPOW is the total signal power contained in the N/2 independent FFT frequencies, CORPOW is the total correlated power summed over all SOS frequencies, and FRREM is the fractional remnant power.

The estimates of correlated power must be considered

approximations because of possible "contamination" by remnant. (Correlated power can exist only at input frequencies, but remnant power is assumed to be distributed smoothly with frequency.) To determine the reliability of a given correlated-power estimate, we must estimate the level of remnant power contained at that SOS frequency. Since we cannot distinguish remnant from input-correlated power in a single FFT measurement, we adopt the following strategy: (1) assume remnant to vary smoothly with frequency, (2) average the remnant estimates in the vicinity of the SOS frequency, and (3) use this average as the estimate of remnant power contained at the SOS frequency.

To elaborate, let us define the (estimated) input-correlated power for the jth SOS frequency as

$$C_j = P(h_j) \qquad (20)$$

Consider the diagram of Figure 3, which shows a hypothetical signal spectrum in the vicinity of the jth SOS frequency. In the VERNAL program (and in other similar programs created by BBN), averaging is performed over a window 1/4 octave wide centered about the input frequency.

Let the upper and lower boundaries of the averaging window be designated as k_j^- and k_j^+, respectively. For a window extending 1/8 octave above and below the SOS frequency, these quantities are computed as

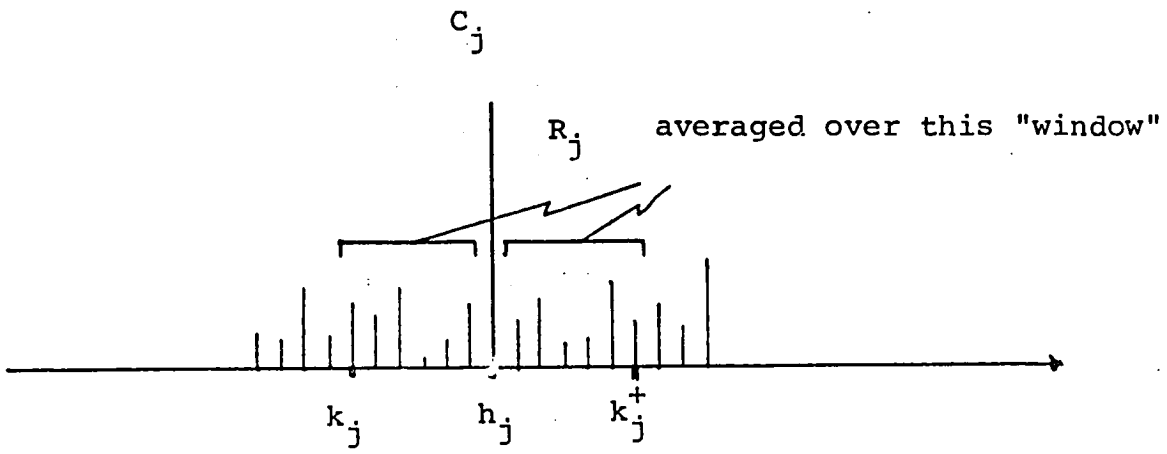


FIG. 3. COMPUTATION OF REMNANT SPECTRUM

$$k_j^- = h_j/2^{(1/8)}$$
$$k_j^+ = h_j \cdot 2^{(1/8)} \qquad (21)$$

where the computed indices are rounded to the nearest integer. The total number of frequency bins spanned is $k_j^+ - k_j^- + 1$, and the number of "remnant frequencies" (total number of bins minus one for the jth SOS frequency) is $k_j^+ - k_j^-$. Thus, the estimate of remnant power associated with the jth SOS frequency is

$$R_j = \frac{1}{k_j^+ - k_j^-} \left[\sum_{k=k_j^-}^{k_j^+} P(k) - C_j \right] \qquad (22)$$

The measures C_j and R_j are pure spectral (rather than spectral density) measures and have units of signal power. We may refer to these measures as "power per bin".

These measures are usually expressed in terms of dB:

$$Cdb_j = 10*\log(C_j)$$
$$Rbd_j = 10*\log(R_j) \qquad (23)$$

where the logarithm is to base ten. To determine the reliability of measures based on correlated power (C(k) plus describing function estimates), we compute the following signal/noise ratio (in dB):

$$P_j = Cdb_j - Rdb_j \qquad (24)$$

A criterion value ρ=6 dB is typically assumed for determining measurement reliability. That is, estimates of correlated power or describing functions at frequencies for which ρ is less than 6 dB are considered "unreliable" and are not used in subsequent analysis (e.g., computation of within- and across-subject statistics).

It is useful to convert spectral measures to units of power per rad/sec (or power/Hz) when the SOS has been constructed to approximate the power distribution of some theoretical continuous power spectral density function, or when one wishes to normalize the data to allow comparison with results obtained using different experimental run lengths (and hence, different frequency bin widths). Remnant is converted by simply dividing the remnant estimate (power/bin) by the bin frequency ω_o. Thus,

$$R'_j = R_j/\omega_o$$
$$Rdb'_j = Rdb_j - 10*\log(\omega_o) \qquad (25)$$

Conversion of correlated power, on the other hand, is not as simple. Recall the discussion in section 2.1 concerning calculation of SOS amplitudes so as to approximate a continuous power spectral density function. Frequency windows were defined by the geometric midpoints of adjacent SOS frequencies ω^- and ω^+ in equations 14-16, and each amplitude was chosen to contain the power within its corresponding window.

To convert input-correlated power to units of power per rad/sec, we approximate the inverse process by a "box-car" representation. That is, we transform C_j into a uniform power spectral density over the frequency region $\omega_j^+ - \omega_j^-$. If we let $W_j = \omega_j^r - \omega_j^-$, the power per rad/sec is

$$C'_j = C_j / W_j$$

$$Cdb'_j = Cdb_j - 10*\log(W_j)$$

2.2.2 Computation of Describing Functions

Analysis of steady-state VER is expected to involve computation of one or more describing functions relating selected pairs of signals. For two transformed signals X and Y, the describing function estimate at the jth SOS frequency index is computed as

$$G_j = \frac{Y(h_j)}{X(h_j)} \qquad (26)$$

where h_j is the FFT index corresponding to the jth SOS frequency

(consistent with the definition of h_j in Section 2.1), $X(h_j)$ and $Y(h_j)$ are the corresponding Fourier coefficients of the "input" (denominator) and "output" (numerator) signals defined for this computation, and G_j is the describing function at that frequency, expressed as a complex number.

For analysis of VER data, the signal X will typically be the SOS visual stimulus I, and Y will be a particular response signal of interest. One may, however, compute the describing function between two VER response channels as well. To keep the discussion general, we shall make no assumptions here as to the specific variables used in the describing function computation.

The complex quantity G_j is usually transformed into a pair of real quantities for presentation. The "gain" (more properly, the "amplitude ratio") is the magnitude of G_j, expressed in dB. Thus,

$$\begin{aligned} a_j &= 20*\log(|G_j|) \\ &= 20*\log(|Y(h_j)|) - 20*\log(|X(h_j)|) \\ &= a_{Y_j} - a_{X_j} \end{aligned} \qquad (27)$$

where a_{Y_j} and a_{X_j} are the magnitudes of Y and X, expressed in dB.

The phase shift is computed as the difference between the relative phase angles of X and I, expressed in degrees. Thus,

$$\phi_j = \angle Y(h_j) - \angle X(h_j) \qquad (28)$$

where the "angle" of X, for example, is computed as

$$\angle X(h_j) = 360*\tan^{-1}(XI(h_j)/XR(h_j))$$

Because phase is a circular function, repeating every 360 degrees, the inverse tangent operation yields a phase estimate within a 360-degree range (typically, 0 to 360, or -180 to +180.) Now, dynamically responding systems that contain a large number of integrating elements and/or significant delays may exhibit a phase-shift change of more than 360 degrees over the frequency range of interest. Therefore, a method of "unwrapping" the phase shift may be required to obtain a true picture of the frequency-dependency of the phase response.

The VERNAL program unwraps the phase by requiring the phase-shift estate at a given SOS index to vary no more than plus or minus 180 degrees from the phase estimate at the preceding index, where a reference phase of 0 degrees is adopted for the SOS index. Thus, the following mathematical constraint is satisfied:

$$\phi_{j-1} - 180 \leq \phi_j \leq \phi_{j-1} + 180 \qquad (29)$$

This algorithm has worked well for analysis of manual control data, where the phase-producing aspects of the man/machine system and the spacing of SOS frequencies usually guarantee a phase change magnitude of less than 180 degrees from one SOS index to the next. Whether or not this algorithm works as well for VER data remains to be seen.

Since the objective of computing the describing function is to provide a characterization of the linear relationship between two signals, the describing function estimates are valid only to the extent that the Fourier coefficients $X(h_j)$ and $Y(h_j)$ reflect response activity linearly correlated with the external SOS stimulus. Therefore, the spectra of X and Y are checked to verify that the signal/noise ratios ρ are greater than some criterion value (say, 6 dB) for <u>both</u> signals. If either spectrum fails this test at a given SOS frequency, the gain and phase shift estimates at that specific frequency are considered invalid and are omitted from further consideration.

2.3 Guidelines for Model Analysis

Studies of manual control research often involve a post-analysis modeling effort in which the time- and frequency-domain measures described above are used to derive parameters of an analytic model. This analysis typically serves two purposes: (a) data compression, in which the measures derived during the primary data reduction are further reduced to a small number of model parameters, and (b) development and validation of theoretical models for operator response behavior. We anticipate the application of analytic model analysis to VER results as well, primarily to achieve an efficient characterization of stimulus/response relationships (or, more precisely, to achieve a parsimonious characterization of the effects of experimental variables on stimulus/response relationships).

Once the frequency-response metrics have been derived from the VER data, three ingredients are needed to allow model analysis:

1. An analytic model that has a well-defined (and, ideally, small) set of independent model parameters and the capability of yielding predicted performance metrics of the type extracted from the data.
2. A scalar metric ("matching error") that defines how well the data are matched by the model predictions.
3. One or more algorithms to identify the set of parameter values that provides the least discrepancy between "predicted" and experimental measurements.

It is important to note that the model parameters identified from a given data set are functions not only of the model structure, but of the definition of the matching error, the search procedure employed in the identification, and possibly the way in which the search procedure is initialized. Unless the model is capable of an exact match to the data -- not likely unless the model itself has been used to generate "data" for test purposes -- the results of the model analysis will be specific to the details of the analysis procedure. Therefore, a consistent model-matching procedure should be used when exploring the effects of experimental variables on the VER, or when exploring inter-subject differences.

2.3.1 Parameter Identification

In this discussion we review a particular scheme for identifying model parameters. This scheme has been extensively

applied to the identification of pilot model parameters from manual tracking data, with apparent success; it is, nevertheless, quite general and can handle a number of model structures.

We recommend that, at least initially, linear model structures be tested, and that parameter identification be based on the describing function (gain and phase) and remnant measures described above in Section 2.2. For reasons that will be clear shortly, we further recommend that model analysis be performed on ensemble statistics of these metrics, rather than measures obtained from a single experimental trial.

Assume for this discussion that some model, having a parameter set $\underline{p}$, is capable of generating predictions for these metrics and is to be tested against VER data. (A specific candidate model structure for VER analysis is considered in Section 2.3.2).

We suggest the following scalar matching criterion, which is similar to that used for manual control analysis:

$$E^2 = \frac{1}{3}\left\{\frac{1}{N_1}\sum_{j=1}^{N_1}\left[\frac{a_j-\hat{a}_j(\underline{p})}{\sigma_{a_j}}\right] + \frac{1}{N_1}\sum_{j=1}^{N_1}\left[\frac{\phi_j-\hat{\phi}_j(\underline{p})}{\sigma_{\phi_j}}\right]^2 + \frac{1}{N_2}\sum_{j=1}^{N_2}\left[\frac{R_{db_j}-\hat{R}_{db_j}(\underline{p})}{\sigma_{R_{db_j}}}\right]^2\right] \quad (30)$$

where:

1. a_j, ϕ_j, and Rdb_j are the gain, phase, and remnant estimates for the jth SOS frequency as defined in Equations 27, 28, and 23, respectively. These quantities represent mean estimates determined by ensemble (point-by-point) averaging across experimental replications and/or across subjects.

2. $\hat{a}_j \underline{p}$, etc., is the model prediction for a particular choice of values for parameter set $\underline{p}$;

3. σ_{a_j}, etc., is the standard deviation of the experimental measurement determined from ensemble averaging;

4. N_1 is the number of frequency components for which reliable gain and phase estimates have been obtained, and N_2 is the number of frequencies yielding reliable remnant estimates. Except for the SOS visual stimulus (which is theoretically remnant-free), N_2 will equal the number of SOS frequencies N_c. N_1 will be equal to or less than N_c, depending on the signal/noise environment at the various SOS frequencies.

Inclusion of the experimental standard deviations in the scalar matching error allows each error component to be weighted inversely by the reliability of the data. In this way, "matching power" is concentrated on the data points that are (presumably) the most repeatable. On the other hand, to prevent the matching scheme from giving excessive weight to data points having unusually low variability, we suggest that the following minimum standard deviations be imposed for computing E^2: 0.5 dB for gain and remnant, 3 degrees for phase shift.

Weighting inversely by standard deviation also converts each error term into a dimensionless number, thereby allowing the accumulation of matching errors across different metrics. The quantity E (the square root of the criterion defined in Equation 30) reflects the average number of standard deviations of mismatch. That is, if every model prediction differed from its corresponding data point by "n" standard deviations, E would have a value of "n".

The matching error E^2 may be expressed as

$$E^2 = \underline{e}'W\underline{e} \tag{31}$$

where each element e_j of the column vector $\underline{e}$ is the difference between the jth experimental data point and the corresponding model prediction, and each element w_i of the diagonal matrix W is a weighting coefficient. For the criterion of Equation 30, $e_1 = a_1 - \hat{a}_1(\underline{p})$, $e_{(N_1+1)} = \phi_1 - \hat{\phi}_1(\underline{p})$, etc., and $\omega_1 = 1/3N_{a1}\sigma_1^2$, $\omega_{(N1+1)} = 1/3N_{\phi 1}\sigma_1^2$, etc.

In a given application, the matching error E^2 will depend on the particular choice of parameter values $\underline{p}$. The objective of the gradient search scheme is to find the $\underline{p}$ that minimizes E^2. To implement the search scheme, we initially assume that model predictions (and therefore prediction error) vary linearly with model parameters. Thus, a change in parameter values yields a change in modeling error characterized as $\Delta\underline{e} = Q'\Delta\underline{p}$, where

$$q(i,j) = \partial e_j / \partial p_i \qquad (32)$$

That is, the matrix Q contains entries quantifying the sensitivity of each prediction error to each model parameter. This matrix is determined empirically using the specific data and parameter sets at hand.

Solving for minimum J as a function of $\Delta \underline{p}$, we obtain

$$\Delta \underline{p} = -[\underline{QWQ}']^{-1} \underline{QWe} \qquad (33)$$

If modeling errors were truly related linearly to model parameters, the desired best-matching parameter set would be obtained by the following three-step procedure:

a. Select an initial parameter set $\underline{p}_o$;

b. Compute the sensitivity matrix Q and the parameter increment $\Delta \underline{p}$ as defined in Equations 32 and 33;

c. Compute the desired parameter set as $\underline{p} = \underline{p}_o + \Delta \underline{p}$

Now, since the relationship between model parameters and model predictions is seldom totally linear, two or more iterations of the above procedure are required until some convergence criterion is satisfied. Because the parameter change computed as per Equation 33 will sometimes yield a scalar matching error greater than the starting value, it is often useful to augment the minimization procedure with a line-search to optimize the magnitude of $\Delta \underline{p}$.

A full discussion of the techniques of implementing the

quasi-Newton gradient search scheme, and of the ramifications of adopting such a procedure, is beyond the scope of this report. Further implementational details may be found in Levison (1981a,b, 1982).

One point to mention here, however, is that the uniqueness of the identified parameters is not guaranteed for any numerical search scheme, including the quasi-Newton procedure. Specifically, a change in the initial guess $\underline{p}_o$ may result in different values for the identified parameters for the same data base. The severity of this potential problem in a given application depends on a number of factors, including:

a. the degree to which the model structure is capable of matching the data;

b. the existence of one or more parameters to which the scalar matching error is relatively insensitive;

c. the degree to which the relation between model parameters and predictions is nonlinear; and

d. the vector "distance" of the initial guess $\underline{p}_o$ from the value of $\underline{p}$ that would provide a global minimum.

To minimize the non-uniqueness problem, therefore, one wishes to test a model that has a structure capable of matching the experimental data with a set of nearly-orthogonal parameters, and to initialize the search scheme with parameter values that are close to optimal. This approach has been quite successful in identifying "pilot-related" parameters of the optimal control model from manual tracking data.

2.3.2 A Candidate Model Structure

As indicated earlier, we have not included model results in this report for two reason: (a) lack of a sufficient data base, and (b) ambiguities in "unwrapping" the phase-shift component of the VER. Nevertheless, we discuss a candidate model structure here for readers who might wish to conduct model analysis of VER once the above constraints have been overcome.

If we had a theoretical quasi-linear model for the VER (as we have, for example, for manual control behavior), we would offer this model for initial testing. Given the lack of such a model, we must examine the experimental data and, relying on our knowledge of control systems, postulate a model structure that is likely to mimic the VER.

We must also decide whether we wish to match describing function and remnant data simultaneously with a single model structure, or to match these quantities independently with either similar or different model structures. Again, given the lack of a firm theoretical basis for deciding whether or not the VER and the background electro-cortical activity are functionally related, we suggest the general approach (i.e., independent models) at this time. If strong correlations are subsequently found between the describing function and remnant models, one can re-analyze the data using a more highly constrained modeling philosophy.

For convenience, the preliminary results shown previously in Figure 1 are repeated here in Figure 4. The following overall trends in the frequency-response measures can be ascertained:

1. Gain appears to reach a maximum in the region of 9-15 Hz, then fall off with increasing frequency.
2. Phase lag (i.e., negative phase shift) increases monotonically, and relatively strongly, with increasing frequency.
3. Remnant peaks in the region of 9-12 Hz, then decreases with increasing frequency.

These trends suggest that one consider a resonant second-order filter with pure delay as a model for the describing function response, and a second-order filter for the remnant response. (Since remnant is a power spectrum and therefore contains no phase or timing information, a delay parameter is not identifiable from the remnant data.)

A second-order model to the describing function might take the form:

$$F(j\omega) = \frac{K_e^{-j\omega T}}{1+\frac{2\zeta\omega_n}{2\omega}+\left(\frac{\omega_n}{j\omega}\right)^2} \qquad (34)$$

where $j\omega$ is radian frequency, expressed as an imaginary number; $F(j\omega)$ is the filter transfer function that will be matched to the experimental describing function; K is the asymptotic low-frequency filter gain; T is a pure delay; ω_n is the natural frequency (approximately the resonant frequency) of the filter, and ζ is the filter damping ratio.

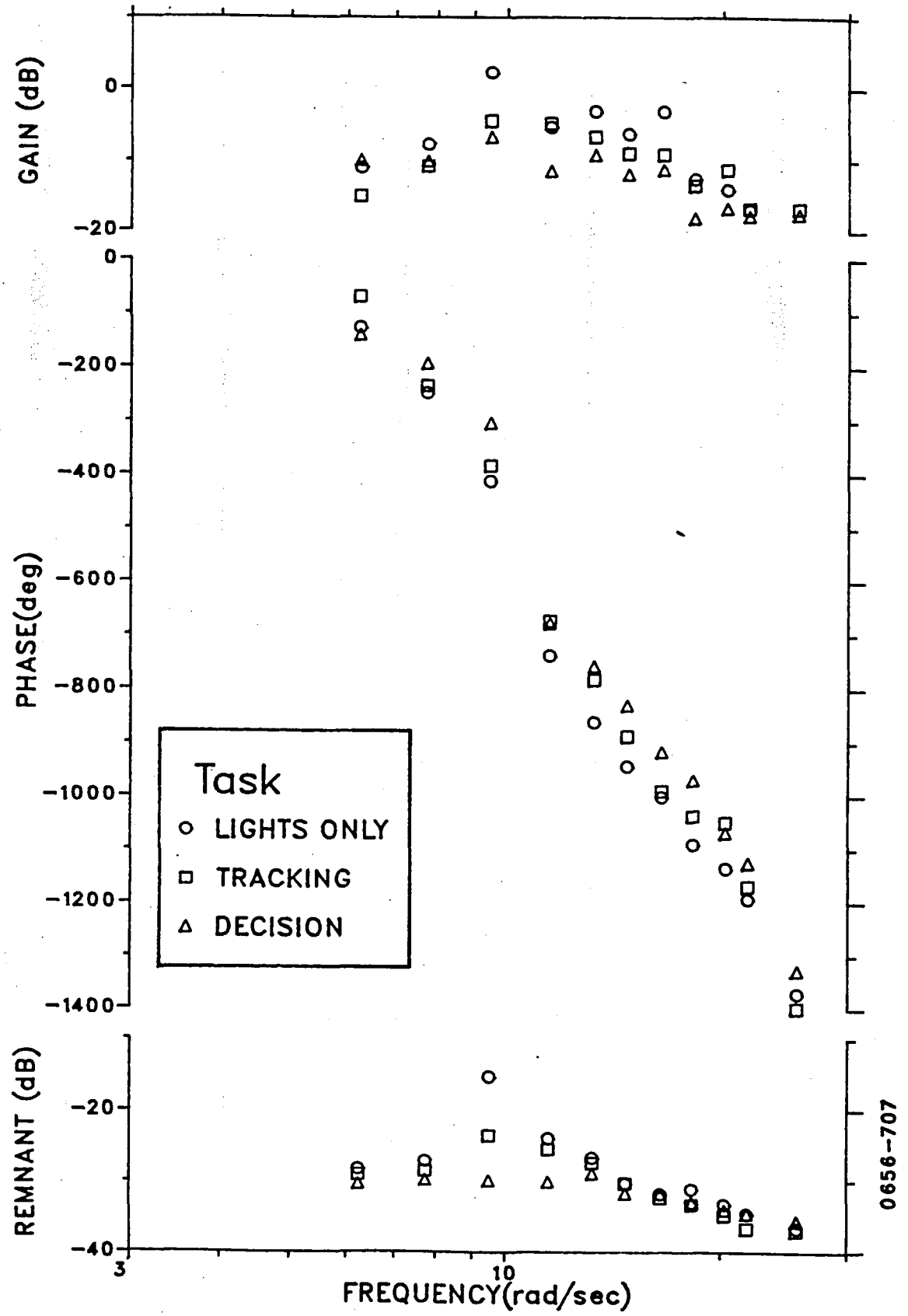


FIG. 4. EFFECTS OF THE TASK ENVIRONMENT ON THE STEADY-STATE VISUALLY EVOKED RESPONSE

$F(j\omega)$ is a theoretical transfer function and therefore is a continuous function of frequency. When used in a scheme for identifying model parameters, however, it will be evaluated only at frequencies corresponding to the experimental describing function measurements -- i.e., the SOS stimulus frequencies. At each such frequency, the complex quantity $F(j\omega)$ will be converted to gain (dB) and phase (deg) to facilitate computation of model/data differences. Since $F(\omega)$ represents a model for the VER describing function only, the scalar matching error will be based on the first two summations contained in Equation 30.

The objective of the gradient search procedure is to identify values for the four independent model parameters -- K, T, ω_n, and ζ -- that minimize the scalar matching error. If we assume a VER experiment employing 10 frequencies (yielding 10 gain and 10 phase estimates), a 5:1 data compression results if the data can be reasonably well matched by the model.

As noted above, success of the identification procedure is contingent on the selection of a suitable initializing set of parameter values. For a low-order model of the type suggested here, selecting a reasonable initial parameter set is relatively straightforward. Once the issue of unwrapping the experimental phase shift has been resolved, the following procedure should yield satisfactory results:

1. Determine K from the asymptotic low-frequency gain exhibited by the data. (Be sure to convert dB to absolute units.)

2. Estimate time delay T from the phase shift at the higher frequencies. Note that the phase shift due to delay is a linear function of frequency: phase in degrees is given by $57.3*\omega T$, or $57.3*2\pi fT$, where "f" is frequency in Hz. High-frequency phase will thus be equal to the asymptotic high-frequency phase shift due to the dynamics response of the filter (exclusive of delay), plus the effects of delay. For a second-order filter, maximum phase shift due to dynamic elements is -180 degrees. Thus, for a given SOS index "j", representing a frequency beyond the filter bypass,

$$\phi_j \approx -180 - 57.3*2\pi f_o T \tag{35}$$

Accordingly, we select the initial delay parameter

$$T = \frac{\phi_j + 180}{57.3*2\pi f_i} \tag{36}$$

3. Let the initial guess for the natural frequency ω_n be the frequency at which the experimental describing function gain is a maximum.

4. Determine the initial value for damping ratio ζ from the ratio of the maximum VER describing function gain to the asymptotic low-frequency gain. For systems with a distinct resonance, the damping ratio is approximately

$$\zeta = \begin{matrix} 1/2\ F' \\ \text{or} \\ 1/(2*10^{(F_{db}'/20)}) \end{matrix} \tag{37}$$

where F' is the ratio of maximum to low-frequency asymptotic gain computed from absolute values, and Fdb' is the same ratio in dB.

Guidelines 3 and 4 apply only when a resonance phenomenon is apparent in the data. Otherwise, set the initial ω_n to the

frequency at which the describing function gain has decreased by about 3 dB from its asymptotic low-frequency value, and set ζ between 0.7 and 1.

Selection of initial parameter values for a remnant model would proceed in the same fashion, except this model would have only three parameters (K, ω_n, and ζ), and step 2 would be omitted.

To demonstrate application of these guidelines, we use the data of Figure 4 (the tracking case) as an example. Taking the gain at the first SOS frequency as the asymptotic low-frequency gain, we set K=0.1 (equivalent to -20 dB). Selecting the second-highest frequency of about 22 Hz as the basis for the time delay computation, we use equation 36 to compute a delay of about 0.12 seconds from the phase shift (about -1200) measured at that frequency. The gain curve seems to peak at around 10-12 Hz, so we let ω_n = 12 Hz. Finally, we note a maximum gain increase of about 10 dB, which, from Equation 37, yields a damping ratio ζ of about 0.16.

Model "predictions" obtained with this initial parameter set are compared to the experimental describing function estimates in Figure 5. Model results are plotted as a continuous function of frequency; data are represented by discrete symbols at SOS frequencies. While not providing a particularly close match to the data, the model results do reflect important frequency trends and, in general, provide a reasonable "ballpark" approximation.

On the basis of our past experience in applying this approach to modeling of manual tracking response, we would expect this initial guess to allow the search procedure to reach a global minimum.

Success of model analysis will be contingent, of course, on the ability to properly unwrap the phase response. The phase curve shown in Figure 5 (not produced by the VERNAL program) conforms to the assumption that the VER phase shift should monotonically decrease with increasing frequency. On the other hand, the VERNAL program as currently configured would have placed the phase measurement at the fourth SOS frequency at a value slightly more positive than the phase at the third frequency, rather than nearly 360 degrees more negative as shown in the Figure. It is not clear at this stage which is the "right" way to unwrap the phase.

In addition, the sign of the VER is arbitrary in terms of theoretical modeling and depends experimentally on the polarity convention adopted in recording the electro-cortical potentials. Thus, one could adopt an analytic model with a negative gain and thereby translate all predicted phase values by plus or minus 180 degrees.

Note that no phase ambiguity exists for an analytic linear model of given structure and parameterization: each differentiation represented in the numerator of the transfer function asymptotically adds 90 degrees phase lead, each

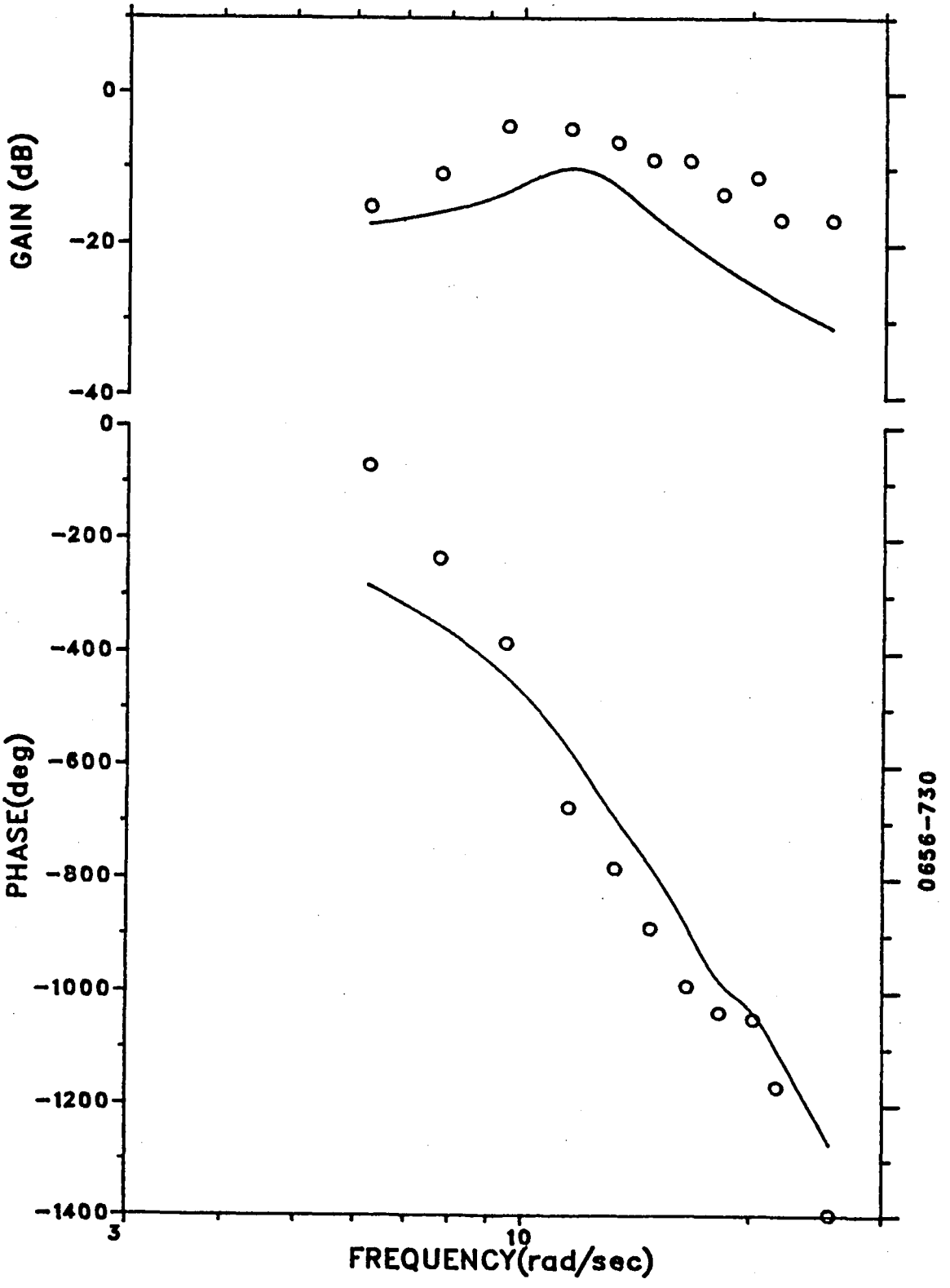


FIG. 5. COMPARISON OF INITIAL MODEL PREDICTIONS WITH EXPERIMENTAL DATA

differentiation represented in the denominator asymptotically adds 90 degrees phase lag, a negative sign adds ±180 degrees, and pure time delay contributes a phase lag that is linear with frequency.

Because linear model predictions are unambiguous, we suggest that a linear model -- rather than some arbitrary criterion of "reasonableness" -- be used to guide the analysis of phase-shift characteristics. Initially, this approach will require a closely-coupled iterative procedure, where the model is used to guide the analysis, and the experimental data are used to define model parameters. As the experimental data base expands, however, we suspect that one or more baseline model structures -- either theoretical or determined empirically -- will emerge to guide this type of analysis. In any case, development of reliable and efficient techniques to perform coupled data and model analysis are suggested as an area for further research.

3. USER'S GUIDE TO VERRUN

3.1 Major Functions

VERRUN is a software system designed to support electroencephalographic (EEG) visual evoked response (VER) experimentation using sum-of-sines (SOS) stimulation as described in Section 2.1. The system is designed to operate in a single-user real-time mini computer-based environment, with modular software to facilitate transportability across systems. Currently, the system is implemented on the Digital Equipment Corporation (DEC) PDP-11/34, using the RSX-11 operating system, and on the PDP-11/23, using the RT-11 operating system. The primary source language is FORTRAN, with some support code written in the MACRO assembly language.

VERRUN is intended for use in the closed-loop stimulus/response environment sketched in Figure 6. The stimulus generator is driven by the software, through a digital-to-analog (D/A) converter, via a commanded SOS signal I_c. The generator, in turn, provides an intensity-modulated visual stimulus I for "driving" the human subject's "steady-state" VER (ssVER). The resulting scalp voltages (E_1 through E_N) are transduced and amplified by the EEG recording hardware, and the measured voltages ($\tilde{E}_1$ through $\tilde{E}_N$) are sampled through a multi-channel analog-to-digital (A/D) converter. A stimulus intensity signal ($\tilde{I}$) is likewise transduced and sampled, through an additional A/D

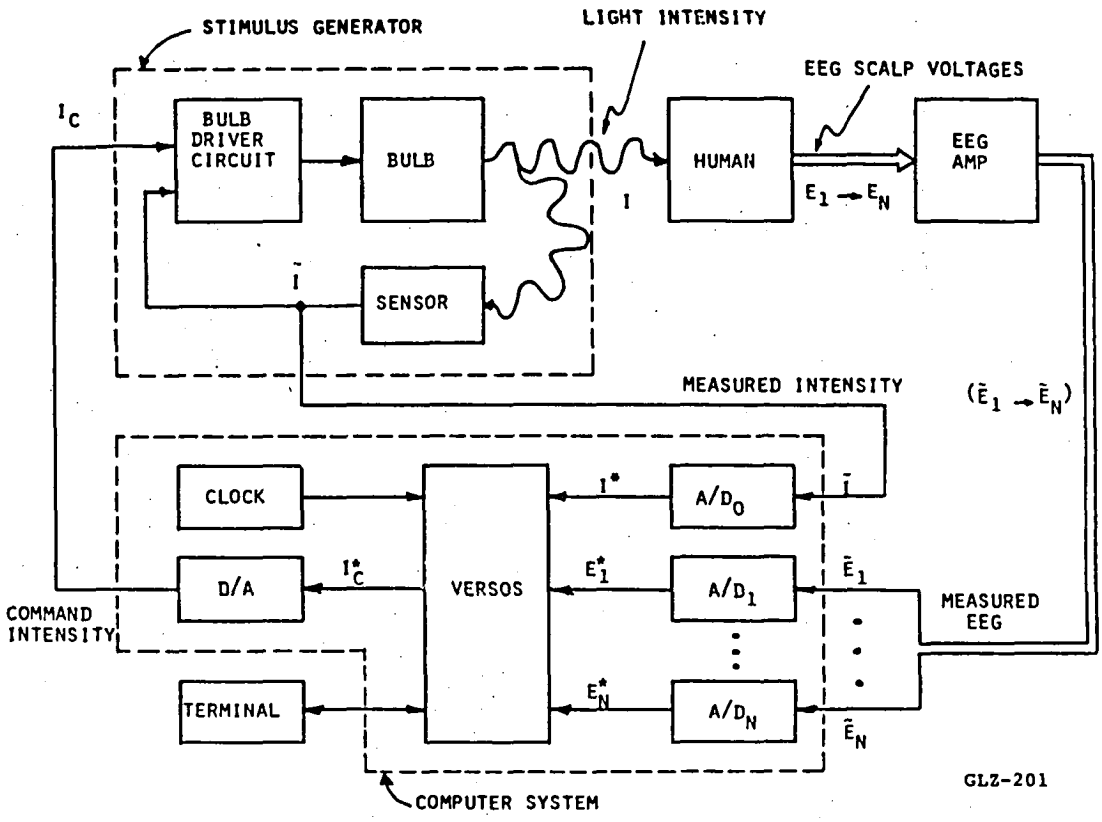


FIG. 6. CLOSED-LOOP STIMULUS/RESPONSE ENVIRONMENT

channel. VERRUN implements four major functions as diagrammed in Figure 7: (1) initial setup and parameter specification, (2) pre-trial initialization, (3) real-time SOS generation and data recording, and (4) post-trial file maintenance. Typically, initial setup and parameter specification is performed only at the start of a multi-trial experimental session, and the remaining three functions are performed in order during each experimental trial.

A user will generally want to use the same time-base and SOS parameters throughout an entire experiment (except for re-randomization of the SOS phases). Since VERRUN can be

initialized with values stored on a previously-created data file, an entire experiment can be run with a minimum of user interaction with the program.

3.1.1 Initial Setup and Parameter Specification

Both time-based and SOS parameters are specified during this initialization phase. Parameters may be specified in one of four ways:

a. Read all parameter values from a previously-created file.

b. Request "nominal" (pre-stored) values for all parameters.

c. Specify all parameters interactively.

d. Request nominal values for time-base (or SOS) parameters and specify SOS (or time-base) parameters interactively.

If parameters are specified interactively, or if nominal values are requested individually for the time-base and SOS parameter sets, the user is provided an opportunity to review and modify parameter values before continuing on. This review/modification option is omitted if the parameters are read from file, or if nominal values have been requested for all parameters.

The user is then asked if he wishes to perform a run. If so, VERRUN executes pre-trial initialization. If not, the parameters are stored on a file specified by the user, and VERRUN provides the options of (a) specifying another parameter set, (b) performing an experimental trial, or (c) terminating the program.

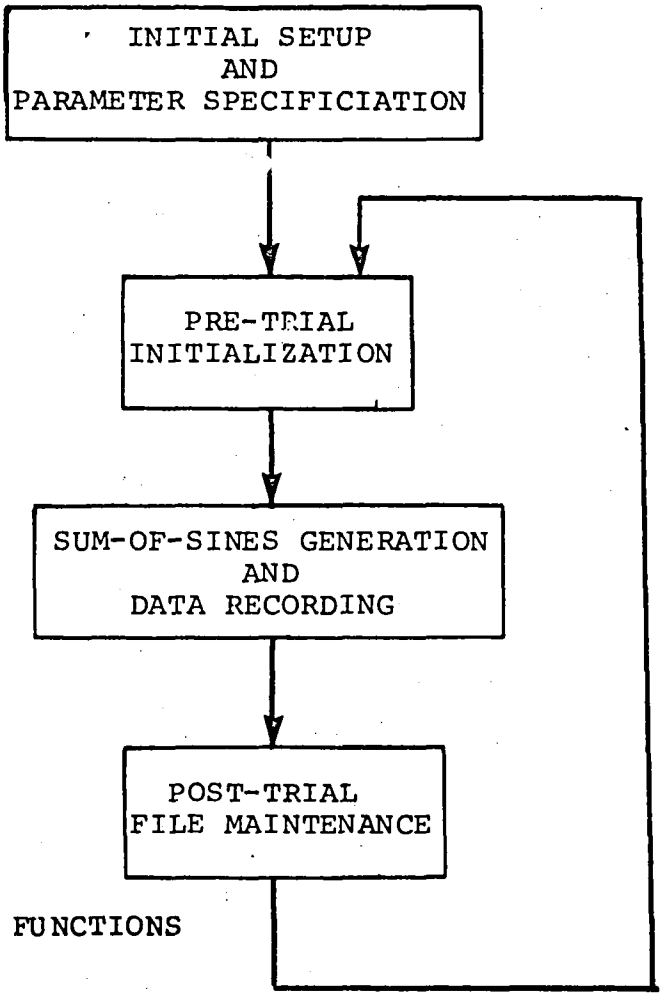


FIG. 7. MAJOR VERRUN FUNCTIONS

With direct user specification of the time-base parameters, the user is prompted to enter the sample interval in milliseconds, I_S, and the overall run length in seconds T_R, defining the duration of an experimental trial. Both entries are checked against minimum and maximum limits; nominal as well as limiting values are shown in Table 1. VERRUN then computes the sample interval in seconds, T_S, and the number of samples per trials, N_R, as follows:

$$T_S = I_S/1000 \tag{38}$$

$$N_R = T_R/T_S + 1 \tag{39}$$

Values specified for I_S and T_R are checked again to make sure that N_R does not exceed the system's preset upper storage limits; nominal values are given in Table 1.

TABLE 1. TIME-BASE PARAMETER VALUES AND LIMITS

Parameter	units	nominal	minimum	maximum	note
I_S	msec	5	1	100	(1)
T_R	sec	5.2	0	100	(1)
N_R	--	1040	0	2250	(2)

note: (1) both parameters are also checked to ensure satisfying limits on N_R

(2) computed via (2.14)

Given the total number of sample points N_R comprising a run, VERRUN specifies the total number of sample points N_o for one period of the SOS signal. For compatibility with the FFT routine to be used later for signal analysis, the value for N_o is chosen to be the largest power of 2 less than or equal to N_R. VERRUN also computes the overall SOS period in seconds T_o, the base frequency in Hz f_o, and the base phase in degrees ϕ_o, as described in Section 2.1. Time-base parameters are then listed for user verification and respecification if not satisfactory.

With direct user specification of the SOS parameters, the user is first prompted to specify the number of sinewave components, N_c. This may be done by specifying the "nominal" value option, or by direct entry, in which case limit checks are provided. Limiting and nominal values are given in Table 2.

The user is then prompted to specify a desired SOS frequency set, f'_j, where j ranges from 1 to N_c, and f'_j is in Hz. This can be done by specifying the "nominal" frequency set option (if N_c is nominally specified), in which case the first N_c components of the nominal frequency set are selected. If the user chooses instead to specify the N_c frequencies directly, VERRUN allows for corrections to be made during data entry, and provides checks to ensure that the chosen frequencies are consistent with the previously-chosen sample and run times. Limiting and nominal values are given in Table 2.

Once the desired frequency set has been specified, VERRUN

TABLE 2. SOS PARAMETER VALUES AND LIMITS

Parameter	units	nominal	minimum	maximum	note
N_c	--	6	1	15	
f'_j	Hz	5,10,..,75	f_o	$f_s/2$	(1)
$\tilde{a}_j$	--	1,1,1,...	0	100	
ϕ'_j	deg	--	0	360	(2)
I_{RMS}	volts	1	0	5	

notes: (1) $f_o = 1/T_o$ and $f_s = 1/T_S$

(2) nominal values set by random number generator

then computes, for each component, the nearest corresponding integer multiplier according to:

$$h_j = [f'/f_o]_i \quad (j=1,...,N_c) \tag{40}$$

This then yields the harmonically related SOS frequencies f_j, where

$$f_i = h_j f_o \quad (j=1,...,N_c) \tag{41}$$

Naturally, progressively smaller values of f_o allow for progressively closer matches between the desired drive frequency sets f_j, and the actual harmonically derived set, f_j. Smaller values of f_o can, in turn, be obtained by increasing T_o.

Once the SOS frequency set has been specified in this fashion, the user is provided the opportunity of listing both desired and actual frequencies, along with the corresponding harmonics. If not satisfactory, VERRUN allows for respecification.

Following SOS frequency specification, VERRUN prompts the user for the distribution of SOS amplitudes with frequency. This is done by specifying normalized (dimensionless) amplitudes $\tilde{a}_j$, which are related to the SOS (dimensioned) amplitudes a_j, by a scale factor f, or:

$$a_j = f\tilde{a}_j \qquad (j=1,\ldots,N_c) \tag{42}$$

so that, with f free, the user can specify the shape of the a_j distribution, independent of the signal RMS level.

The normalized amplitudes $\tilde{a}_j$ may be set by specifying the "nominal" amplitude set option (if N_c is nominally specified), or by direct entry of the N_c normalized amplitudes. If the user chooses the latter, VERRUN allows for corrections to be made during data entry, and provides checks to ensure that the chosen amplitudes are within prespecified limits. Limiting and nominal values are given in Table 2.

Once the normalized amplitude set has been specified, VERSOS then prompts the user for the desired RMS signal level of the SOS signal, I_{RMS}. This may be done by specifying the "nominal" value option, or by direct entry, in which case limit checks are provided (limiting and nominal values are given in Table 2). VERRUN then computes the amplitude scale factor f according to:

$$f = \sqrt{2}\, I_{RMS} \left(\sum_{j=1}^{N_c} \tilde{a}_j^2 \right)^{-1/2} \tag{43}$$

By then computing the SOS amplitudes according to (42), VERRUN ensures that the SOS signal I(t) will have the desired RMS level, since

$$\overline{I^2(t)} = \sum_{j=1}^{N_c} \frac{1}{2} a_j^2 = \frac{1}{2} f^2 \sum_{j=1}^{N_c} \tilde{a}_j^2 = I_{RMS}^2 \qquad (44)$$

Following SOS amplitude specification, VERRUN prompts the user for a desired SOS phase set, ϕ_j, where j ranges from 1 to N_c. Phases can be selected in one of three ways (1) the "nominal" selection procedure, (2) specification of a "seed" for picking a set of random phases, or (3) direct specification of phases. If the user chooses the nominal option, VERRUN uses a random number generator to select uniformly distributed values between 0 and 360 deg; the "seed" of the random number generator is automatically changed from run to run to allow for a consistent means of randomizing the phase sets each run (and thus the SOS time history). If the user specifies the seed for phase randomization, or specifies phases directly, VERRUN allows for corrections to be made during data entry, and provides checks to ensure that the chosen phases are within prespecified limits (given in Table 2).

Once the desired phase set has been specified, VERRUN then computes, for each component, the nearest corresponding integer phase multiplier, p_i, according to:

$$p_j = [\phi'_j/\phi_o] \qquad (j=1,\ldots,N_c) \qquad (45)$$

The SOS phases can then be computed as integral multiples of the base phase as

$$\phi_j = p_j \cdot \phi_o \qquad (46)$$

This, of course, quantizes the phase choices, but progressively smaller values of ϕ_o allow for progressively closer matches between the desired phase set ϕ'_j and the actual set ϕ_j. Smaller values for ϕ_o can, in turn, be obtained by reducing the ratio of T_S/T_o.

3.1.2 Pre-Trial Initialization

Pre-trial initialization consists of four basic steps. First, if an experimental trial has just been completed, and the user has requested another run, the user is provided the option to change all, some, or none of the time-base and SOS parameters. If the user requests no changes, SOS component phases are automatically re-randomized. (This step is omitted on the first trial following initial setup and parameter specification.)

Next VERRUN displays the date, time, and run number selected for the upcoming trial. (The run number is set to 1 during initial setup and is automatically incremented by 1 for successive trials.) The user either accepts or modifies the run number and then specifies up to 6 lines of commentary.

VERRUN then prompts the user for a filename for parameter and data storage. After some simple legality checks on the

entered name, VERSOS opens a file and writes out the "header": that portion of the data file comprised of the (previously-defined) run parameter values, along with miscellaneous "housekeeping" parameters and tags to aid in later data file maintenance.

Finally, VERRUN generates a "pre-stored" version of the entire SOS signal to be used. This is done by first generating and storing a "quarterwave" sine table associated with the sample and base periods, T_S and T_o, of the SOS signal, using the tabular sinusoidal function S_n, as described in Section 2.1. With this table, the sampled-time version of the SOS signals is then computed for all N_R samples which comprise a complete run. Each sample value is then scaled for eventual conversion by the D/A hardware, and then stored in a linear data array. With the SOS signal generated and stored, VERSOS prompts the user for a "run start" signal, and waits for the user's response.

3.1.3 Real-Time Control

Once a start signal is received from the user, VERRUN zeros the D/A channels and starts the digital clock "ticking" at a pre-specified rate (nominal clock rate is 100 kHz). After the clock has counted down the number of ticks corresponding to the desired sample interval T_S, D/A and A/D conversions are performed. This cycle is repeated N_R times to generate an experimental trial of the desired length T_R seconds, after which the clock is stopped and the D/A channels zeroed.

Two signals are generated each sample interval: (a) a square wave alternating between maximum positive and negative values on D/A channel Ø, to be used for test purposes, and (b) the SOS signal on channel 1, obtained by table lookup.

Three signals are recorded by A/D channels 1-3 and are stored in the same linear array containing the SOS stimulus signal. The data sequences recorded from the three A/D channels are interleaved with each other and with the SOS stimulus. That is, the first element of the linear data array contains the first SOS sample, the second through fourth elements contain the first samples recorded from A/D channels 1-3, respectively, the fifth element contains the second SOS sample, and so forth. The linear data array will therefore contain $4*N_R$ samples at the end of the experimental trial.

3.1.4 Post-Run File Maintenance and Multi-Run Control

VERRUN "closes-out" a run by first writing the recorded data strings onto the file opened at the beginning of the run, thus appending the data to the parameter set used to specify the run. The file is then closed, and the user is provided the options of (a) performing another run, (b) setting up a parameter file, or (c) terminating the program. If another run is requested, the run number is incremented, and VERRUN proceeds with pre-trial initialization as described in Section 3.1.2. Request for a new parameter file returns the program to the initialization mode described in Section 3.1.1.

3.2 Program Generation and Operation

VERRUN was designed to run efficiently under DEC's RT-11 operating system, but program development can be conveniently done under the RSX-11 operating system in a time-shared mode.

3.2.1 Program Generation

An executable file of the VERRUN software system is generated within the RSX-11 Operating System by the command:

```
TKB @VERRUN.CMD
```

where the file VERRUN.CMD contains the following text:

```
VERRUN=VERRUN
PARSET
TIMPAR
SOSPAR
SOSNCP
SOSHMC
SOSAMP
SOSPHS
SOSGEN
LOOP
RWHEAD
RWDATA
TITLER
UTLLIB/LB
IOLIB/LB
/
RESLIB=(1,54)DEVCOM/RW=7
//
```

The last three lines of the CMD file exercises the option to access a specific file in the resident library. This file is required to allow real-time operations by the RSX system.

3.2.2 Program Operation

Two examples of VERRUN operation are shown in this section. First, we illustrate the procedure one might follow when defining parameters for a new experiment. The second example illustrates the more typical operating mode in which minimal trial-to-trial changes are made. For expository purposes, the user input is circled in these examples.

Figure 8 illustrates a sample dialog for an initial experimental trial. User entries are circled; other text is generated by the program. Section A shows that the user has refused to accept nominal time-base parameters and has interactively specified the sample period and run time (trial duration). Upon request, VERRUN lists the specified and computed time-base parameters, along with the base frequency.

Section B illustrates interactive specification of component SOS frequencies, followed by a listing of the final set of harmonic indices and frequencies. Note that the actual frequencies differ slightly from the desired (user-specified) frequencies because of the requirement for VERRUN to use integral harmonics of the base frequency. In Section C, the user specifies component amplitudes and overall signal rms level, and VERRUN lists both the relative amplitudes specified by the user as well as the adjusted amplitudes that will be used later to generate an SOS signal of the specified rms level.

```
>RUN VERRUN

PARAMETERS FROM A FILE? (N)

NOMINAL PARAMETERS? (N)


**********TIME BASE PARAMETERS**********
NOMINAL TIME BASE? (N)

SAMPLE PERIOD (MSEC) = (5)
RUN TIME (SEC) = (6)

LIST TIME BASE PARAMETERS? (Y)


SAMPLE PERIOD =    5    (MSEC)
RUN LENGTH =       6.00 (SEC) WITH 1201 SAMPLES
SOS PERIOD =       5.12 (SEC) WITH 1024 SAMPLES
BASE FREQ =        0.20 (HZ), BASE PHASE =      0.35 (DEG)
OK? Y
```

(A)

FIG. 8. SAMPLE DIALOG FOR INITIAL OPERATION OF VERRUN

```
**********SOS PARAMETERS**********
NOMINAL SOS? (N)

NOMINAL NUMBER OF SINES? (N)


NUMBER OF SINES= (7)

NOMINAL FREQUENCIES? (N)


ENTER DESIRED FREQUENCIES (HZ):

F( 1)=6
F( 2)=7.5
F( 3)=9
F( 4)=10.5
F( 5)=12
F( 6)=13.5
F( 7)=15

ANY CHANGES? (N)

WANT FREQUENCIES LISTED? (Y)

COMP        HARM        FRQ         FRQ(DES)

  1           31         6.05          6.00
  2           38         7.42          7.50
  3           46         8.98          9.00
  4           54        10.55         10.50
  5           61        11.91         12.00
  6           69        13.48         13.50
  7           77        15.04         15.00

OK? (Y)
```

(B)

FIG. 8. (Cont'd)

```
NOMINAL AMPLITUDES? (N)

ENTER (RELATIVE) AMPLITUDES:

A( 1)=1
A( 2)=1
A( 3)=.5
A( 4)=.5
A( 5)=.5
A( 6)=1
A( 7)=1

ANY CHANGES? (N)

NOMINAL RMS LEVEL? (N)

RMS LEVEL (VOLT) = (2.0)
LIST AMPLITUDES? (Y)

COMP        AMP         AMP (REL)

  1         1.30          1.00
  2         1.30          1.00
  3         0.65          0.50
  4         0.65          0.50
  5         0.65          0.50
  6         1.30          1.00
  7         1.30          1.00

OK? (Y)

NOMINAL PHASES? (Y)

LIST PHASES? (Y)

COMP        PMUL          PHS          PHS(DES)

  1         1018        357.89         357.87
  2          563        197.93         197.90
  3          101         35.51          35.34
  4          322        113.20         113.04
  5          197         69.26          69.39
  6          680        239.06         239.10
  7          714        251.02         251.02

OK? (Y)
```

(C)

(D)

FIG. 8. (Cont'd)

```
LIST SOS PARAMETERS? (Y)

COMP      HARM        FREQ        AMP         PHASE

   1        31        6.05        1.30        357.89
   2        38        7.42        1.30        197.93
   3        46        8.98        0.65         35.51
   4        54       10.55        0.65        113.20
   5        61       11.91        0.65         69.26
   6        69       13.48        1.30        239.06
   7        77       15.04        1.30        251.02

OK? (Y)

DOING A RUN NOW? (Y)

RUN NUMBER:     1     DATE: 15-DEC-83     TIME: 11:04:56

CHANGING THE RUN NUMBER? (N)

NUMBER OF COMMENT LINES: (1)
!TEST OF VERRUN PROGRAM
ENTER FILENAME FOR OUTPUT: TEST1.VER
GENERATING SOS SIGNAL NOW...

TYPE S TO START: (S)
STORING DATA NOW...

DOING ANOTHER RUN? (N)

SET UP A PARAMETER FILE? (N)

TTO  --  STOP

>
```

(E)

(F)

FIG. 8. (Concl'd)

Section D shows the user selecting nominal phases (i.e., VERRUN selects a random phase set). Direct user specification of phases would be highly unlikely even in the initial setup mode and would most likely be employed only for program testing and debugging. Note that, after each set of parameters has been specified, VERRUN asks the user if he is satisfied with the results. If the user responds with "N", the particular set is re-specified.

In Section E the user requests a review of the entire set of SOS parameters and accepts the results. If the user were to respond "N" to the query, VERRUN would repeat Sections B through E, affording the user an opportunity to modify any or all SOS parameter subsets.

Section F illustrates the following sequence of events:

1. The user decides to conduct an experimental trial. (The alternative would be to save only the parameters on file.)
2. The user accepts the run number, which is automatically initialized to "1".
3. The user specifies one line of comment and names the output file.
4. Real-time SOS generation and data recording are initialized by responding "S" to the prompt.
5. The user terminates VERRUN by declining to perform another run or another problem initialization.

Figure 9 shows the type of terminal interaction that might occur in a "production-run" mode where the user performs a

sequence of experimental trials with a statistically invariant SOS stimulus. Section A assumes that the user initializes the first such trial from the data file created in the sample case discussed above. After specifying the name for the new data file, the user changes the run number to "2", as this is the second trial to be performed the same day. The user then provides a single line for commentary, initiates real-time operation, and requests another run.

The type of interaction that will occur for most experimental trials is shown in Section B. The user requests no changes from the previous run, causing VERRUN to retain all previous parameter values except for re-randomization of the phases. The user then accepts the new run number, types a comment line, initiates real-time operations, and requests another run.

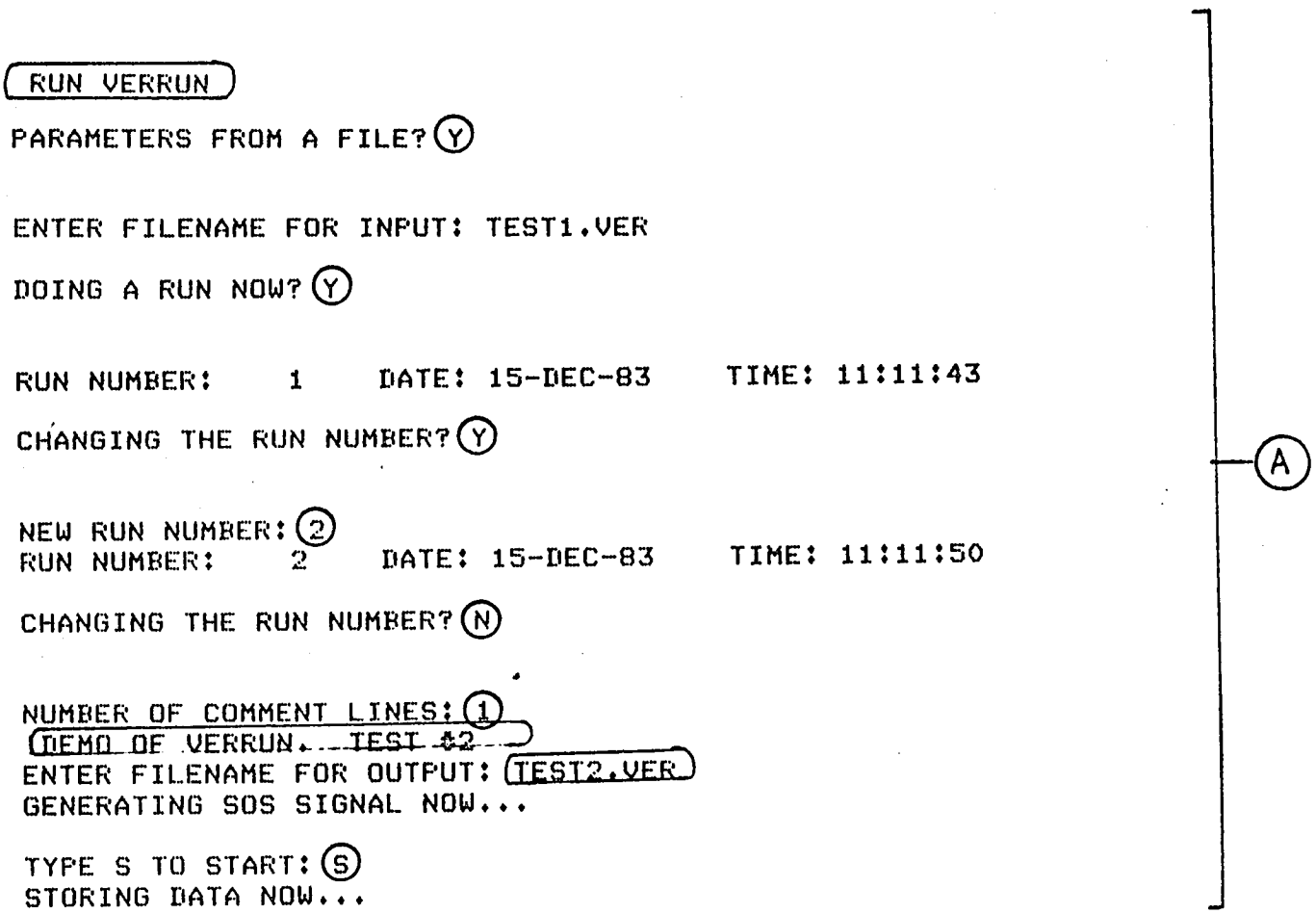


FIG. 9. SAMPLE DIALOG FOR CONTINUING OPERATION OF VERRUN

```
DOING ANOTHER RUN? (Y)

ANY CHANGES? (N)

RUN NUMBER:     3     DATE: 15-DEC-83     TIME: 11:12:55

CHANGING THE RUN NUMBER? (N)

NUMBER OF COMMENT LINES: (3)
!DEMO OF VERRUN
!PRODUCTION RUN
!TEST #3
ENTER FILENAME FOR OUTPUT: (TEST3.VER)
GENERATING SOS SIGNAL NOW...

TYPE S TO START: (S)
STORING DATA NOW...


DOING ANOTHER RUN? (Y)
```

(B)

FIG. 9. (Concl'd)

4. USER'S GUIDE TO VERNAL

VERNAL is a digital computer program for performing post-experiment analysis of VER data obtained using the VERRUN program described in Chapter 3. VERNAL is written entirely in FORTRAN and is implemented on the PDP-11/34, using the RSX-11 operating system, and the PDP-11/23, using the RT-11 operating system.

4.1 Major Functions

VERNAL performs the five major operations shown in Figure 10. This program is "menu-driven" in that the user specifies interactively, via a "part" number, the operation VERNAL is to perform. Upon completion of a given operation, the user specifies the next operation to be performed. A part number of 0 displays the options shown in Figure 9, and a part number of -1 terminates the program.

Part 1 (read header) must be performed first; otherwise, program parts may be executed in any order. Figure 10 shows the typical order in which program functions are executed. These functions are described individually below.

4.1.1 Part 1: Read Header

Once the user has specified the name of the data file, VERNAL opens the file, reads the header information, and leaves the file open for subsequent reading of the experimental data.

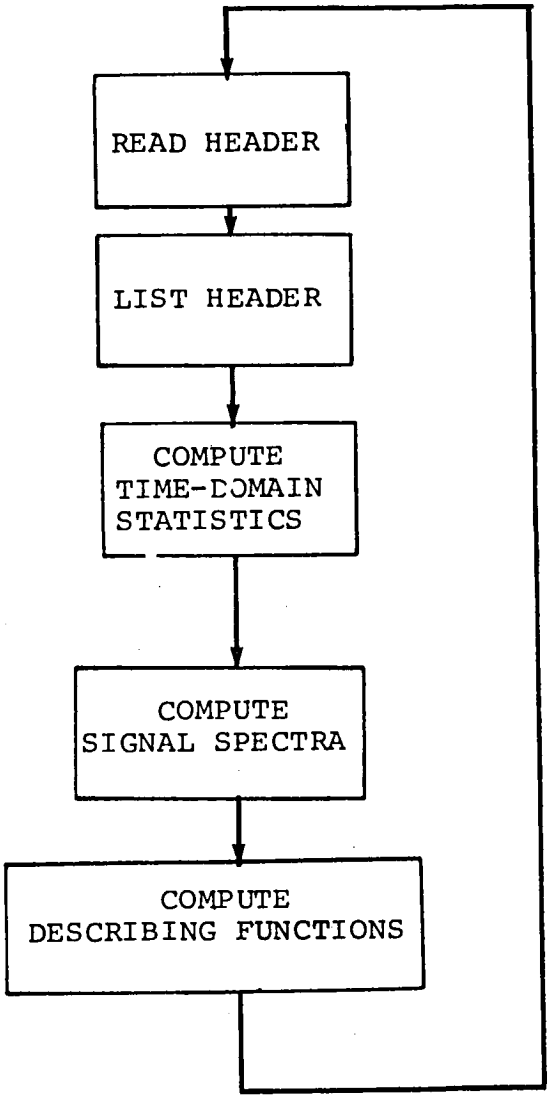


FIG. 10. MAJOR VERNAL FUNCTIONS

4.1.2 Part 2: List Header

Header information consisting of run identification, problem parameters, and user commentary, is displayed on the user's terminal. If the user then discovers he has not requested a file of interest, he may next request re-execution of Part 1, in which case the current file is closed, and a new file is requested and opened.

4.1.3 Part 3: Time-Domain Statistics

When a statistical computation (either time- or frequency-domain) is first requested for a given data file, VERNAL reads the experimental data from the current file, stores the data in a linear array, and closes the file. The user is informed of the currently specified starting point for calculations, and is given the option to change the start point, which must lie within the range of 1 to $N_R - N_o + 1$, where N_R is the number of samples/channel in the experimental trial, and N_o is the number of samples in the SOS period. This restriction guarantees that N_o samples will be available for computation. The user will typically request a start point greater than 1 to minimize the influence of the transients that most likely followed the onset of the SOS stimulus.

Before computing time-domain statistics, VERNAL provides the option to list the entire data base stored in the array IDATA, or to list an array XDATA of data from a single channel of the

user's choosing. Unless the user is debugging the program, or suspects unusual response behavior, this option will typically not be exercised.

The primary function of this part is to compute mean, standard deviation, and rms amplitude as defined in Section 2.2, Equation 17. These quantities are computed for all data channels and displayed on the user's terminal.

4.1.4 Part 4: Spectra

Part 4 computes the spectra of one or more signals of the user's choosing, using fast-Fourier transform (FFT) techniques as described in Section 2.2. Once the spectrum has been computed for a specified data channel, the user has the option of listing either the entire spectrum (i.e., at all FFT frequencies) or the spectral components at SOS frequencies. Again, unless the user is debugging the program or looking for some specific spectral feature (say, evidence of significant nonlinear response behavior), this option will typically not be exercised.

Whether or not the listing option is exercised, VERNAL will list, for each input frequency: (1) correlated power per measurement bin, (2) remnant power per bin, (3) the ratio of the correlated to remnant power, (4) correlated power per rad/sec, (5) remnant power per rad/sec, (6) the ratio for correlated power to remnant power (rad/sec), and (7) the number of frequency bins included in the remnant averaging window. These spectral

quantities are given in dB. Conversion of power per bin to power per rad/sec is discussed in Section 2.2.1.)

The following overall statistics (in problem units) are then listed: (1) correlated power summed over all input frequencies, (2) rate of correlated to total signal power, (3) remnant power summed over all non-input frequencies, (4) rate of remnant to total power, and (5) total signal power(i.e., sum of all spectral computations over all frequencies). The user is then given the option to perform another spectral analysis or to specify another program part.

4.1.5 Part 5: Describing Functions

Part 5 performs a describing function analysis as defined in Section 2.2.2. When execution is begun, VERNAL prompts the user for indices corresponding to the numerator and denominator channels. After the requested describing function h as been computed, gain (in dB) and phase (in degrees) are printed out at each SOS frequency, except that computations failing the 6 dB signal/noise ratio test (Section 2.2.1) are flagged by a printout of the string (****). The user then has the option of computing another describing function or specifying another program part.

4.2 Program Generation and Operation

VERNAL has been implemented to run under the DEC RT-11 and RSX-11 operating systems. This program performs post-experiment analysis with no requirement for real-time operation.

4.2.1 Program Generation

An executable file of the VERNAL software system is generated within the RSX-11 Operating System by the command:

```
TKB @VERNAL.CMD
```

where the file VERNAL.CMD contains the following text:

```
VERNAL=VERNAL
PART
SIGNAL
STATS
SPECT
DFCN
REMPWR
FFT
RWHEAD
RWDATA
TITLER
FFTPKG
IOLIB/LB
```

4.2.2 Program Operation

A sample dialog with VERNAL is shown in Figure 11. Section A shows that the user has requested the file "TEST3" and, by requesting execution of Part 2, has caused VERNAL to display the parameter values and other descriptive information for this data file.

In Section B, the user requests execution of Part 3 to obtain time-domain statistics. The start point (initialized to unity when VERRUN is first started) is changed to 150 to allow statistical analysis to begin 0.75 seconds into the run. After the options to list time histories are waived, VERNAL displays

```
:RUN VERNAL
TO PART (0-6): 1
ENTER FILENAME FOR INPUT: TEST3.VER
TO PART (0-6): 2
VERSION NUMBER: 2

***RUN IDENTIFICATION***
FILE: TEST3.VER        RUN NO:    3     DATE: 15-DEC-83    TIME: 11:12:55
 DEMO OF VERRUN
 PRODUCTION RUN
 TEST #3


***TIME BASE PARAMETERS***                                          A
SAMPLE PERIOD:              5 MSEC
BASE FREQUENCY:    1.953E-01   HZ    BASE PHASE:    3.516E-01 DEG
SOS PERIOD:        5.120E+00   SEC   WITH:                1024 PTS
RUN LENGTH:        6.000E+00   SEC   WITH:                1201 PTS

***SOS SIGNAL PARAMETERS***
# OF SOS COMPONENTS:    7
 COMP     HARM       FREQ        AMP         PMUL        PHS
   1        31       6.05       1.298        500        175.8
   2        38       7.42       1.298        614        215.9
   3        46       8.98       0.649        713        250.7
   4        54      10.55       0.649        131         46.1
   5        61      11.91       0.649        907        318.9
   6        69      13.48       1.298        848        298.2
   7        77      15.04       1.298        916        322.1
TO PART (0-6): 3
READING IN DATA NOW....

SCORING STARTS AT POINT     1  WANT TO CHANGE? Y


ENTER START POINT IN RANGE 1 THRU   178:150

**TEST CODE: WANT IDATA LISTED? N


**TEST CODE: WANT XDATA LISTED? N                                   B

DOING STATS NOW...

FILE: TEST3.VER        RUN NO:    3     DATE: 15-DEC-83    TIME: 11:12:55


     CHAN        AVG          S.D.          RMS
      1        -0.001        2.000         2.000
      2        -0.002        4.001         4.001
      3         1.998        2.000         2.827
      4         0.004        2.061         2.061
```

FIG. 11. SAMPLE DIALOG FOR OPERATION OF VERNAL

```
TO PART (0-6): (4)
SPECTRUM FOR CHANNEL #: (1)
DOING FFT...
**TEST CODE: WANT SPECTRUM LISTOUT? (N)


FILE: TEST3.VER        RUN NO:    3     DATE: 15-DEC-83    TIME: 11:12:55

                        SPECTRUM FOR CHANNEL #  1

                              PWR/BIN                          PWR/HZ
COMP    FREQ  *     COR        REM       C/R   *     COR        REM       C/R   * NREM
                    *                          *                                 *
  1     6.05  *   -0.74     -89.83     89.08  *   -9.01     -82.73     73.73  *    6
  2     7.42  *   -0.75     -90.37     89.62  *   -2.40     -83.27     80.88  *    6
  3     8.98  *   -6.77     -91.23     84.46  *   -8.72     -84.14     75.42  *    8
  4    10.55  *   -6.77     -91.55     84.79  *   -8.45     -84.46     76.01  *    9
  5    11.91  *   -6.77     -89.84     83.07  *   -8.41     -82.75     74.33  *   11
  6    13.48  *   -0.75     -89.81     89.07  *   -2.69     -82.72     80.03  *   12
  7    15.04  *   -0.75     -90.45     89.70  *   -2.92     -83.35     80.43  *   13

     COR PWR =       4.00      COR/TOT PWR =       1.00
     REM PWR =       0.00      REM/TOT PWR =       0.00

     TOT PWR =       4.00


ANOTHER SPECTRUM? (N)


TO PART (0-6): (5)
CHANNEL # FOR DFCN   NUM: [2]
CHANNEL # FOR DFCN DENOM: [1]
DOING FFT...

DOING FFT...

FILE: TEST3.VER        RUN NO:    3     DATE: 15-DEC-83    TIME: 11:12:55

                    DFCN FOR (CHAN 2)/(CHAN 1)

                  COMP      FREQ       GAIN       PHASE
                    1       6.05        6.0        0.0
                    2       7.42        6.0        0.0
                    3       8.98        6.0        0.0
                    4      10.55        6.0        0.0
                    5      11.91        6.0        0.0
                    6      13.48        6.0        0.0
                    7      15.04        6.0        0.0

ANOTHER DFCN? (N)
```

FIG. 11. (Concl'd)

```
TO PART (0-6): (-1)
TTO  --  STOP

>
```

the average, standard deviation, and rms levels for all four data channels.

The user then requests that VERNAL compute the spectrum of data channel No. 1 (Section C). VERNAL performs the required FFT analysis, computes input-correlated and remnant spectral components, and displays the results. The user declines the option to compute another spectrum.

In Section D the describing function computation is initialized by specification of the data channels corresponding to the numerator and denominator variables. After FFT's have been computed for both channels, gain and phase shift are computed and displayed. The user then declines to compute another describing function and terminates VERNAL by specifying execution of Part No. -1.

PROGRAMMER'S GUIDE TO VERRUN AND VERNAL

The software system described in this Programmer's Guide consists of two main programs, several major FORTRAN subprograms, a FORTRAN library of input/output support routines, and a MACRO library of programs used for real-time operations and for random number generation. Description of the various software elements is organized into five sections as follows: (A) the VERRUN main program and the major FORTRAN subprograms called by VERRUN; (B) the VERNAL main program and the major FORTRAN subprograms called by VERNAL; (C) additional major FORTRAN subprograms called by both VERRUN and VERNAL; (D) the I/O FORTRAN library, and (E) the MACRO library.

APPENDIX A
THE VERRUN SOFTWARE SYSTEM

A.1 Program Structure

The organization of the VERRUN software system is shown in Figure A.1. The main program VERRUN will, in the normal course of events, call the six main subprograms PARSET, TITLER, RWHEAD, SOSGEN, LOOP, and RWDATA. They, in turn, call the routines indicated by the line connections made to their respective blocks. In general, the calling sequence at any given level corresponds to the top-to-bottom ordering shown in the diagram. Thus, PARSET calls TIMPAR, SOSPAR, and RWHEAD in that order.

Subprograms belonging to the assembly-language MACRO library are indicated by cross-hatching. All other subprograms are written in FORTRAN, and most of these programs use one or more routines in the I/O library. In the interest of minimizing clutter, calls to the I/O library are not shown explicitly in this and in the ensuing flow diagrams.

A.2 Software Description

Table A.1 contains brief descriptions of each of the FORTRAN routines contained in the VERRUN software system. The remainder of this Appendix provides documentation for each of the routines listed in the Table except for TITLER, RWHEAD, and RWDATA (which are common

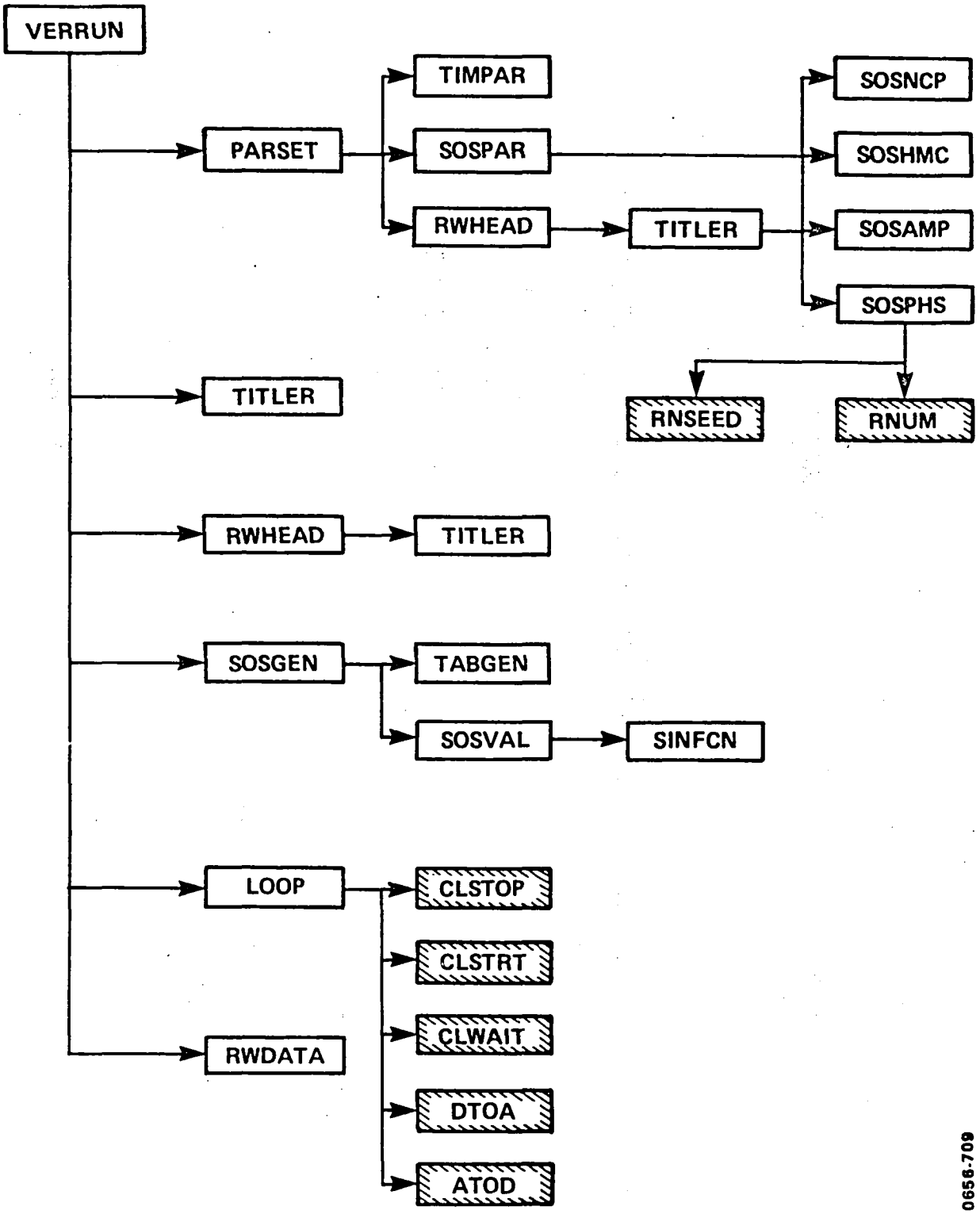


FIG. A.1 ORGANIZATION OF THE VERRUN SOFTWARE SYSTEM

to both VERRUN and VERNAL and are described separately in Appendix C). The documentation for each item consists of (1) a brief written description, a flow diagram, and a program listing. Except as noted above, program descriptions are provided in the order shown in Table A.1.

The written description consists of sections as follows:

FUNCTION:	a brief statement of the routine's function.
OPERATION:	a more detailed description of the routine's operation, and how the function is carried out.
INPUTS/OUTPUTS:	lists of the input and output variable which are passed by the routine's own argument list, or by COMMONs accessed by this routine.
LOCAL:	important variables not included in the argument list or in common blocks, especially variables passed to other routines.
CALLER/CALLS:	the name of the calling routine, and the names of any routines called.

In the case of the main programs VERRUN and VERNAL, only the calls are indicated; there are no inputs or outputs to a superior calling routine, and all variables are "local".

In the following program descriptions, variable names written entirely in capital letters indicate FORTRAN variables, whereas variable names written in lower case (or upper case with subscripts) refer to problem variables discussed in Chapter 2. The "=" symbol indicates either identity or replacement, as will be clear from the context. For example, the phrase "t =TSAMP" appearing in the
s

TABLE A.1 FUNCTIONS OF THE VERRUN ROUTINES

ROUTINE	FUNCTION
VERRUN	Main Program. Controls pre-run parameter setup, real-time SOS stimulus generation and response recording, and post-run file maintenance.
PARSET	Sets problem parameters interactively or by reading from existing file.
TIMPAR	Defines the time-base parameters during interactive user setup.
SOSPAR	Defines the SOS parameters during interactive user setup.
SOSNCP	Specifies the number of SOS components.
SOSHRC	Specifies the SOS harmonics.
SOSAMP	Specifies the SOS amplitudes.
SOSPHS	Specifies the SOS phase multipliers.
RWHEAD	Reads and writes header information.
TITLER	Reads and writes title information.
SOSGEN	Computes, scales and stores the SOS signal time history, before the start of each run.
TABGEN	Generates the basic quarter-wave sine table used for SOS generation.
SOSVAL	Generates a new SOS value for each call and increments the phase multiplier.
SINFCN	Generates one value of the tabular sinusoidal function for each call.
LOOP	Control real-time operation of the program, including (a) maintenance of the timing loop, (b) generation of the SOS stimulus signal, and (c) sampling and storing of data.

discussion of the routine TIMPAR signifies that the problem variable t_s is represented by the program variable TSAMP, whereas the phrase "TSAMP=ISAMP/1000.0" indicates a replacement operation executed within the routine.

The general format for a flow diagram is shown in Figure A.2. The routine of immediate interest is indicated by the block drawn with thick lines; the calling routine is shown above, and any routines called are shown below. The connecting "flow lines" are used to indicate the flow of information between routines via the argument list, where one routine's output becomes the other routine's input. Labels on these lines indicate the particular variables involved. Information flow via COMMON are indicated by flow lines circled and labelled with the name of the specific COMMON list in brackets. Because of their complexity, and because they have no calling routines, the main program VERRUN and VERNAL deviate somewhat from this format.

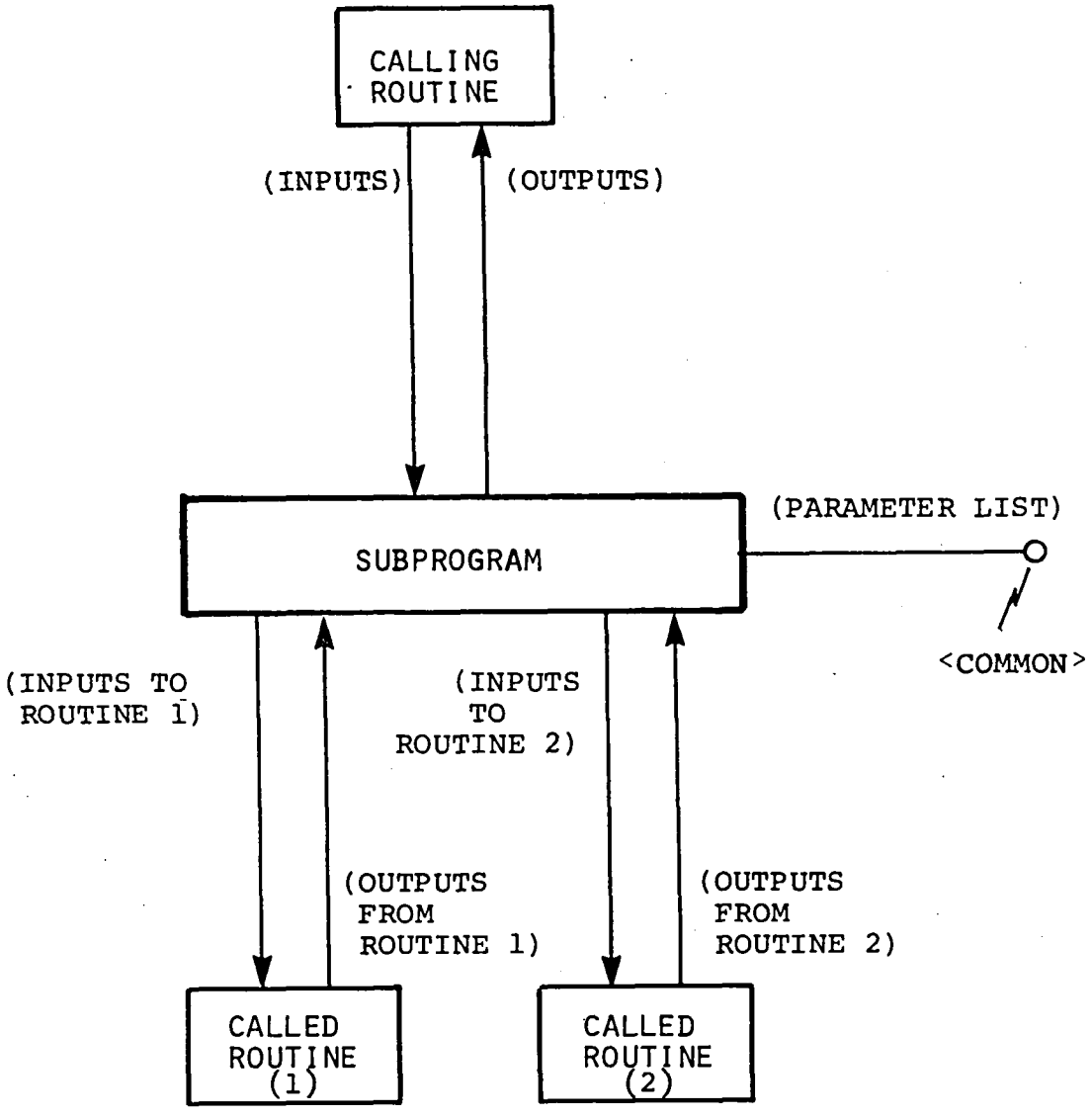


FIG. A.2 FLOW DIAGRAM FORMAT

program VERRUN

FUNCTION: Controls pre-run parameter setup, real-time SOS stimulus generation and response recording, and post-run file maintenance.

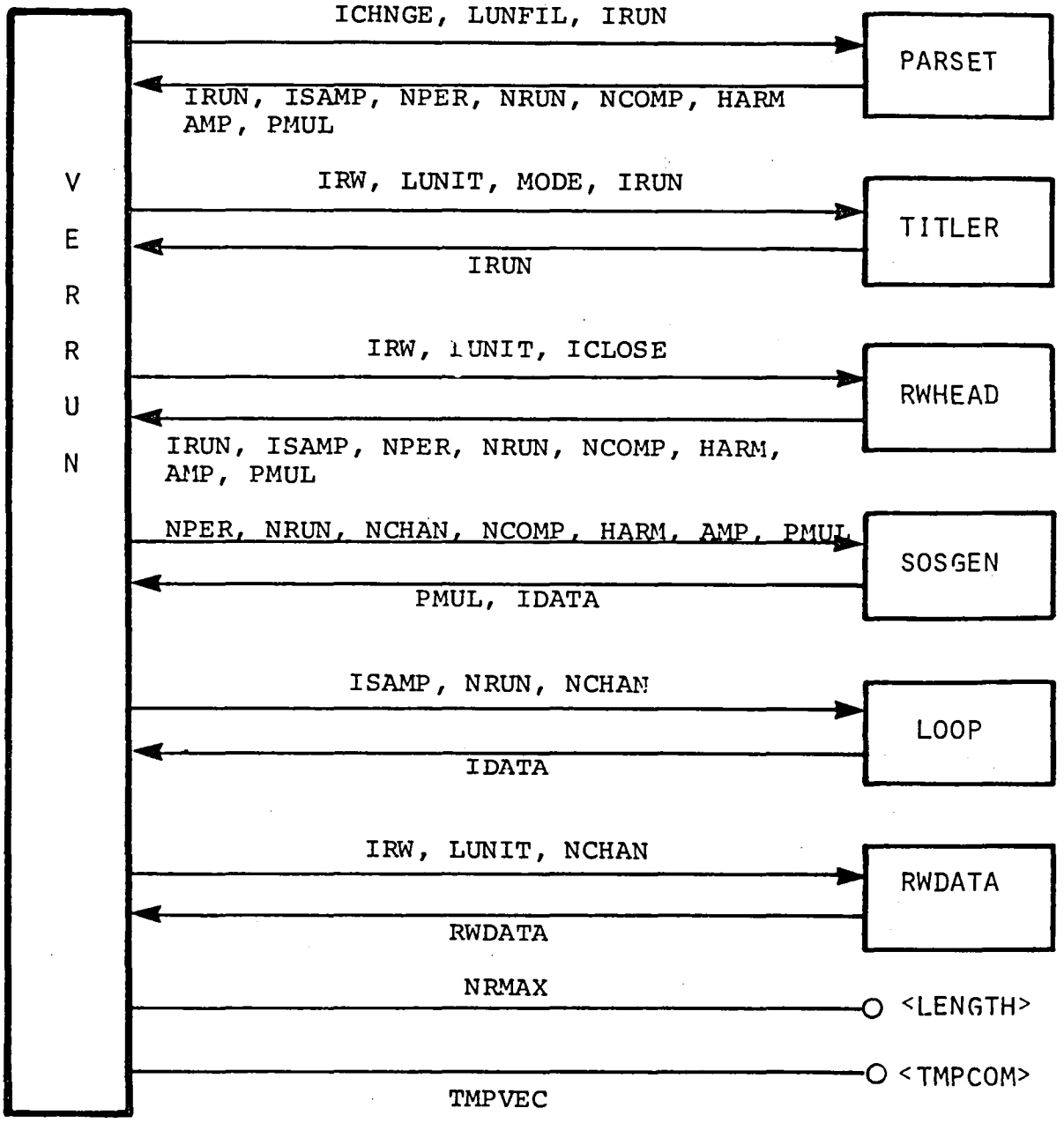
OPERATION: Operation begins with a call to PARSET to allow the user to specify problem parameters interactively or from a previously stored data file. If the user indicates he is not ready to complete a run, the parameters are stored on a new file, and pre-run parameter setup is again initiated by a call to PARSET.

Once the header is ready to run, header information is stored in the output file by a call to RWHEAD, and the entire SOS time history is computed and stored by a call to SOSGEN. VERRUN then waits for a run start signal from the user. Upon this signal, the routine LOOP is called to provide real-time stimulus generation, response recording, and in-memory storage of the data in the array IDATA.

Upon completion of the run, VERRUN writes out the data array IDATA via a call to RWDATA, closes the data file, and returns to the pre-run parameter setup portion of the program.

CALLS: PARSET, TITLER, RWHEAD, SOSGEN, LOOP, RWDATA

program VERRUN



0656-725

```
      PROGRAM VERRUN
C
C     CHANGES BY W.H. LEVISON,  12/9/83
C     1.  DEFINE LUNFIL AS UNIT 3
C
      COMMON /LENGTH/NRMAX
      COMMON /TMPCOM/TMPVEC
C
      LOGICAL*1 LASK, LANS, ICHNGE, MODE, FILNAM(11), TITLE(200)
      INTEGER HARM(15), PMUL(15)
      DIMENSION TMPVEC(20), AMP(15)
      DIMENSION IDATA (9000)
C
      DATA IDIM/9000/       !DIMENSION OF IDATA
      DATA LUNFIL/3/           !LUN FOR DATA FILE
      DATA LUNTTY/5/           !LUN FOR TTY
      DATA NCHAN /4/
      DATA MODE /'U'/       !START WITH UNDEFINED MODE
C
C     SET UP PARAMETERS...
C
 100  NRMAX = IDIM/NCHAN
      ICHNGE = 'Y'
      IRUN = 1
 110  CALL PARSET (ICHNGE,LUNFIL,IRUN,ISAMP,NPER,
     1              NRUN,NCOMP,HARM,AMP,PMUL)
C
      IF (MODE .NE. 'U') GO TO 120     !SET MODE TO P OR R
      MODE = 'P'
      IF (LASK ('DOING A RUN NOW? ') .EQ. 'Y')  MODE = 'R'
 120  IF (MODE .EQ. 'P') GOTO 300      !GO SET PARAMETERS
C
C             NORMAL RUN MODE
C
      DO 150 I = 1,IDIM                !ZERO OUT IDATA
 150  IDATA(I) = 0
C
 200  IRW = 1                   !READ TITLE FROM TTY
      CALL TITLER (IRW, LUNTTY, MODE, IRUN)
      IRW = 2                      !WRITE HEADER ONTO FILE
      ICLOSE = 2                   !AND LEAVE OPEN
      CALL RWHEAD (IRW,LUNFIL,ICLOSE,IRUN,ISAMP,NPER,
     1              NRUN,NCOMP,HARM,AMP,PMUL)
      CALL TTYOUT ('GENERATING SOS SIGNAL NOW...')
      CALL SOSGEN (NPER, NRUN, NCHAN, NCOMP, HARM, AMP, PMUL, IDATA)
      CALL TTYOUT ('TYPE S TO START: $')
      CALL LANS ('S', 'S')
      CALL LOOP (ISAMP, NRUN, NCHAN, IDATA)
      CALL TTYOUT ('STORING DATA NOW...')
      IRW = 2                   !WRITE DATA TO FILE & CLOSE IT
```

```
        CALL RWDATA (IRW, LUNFIL, NRUN, NCHAN, IDATA)
C
C
        CALL TTYOUT (' ')
        IF (LASK ('DOING ANOTHER RUN? ') .EQ. 'N') GOTO 210
        ICHNGE = LASK ('ANY CHANGES? ')
        IRUN = IRUN + 1          !INCREMENT RUN NUMBER
        GOTO 110
C
 210    IF (LASK ('SET UP A PARAMETER FILE? ') .EQ. 'N')  STOP
C
        MODE = 'P'
        GOTO 100
C
C               PARAMETER FILE SET UP MODE
C
 300    IRW = 1                  !READ TITLE FROM TTY
        CALL TITLER (IRW, LUNTTY, MODE, IRUN)
        IRW = 2                  !WRITE HEADER ONTO FILE
        ICLOSE = 1               !AND CLOSE IT
        CALL RWHEAD (IRW,LUNFIL,ICLOSE,IRUN,ISAMP,NPER,
     1               NRUN,NCOMP,HARM,AMP,PMUL)
C
C              USER SPECIFIES WHAT'S NEXT
G
        CALL TTYOUT ('-')
        IF (LASK ('ANOTHER PARAMETER FILE? ') .EQ. 'Y')  GOTO 110
        IF (LASK ('DOING A RUN NOW? ') .EQ. 'N')  STOP
C
        MODE = 'R'
        GOTO 100
        END
```

subroutine PARSET

FUNCTION: Sets problem parameters interactively or by reading from an existing file

OPERATION: If PARSET is called with the flag ICHNGE set to 'N', indicating no changes to previously-defined problem parameters, a call is made to the routine SOSPHS (via the routine SOSPAR) for re-randomization of phase multiplers PMUL(J). If ICHNGE indicates changes are to be made, the user has the option of initializing problem parameters from an existing file through a call to RWHEAD. If the user selects to define parameters directly, the flag NOMPAR is set to indicate whether or not parameters are to be selected interactively or selected from a stored set of nominal values. Time base and SOS parameters are then specified through calls to TIMPAR and SOSPAR, respectively, and control is returned to the main program VERRUN.

INPUTS: ARGLST: ICHNGE, LUNFIL, IRUN

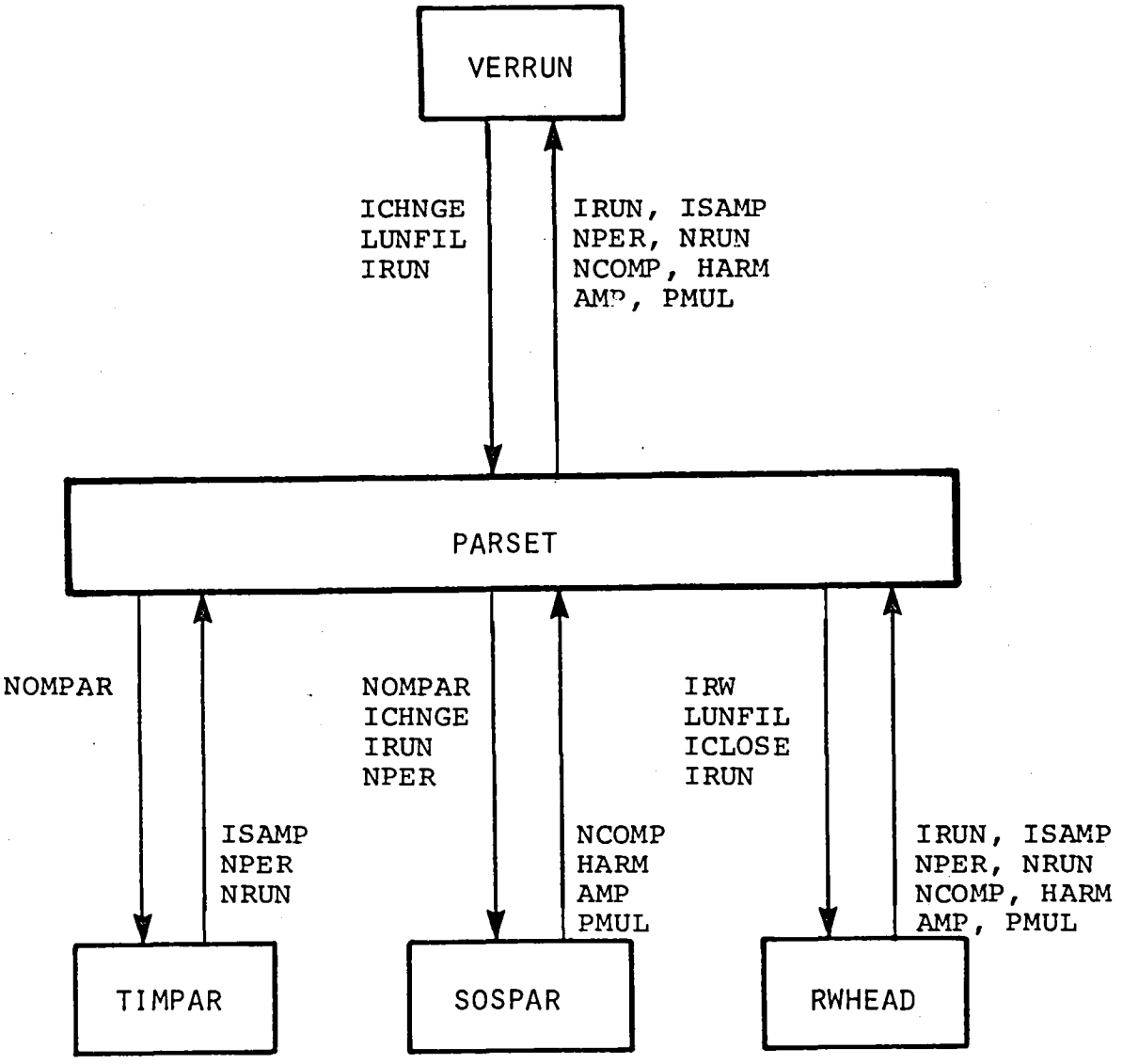
OUTPUTS: ARGLST: IRUN, ISAMP, NPER, NRUN, NCOMP, HARM, AMP, PMUL

LOCAL: NOMPAR, IRW, ICLOSE

CALLER: VERRUN

CALLS: TIMPAR, SOSPAR, RWHEAD

subroutine PARSET



```
      SUBROUTINE PARSET(ICHNGE,LUNFIL,IRUN,ISAMP,NPER,
     1                  NRUN,NCOMP,HARM,AMP,PMUL)
C
C       SETS THE PROBLEM PARAMETERS BY USER-SPECIFIED INPUTS,
C                         OR...BY READING FROM AN OLD FILE
C
C       INPUTS:  (VIA ARGLST)   ICHNGE, LUNFIL, IRUN
C       OUTPUTS: (VIA ARGLST)   ISAMP
C                (     "    )   NPER, NRUN, NCOMP
C                (     "    )   HARM, AMP, PMUL
C
C
      LOGICAL*1 LASK, NOMPAR, ICHNGE
      INTEGER HARM(1), PMUL(1)
      DIMENSION AMP(1)
C
      IF (ICHNGE .EQ. 'N') GOTO 200
      CALL TTYOUT (' ')
      IF (LASK ('PARAMETERS FROM A FILE? ') .EQ. 'Y') GOTO 300
C
C     GET PARAMETERS DIRECTLY FROM USER
C
 100  NOMPAR = LASK('NOMINAL PARAMETERS? ')
      CALL TIMPAR(NOMPAR, ISAMP, NPER, NRUN)
 200  CALL SOSPAR (NOMPAR,ICHNGE, IRUN, NPER, NCOMP, HARM, AMP,
      PMUL)
      RETURN
C
C     GET PARAMETERS FROM AN OLD FILE
C
 300  IRW = 1          !READ HEADER FROM FILE
      ICLOSE = 1       !AND CLOSE IT
      CALL RWHEAD(IRW,LUNFIL,ICLOSE,IRUN,ISAMP,NPER,
     1            NRUN,NCOMP,HARM,AMP,PMUL)
      CALL TTYOUT (' ')
      RETURN
      END
```

subroutine TIMPAR

FUNCTION: Defines the time-base parameters during interactive user setup

OPERATION: The following parameters are defined:

a. Intersample interval (msec) ISAMP

b. Intersample interval (seconds) t_s =TSAMP

c. Run length in seconds t_r =TRUN

d. Number of sample intervals in run N_r =NRUN

e. Number of sample intervals in measurement interval N_o =NPER

f. Minimum phase increment ϕ_o =PZERO

g. Minimum frequency increment f_o =FZERO

If the flag NOMPAR indicates selection of nominal parameters, ISAMP and TRUN are set to pre-stored values, remaining parameters are calculated as described below, and control is returned to the calling routine PARSET. Otherwise, the user specifies ISAMP and TRUN. Entered values are checked against nominal (stored) limits; if exceeded, the user is prompted to reenter.

TIMPAR computes timebase parameters as follows:

a. TSAMP=ISAMP/1000.0

b. NRUN=(TRUN/TSAMP)+1, rounded to the nearest integer. If NRUN exceeds a nominal (stored) limit NRMAX, the user is requested to re-specify the time base parameters.

c. NPER is set to the largest 2^k contained in NRUN, where k is an integer

d. PZERO=360.0/NPER

e. FZERO=1.0/TPER

If ISAMP and TRUN have been specified by the user, the user is allowed to review the entire set of time base parameters and to re-specify ISAMP and TRUN if desired before control is returned to PARSET.

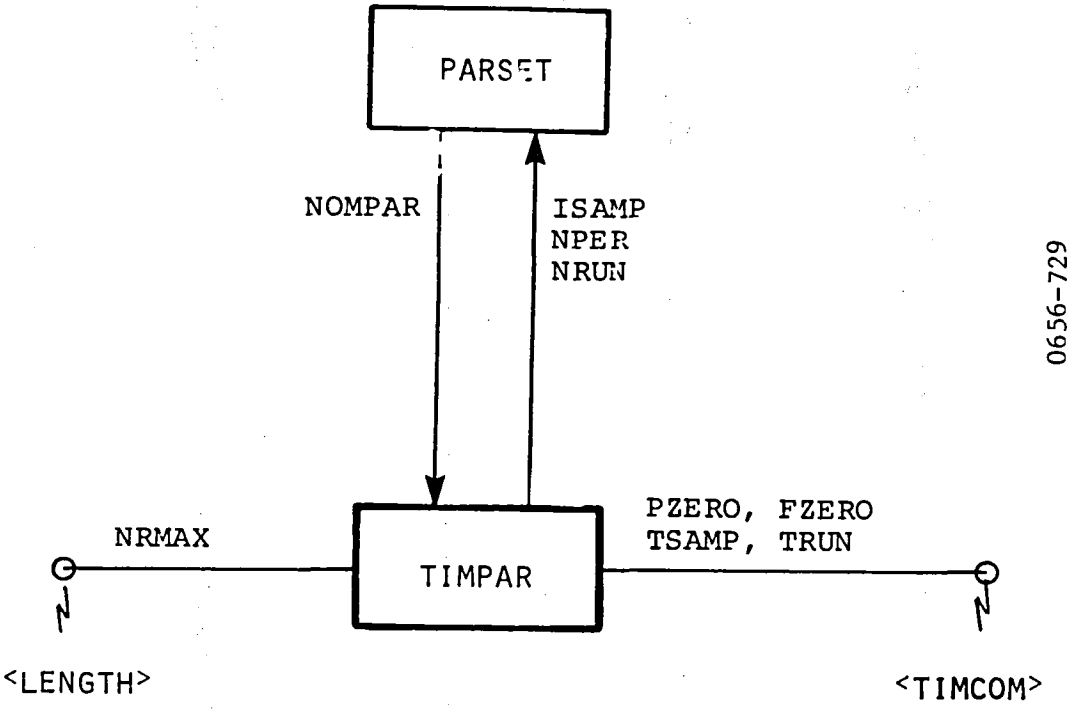
INPUTS: ARGLST: NOMPAR
<LENGTH>: NRMAX

OUTPUTS: ARGLST: ISAMP, NPER, NRUN
<TIMCOM>: PZERO, FZERO, TSAMP, TRUN

CALLER: PARSET

CALLS: ---

subroutine TIMPAR



```
      SUBROUTINE TIMPAR (NOMPAR, ISAMP, NPER, NRUN)
C
C     TIMPAR SETS UP THE TIME BASE PARAMETERS FOR VERRUN
C     TIME PARAMETERS ARE EITHER USER SPECIFIED, OR
C                                  SET TO NOMINAL VALUES
C     INPUTS: (VIA ARGLST)     NOMPAR
C             (VIA LENGTH)     NRMAX
C
C     OUTPUTS:(VIA ARGLST)     ISAMP, NPER, NRUN
C             (VIA TIMCOM)     PZERO, FZERO, TSAMP, TRUN
C
      COMMON /LENGTH/NRMAX
      COMMON /TIMCOM/PZERO, FZERO, TSAMP, TRUN
C
      LOGICAL*1 LASK, NOMPAR
C
      DATA ISMIN, ISNOM, ISMAX /1, 5, 100/
      DATA IMAX /32767/
      DATA TRNOM, TRMAX /5.2, 100.0/
C
      IF (NOMPAR .EQ. 'Y') GOTO 110
      CALL TTYOUT ('*******TIME BASE PARAMETERS******')
 100  IF (LASK ('NOMINAL TIME BASE? ') .EQ. 'Y') GOTO 110
C
      CALL TTYOUT ('$SAMPLE PERIOD (MSEC) = $')
      ISAMP = IANS (ISMIN, ISMAX)
      TSAMP = ISAMP/1000.0
      CALL TTYOUT ('RUN TIME (SEC) = $')
      TRUN = RANS (TSAMP, TRMAX)
      CALL TTYOUT (' ')
      GOTO 120
 110  ISAMP = ISNOM
      TSAMP = ISAMP/1000.0
      TRUN = TRNOM
 120  TEMP = TRUN/TSAMP + 1.5
      IF (TEMP .LE. IMAX) GOTO 125
      WRITE (5, 200) IMAX
 200  FORMAT (' SAMPLE COUNT EXCEEDS INTEGER LIMIT OF ',I6
     1 '; TRY AGAIN')
      GOTO 126
 125  NRUN = TEMP
      IF (NRUN .LE. NRMAX) GOTO 130
      WRITE (5, 201) NRUN, NRMAX
 201  FORMAT(' SAMPLE COUNT',I6,' EXCEEDS FRAME LIMIT OF',I6
     1 '; TRY AGAIN')
 126  TNEED = (NRMAX - 1) * TSAMP
      WRITE (5, 202) ISAMP, TNEED
 202  FORMAT (' WITH SAMPLE PERIOD =', I4,
     1        ' (MSEC), NEED RUN TIME .LE. ',F6.3,' (SEC)',/)
      GOTO 100
```

```
C
 130    NPER = 1
        DO 140 J = 1, 20
        NPER = 2 * NPER
 140    IF (NPER .GT. NRUN) GOTO 150
 150    NPER = NPER/2
C
        TPER = NPER * TSAMP
        TRUN = (NRUN - 1) * TSAMP
C
        PZERO = 360.0/NPER
        FZERO = 1.0/TPER
C
        IF (NOMPAR .EQ. 'Y') RETURN
        IF (LASK ('LIST TIME BASE PARAMETERS? ') .EQ. 'N') RETURN
        WRITE (5, 205) ISAMP
 205    FORMAT (' SAMPLE PERIOD =', I4, '   (MSEC)')
        WRITE (5, 206) TRUN, NRUN
 206    FORMAT (' RUN LENGTH =', F10.2, ' (SEC) WITH', I5, ' SAMPLES')
        WRITE (5, 207) TPER, NPER
 207    FORMAT (' SOS PERIOD =', F10.2, ' (SEC) WITH', I5, ' SAMPLES')
        WRITE(5, 208) FZERO,PZERO
 208    FORMAT (' BASE FREQ = ', F10.2, ' (HZ), BASE PHASE =',
     1           F10.2,' (DEG)',/)
        IF (LASK ('OK? ') .EQ. 'N') GOTO 100
        RETURN
        END
```

subroutine SOSPAR

FUNCTION: Defines the SOS parameters during interactive user setup

OPERATION: SOSPAR specifies the following parameter sets:

a. the number of sinusoidal components N_c =NCOMP through a call to SOSNCP

b. the SOS harmonic indices h_j =HARM(J) through a call to SOSHMC

c. the SOS amplitudes a_j =AMP(J) through a call to SOSAMP

d. the SOS phase multipliers p_j =PMUL(J) through a call to SOSPHS

SOSPAR is called with the argument ICHNGE to indicate whether any parameter changes are to be made, and (if changes are to be made) the argument NOMPAR to indicate whether or not nominal parameter values are to be selected. If ICHNGE is set to 'N', phase multipliers are re-randomized, and control returns to the calling routine PARSET.

If both ICHNGE and NOMPAR are set to 'Y', all SOS parameters are (re)set to nominal values, and control returns to PARSET. If SOSPAR is called with NOMPAR='N', the user has the option to (a) request nominal values for all parameters (set NOMSOS='Y'), or (b) interactively specify values for all parameter sets (set NOMSOS='N'). If parameters are specified interactively, the user is allowed to review the parameter settings and re-specify the entire set if desired. Upon completion of this operation, control returns to PARSET.

INPUTS: ARGLST: NOMPAR, ICHNGE, IRUN, NPER
<TIMCOM>: PZERO, FZERO, TSAMP

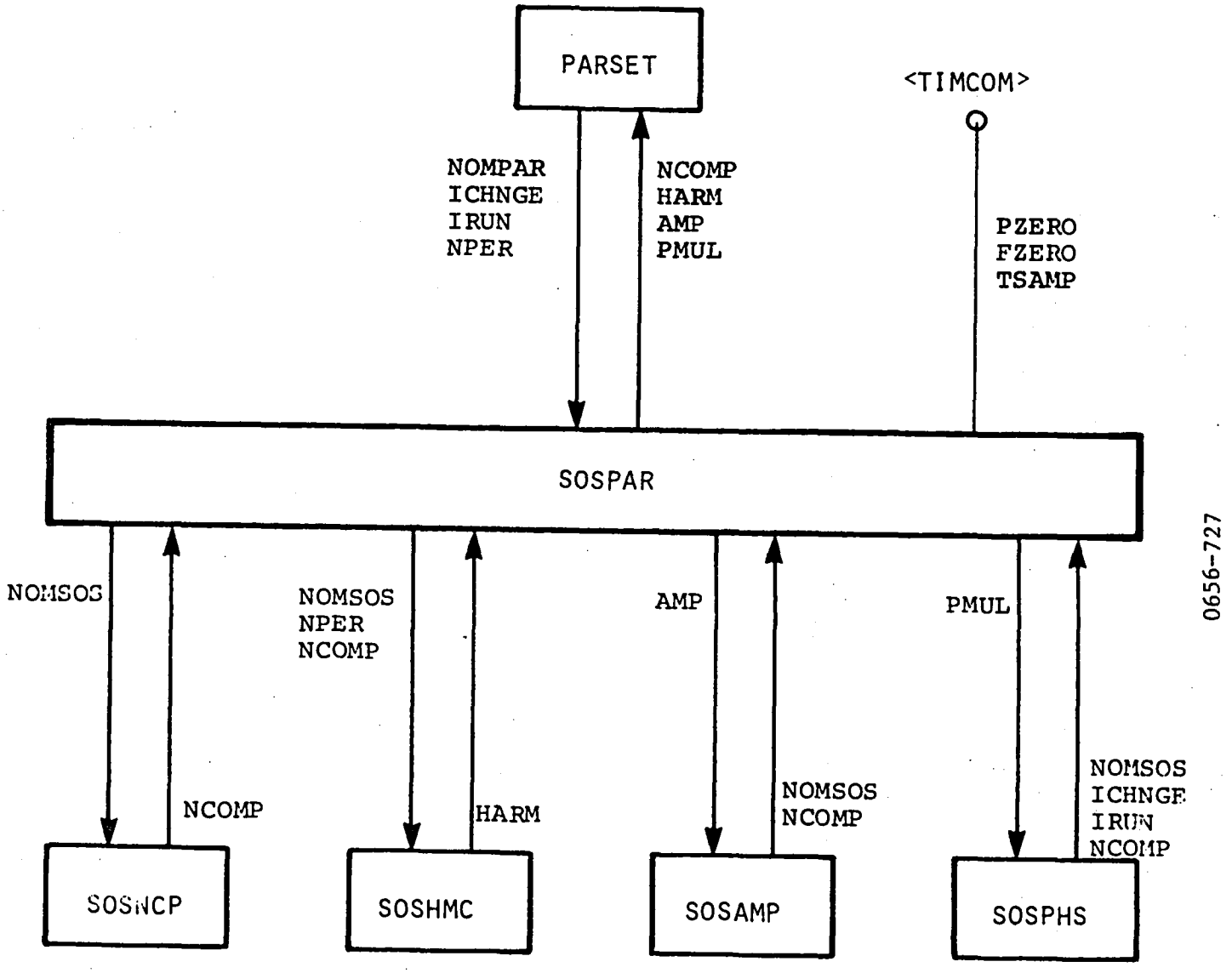
OUTPUTS: ARGLST: NCOMP, HARM, AMP, PMUL
<TIMCOM>: PZERO, FZERO, TSAMP

LOCAL: NOMSOS

CALLER: PARSET

CALLS: SOSNCP, SOSHMC, SOSAMP, SOSPHS

subroutine SOSPAR

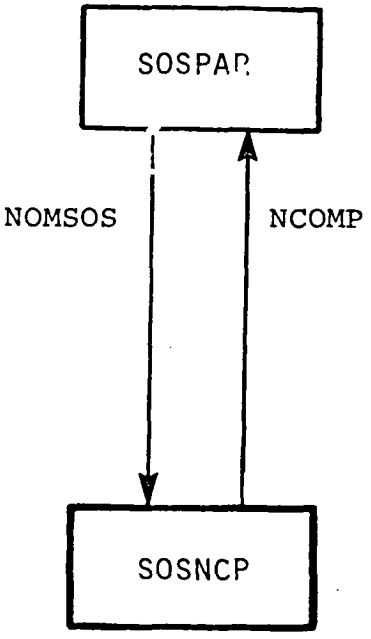


```
      SUBROUTINE SOSPAR(NOMPAR,ICHNGE,IRUN,NPER,NCOMP,HARM,AMP,
      PMUL)
C
C     SOSPAR SETS UP THE SOS PARAMETERS FOR SOSGEN
C     SOS PARAMETERS ARE EITHER USER SPECIFIED, OR
C                              SET TO NOMINAL VALUES
C
C     INPUTS: (VIA ARGLST)     NOMPAR, ICHNGE, IRUN, NPER
C     OUTPUTS:(VIA ARGLST)     NCOMP, HARM, AMP, PMUL
C
      COMMON /TIMCOM/ PZERO, FZERO, TSAMP
C
      LOGICAL*1 LASK, NOMPAR, NOMSOS, ICHNGE
      INTEGER HARM (1), PMUL (1)
      DIMENSION AMP (1)
C
      IF (ICHNGE .EQ. 'N') GOTO 300
      NOMSOS = 'Y'
      IF (NOMPAR .EQ. 'Y') GOTO 110
      CALL TTYOUT ('************SOS PARAMETERS************')
 100  NOMSOS = LASK ('NOMINAL SOS? ')
C
 110  CALL SOSNCP (NOMSOS, NCOMP)
      CALL SOSHMC (NOMSOS, NCOMP, NPER, HARM)
      CALL SOSAMP (NOMSOS, NCOMP, AMP)
 200  CALL SOSPHS (NOMSOS, ICHNGE, IRUN, NCOMP, PMUL)
      IF (NOMPAR .EQ. 'Y') RETURN
      IF (LASK ('LIST SOS PARAMETERS? ') .EQ. 'N') RETURN
      WRITE (5, 1000)
1000  FORMAT (1X,'COMP',5X,'HARM',7X,'FREQ',7X,'AMP',8X,'PHASE',/)
      WRITE (5, 1001) (J, HARM(J), FZERO * HARM(J), AMP(J),
      1 PZERO * PMUL(J), J = 1, NCOMP)
1001  FORMAT (I5, 5X, I4, 5X, F6.2, 5X, F6.2, 5X, F8.2)
      CALL TTYOUT (' ')
      IF (LASK ('OK? ') .EQ. 'N') GOTO 100
      RETURN
C
 300  CALL SOSPHS(NOMSOS, ICHNGE, IRUN, NCOMP, PMUL)
      RETURN
      END
```

subroutine SOSNCP

FUNCTION:	Specifies the number of SOS components N_c =NCOMP
OPERATION:	If the user has specified that all SOS parameters take on their nominal (stored) values (via the flag NOMSOS), NCOMP is set to its nominal value, and control is returned to SOSPAR. Otherwise, specification of NCOMP can be either by choosing a nominal (stored) value or by entering a value from the terminal. The entered value is checked against nominal (stored) limits; if exceeded, the user is prompted to reenter.
INPUTS:	ARGLST: NOMSOS
OUTPUTS:	ARGLST: NCOMP
CALLER:	SOSPAR
CALLS:	----

subroutine SOSNCP



0656-717

```
      SUBROUTINE SOSNCP (NOMSOS, NCOMP)
C
C     INPUTS: (VIA ARGLST)     NOMSOS
C     OUTPUTS:(VIA ARGLST)     NCOMP
C
      LOGICAL*1 LASK, NOMSOS
C
      DATA NCPMIN, NCPNOM, NCPMAX /1,6,15/
C
      IF (NOMSOS .EQ. 'Y') GOTO 200
      IF (LASK ('NOMINAL NUMBER OF SINES? ') .EQ. 'Y') GOTO 200
C
 100  CALL TTYOUT ('NUMBER OF SINES= $')
      NCOMP = IANS (NCPMIN, NCPMAX)
      CALL TTYOUT (' ')
      RETURN
C
 200  NCOMP = NCPNOM
      RETURN
      END
```

subroutine SOSHMC

FUNCTION: Specifies the SOS harmonics h_j =HARM(J)

OPERATION: If the user has specified that all SOS parameters take on their nominal values (via the flag NOMSOS), the desired frequencies f'_j=FRQTMP(J) are set to their nominal values, which range in 5 Hz increments from 5 to 75 Hz. Harmonic indices are computed as

HARM(J)=FRQTMP(J)/FZERO J=1, NCOMP)

where FZERO is the minimum frequency increment f_o computed in TIMPAR. The HARM(J) are rounded to the nearest integer.

Control is then returned to the calling routine, SOSPAR.

If the user has <u>not</u> specified that all SOS take on their nominal values, the user is given the option to choose the nominal frequency set. If he so chooses, then SOSHMC operates as described above. If the user does not choose this option, he is then allowed to enter the desired SOS frequencies. Each entered value is checked against nominal (calculated) limits; if exceeded, the user is prompted to reenter. Once all frequencies are entered, the harmonic indices are calculated as above.

SOSHMC then allows the user to review/change the chosen parameter set; if satisfactory, control is returned to the calling routine SOSPAR.

INPUTS: ARGLST: NOMSOS, NCOMP, NPER
<TIMCOM>: PZERO, FZERO, TSAMP

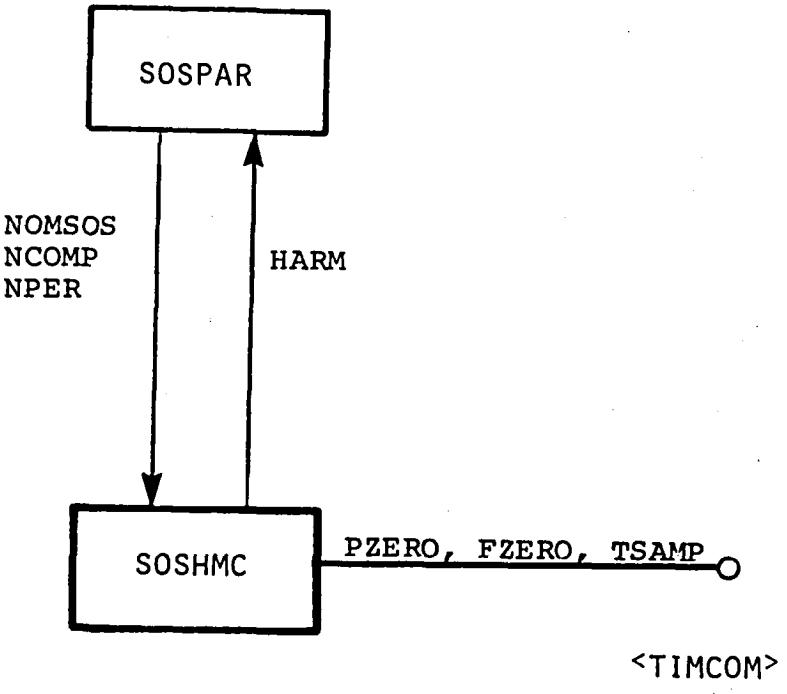
OUTPUTS: ARGLST: HARM

LOCAL: <TMPCOM>: FRQTMP

CALLER: SOSPAR

CALLS: ----

subroutine SOSHMC



```
      SUBROUTINE SOSHMC (NOMSOS, NCOMP, NPER, HARM)
C
C     INPUTS   (VIA ARGLST)     NOMSOS, NCOMP, NPER
C              (VIA TIMCOM)     FZERO, TSAMP
C
C     OUPUTS   (VIA ARGLST)     HARM
C
      COMMON /TMPCOM/ FRQTMP
      COMMON /TIMCOM/ PZERO, FZERO, TSAMP
C
      LOGICAL*1 LANS, LASK, NOMSOS
      INTEGER HARM (1)
      DIMENSION FRQNOM (15), FRQTMP (15)
C
      DATA FRQNOM /5., 10., 15., 20., 25., 30., 35., 40., 45.,
     1 50., 55., 60., 65., 70., 75./
C
      FMIN = FZERO
      FMAX = 1.0/(2.0 * TSAMP)
  100 IF (NOMSOS .EQ. 'Y') GOTO 120
      IF (LASK ('NOMINAL FREQUENCIES? ') .EQ. 'Y')  GOTO 120
C
  110 CALL TTYOUT ('ENTER DESIRED FREQUENCIES (HZ): ')
      CALL VECTIN (1, 'FREQ', NCOMP, FRQTMP, FMIN, FMAX)
      GOTO 140
C
  120 DO 130 J = 1, NCOMP
  130 FRQTMP (J) = FRQNOM (J)
  140 IERR = 0                !CHECK FOR LIMIT EXCEEDANCE
      DO 150 J = 1, NCOMP
      FTEMP = FRQTMP (J)
      IF ((FTEMP .LT. FMIN) .OR. (FTEMP .GT. FMAX)) IERR = 1
  150 CONTINUE
      IF (IERR .EQ. 0) GOTO 160    !SKIP BELOW IF WITHIN
      LIMITS CALL TTYOUT ('ONE OR MORE FREQUNCIES EXCEED
      LIMITS') WRITE (5, 151) FMIN, FMAX
  151 FORMAT (' FMIN=', F7.2, 3X, 'FMAX=', F7.2)
      CALL TTYOUT ('$ ')
      IF (LASK ('WANT FREQUENCIES LISTED? ') .EQ. 'N') GOTO 153
      WRITE (5, 152) (J, FRQTMP (J), J = 1, NCOMP)
  152 FORMAT (1X, I4, 5X, F7.2)
      CALL TTYOUT ('$ ')
  153 CALL TTYOUT ('CHANGE FREQENCIES OR TIME BASE? (F/T) $')
      IF (LANS ('F', 'T') .EQ. 'F') GOTO 110
      CALL TTYOUT ('TIME BASE CHANGE OPTION NOT IMPLEMENTED')
      GOTO 153
C
  160 DO 170 J = 1, NCOMP
  170 HARM (J) = FRQTMP (J)/FZERO + 0.5
C
```

```
      IF (NOMSOS .EQ. 'Y') RETURN
      IF (LASK ('WANT FREQUENCIES LISTED? ') .EQ. 'N') RETURN
C
      WRITE (5, 200)
 200  FORMAT (1X, 'COMP', 7X, 'HARM', 8X, 'FRQ', 8X, 'FRQ(DES)', /)
      WRITE (5, 201) (J,HARM(J),FZERO*HARM(J),FRQTMP(J), J=1,NCOMP)
 201  FORMAT (I4, 5X, I6, 6X, F7.2, 6X, F7.2)
      CALL TTYOUT (' ')
      IF (LASK ('OK? ') .EQ. 'N') GOTO 100
      RETURN
      END
```

subroutine SOSAMP

FUNCTION: Specifies the SOS amplitudes a_j =AMP(J)

OPERATION: If the user has specified that all SOS parameters take on their nominal values (via the flag NOMSOS), the normalized amplitudes, $\tilde{a}_j$, are set to their nominal values. Otherwise, the user has the option to enter the values from the TTY. Entered values are checked against nominal (stored) limits; if exceeded, the user is prompted to reenter. Next specified is the RMS SOS level, RMSLVL. This can be done either by choosing a nominal (stored) value, or by entering a value from the terminal. The entered value is checked against nominal (stored) limits; if exceeded, the user is prompted to reenter. SOSAMP then scales the normalized amplitudes, $\tilde{a}_j$, to obtain the SOS amplitudes, a_j, which yield the desired RMS level according to:

$$\text{RMSLVL} = \left[\sum_{j=1}^{N} \frac{1}{2} a_j^2 \right]^{1/2}$$

SOSAMP then allows the user to review/change the chosen parameter set; if satisfactory, control is returned to the calling routine, SOSPAR

INPUTS: ARGLST: NONSOS, NCOMP

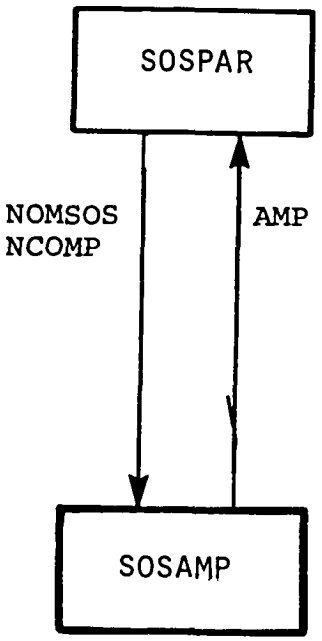
OUTPUTS: ARGLST: AMP

CALLER: SOSPAR

CALLS: ----

LOCAL: <TMPCOM>: AMPTMP

subroutine SOSAMP



0656-716

```
      SUBROUTINE SOSAMP (NOMSOS, NCOMP, AMP)
C
C     INPUTS: (VIA ARGLST)     NOMSOS, NCOMP
C     OUTPUTS:(VIA ARGLST)     AMP
C
      COMMON /TMPCOM/AMPTMP
C
      LOGICAL*1 LASK, NOMSOS
      DIMENSION AMP(1), AMPNOM(15), AMPTMP(15)
C
      DATA AMPNOM /15 * 1./
      DATA RMSMIN, RMSNOM, RMSMAX /0., 1., 5./
      DATA AMIN, AMAX /0., 100./
C
      IF (NOMSOS .EQ. 'Y') GOTO 120
 100  IF (LASK ('NOMINAL AMPLITUDES? ') .EQ. 'Y') GOTO 120
C
 110  CALL TTYOUT ('ENTER (RELATIVE) AMPLITUDES: ')
      CALL VECTIN (1, 'AMP', NCOMP, AMPTMP, AMIN, AMAX)
      GOTO 140
C
 120  DO 130 J = 1, NCOMP
 130  AMPTMP(J) = AMPNOM(J)
C
 140  RMSLVL = RMSNOM
      IF (NOMSOS .EQ. 'Y') GOTO 150
      IF (LASK ('NOMINAL RMS LEVEL? ') .EQ. 'Y') GOTO 150
      CALL TTYOUT ('RMS LEVEL (VOLT) = $')
      RMSLVL = RANS(RMSMIN, RMSMAX)
C
 150  SUMSQ = 0.0
      DO 160 J = 1, NCOMP
 160  SUMSQ = SUMSQ + AMPTMP(J) * AMPTMP(J)
C
      SCALE = RMSLVL * SQRT(2.0/SUMSQ)
C
      DO 170 J = 1, NCOMP
 170  AMP(J) = SCALE * AMPTMP(J)
C
      IF (NOMSOS .EQ. 'Y') RETURN
      CALL TTYOUT ('$ ')
      IF (LASK ('LIST AMPLITUDES? ') .EQ. 'N') RETURN
C
      WRITE (5, 200)
 200  FORMAT (1X, 'COMP', 7X, 'AMP', 7X, 'AMP (REL)', /)
      WRITE (5, 201) (J, AMP (J), AMPTMP (J), J = 1, NCOMP)
 201  FORMAT (I4, 5X, F7.2, 5X, F7.2)
      CALL TTYOUT (' ')
      IF (LASK ('OK? ') .EQ. 'N') GOTO 100
      RETURN
```

```
END
```

subroutine SOSPHS

FUNCTION: Specifies the SOS phase multipliers p_j =PMUL(J)

OPERATION: If the user has specified that all SOS parameters take on their nominal values (via the flag NOMSOS), or the program is updating automatically for a new run (indicated by the flag NOMPAR), then the desired phases ϕ'_j=PHSTMP(J) are generated via a uniform random number generator which operates over the range Ø to 360 degrees, and which is started by a nominal (stored) integer "seed", incremented by the run number. The corresponding phase multipliers are then calculated as:

PMUL(J)=PHSTMP/PZERO (j=1, NCOMP)

where PZERO is the minimum phase increment, in degrees, computed in TIMPAR. The PMUL(J) are rounded to the nearest integer.

Control is then returned to the calling routine SOSPAR.

If the user has <u>not</u> specified a nominal selection of all SOS parameter, then the user is given the option of choosing either randomized phases, or specified phases. If randomized, the user enters an integer "seed" value, and the desired phases are generated as above. If specified, the user enters the individual phases. Each entered value is checked against nominal (stored) limits; if exceeded, the user is prompted to reenter. With phases then specified, the phase multipliers PMUL(J) are calculated as above. SOSPHS then allows the user to review/change the chosen parameter set; if satisfactory, control is returned to the calling routine, SOSPAR.

INPUTS: ARGLST: NOMSOS, ICHNGE, IRUN, NCOMP
<TIMCOM>: PZERO

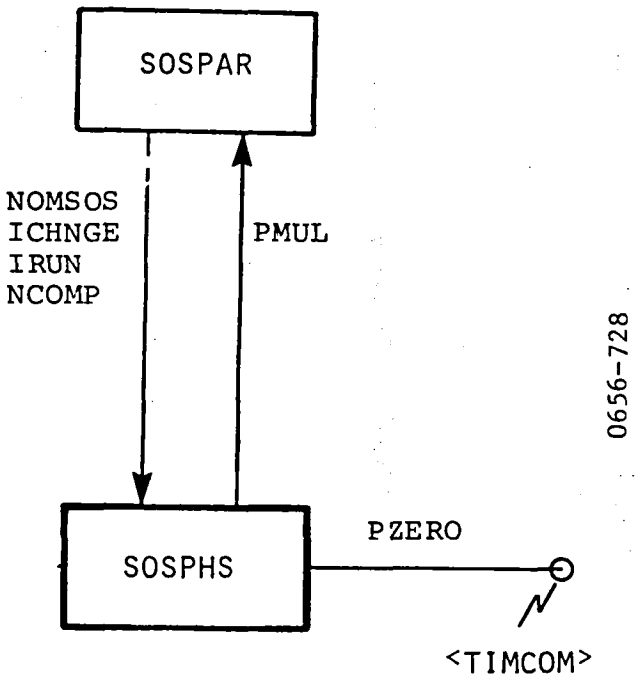
OUTPUTS: ARGLST: PMUL

LOCAL: <TMPCOM>: PHSTMP

CALLER: SOSPAR

CALLS: ----

subroutine SOSPHS



```
      SUBROUTINE SOSPHS (NOMSOS, ICHNGE, IRUN, NCOMP, PMUL)
C
C       INPUTS: (VIA ARGLST)    NOMSOS, ICHNGE
C               (VIA ARGLST)    IRUN, NCOMP
C               (VIA TIMCOM)    PZERO
C
C       OUTPUTS:(VIA ARGLST)    PMUL
C
        COMMON /TMPCOM/PHSTMP
        COMMON /TIMCOM/ PZERO
C
        LOGICAL*1 LASK, NOMSOS, ICHNGE
        INTEGER PMUL (1)
        DIMENSION PHSTMP(15)
C
        DATA PMIN, PMAX /0., 360./
        DATA IMAX,TMAX /32767,32767./
C
        IF (ICHNGE .EQ. 'N') GOTO 130
 100    IF (NOMSOS .EQ. 'Y') GOTO 130
        IF (LASK ('NOMINAL PHASES? ') .EQ. 'Y') GOTO 130
C
 110    IF (LASK ('RANDOM PHASES? ') .EQ. 'Y') GOTO 120
C
        CALL TTYOUT ('ENTER (DESIRED) PHASES (DEG): ')
        CALL VECTIN (1, 'PHASE', NCOMP, PHSTMP, PMIN, PMAX)
        GOTO 160
C
 120    CALL TTYOUT ('$RANDOM PHASE SEED (POS INT) = $')
        ISEED = IANS (0, IMAX)
        CALL TTYOUT (' ')
        GOTO 140
C
 130    ISEED = IRUN + 1                !NORMAL SEED = RUN #  +  1
 140    CALL RNSEED (0, ISEED)          !SET GENERATOR
        DO 145 I = 1, 100                !WARM UP GENERATOR
 145    CALL RNUM (ITEMP,1)
C
        DO 150 J = 1, NCOMP
        CALL RNUM (ITEMP, 1)
        TEMP = ITEMP
        TEMP = (TEMP + TMAX)/(2.*TMAX)
 150    PHSTMP (J) = PMAX * TEMP
C
 160    DO 170 J = 1, NCOMP
 170    PMUL (J) = (PHSTMP (J) / PZERO) + 0.5
C
        IF (ICHNGE .EQ. 'N') RETURN
        IF (NOMSOS .EQ. 'Y') RETURN
        IF (LASK ('LIST PHASES? ') .EQ. 'N') RETURN
```

```
C
      WRITE (5, 200)
 200  FORMAT (1X, 'COMP', 6X, 'PMUL', 8X, 'PHS', 8X, 'PHS(DES)', /)
      WRITE (5, 201) (J, PMUL(J), PZERO*PMUL(J), PHSTMP(J),
      J=1,NCOMP)
 201  FORMAT (I4, 5X, I6, 6X, F7.2, 6X, F7.2)
      CALL TTYOUT (' ')
      IF (LASK ('OK? ') .EQ. 'N') GOTO 100
      RETURN
      END
```

subroutine SOSGEN

FUNCTION: Computes, scales and stores the SOS signal time history, before the start of each run.

OPERATION: SOSGEN first sets up the basic quarter-wave sine table SINTAB, via a call to TABGEN

SOSGEN then "loops" for NRUN times, where NRUN is the number of samples in the entire run, and is set by TIMPAR. For each kth sample, SOSGEN:

a. calculates a new SOS value via a call to SOSVAL

b. scales it for later D/A conversion

c. stores it in the scaled indexed array IDNTA

INPUTS: ARGLST: NPER, NRUN, NCHAN, NCOMP, HARM, AMP, PMUL

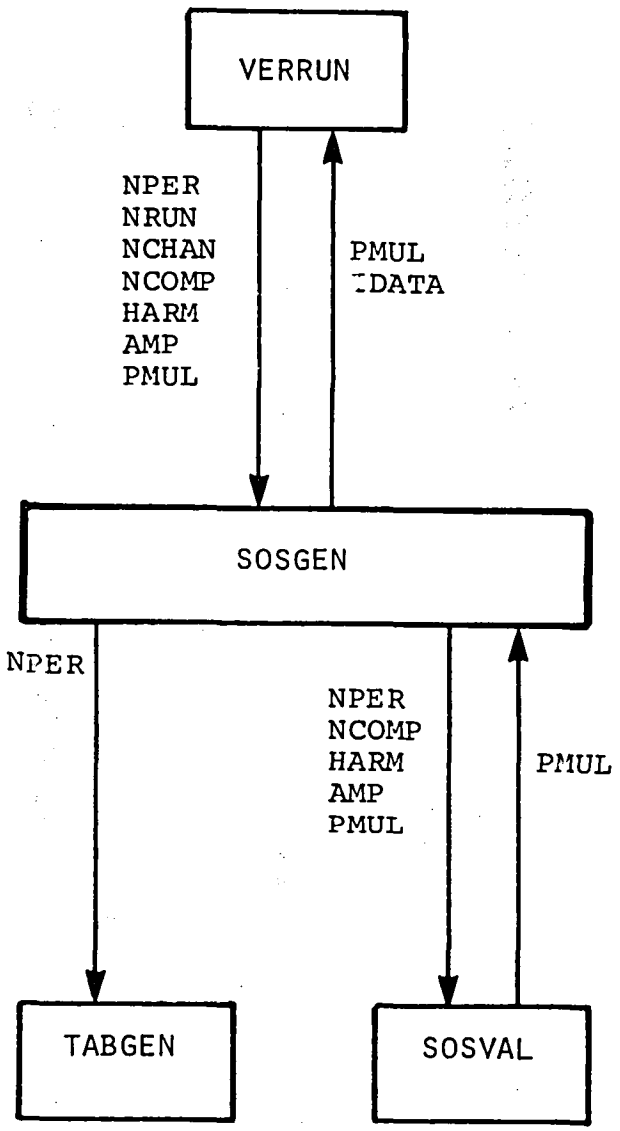
OUTPUTS: ARGLST: PMUL, IDATA

LOCAL: SOS

CALLER: VERRUN

CALLS: TABGEN, SOSVAL

subroutine SOSGEN



```
      SUBROUTINE SOSGEN(NPER,NRUN,NCHAN,NCOMP,HARM,AMP,PMUL,IDATA)
C
C     SOSGEN GENERATES SOS SIGNAL & LOADS IT INTO FIRST CHANNEL OF
      IDATA
C
C             INPUTS:  (VIA ARGLST) NPER,NRUN,NCHAN,NCOMP,HARM,AMP,
              PMUL
C             OUTPUTS: (VIA ARGLST) PMUL,IDATA
C
C     NOTES:  1)SOSGEN KEEPS HARMONIC COUNTER IN PMUL,OVERWRITING IT
C             2)SOS SCALING ASSUMES PLUS/MINUS 5 VOLT D/A
C
      INTEGER HARM(1),PMUL(1)
      DIMENSION AMP(1)
      DIMENSION IDATA (1)
C
      DATA IMAX,VMAX/2048,5./
C
      I = 1
      SCALE=IMAX/VMAX
      CALL TABGEN(NPER)
      DO 10 IFRAME = 1, NRUN
      CALL SOSVAL(NPER,NCOMP,HARM,AMP,PMUL,SOS)
      IDATA(I)=SCALE*SOS + IMAX
10    I = I + NCHAN
      RETURN
      END
```

subroutine TABGEN

FUNCTION: Generates the basic quarter-wave sine table $\tilde{S}_n$ =SINTAB used for SOS generation.

OPERATION: TABGEN first calculates the half-wave counter NHALF and quarter-wave counter NQUART, according to:

NHALF=NPER/2 ; NQUART=NHALF/2

where NPER is the SOS period set by TIMPAR. The quarter-wave table $\tilde{S}_n$ is then calculated according to:

$$\tilde{S}_n = \sin\left[2\pi\left(\frac{n}{N_o}\right)\right] \qquad (n=0,\ldots,N_Q)$$

and stored with an index shift of 1 so that SINTAB(N+1) is associated with $\tilde{S}_n$, assuring unity (and non-zero) indexing for the first array element.

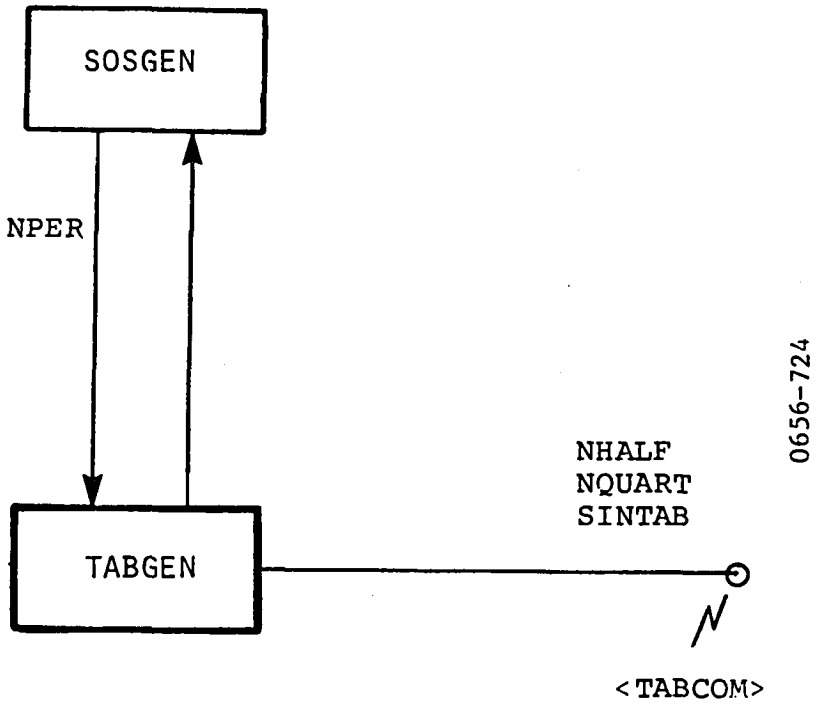
INPUTS: ARGLST: NPER

OUTPUTS: <TABCOM>: NHALF, NQUART, SINTAB

CALLER: SOSGEN

CALLS: ----

subroutine TABGEN



```
      SUBROUTINE TABGEN(NPER)
C
C     TABGEN CALCULATES HALF & QUARTER WAVE INDICES NHALF
C            & NQUART AND SETS UP QUARTER WAVE SINE TABLE
C            SINTAB
C
C             WHERE SINTAB (N+1)=SIN (2 * PI * (N/NPER))
C             FOR Ø .LE. N .LE. (NPER / 4)
CC    INPUT: (VIA ARGLST) NPER
C     OUTPUT: (VIA TABCOM) NHALF,NQUART,SINTAB
C
C     NOTE:   CURRENTLY ASSUMES NPER .LE. 2Ø48
C
      DIMENSION SINTAB(513)
C
      COMMON/TABCOM/NHALF,NQUART,SINTAB
C
      IF(NPER.LE.2Ø48)GO TO 1Ø
      CALL TTYOUT('******TABGEN: NPER TOO BIG')
      STOP
C
1Ø    IF (NPER .NE. Ø) GOTO 15
      STOP '**********TABGEN ZERO DIVIDE**********'
15    TWOPI=2.*3.14159
      NHALF=NPER/2
      NQUART=NHALF/2
      TEMP=TWOPI/NPER
C
      DO 2Ø N=Ø,NQUART
 2Ø   SINTAB(N+1)=SIN(N*TEMP)
C
      RETURN
      END
```

subroutine SOSVAL

FUNCTION: Generates a new SOS value I_k =SOS for each call and increments the phase multiplier PMUL

OPERATION: SOSVAL first sets the sine table index N equal to the phase multiplier PMUL(J). This index is then adjusted, modulo NPER, to lie between Ø and NPER-1.

SOSVAL calculates a new SOS value according to

$$I_k = \sum_{j=1}^{N_c} a_j S_{n_{j,k}} \qquad (j=1,\ldots,N_c)$$

where a_j are the SOS amplitudes AMP(J) (set by the routine SOSAMP) and S_n is the tabular sinusoidal function defined by

$$S_n = \sin\left[2\pi\left(\frac{n}{N_o}\right)\right] \qquad (n=0,\ldots,N_o)$$

This calculation of S_n is done via a direct call to SINFCN. The following operations are performed during each increment of the component index J:

a. The sine table index N is set to the corresponding phase multiplier PMUL(J).

b. This index is adjusted modulo NPER to lie between Ø and NPER-1.

c. The quantity I_k is incremented as defined above.

d. A new value for PMUL(J), to be used during the subsequent sample interval, is computed as

PMUL(J) = N+HARM(J)

where HARM(J) are the harmonic indices set by the routine SOSHMC.

INPUTS: ARGLST: NPER, NCOMP, HARM, AMP, PMUL

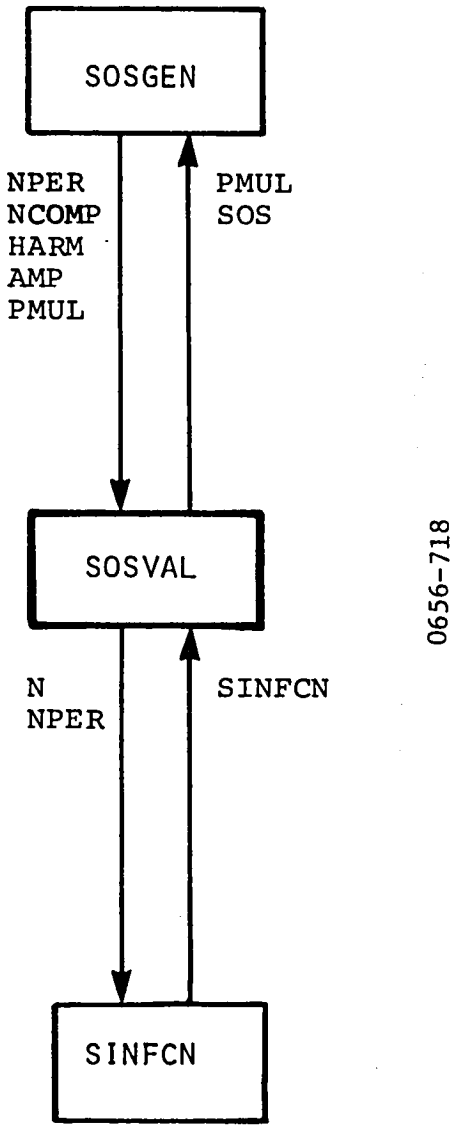
OUTPUTS: ARGLST: PMUL, SOS

LOCAL: N

CALLER: SOSGEN

CALLS: SINFCN

subroutine SOSVAL



```
      SUBROUTINE SOSVAL(NPER,NCOMP,HARM,AMP,PMUL,SOS)
C
C     CALCULATES NEW SOS VALUE FOR EACH CALL
C     AND INCREMENTS PMUL BY HARM
C
C      INPUT: (VIA ARGLST) NPER,NCOMP,HARM,AMP,PMUL
C     OUTPUT: (VIA ARGLST) PMUL,SOS
C
      INTEGER HARM(1),PMUL(1)
C
      DIMENSION AMP(1)
C
      SOS=0.
C
      DO 10 J=1,NCOMP
      N=PMUL(J)
      IF(N.GE.NPER)N=MOD(N,NPER)
      SOS=SOS + AMP(J)*SINFCN(N,NPER)
      N=N +HARM(J)
      PMUL(J)=N
10    CONTINUE
C
      RETURN
      END
```

function SINFCN

FUNCTION: Generates one value of the tabular sinusoidal function SINFCN, for each call

OPERATION: SINFCN generates the sinusoidal function S_n =SINFCN, where

$$S_n = \sin\left[2\pi\left(\frac{n}{N_o}\right)\right] \qquad (n=0,\ldots,N_o)$$

where N_o =NPER is the SOS period, set by TIMPAR. SINFCN does this by "reflecting" n into the first quadrant (module N_o), and then using the precalculated quarter-wave table $\underset{\sim}{S}_n$ =SINTAB, generated by TABGEN, to assign the appropriate sinusoidal value.

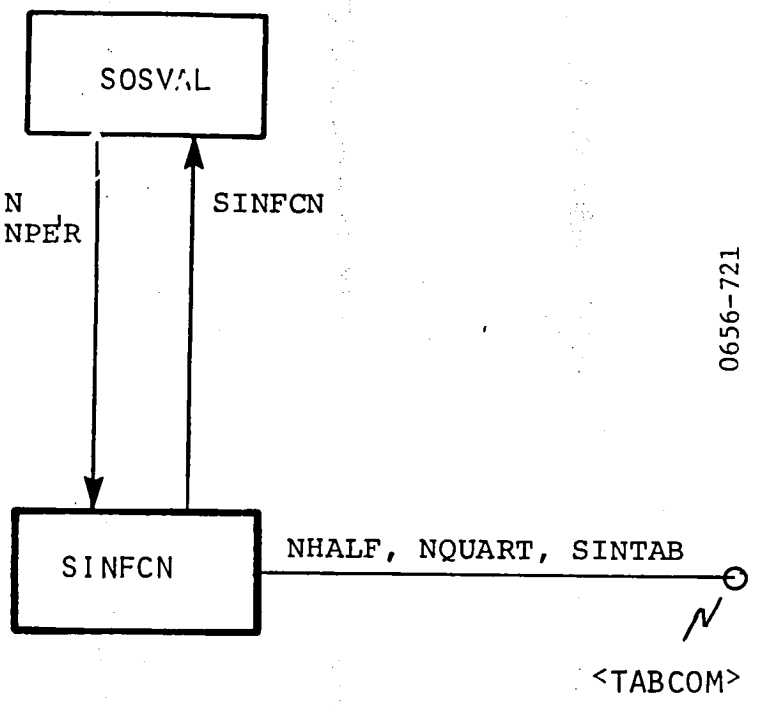
INPUTS: ARGLST: N, NPER
<TABCOM>: NHALF, NQUART, SINTAB

OUTPUTS: ARGLST: SINFCN

CALLER: SOSVAL

CALLS: ----

subroutine SINFCN



```
      FUNCTION SINFCN(N,NPER)
C
C     CALCULATES SINFCN (N) = SIN (2 * PI (N / NPER))
C             FOR Ø .LE. N .LE. (NPER-1)
C     USES QUARTER WAVE SINE TABLE SINTAB
C
C      INPUT: (VIA ARGLST) N,NPER
C             (VIA TABCOM) NHALF,NQUART,SINTAB
C
C     OUTPUT:              SINFCN
C
      DIMENSION SINTAB(1)
C
      COMMON/TABCOM/NHALF,NQUART,SINTAB
C
      NTEMP=N
      IF(NTEMP.GT. NHALF) NTEMP=NPER-NTEMP
      IF(NTEMP.GT.NQUART) NTEMP=NHALF-NTEMP
      SINFCN=SINTAB(NTEMP+1)
      IF(N.GT.NHALF) SINFCN=-SINFCN
      RETURN
      END
```

subroutine LOOP

FUNCTION: Control real-time operation of the program, including (a) maintenance of the timing loop, (b) generation of the SOS stimulus signal, and (c) sampling and storing of data.

OPERATION: LOOP selects a clock rate of 100 kHz by setting the variable IRATE to 2. The number of clock "ticks" NTIC in a sample interval is determined by multiplying the number of clock ticks per msec (in this case, 100) by the number of msec per sample interval (ISAMP). The clock is first stopped via a call to CLSTOP; D/A channels 0 and 1 are initialized to IZERO=2048, the integer corresponding to zero volts; and the clock is started with a count of NTIC via a call to CLSTRT.

LOOP "loops" for NRUN sample intervals and, for each interval, performs the following operations:

1. A call to CLWAIT checks the clock count. If the count has reached zero, a message indicating a "bad interval" is typed and the program is stopped. Otherwise, the program waits until the clock count reaches zero.

2. D/A conversions are performed by D/A units 0 and 1 which contain, respectively, a test signal ITEST which alternates between 0 and 4095, and the SOS input signal IDATA(I).

3. A/D conversions are performed via A/D devices 1 through 3, and the converted data are stored in the array IDATA.

4. The test signal is "flipped".

Upon completion of NRUN cycles, the clock is stopped, and D/A channels 0 and 1 are again initialized to 2048.

INPUTS: ARGLST: ISAMP, NRUN, NCHAN, IDATA

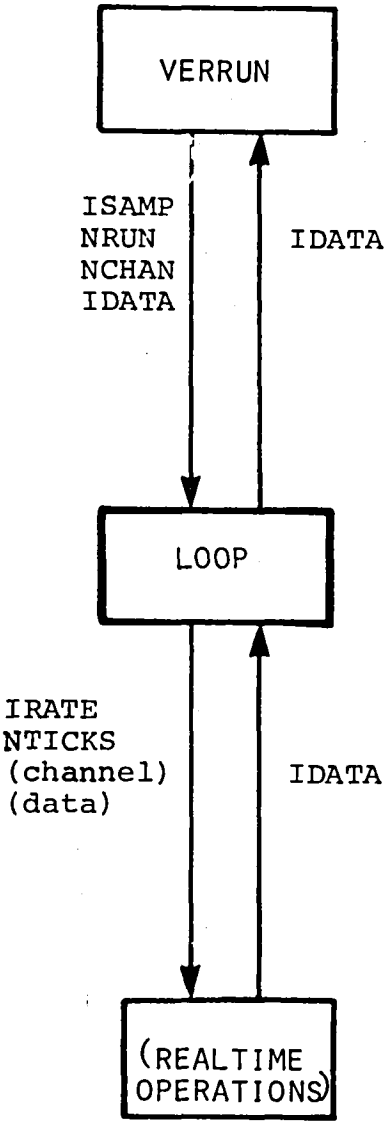
OUTPUTS: ARGLST: IDATA

LOCAL: IRATE, NTICKS

CALLER: VERRUN

CALLS: CLSTOP, CLSTRT, CLWAIT, DTOA, ATOD

subroutine LOOP



```
      SUBROUTINE LOOP (ISAMP, NRUN, NCHAN, IDATA)
C
C             INPUTS: (VIA ARGLST)     ISAMP, NRUN, NCHAN, IDATA
C             OUTPUTS:(VIA ARGLST)     IDATA
C
      LOGICAL CLWAIT
C
      DIMENSION IDATA (1)
C
      DATA IRATE /2/                                !SET CLOCK  100KHZ
      DATA IZERO /2048/
      DATA IFLIP,ITEST/1,0/                         !TEST CODE
      DATA TMAX/32767./                             !TEST CODE
C
      NTEMP = 10.**(4-IRATE) + 0.1                  !GET TICK COUNT
      NTICKS = NTEMP * ISAMP
      I = 1                                         !SET IDATA INDEX
C
      CALL CLSTOP                          !STOP CLOCK & ZERO D/A'S
      CALL DTOA (0, IZERO)
      CALL DTOA (1, IZERO)
      CALL CLSTRT (IRATE, NTICKS)      !THEN START CLOCK
C
      DO 100 IFRAME = 1, NRUN
      IF (CLWAIT()) GOTO 10
C     CALL TTYOUT ('*****LOOP: BAD TIME INTERVAL*****')
C     STOP
 10   CONTINUE
C
      CALL DTOA (0, ITEST)                          !TEST CODE
      CALL DTOA (1, IDATA (I))
      CALL ATOD (1, IDATA (I+1))
      CALL ATOD (2, IDATA (I+2))
      CALL ATOD (3, IDATA (I+3))
C
      SCALE=5./IZERO                                !TEST CODE
      XSIG=SCALE*(IDATA(I)-IZERO)                   !TEST CODE
      IDATA(I+1)=   (2.*XSIG)/SCALE + IZERO         !TEST CODE
      IDATA(I+2)=   (XSIG+2.)/SCALE + IZERO         !TEST CODE
      CALL RNUM(ITEMP,1)                            !TEST CODE
      TEMP = ITEMP                                  !TEST CODE
      TEMP = TEMP/TMAX                              !TEST CODE
      IDATA(I+3)= (XSIG+TEMP)/SCALE + IZERO         !TEST CODE
C
      I = I + NCHAN
C
      IFLIP = -IFLIP                                !TEST CODE FOR D/A 0
      IF(IFLIP .EQ. 1) ITEST = 0                    !TEST CODE
      IF(IFLIP .EQ.-1) ITEST = 4095                 !TEST CODE
C
```

```
 100    CONTINUE
C
        CALL CLSTOP                          !STOP CLOCK & ZERO D/A'S
        CALL DTOA (0, IZERO)
        CALL DTOA (1, IZERO)
        RETURN
        END
```

APPENDIX B
THE VERNAL SOFTWARE SYSTEM

B.1 Program Structure

The organization of the VERNAL software system is shown in Figure B.1. The main program VERNAL will, in general, call the eight main subprograms PART, RWHEAD, RWDATA, SIGNAL, STATS, TITLER, SPECT, DFCN. They, in turn, call the routines indicated by the line connections made to their respective blocks. All programs are written in FORTRAN. In order to minimize clutter, calls to the FORTRAN I/O library are not shown explicitly in the flow diagrams contained in this Appendix.

B.2 Software Description

Table B.1 contains brief descriptions of each of the routines contained in the VERNAL software system. The remainder of this Appendix provides documentation for each of the routines listed in the Table (and in that order), except for TITLER, RWHEAD, and RWDATA (which are common to both VERRUN and VERNAL and are described separately in Appendix C). Documentation is of the same format as that used in Appendix A (see Section A.2).

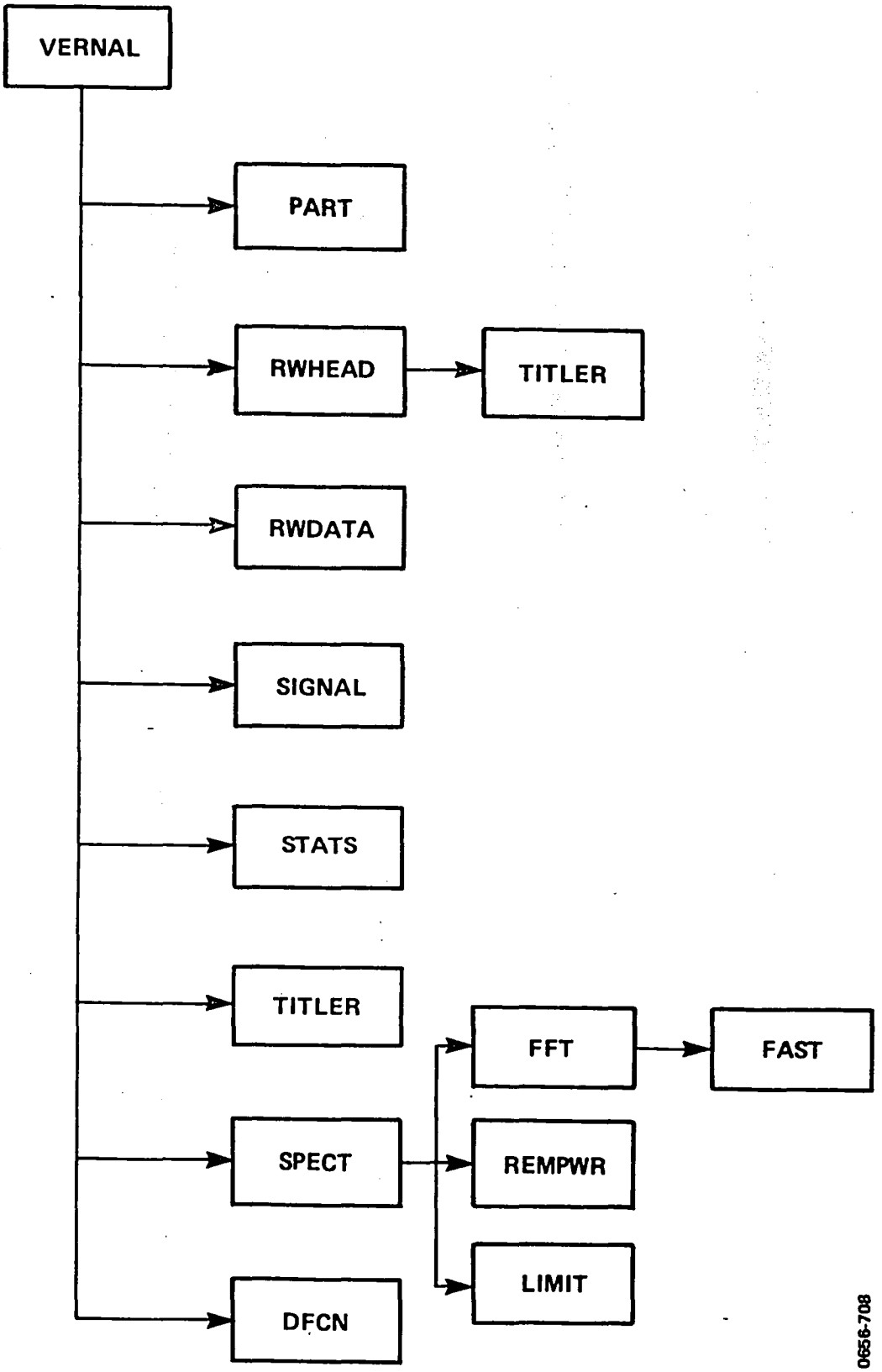


FIG. B.1 ORGANIZATION OF THE VERNAL SOFTWARE SYSTEM

TABLE B.1 FUNCTIONS OF THE VERNAL ROUTINES

VERNAL	Controls time-domain and frequency-domain analysis of VER time histories.
PART	Allows user to specify the section of code to be executed by the program VERNAL.
RWHEAD	Reads and writes header information.
TITLER	Reads and writes title information.
RWDATA	Reads and writes time history data.
SIGNAL	Extracts and scales a single channel of data for subsequent processing.
STATS	Calculates mean, standard deviation, and rms value for a time history.
SPECT	Computes frequency-response statistics for a single data channel.
FFT	Returns N-point fast-Fourier transform of a time history.
FAST	Computes discrete fast-Fourier transform.
REMPWR	Computes remnant power over a specific frequency "window".
LIMIT	Maintain variable within limits.
DFCN	Compute the describing function between two channels.

program VERNAL

FUNCTION: Controls time-domain and frequency-domain analysis of VER time histories

OPERATION: VERNAL is "menu-driven" in that the user specifies interactively, via a "part" number, the operation he wishes VERNAL to perform. Upon completion of a given operation, the user specifies the next operation to be performed. A part number of Ø displays the program options, and a part number of -1 causes VERNAL to terminate.

The program parts are:

Part 1: Read header from data file

Part 2: List header on the terminal

Part 3: Compute time-domain statistics

Part 4: Compute signal spectra

Part 5: Compute describing functions

Part 6: (not currently implemented)

Part 7: Read data from file

Part 1 must be performed first, and Part 7 must be performed before data analysis can be undertaken. Otherwise, the parts may be requested in any order.

VERNAL is initialized with the flag INFILE set to 'N'. Operation then proceeds with activation of Part 1, wherein a call to RWHEAD causes a data file to be specified by the operator, header information to be read from the requested file, and the file to be left open for possible subsequent read-in of data. The flag LIDATA is set to Ø to signify that time-history data have not been read from this file, and INFILE is set to "Y".

Execution of Part 2 writes the header information of the currently opened data file to the terminal. If the user decides he would rather analyze a different file, he again executes Part 1 to close the current file and open a new one.

Whenever Parts 3, 4, or 5, are specified, the flag LIDATA is checked to determine whether or not data have been read from the current file. If not, data are read via a call to RWDATA, the data file is closed, and LIDATA set to 1. The user then specifies the sample index NSTART at which analysis is to begin. This index is constrained to allow analysis of NPER samples. Subsequent execution of Parts 3-5 will operate on the same data base. In order to analyze a new data file, or to redefine the start point, the user must execute Part 1 followed by Part 3, 4, or 5.

Execution of Part 3 (time-domain statistics) begins with an option for the user to list on the terminal the entire data base stored in the array IDATA, or to list data from a single channel, which is stored in the temporary array XDATA. Via successive calls to SIGNAL and STATS, VERNAL computes the mean, standard deviation, and rms for each time history (NPER points beginning at NSTART) and lists the results on the terminal.

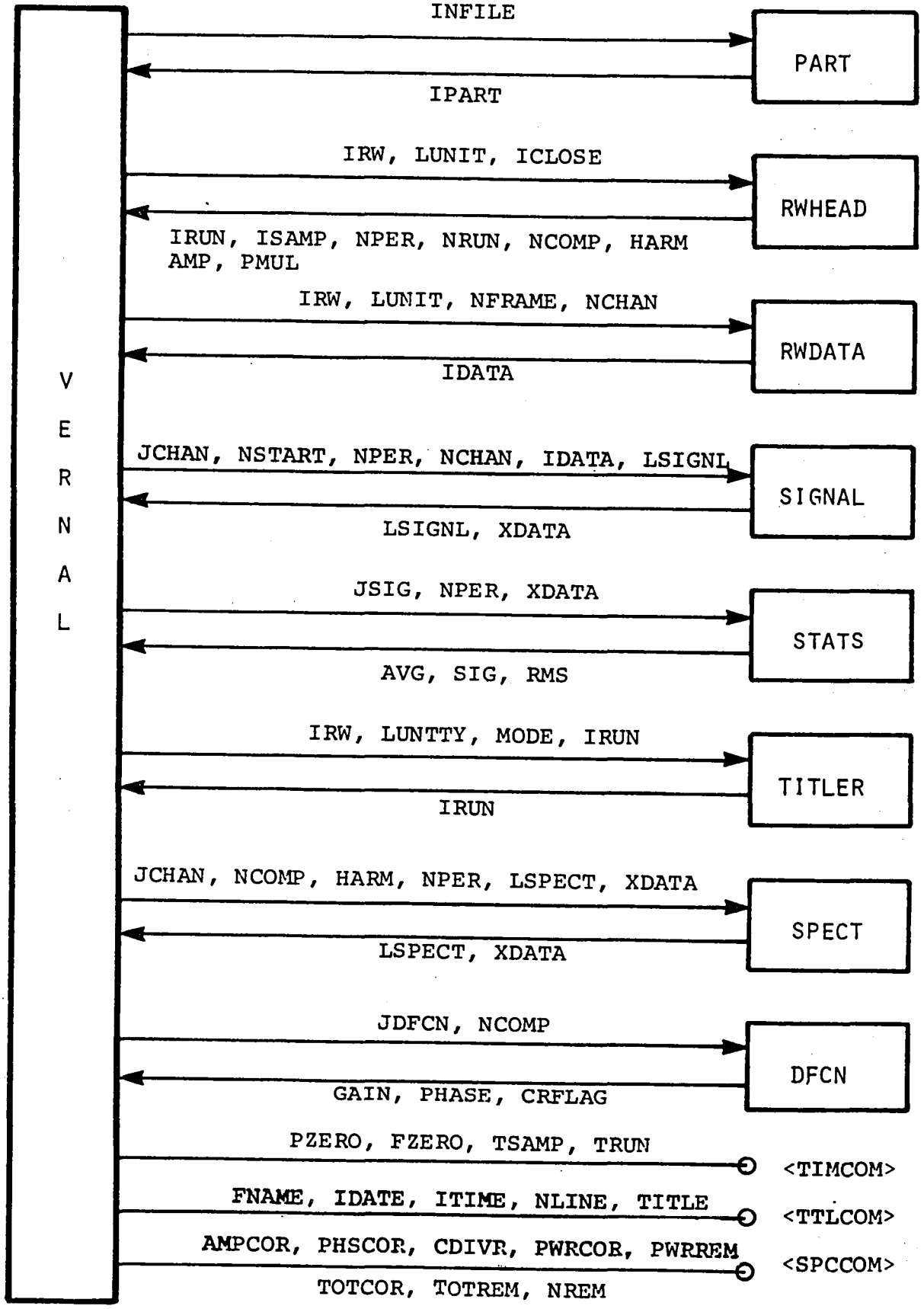
To compute a signal spectrum, the user requests Part 4 and then specifies the channel to be analyzed. Successive calls to SIGNAL and SPECT yield the desired spectrum. The user then has the option to list the spectrum (typically exercised to test the program on a short test signal). If the listing option is exercised, the user has the further option ot listing either the entire signal or only the signal components at input frequencies.

VERNAL then lists, for each input frequency: (1) correlated power per measurement bin, (2) remnant power per bin, (3) the ratio of the correlated to remnant power, (4) correlated power per rad/sec, (5) remnant power per rad/sec, (6) the ratio of correlated power to remnant power (rad/sec), and (7) the number of frequency bins included in the remnant averaging window. These spectral quantities are given in dB. The following overall statistics (in problem units) are then listed: (1) correlated power summed over all input frequencies, (2) ratio of correlated to total signal power, (3) remnant power summed over all non-input frequencies, (4) ratio of remnant to total power, and (5) total signal power (i.e., sum of all spectral computations overall frequencies). The user is then given the option to perform another spectral analysis or to specify another program part.

When execution of Part 5 (describing function analysis) is begun, VERNAL prompts the user for indices corresponding to the numerator and denominator channels. Calls to SIGNAL and SPECT provide the gain and phase information subsequently used by the routine DFCN to compute the specified describing function. Gain and phase at each frequency are printed out, and computations failing the signal/noise test within DFCN are flagged by a printout of the string (****).

CALLS: PART, RWHEAD, RWDATA, SIGNAL, TITLER, SPECT, DFCN

program VERNAL



0656-713

```
      PROGRAM VERNAL
C
C     CHANGES BY W.H. LEVISON,  12/15/83
C     1.  REVISE STATEMENT 3000.
C     2.  REVISE STATEMENT 4011 (PWR/HZ).
C     3.  ADD COMPUTATION OF PWR/HZ.
C     4.  GOTO 401 INSTEAD OF 410.
C     5.  CORRECT COMPUTATION OF TOTPWR
C
      COMMON /TIMCOM/ PZERO, FZERO, TSAMP, TRUN
      COMMON /TTLCOM/ FNAME, IDATE, ITIME, NLINE, TITLE
      COMMON /SPCCOM/ AMPCOR, PHSCOR, CDIVR, PWRCOR, PWRREM,
     1                TOTCOR, TOTREM, NREM
C
      LOGICAL*1 LANS,LASK,LSOS
      LOGICAL*1 MODE, INFILE, TITLE (200), IDATE(9), FNAME(11)
      INTEGER HARM (15), PMUL (15), HOURS, SECONS
      DIMENSION AMP(15), IDATA(5000), XDATA(1250)
      DIMENSION AVG(4),SIG(4),RMS(4)
      DIMENSION AMPCOR(15,4),PHSCOR(15,4),CDIVR(15,4),
     1          PWRCOR(15,4),PWRREM(15,4),TOTCOR(4),TOTREM(4),
     1           NREM(15)
      DIMENSION JDFCN(2),GAIN(15),PHASE(15),CRFLAG(15)
C
      DATA LUNFIL, LUNTTY /3, 5/
      DATA INFILE /'N'/
      DATA RTD /57.296/
      DATA NSTART/1/          !TEMP CODE
      DATA NCHAN/4/           !TEMP CODE
C
 10   CALL PART (INFILE, IPART)
      IF (IPART .LT. 0) STOP
C
  20  GOTO (100, 200, 300, 400, 500, 600) IPART
C
C             PART1:  READ HEADER FROM DATA FILE
C
 100  CONTINUE
      IRW = 1           !READ HEADER FROM FILE
      ICLOSE = 2        !AND LEAVE OPEN
      CALL RWHEAD (IRW,LUNFIL,ICLOSE,IRUN,ISAMP,NPER,
     1             NRUN,NCOMP,HARM,AMP,PMUL)
      INFILE = 'Y'      !INDICATE WE HAVE AN INPUT FILE
      LIDATA = 0        !          IDATA NOT LOADED
      LSIGNL = 0        !          XDATA NOT COMPUTED
      LSTATS = 0        !          STATS NOT COMPUTED
      LSPECT = 0        !          SPECTRA NOT COMPUTED
      GOTO 10
C
C             PART2:  LIST HEADER
```

```
C
 200    CONTINUE
        IRW = 2          !WRITE HEADER TO TTY
        CALL RWHEAD (IRW,LUNTTY,ICLOSE,IRUN,ISAMP,NPER,
     1                  NRUN,NCOMP,HARM,AMP,PMUL)
        GOTO 10
C
C               PART3:  COMPUTE SIGNAL STATISTICS
C
 300    CONTINUE
        IF (LIDATA .EQ. 0) GOTO 700
C
C**********************START TEST CODE**************************
C
        CALL TTYOUT(' ')
        IF(LASK('**TEST CODE: WANT IDATA LISTED?').EQ.'N')GOTO 301
        IRW=2            !WRITE DATA ONTO TTY
        CALL RWDATA (IRW,LUNTTY,NRUN,NCHAN,IDATA)
C
 301    CALL TTYOUT(' ')
        IF(LASK('**TEST CODE: WANT XDATA LISTED? ').EQ.'N')GOTO 302
        CALL TTYOUT('ENTER JCHAN $')
        JCHAN=IANS(1,NCHAN)
        CALL SIGNAL (JCHAN,NSTART,NPER,NCHAN,IDATA,LSIGNL,XDATA)
        WRITE(5,1010)(I,XDATA(I),I=1,NRUN)
1010    FORMAT(I5,1PE12.4)
 302    CONTINUE
C
C************************END TEST CODE**************************
C
        IF (LSTATS .EQ. 1) GOTO 320
C
        CALL TTYOUT('DOING STATS NOW...')
        DO 310 JCHAN = 1, NCHAN
        CALL SIGNAL (JCHAN,NSTART,NPER,NCHAN,IDATA,LSIGNL,XDATA)
        CALL STATS (JCHAN,NPER,XDATA,AVG,SIG,RMS)
 310    CONTINUE
        LSTATS = 1       !INDICATE STATS COMPUTED
C
 320    IRW = 2          !WRITE TITLE ONTO TTY
        MODE = 'S'       !BUT SUPPRESS COMMENTS
        CALL TITLER (IRW, LUNTTY, MODE, IRUN)
        WRITE (5,3000)
 3000   FORMAT(//,5X,'CHAN',8X,'AVG',10X,'S.D.',10X,'RMS')
        WRITE (5,3010) (J,AVG(J),SIG(J),RMS(J),J=1,NCHAN)
 3010   FORMAT(2X,I5,4X,F10.3,3X,F10.3,4X,F10.3)
        WRITE (5,3020)
 3020   FORMAT(//)
        GOTO 10
C
```

```
C               PART4:  COMPUTE SIGNAL SPECTRA
C
 400    CONTINUE
        BINLOG=10.0*ALOG10(FZERO)
        IF (LIDATA .EQ. 0) GOTO 700
C
 401    CALL TTYOUT ('SPECTRUM FOR CHANNEL #: $')
        JCHAN = IANS (1,NCHAN)
C
        CALL SIGNAL (JCHAN,NSTART,NPER,NCHAN,IDATA,LSIGNL,XDATA)
        CALL SPECT  (JCHAN,NCOMP,HARM,NPER,LSPECT,XDATA)
C
C**********START TEST CODE*****************************
C
        IF(LASK('**TEST CODE: WANT SPECTRUM LISTOUT? ').EQ.'N')
        GOTO 405
        LSOS = LASK('**TEST CODE: ALL FREQS? ')
C
        NHALF = NPER/2
        DO 404 K=0,NHALF
        IF (LSOS .EQ. 'Y') GOTO 403
        DO 402 L = 1,NCOMP
 402    IF (K .EQ. HARM(L)) GOTO 403
        GOTO 404
C
 403    INDEX = 2*K + 1
        WRITE(5,4000) K,XDATA(INDEX),RTD*XDATA(INDEX+1)
 4000   FORMAT(I5,2F10.3)
 404    CONTINUE
 405    CONTINUE
C
C**********END TEST CODE*******************************
C
        CALL TTYOUT(' ')
        IRW = 2                                  !WRITE TITLE ONTO TTY
        MODE = 'S'                               !BUT SUPPRESS COMMENTS
        CALL TITLER (IRW,LUNTTY,MODE,IRUN)
C
        WRITE (5,4010) JCHAN
 4010   FORMAT (/,32X, 'SPECTRUM FOR CHANNEL # ', I2, /)
        WRITE (5,4011)
 4011   FORMAT (27X, 'PWR/BIN', 18X, 'PWR/HZ')
        WRITE (5,4012)
 4012   FORMAT (/,1X,'COMP   FREQ *    COR      REM      C/R',
     1                          '  *    COR      REM      C/R',
     1                            '  * NREM')
        WRITE (5,4013)
 4013   FORMAT(13X,'*',27X,'*',27X,'*')
        A0=0
        DO 420 J = 1,NCOMP
```

```
        AFREQ=FLOAT(HARM(J))
        IF(J.EQ.NCOMP) GO TO 410
        A1=SQRT(AFREQ*FLOAT(HARM(J+1)))
        GO TO 415
410     A1=(AFREQ**2)/A0
415     WIDTH=10.0*ALOG10(A1-A0)
        A0=A1
        AFREQ=AFREQ*FZERO
        TEMP1=PWRCOR(J,JCHAN)-WIDTH-BINLOG
        TEMP2=PWRREM(J,JCHAN)-BINLOG
        TEMP3=TEMP1-TEMP2
        WRITE (5,4020)  J,AFREQ,PWRCOR(J,JCHAN),PWRREM(J,JCHAN),
     1                  CDIVR(J,JCHAN),TEMP1,TEMP2,TEMP3,NREM(J)
 420    CONTINUE
 4020   FORMAT (I4,F8.2,' *',F8.2,2F9.2,' *',F8.2,2F9.2,' *',I4)
C
        TOTPWR=TOTCOR(JCHAN)+TOTREM(JCHAN)
        WRITE (5,4031) TOTCOR(JCHAN), TOTCOR(JCHAN)/TOTPWR
        WRITE (5,4032) TOTREM(JCHAN), TOTREM(JCHAN)/TOTPWR
        WRITE (5,4033) TOTPWR
 4031   FORMAT(//,5X, 'COR PWR = ', F9.2, 5X, 'COR/TOT PWR = ', F9.2)
 4032   FORMAT(   5X, 'REM PWR=', F9.2, 5X, 'REM/TOT PWR= ', F9.2,/)
 4033   FORMAT(   5X, 'TOT PWR = ', F9.2,//)
C
        CALL TTYOUT(' ')
        IF (LASK ('ANOTHER SPECTRUM? ') .EQ. 'Y') GOTO 401
C
        GOTO 10
C
C               PART5:  COMPUTE TRANSFER FUNCTIONS
C
 500    CONTINUE
        IF (LIDATA .EQ. 0) GOTO 700
C
 510    CALL TTYOUT ('CHANNEL # FOR DFCN   NUM: $')
        JDFCN(1) = IANS(1,NCHAN)
        CALL TTYOUT ('CHANNEL # FOR DFCN DENOM: $')
        JDFCN(2) = IANS(1,NCHAN)
C
        DO 520 I = 1,2                      !GET SPECTRA FOR NUM & DENOM
        JCHAN = JDFCN(I)
        CALL SIGNAL (JCHAN,NSTART,NPER,NCHAN,IDATA,LSIGNL,XDATA)
        CALL SPECT  (JCHAN,NCOMP,HARM,NPER,LSPECT,XDATA)
 520    CONTINUE
C
        CALL DFCN (JDFCN, NCOMP, GAIN, PHASE, CRFLAG)
C
        IRW = 2                             !WRITE TITLE ONTO TTY
        MODE = 'S'                          !BUT SUPPRESS COMMENTS
        CALL TITLER (IRW, LUNTTY, MODE, IRUN)
```

```
      WRITE (5, 5010) JDFCN
 5010 FORMAT (/,25X,'DFCN FOR (CHAN ',I1,')/(CHAN ',I1,')',//)
      WRITE (5,5015)
 5015 FORMAT (20X,'COMP     FREQ      GAIN      PHASE')
      DO 530 J = 1, NCOMP
 530  WRITE (5, 5020) J, FZERO*HARM(J), GAIN(J), RTD*PHASE(J),
     1                CRFLAG(J)
 5020 FORMAT (20X, I4, F9.2, F10.1, F10.1, A8)
      CALL TTYOUT(' ')
C
      IF (LASK ('ANOTHER DFCN? ') .EQ. 'Y') GOTO 510
C
      GOTO 10
C
C             PART6:  SUMMARY
C
 600  CONTINUE
      IF (LIDATA .EQ. 0) GOTO 700
      GOTO 10
C
C             PART7:  READ DATA FROM FILE; SET START POINT
C
 700  CONTINUE
      CALL TTYOUT ('READING IN DATA NOW....')
      IRW = 1          !READ DATA FROM FILE & CLOSE IT
      CALL RWDATA (IRW,LUNFIL,NRUN,NCHAN,IDATA)
      LIDATA = 1       !INDICATE IDATA IS LOADED
C
      CALL TTYOUT ('SCORING STARTS AT POINT $')  !SET UP START
      POINT
      WRITE (5,105) NSTART
 105  FORMAT (1H+,I5$)
      IF (LASK('  WANT TO CHANGE? ') .EQ. 'N') GOTO 20
      NTEMP = NRUN - NPER + 1
      CALL TTYOUT ('ENTER START POINT IN RANGE 1 THRU $')
      WRITE (5,105) NTEMP
      CALL TTYOUT ('$:$')
      NSTART = IANS(1,NTEMP)
      GOTO 20
C
      END
```

subroutine PART

FUNCTION: Allows user to specify the section of code to be executed by the program VERNAL

OPERATION: PART first prompts the user to specify a program part within the range -1 to 6. A value of -1 causes the routine to return to the calling program; a value of zero causes a printout of the part definitions, followed by another prompt for a program part.

If the user specifies a number between 1 and 6, PART checks the flag INFILE to determine whether or not a data file has been specified for input. If such an input has been specified, PART returns with the part number specified by the user; otherwise, the part number is set to 1, and the user is informed of the need to specify a data file.

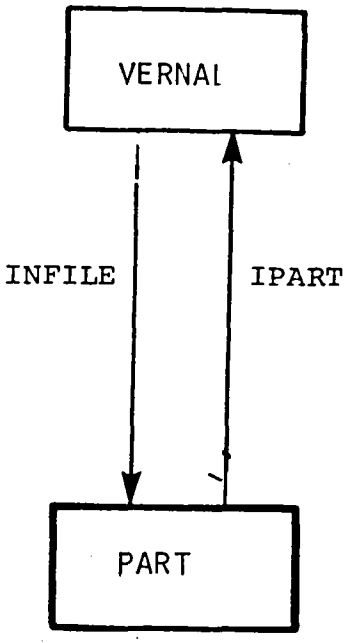
INPUTS: ARGLST: INFILE

OUTPUTS: ARGLST: IPART

CALLER: VERNAL

CALLS: ----

subroutine PART



0656-712

subroutine SIGNAL

FUNCTION: Extracts and scales a single channel of data for subsequent processing

OPERATION: On the first call to SIGNAL, the (uniform) scale factors S_j = SCALE(J) are defined:

$$S_j = VMAX/IDATA$$

where VMAX is the maximum A/D and D/A voltage (defined as 5 volts), and IMAX is one half the maximum peak-to-peak variations allowed in the stored integer data (defined as 2048). This operation is bypassed on subsequent calls to SIGNAL.

Data for the signal channel JCHAN, starting at time frame NSTART, are extracted from the interleaved data vector d_i =IDATA(I) and stored in x_k =XDATA(K) for further processing. The following conversion is performed for each x_k :

$$x_k = S_j \cdot (d_i - d_o)$$

where d_o = IZERO (defined as 2048) is the zero offset of the data stored in IDATA.

INPUTS: ARGLST: JCHAN, NSTART, NPER, NCHAN, IDATA, LSIGNL

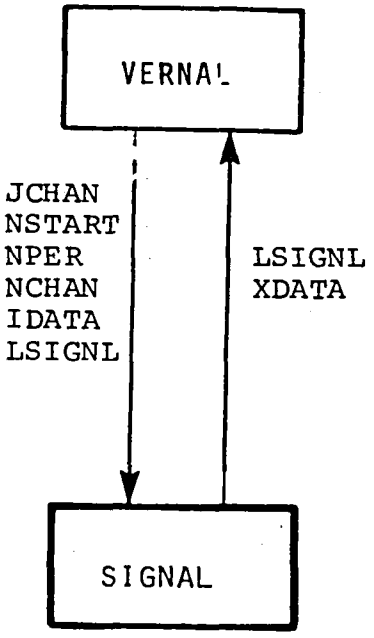
OUTPUTS: ARGLIST: LSIGNL, XDATA

LOCAL: SCALE, IZERO

CALLER: VERNAL

CALLS: ----

subroutine SIGNAL



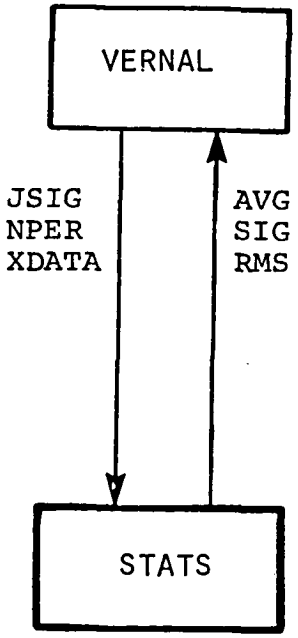
0656-715

```
      SUBROUTINE SIGNAL (JCHAN,NSTART,NPER,NCHAN,IDATA,LSIGNL
      XDATA)
C
C     SIGNAL LOADS XDATA WITH DATA CHANNEL JCHAN, TAKEN
C     FROM IDATA, STARTING AT POINT NSTART IN THE IDATA ARRAY
C
C             INPUTS: (VIA ARGLST)    JCHAN,NSTART,NPER,NCHAN,IDATA
C                     (VIA ARGLST)    LSIGNL (0=INIT PASS, 1=OTHERS)
C             OUTPUTS:(VIA ARGLST)    LSIGNL,XDATA
C
      DIMENSION IDATA(1),XDATA(1),SCALE(4)
C
      DATA IMAX,VMAX/2048,5./
      DATA IZERO/2048/
C
      IF (LSIGNL .EQ. 1) GOTO 20
C
      DO 10 J = 1,NCHAN                 !INIT SCALES  FIRST TIME THRU
 10   SCALE(J) = VMAX/IMAX
      LSIGNL = 1                        !& INDICATE DONE
C
 20   SFACT = SCALE(JCHAN)
      I = (NSTART-1)*NCHAN + JCHAN
C
      DO 30 K = 1,NPER
      XDATA(K) = SFACT*(IDATA(I)-IZERO)
 30   I = I + NCHAN
C
      RETURN
      END
```

subroutine STATS

FUNCTION:	Calculates mean, standard deviation, and rms value for a time history
OPERATION:	Statistics are computed for the data vector XDATA, of length NPER, defined in a preceding call to SIGNAL. Mean, standard deviation, and rms are stored in the vectors AVG(J), SIG(J), and RMS(J), respectively.
INPUTS:	ARGLST: JSIG, NPER, XDATA
OUTPUTS:	ARGLST: AVG, SIG, RMS
CALLER:	VERNAL
CALLS:	----

subroutine STATS



0656-719

```
      SUBROUTINE STATS (JSIG, NPER, XDATA, AVG, SIG, RMS)
C
C     STATS CALCULATES TIME-AVERAGED MEAN, SD, & RMS VALUES
C     FOR THE DATA STRING CONTAINED IN XDATA
C     CALCULATIONS ARE DONE FOR THE FIRST NPER POINTS IN XDATA
C     RESULTS ARE LOADED IN JSIG COMPONENTS OF AVG,SIG,&RMS
C
C             INPUTS: (VIA ARGLST)     JSIG,NPER,XDATA
C             OUTPUTS:(VIA ARGLST)     AVG,SIG,RMS
C
      DIMENSION XDATA(1), AVG(1), SIG(1), RMS(1)
C
      SUM = 0.
      SUMSQ = 0.
C
      DO 10 I =1, NPER
      TEMP = XDATA(I)
      SUM = SUM + TEMP
      SUMSQ = SUMSQ + TEMP**2
 10   CONTINUE
C
      AVG(JSIG) = SUM/NPER
      RMS(JSIG) = SQRT(SUMSQ/NPER)
      SIG(JSIG) = SQRT (ABS (RMS(JSIG)**2 - AVG(JSIG)**2))
      RETURN
      END
```

subroutine SPECT

FUNCTION: Computes frequency-response statistics for a single data channel

OPERATION: SPECT computes the following statistics for the Fourier-transformed data contained in the array XDATA:

a. Amplitude and phase shift for each SOS frequency index defined by HARM(J).

b. The input-correlated power at each SOS frequency, the average remnant power in the vicinity of each such frequency, and the ratio of correlated to remnant power.

c. Total power, total correlated power, and total remnant power contained in the signal, plus the ratios of correlated and remnant power to total power.

When first called by VERNAL, certain constants are computed, and the flag LSPECT is set to unity so that these computations are bypassed on subsequent calls. SPECT then calls the routine FFT to compute the discrete fast Fourier transform of the time-history data contained in the array XDATA. The results of this transformation are returned in the array XDATA as alternate estimates of magnitude and phase. The following computations are then performed:

$$a_j = 20.\mathrm{LOG}(x_k)$$

$$\phi_j = x_{k+1}$$

$$P_j = 10.\mathrm{LOG}(x_k^2/2)$$

where a_j = AMPCOR(J) is the amplitude, in dB, of the

signal at the jth SOS harmonic index, ϕ_j = PHSCOR(J) is the phase shift at that frequency, and P_j = PWRCOR(J) is the signal power in dB. x_k represents the values of XDATA at index "k", where, because of the interleaving of magnitude and phase results, $k=2h_j+1$. The variable SUMCOR is incremented by the jth correlated power computation (in experimental units, not dB) in order to determine the total amount of input-correlated power contained in the signal.

To compute the remnant power PWRREM(J) for the jth SOS index, the indices KLOW and KHIGH (for array XDATA) are computed to be approximately 1/8 octave below and above the jth SOS harmonic index. The routine REMPOW is then called to yield the accumulated remnant SUMREM and to determine the number of frequency "bins" NREM(J) utilized in the (local) remnant computation. The remnant estimate PWRREM(J) is determined by dividing SUMREM by NCOUNT and converting to dB. The ratio of correlated remnant power CDIVR(J), in dB, is computed by subtracting the remnant power (in dB) from the correlated power (in dB). Correlated and remnant powers are limited to a minimum of -99.99 dB, and a call to LIMIT maintains the signal/noise ratio between -99.99 and +99.99 dB.

After completing the above calculations for each SOS index, SPECT computes the total remnant power via a call to REMPOW, with indices KLOW and KHIGH set to include the entire spectrum. Total correlated and remnant power for the signal are stored as TOTCOR and TOTREM, respectively.

INPUTS: ARGLST: JCHAN, NCOMP, HARM, NPER, LSPECT, XDATA

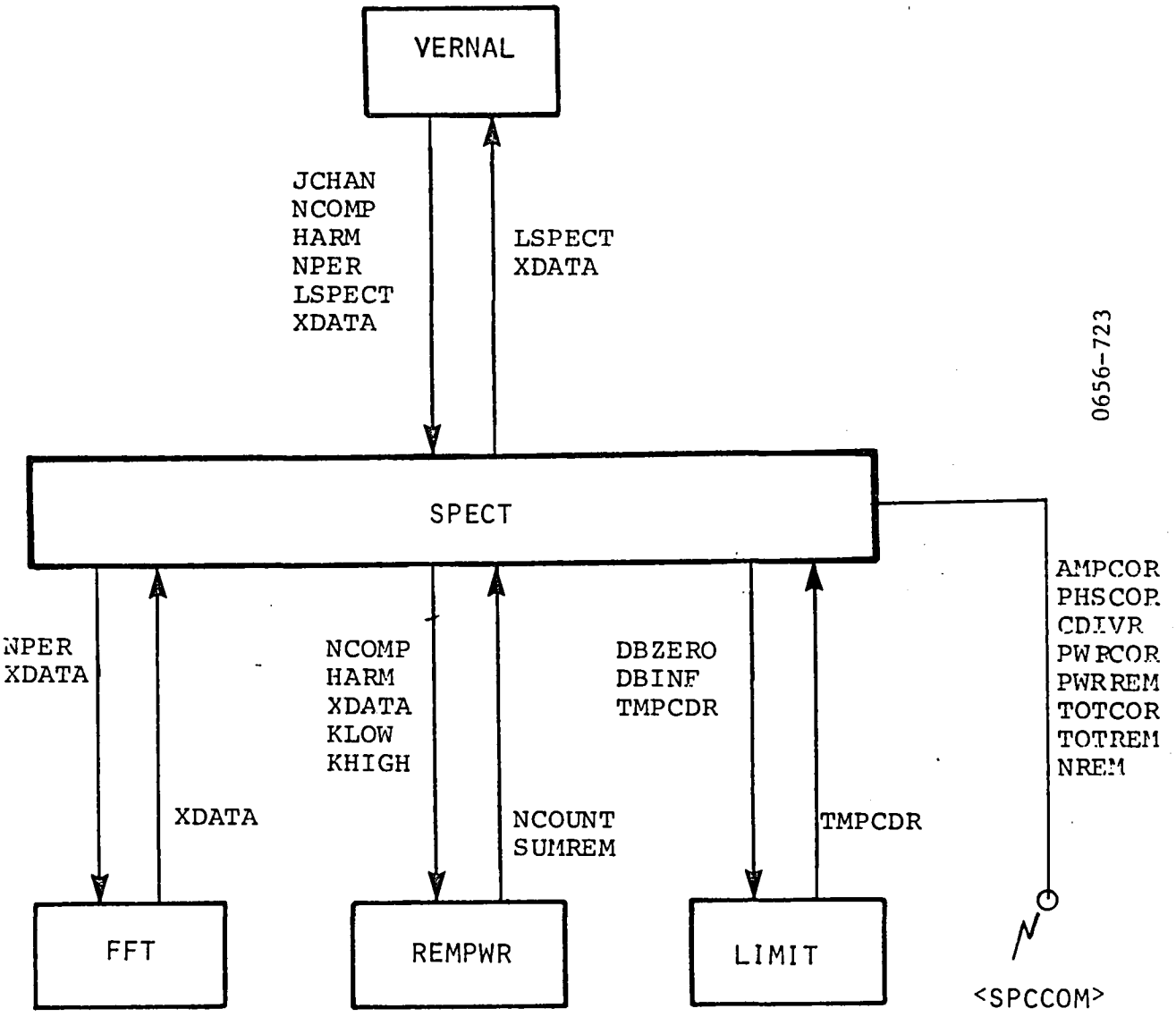
OUTPUTS: ARGLST: LSPECT, XDATA
<SPCCOM>: AMPCOR, PHSCOR, CDIVR, PWRCOR, PWRREM, TOTCOR, TOTREM, NREM

LOCAL: KLOW, KHIGH, NCOUNT

CALLER: VERNAL

CALLS: FFT, REMPWR, LIMIT

subroutine SPECT



```
      SUBROUTINE SPECT (JCHAN, NCOMP, HARM, NPER, LSPECT, XDATA)
C
C     FOR THE SIGNAL IN XDATA, SPECT CALCULATES, AT EACH SOS FREQ:
C           1)THE CORRELATED AMP AND PHS
C           2)THE CORRELATED & REMNANT POWER (PER MSMT BIN)
C           3)THE COR-TO-REM POWER RATIO     (PER MSMT BIN)
C     SPECT ALSO CALCULATES THE TOTAL CORRELATED AND REMNANT POWER
C
C           INPUTS: (VIA ARGLST)    JCHAN
C                   (    "     )    NCOMP,HARM,NPER
C                   (    "     )    LSPECT (0=INIT PASS, 1=OTHERS)
C                   (    "     )    XDATA
C           OUTPUTS:(VIA ARGLST)    LSPECT,XDATA
C                   (VIA SCRCOM)    AMPCOR,PHSCOR,CDIVR
C                   (VIA SCRCOM)    PWRCOR,PWRREM,TOTCOR,TOTREM,
C                                     NREM
C
      COMMON /SPCCOM/ AMPCOR,PHSCOR,CDIVR,PWRCOR,PWRREM,
     1                TOTCOR,TOTREM,NREM
C
      INTEGER HARM(1)
      DIMENSION XDATA(1)
      DIMENSION AMPCOR(15,4),PHSCOR(15,4),CDIVR(15,4),
     1          PWRCOR(15,4),PWRREM(15,4),TOTCOR(4),TOTREM(4),
     1           NREM(15)
C
      DATA HALF /0.50/
      DATA WINDOW /0.25/                      !1/4 OCTAVE REM WINDOW
      DATA DBZERO, DBINF /-99.99,+99.99/      !ZERO & INF IN DB UNITS
C
      IF (LSPECT .EQ. 1) GOTO 10
      NHALF = NPER/2                          !DO FIRST PASS CALCS
      DBTWO = 10.*ALOG10(2.)
      RATIO = 2.**(HALF*WINDOW)
      LSPECT = 1                              !& INDICATE DONE
C
 10   CALL TTYOUT ('DOING FFT...')
      CALL FFT (NPER, XDATA)
C
      SUMCOR = 0.                             !ZERO THE COR PWR SUM
C
      DO 30 J =1,NCOMP
      KHARM = HARM(J)                         !GET JTH HARMONIC
      INDEX = 2*KHARM + 1                     !& ITS XDATA INDEX
C
C     DO AMP, PHS, PWR CALCULATIONS FOR SOS FREQS (CORRELATED)
C
      AMPTMP = XDATA (INDEX)                  !GET AMP & ITS SQUARE
      AMPSQR = AMPTMP*AMPTMP
      AMPTMP = 20.*ALOG10(AMPTMP)             !GET AMP & PWR IN DB
```

```
      PWRTMP = AMPTMP - DBTWO
      PHSTMP = XDATA (INDEX+1)          !GET PHASE IN RAD
C
      AMPCOR (J,JCHAN) = AMPTMP         !LOAD COR AMP,PHS,PWR
      PHSCOR (J,JCHAN) = PHSTMP
      PWRCOR (J,JCHAN) = PWRTMP
C
      SUMCOR = SUMCOR + AMPSQR          !ACCUMULATE 2*PWR
C
C     DO PWR CALCULATIONS FOR NON-SOS FREQS (REMNANT)
C
      KLOW = KHARM/RATIO + HALF         !GET LOW & HIGH HARMS
      KHIGH= KHARM*RATIO + HALF         !WHICH DEFINE REM WINDOW
      IF ( KLOW .LT.     1)  KLOW = 1 !& LIMIT THEM
      IF (KHIGH .GT. NHALF) KHIGH = NHALF
      CALL REMPWR (NCOMP,HARM,XDATA,KLOW,KHIGH,NCOUNT,SUMREM)
C
      PWRTMP = DBZERO                   !CALC AVG REM PWR IN WINDOW
      IF ( (NCOUNT .GT. 0) .AND. (SUMREM .GT. 0.) )
     1  PWRTMP = 10.*ALOG10(SUMREM/NCOUNT)
      PWRREM (J,JCHAN) = PWRTMP
      NREM (J) = NCOUNT                 !LOAD # OF REM FREQS IN AVG
C
C     DO CALCULATIONS FOR COR-TO-REM POWER RATIO
C
      TMPCOR = PWRCOR (J,JCHAN)                 !GET COR & REM PWR
      TMPREM = PWRREM (J,JCHAN)
      IF (TMPCOR .GT. DBZERO) GOTO 18           !SET C/R TO ZERO WHEN
      TMPCDR = DBZERO                           !COR PWR IS ZERO
      GOTO 20
 18   IF (TMPREM .GT. DBZERO) GOTO 19           !SET C/R TO INF WHEN
      TMPCDR = DBINF                            !REM PWR IS ZERO
      GOTO 20
 19   TMPCDR = TMPCOR - TMPREM                  !SET C/R TO DIF IN DB
      CALL LIMIT (DBZERO,DBINF,TMPCDR)          !AND LIMIT
 20   CDIVR (J,JCHAN) = TMPCDR                  !LOAD C/R VECTOR
C
 30   CONTINUE
C
C     DO TOTAL POWER CALCS
C
      SUMCOR = SUMCOR/2.                !GET TOTAL COR PWR
C
      KLOW  = 0                         !GET TOTAL REM PWR (INCL DC)
      KHIGH = NHALF
      CALL REMPWR (NCOMP,HARM,XDATA,KLOW,KHIGH,NCOUNT,SUMREM)
C
      TOTCOR (JCHAN) = SUMCOR           !LOAD TOTAL PWR FIGURES
      TOTREM (JCHAN) = SUMREM
C
```

```
RETURN
END
```

subroutine FFT

FUNCTION: Returns N-point fast-Fourier transform of a time history

OPERATION: A time history of length N, stored in the array X, is processed by the routine FAST, which overwrites the time history and returns (to FFT) its discrete Fourier transform in the array X.

The first element of X contains the absolute value of the mean of the time history. The second element contains Ø if the signal mean is positive; otherwise, it contains π. The remaining elements contain magnitude and phase information as follows:

$$\left.\begin{aligned} x(i) &= \frac{2}{N_o}\,[f^2(i) + f^2(i+1)]^{1/2} \\ x(i+1) &= \tan^{-1}(-f(i+1)/f(i)) \end{aligned}\right\} \quad i=3,5,\ldots$$

where "i" is the index in the array X, F signifies the real and imaginary components of the Fourier transform returned by the routine FAST, and x(i) signifies the resulting gain and phase data placed in the array X before returning control to the calling routine.

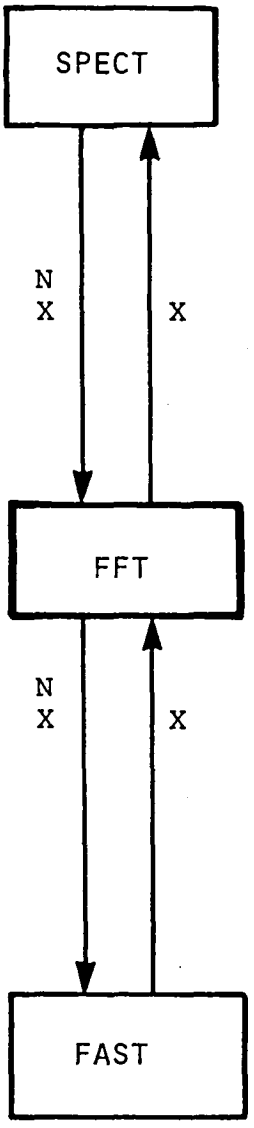
INPUTS: ARGLST: N, X

OUTPUTS: ARGLST: X

CALLER: SPECT

CALLS: FAST

subroutine FFT



```
      SUBROUTINE FFT (N,X)
C
C     RETURNS N-POINT FFT OF X, IN X, WHERE N IS A PWR OF 2
C     (AMP,PHS) FOR JTH HARMONIC STORED IN (X(2J+1),X(2J+2)),
C                                        FOR J = 1 THRU N/2-1
C     (AMP,PHS) FOR   ØTH HARMONIC STORED IN (X(1),  X(2))
C                N/2TH                       (X(N+1),X(N+2))
C
C            INPUTS: (VIA ARGLST)     N,X
C            OUTPUTS:(VIA ARGLST)     X
C
      DIMENSION X(2)
C
      DATA PI /3.14159/
C
      CALL FAST (N,X)
C
      NHALF = N/2
      TWODN = 1./NHALF
C
      TEMP = X(1)/N                        !DO ZEROTH HARMONIC (DC)
      X(1) = ABS(TEMP)
      X(2) = Ø.
      IF (TEMP .LT. Ø.) X(2) = PI
C
      DO 1ØØ I = 1, (NHALF-1)     !DO HARMONICS FROM 1 TO (NHALF-1)
      JODD  = 2*I + 1
      JEVEN = JODD + 1
      TEMP1 = X( JODD)
      TEMP2 = X(JEVEN)
      X( JODD) = TWODN*SQRT (TEMP1*TEMP1 + TEMP2*TEMP2)
      X(JEVEN) = ATAN2 (TEMP1,-TEMP2)
 1ØØ  CONTINUE
C
      TEMP = X(N+1)                        !DO N/2 HARMONIC (NYQUIST)
      X(N+1) = TWODN*ABS (TEMP)
      X(N+2) = Ø.
C
      RETURN
      END
```

subroutine FAST

FUNCTION: computes discrete fast-Fourier transform.

OPERATION: A discrete Fourier transform is performed on the N-point time history provided in the array B where N must be 2 raised to an integral power. The mean value of the time history is returned in element B(1), and B(2) is set to zero. The Jth Fourier harmonic is returned as a complex number, with the real part in element B(2*J+1) and the imaginary part in B(2*J+2). The N/2 harmonic is returned in B(N+1) with B(N+2) set to zero. Thus, the array B must have a minimum dimension of N+2.

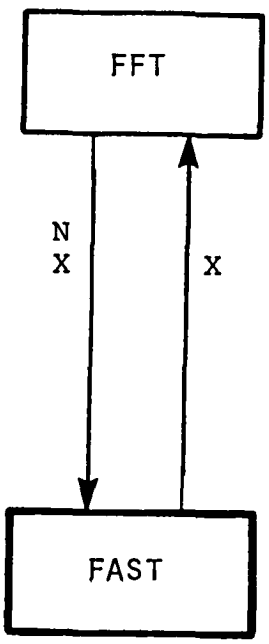
INPUTS: N, B

OUTPUTS: B

CALLER: FFT

CALLS: ----

subroutine FAST



```
C SUBROUTINE:  FAST
C REPLACES THE REAL VECTOR B(K), FOR K=1,2,...,N,
C WITH ITS FINITE DISCRETE FOURIER TRANSFORM
C-------------------------------------------------------------
C
      SUBROUTINE FAST(N,B)
C
C THE DC TERM IS RETURNED IN LOCATION B(1) WITH B(2) SET TO 0.
C THEREAFTER THE JTH HARMONIC IS RETURNED AS A COMPLEX
C NUMBER STORED AS  B(2*J+1) + I B(2*J+2).
C THE N/2 HARMONIC IS RETURNED IN B(N+1) WITH B(N+2) SET TO 0.
C HENCE, B MUST BE DIMENSIONED TO SIZE N+2.
C THE SUBROUTINE IS CALLED AS  FAST(N,B) WHERE N=2**M AND
C B IS THE REAL ARRAY DESCRIBED ABOVE.
C
      DIMENSION B(2)
      COMMON /CONS/ PII, P7, P7TWO, C22, S22, PI2
C
C IW IS A MACHINE DEPENDENT WRITE DEVICE NUMBER
C
      IW = 5
C
      PII = 4.*ATAN(1.)
      PI8 = PII/8.
      P7 = 1./SQRT(2.)
      P7TWO = 2.*P7
      C22 = COS(PI8)
      S22 = SIN(PI8)
      PI2 = 2.*PII
      DO 10 I=1,15
        M = I
        NT = 2**I
        IF (N.EQ.NT) GO TO 20
  10  CONTINUE
      WRITE (IW,9999)
9999  FORMAT (33H N IS NOT A POWER OF TWO FOR FAST)
      STOP
  20  N4POW = M/2
C
C DO A RADIX 2 ITERATION FIRST IF ONE IS REQUIRED.
C
      IF (M-N4POW*2) 40, 40, 30
  30  NN = 2
      INT = N/NN
      CALL FR2TR(INT, B(1), B(INT+1))
      GO TO 50
  40  NN = 1
C
C PERFORM RADIX 4 ITERATIONS.
C
```

```
   50  IF (N4POW.EQ.0) GO TO 70
       DO 60 IT=1,N4POW
         NN = NN*4
         INT = N/NN
         CALL FR4TR(INT, NN, B(1), B(INT+1), B(2*INT+1), B(3*INT+1),
      *       B(1), B(INT+1), B(2*INT+1), B(3*INT+1))
   60  CONTINUE
C
C PERFORM IN-PLACE REORDERING.
C
   70  CALL FORD1(M, B)
       CALL FORD2(M, B)
       T = B(2)
       B(2) = 0.
       B(N+1) = T
       B(N+2) = 0.
       DO 80 IT=4,N,2
         B(IT) = -B(IT)
   80  CONTINUE
       RETURN
       END
C-----------------------------------------------------------------
C SUBROUTINE:  FR2TR
C RADIX 2 ITERATION SUBROUTINE
C-----------------------------------------------------------------
C
       SUBROUTINE FR2TR(INT, B0, B1)
       DIMENSION B0(2), B1(2)
       DO 10 K=1,INT
         T = B0(K) + B1(K)
         B1(K) = B0(K) - B1(K)
         B0(K) = T
   10  CONTINUE
       RETURN
       END
C
C-----------------------------------------------------------------
C SUBROUTINE:  FR4TR
C RADIX 4 ITERATION SUBROUTINE
-----------------------------------------------------------------
C
       SUBROUTINE FR4TR(INT, NN, B0, B1, B2, B3, B4, B5, B6, B7)
       DIMENSION L(15), B0(2), B1(2), B2(2), B3(2), B4(2), B5(2),
       B6(2),
      *    B7(2)
       COMMON /CONS/ PII, P7, P7TWO, C22, S22, PI2
       EQUIVALENCE (L15,L(1)), (L14,L(2)), (L13,L(3)), (L12,L(4)),
      *    (L11,L(5)), (L10,L(6)), (L9,L(7)), (L8,L(8)), (L7,L(9)),
      *    (L6,L(10)), (L5,L(11)), (L4,L(12)), (L3,L(13)), (L2,L(14)),
      *    (L1,L(15))
```

```
C
C JTHET IS A REVERSED BINARY COUNTER, JR STEPS TWO AT A TIME TO
C LOCATE THE REAL PARTS OF INTERMEDIATE RESULTS, AND JI LOCATES
C THE IMAGINARY PART CORRESPONDING TO JR.
C
      L(1) = NN/4
      DO 40 K=2,15
        IF (L(K-1)-2) 10, 20, 30
  10    L(K-1) = 2
  20    L(K) = 2
        GO TO 40
  30    L(K) = L(K-1)/2
  40  CONTINUE
C
      PIOVN = PII/FLOAT(NN)
      JI = 3
      JL = 2
      JR = 2
C
      DO 120 J1=2,L1,2
      DO 120 J2=J1,L2,L1
      DO 120 J3=J2,L3,L2
      DO 120 J4=J3,L4,L3
      DO 120 J5=J4,L5,L4
      DO 120 J6=J5,L6,L5
      DO 120 J7=J6,L7,L6
      DO 120 J8=J7,L8,L7
      DO 120 J9=J8,L9,L8
      DO 120 J10=J9,L10,L9
      DO 120 J11=J10,L11,L10
      DO 120 J12=J11,L12,L11
      DO 120 J13=J12,L13,L12
      DO 120 J14=J13,L14,L13
      DO 120 JTHET=J14,L15,L14
        TH2 = JTHET - 2
        IF (TH2) 50, 50, 90
  50    DO 60 K=1,INT
          T0 = B0(K) + B2(K)
          T1 = B1(K) + B3(K)
          B2(K) = B0(K) - B2(K)
          B3(K) = B1(K) - B3(K)
          B0(K) = T0 + T1
          B1(K) = T0 - T1
  60    CONTINUE
C
        IF (NN-4) 120, 120, 70
  70    K0 = INT*4 + 1
        KL = K0 + INT - 1
        DO 80 K=K0,KL
          PR = P7*(B1(K)-B3(K))
```

```
            PI = P7*(B1(K)+B3(K))
            B3(K) = B2(K) + PI
            B1(K) = PI - B2(K)
            B2(K) = B0(K) - PR
            B0(K) = B0(K) + PR
   80     CONTINUE
          GO TO 120
C
   90     ARG = TH2*PIOVN
          C1 = COS(ARG)
          S1 = SIN(ARG)
          C2 = C1**2 - S1**2
          S2 = C1*S1 + C1*S1
          C3 = C1*C2 - S1*S2
          S3 = C2*S1 + S2*C1
C
          INT4 = INT*4
          J0 = JR*INT4 + 1
          K0 = JI*INT4 + 1
          JLAST = J0 + INT - 1
          DO 100 J=J0,JLAST
            K = K0 + J - J0
            R1 = B1(J)*C1 - B5(K)*S1
            R5 = B1(J)*S1 + B5(K)*C1
            T2 = B2(J)*C2 - B6(K)*S2
            T6 = B2(J)*S2 + B6(K)*C2
            T3 = B3(J)*C3 - B7(K)*S3
            T7 = B3(J)*S3 + B7(K)*C3
            T0 = B0(J) + T2
            T4 = B4(K) + T6
            T2 = B0(J) - T2
            T6 = B4(K) - T6
            T1 = R1 + T3
            T5 = R5 + T7
            T3 = R1 - T3
            T7 = R5 - T7
            B0(J) = T0 + T1
            B7(K) = T4 + T5
            B6(K) = T0 - T1
            B1(J) = T5 - T4
            B2(J) = T2 - T7
            B5(K) = T6 + T3
            B4(K) = T2 + T7
            B3(J) = T3 - T6
  100     CONTINUE
C
          JR = JR + 2
          JI = JI - 2
          IF (JI-JL) 110, 110, 120
  110     JI = 2*JR - 1
```

```
        JL = JR
 120  CONTINUE
      RETURN
      END
C
C-----------------------------------------------------------------
C SUBROUTINE:  FR4SYN
C RADIX 4 SYNTHESIS
C-----------------------------------------------------------------
C
C
      SUBROUTINE FR4SYN(INT, NN, B0, B1, B2, B3, B4, B5, B6, B7)
      DIMENSION L(15), B0(2), B1(2), B2(2), B3(2), B4(2), B5(2),
      B6(2),
     *    B7(2)
      COMMON /CONST/ PII, P7, P7TWO, C22, S22, PI2
      EQUIVALENCE (L15,L(1)), (L14,L(2)), (L13,L(3)), (L12,L(4)),
     *    (L11,L(5)), (L10,L(6)), (L9,L(7)), (L8,L(8)), (L7,L(9)),
     *    (L6,L(10)), (L5,L(11)), (L4,L(12)), (L3,L(13)), (L2,L(14)),
     *    (L1,L(15))
C
      L(1) = NN/4
      DO 40 K=2,15
        IF (L(K-1)-2) 10, 20, 30
  10    L(K-1) = 2
  20    L(K) = 2
        GO TO 40
  30    L(K) = L(K-1)/2
  40  CONTINUE
C
      PIOVN = PII/FLOAT(NN)
      JI = 3
      JL = 2
      JR = 2
C
      DO 120 J1=2,L1,2
      DO 120 J2=J1,L2,L1
      DO 120 J3=J2,L3,L2
      DO 120 J4=J3,L4,L3
      DO 120 J5=J4,L5,L4
      DO 120 J6=J5,L6,L5
      DO 120 J7=J6,L7,L6
      DO 120 J8=J7,L8,L7
      DO 120 J9=J8,L9,L8
      DO 120 J10=J9,L10,L9
      DO 120 J11=J10,L11,L10
      DO 120 J12=J11,L12,L11
      DO 120 J13=J12,L13,L12
      DO 120 J14=J13,L14,L13
      DO 120 JTHET=J14,L15,L14
```

```
      TH2 = JTHET - 2
      IF (TH2) 50, 50, 90
   50 DO 60 K=1,INT
        T0 = B0(K) + B1(K)
        T1 = B0(K) - B1(K)
        T2 = B2(K)*2.0
        T3 = B3(K)*2.0
        B0(K) = T0 + T2
        B2(K) = T0 - T2
        B1(K) = T1 + T3
        B3(K) = T1 - T3
   60 CONTINUE
C
      IF (NN-4) 120, 120, 70
   70 K0 = INT*4 + 1
      KL = K0 + INT - 1
      DO 80 K=K0,KL
        T2 = B0(K) - B2(K)
        T3 = B1(K) + B3(K)
        B0(K) = (B0(K)+B2(K))*2.0
        B2(K) = (B3(K)-B1(K))*2.0
        B1(K) = (T2+T3)*P7TWO
        B3(K) = (T3-T2)*P7TWO
   80 CONTINUE
      GO TO 120
   90 ARG = TH2*PIOVN
      C1 = COS(ARG)
      S1 = -SIN(ARG)
      C2 = C1**2 - S1**2
      S2 = C1*S1 + C1*S1
      C3 = C1*C2 - S1*S2
      S3 = C2*S1 + S2*C1
C
      INT4 = INT*4
      J0 = JR*INT4 + 1
      K0 = JI*INT4 + 1
      JLAST = J0 + INT - 1
      DO 100 J=J0,JLAST
        K = K0 + J - J0
        T0 = B0(J) + B6(K)
        T1 = B7(K) - B1(J)
        T2 = B0(J) - B6(K)
        T3 = B7(K) + B1(J)
        T4 = B2(J) + B4(K)
        T5 = B5(K) - B3(J)
        T6 = B5(K) + B3(J)
        T7 = B4(K) - B2(J)
        B0(J) = T0 + T4
        B4(K) = T1 + T5
        B1(J) = (T2+T6)*C1 - (T3+T7)*S1
```

```
          B5(K) = (T2+T6)*S1 + (T3+T7)*C1
          B2(J) = (T0-T4)*C2 - (T1-T5)*S2
          B6(K) = (T0-T4)*S2 + (T1-T5)*C2
          B3(J) = (T2-T6)*C3 - (T3-T7)*S3
          B7(K) = (T2-T6)*S3 + (T3-T7)*C3
 100    CONTINUE
        JR = JR + 2
        JI = JI - 2
        IF (JI-JL) 110, 110, 120
 110    JI = 2*JR - 1
        JL = JR
 120  CONTINUE
      RETURN
      END
C
C-----------------------------------------------------------------------
C SUBROUTINE:  FORD1
C IN-PLACE REORDERING SUBROUTINE
C-----------------------------------------------------------------------
C
      SUBROUTINE FORD1(M, B)
      DIMENSION B(2)
C
      K = 4
      KL = 2
      N = 2**M
      DO 40 J=4,N,2
        IF (K-J) 20, 20, 10
  10    T = B(J)
        B(J) = B(K)
        B(K) = T
  20    K = K - 2
        IF (K-KL) 30, 30, 40
  30    K = 2*J
        KL = J
  40  CONTINUE
      RETURN
      END
C
C-----------------------------------------------------------------------
C SUBROUTINE:  FORD2
C IN-PLACE REORDERING SUBROUTINE
C-----------------------------------------------------------------------
C
      SUBROUTINE FORD2(M, B)
      DIMENSION L(15), B(2)
      EQUIVALENCE (L15,L(1)), (L14,L(2)), (L13,L(3)), (L12,L(4)),
     *    (L11,L(5)), (L10,L(6)), (L9,L(7)), (L8,L(8)), (L7,L(9)),
     *    (L6,L(10)), (L5,L(11)), (L4,L(12)), (L3,L(13)), (L2,L(14)),
     *    (L1,L(15))
```

```
      N = 2**M
      L(1) = N
      DO 10 K=2,M
        L(K) = L(K-1)/2
10    CONTINUE
      DO 20 K=M,14
        L(K+1) = 2
20    CONTINUE
      IJ = 2
      DO 40 J1=2,L1,2
      DO 40 J2=J1,L2,L1
      DO 40 J3=J2,L3,L2
      DO 40 J4=J3,L4,L3
      DO 40 J5=J4,L5,L4
      DO 40 J6=J5,L6,L5
      DO 40 J7=J6,L7,L6
      DO 40 J8=J7,L8,L7
      DO 40 J9=J8,L9,L8
      DO 40 J10=J9,L10,L9
      DO 40 J11=J10,L11,L10
      DO 40 J12=J11,L12,L11
      DO 40 J13=J12,L13,L12
      DO 40 J14=J13,L14,L13
      DO 40 JI=J14,L15,L14
        IF (IJ-JI) 30, 40, 40
30      T = B(IJ-1)
        B(IJ-1) = B(JI-1)
        B(JI-1) = T
        T = B(IJ)
        B(IJ) = B(JI)
        B(JI) = T
40      IJ = IJ + 2
      RETURN
      END
```

subroutine REMPWR

FUNCTION: Computes remnant power over a specific frequency "window".

OPERATION: Once a power spectrum has been computed by the subroutine SPECT and stored in the vector XDATA, the routine REMPWR computes the accumulated power in XDATA between the frequency indices KLOW and KHIGH, exclusive of power at SOS indices defined by HARM. Remnant power is returned as SUMREM, with NREM indicating the number of frequency indices used in computing the remnant power.

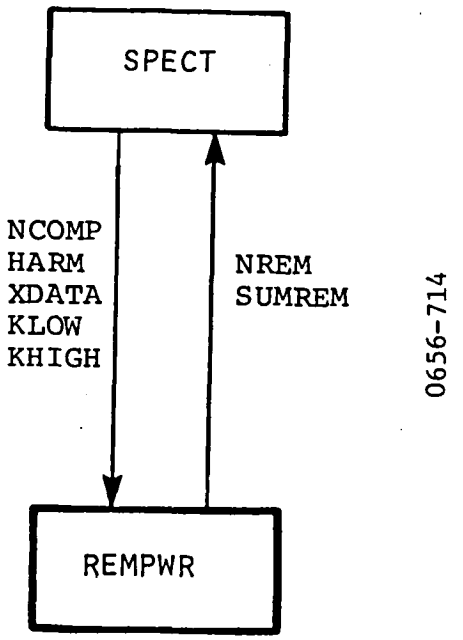
INPUTS: ARGLST: NCOMP, HARM, XDATA, KLOW, KHIGH

OUTPUTS: ARGLST: NREM, SUMREM

CALLER: SPECT

CALLS: ----

subroutine REMPWR

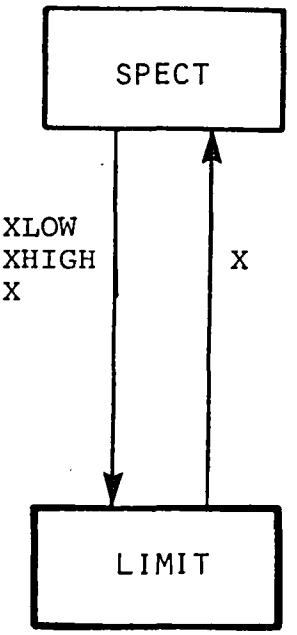


```
      SUBROUTINE REMPWR (NCOMP,HARM,XDATA,KLOW,KHIGH,NREM,SUMREM)
C
C     REMPWR COMPUTES THE SUMMED REMNANT POWER OVER THE
C     HARMONIC WINDOW DEFINED BY (KLOW,KHIGH)
C     SUMREM EXCLUDES POWER AT THE SOS HARMONICS DEFINED
C     BY HARM, AND RETURNS THE NUMBER OF REMNANT FREQS SUMMED
C
C           INPUTS: (VIA ARGLST)    NCOMP,HARM,XDATA
C                   (VIA ARGLST)    KLOW,KHIGH
C           OUTPUTS:(VIA ARGLST)    NREM, SUMREM
C
      INTEGER HARM(1)
      DIMENSION XDATA(1)
C
      NREM = 0                              !ZERO COUNTER & SUMMER
      SUMREM = 0.
C
      DO 20 K = KLOW,KHIGH                  !SUM FROM KLOW TO KHIGH
C
      DO 10 L = 1,NCOMP                     !EXCLUDE SOS HARMONICS
 10   IF (K .EQ. HARM(L)) GOTO 20
C
      INDEX = 2*K + 1                       !GET REMNANT AMP
      AMPREM = XDATA (INDEX)
      SUMREM = SUMREM + AMPREM*AMPREM       !ACCUMULATE 2*PWR
      NREM = NREM + 1                       !INCREMENT COUNTER
C
 20   CONTINUE
C
      SUMREM = SUMREM/2.                    !CALC SUMMED REM PWR
      RETURN
      END
```

subroutine LIMIT

FUNCTION:	Maintain variable within limits
OPERATION:	LIMIT first checks that the desired minimum value XLOW is less than or equal to XHIGH. If the test fails, an error message is sent to the terminal, and the program stops. Otherwise, the variable X is adjusted, if necessary, to lie between XLOW and XHIGH.
INPUTS:	ARGLST: XLOW, XHIGH, X
OUTPUTS:	ARGLIST: X
CALLER:	SPECT
CALLS:	----

subroutine LIMIT



0656-710

```
      SUBROUTINE LIMIT (XLOW, XHIGH, X)
C
      IF (XLOW .LE. XHIGH) GOTO 10
      CALL TTYOUT ('*****LIMIT: LOW/HIGH LIMITS REVERSED*****')
      STOP
C
 10   IF (X .LT.  XLOW) X = XLOW
      IF (X .GT. XHIGH) X = XHIGH
      RETURN
      END
```

subroutine DFCN

FUNCTION: Compute the describing function between two channels

OPERATION: For each SOS index "j", DFCN computes the describing function gain Δa_j =GAIN(J) and relative phase shift $\Delta\phi_j$ = PHASE(J) between two channels as follows:

$$\Delta a_j = a_{j,1} - a_{j,2}$$

$$\Delta\phi_j = \phi_{j,1} - \phi_{j,2}$$

where $a_{j,1}$ =AMPCOR(J,I) is the amplitude of signal I in dB, and $\phi_{j,i}$ =PHSCOR(J,I) is the phase shift of signal I in degrees. AMPCOR and PHSCOR are determined by previous calls to SPECT, and the indices I are set in VERNAL to point to the channels specified by the user to serve as the numerator (I=1) and denominator (I=2) quantities for describing function computation. Because phase shift is a circular function, repeating every 360 degrees, a scheme for "unwrapping" the phase is employed in an attempt to maintain a smoothly varying function of frequency. Specifically, the phase computation at a given SOS frequency is adjusted up or down by an integral multiple of 360, if necessary, to yield a result that is within $\pm$ 180 degrees of the phase estimate at the previous SO frequency. (The reference phase PHSOLD is initialized to zero for the first SOS frequency.)

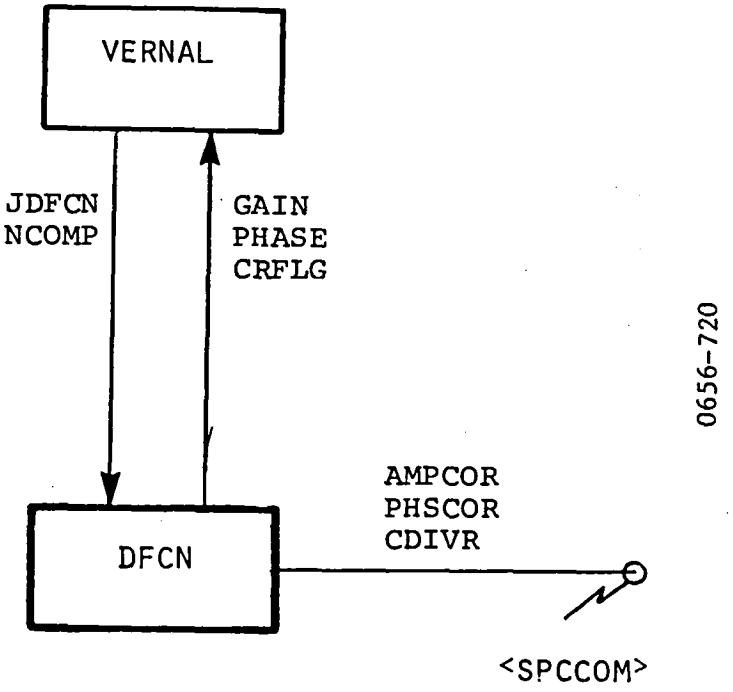
The signal/noise ratios CDIV are checked for both the numerator denominator signals; if either ratio is less than 6 dB, the flag CRFLAG is set from subsequent printout of "stars" (****) to indicate an unreliable describing function estimate at that frequency.

INPUTS: ARGLST: JDFCN, NCOMP
<SPCCOM>: AMPCOR, PHSCOR, CDIVR

OUTPUTS: ARGLIST: GAIN, PHASE, CRFLAG

LOCAL: PHSOLD

subroutine DFCN



```
      SUBROUTINE DFCN (JDFCN, NCOMP, GAIN, PHASE, CRFLAG)
C
C     DFCN COMPUTES DESCRIBING FUNCTION FOR TWO CHANNELS
C     CHANNEL NUMBERS FOR (NUM,DENOM) ARE (JDFCN(1),JDFCN(2))
C     GAIN/PHASE IS DIFFERENCE IN DB/RAD OF CORRELATED SIGNAL
C     AMPS/PHASES
C     PHASE CHANGE WITH FREQUENCY IS LIMITED , AND A FLAG IS
C     SET WHEN THE C/R RATIO IS LOW, FOR EITHER CHANNEL
C
C            INPUTS: (VIA ARGLST)    JDFCN, NCOMP
C                    (VIA SPCCOM)    AMPCOR,PHSCOR,CDIVR
C           OUTPUTS:(VIA ARGLST)     GAIN,PHASE,CRFLAG
C
      COMMON /SPCCOM/ AMPCOR,PHSCOR,CDIVR
C
      DIMENSION JDFCN(2), GAIN(1), PHASE(1), CRFLAG(1),
     1          AMPCOR(15,4), PHSCOR(15,4), CDIVR(15,4)
C
      DATA BLANK, STARS/'    ', '****'/
      DATA SIXDB /6./
      DATA PI, TWOPI /3.14159, 6.28318/
C
      PHSOLD =0.
C
      DO 40 J= 1,NCOMP
      JNUM = JDFCN(1)
      JDENOM =JDFCN(2)
      GAIN(J) = AMPCOR(J,JNUM) - AMPCOR(J,JDENOM)      !GET GAIN
C
      PHSTMP =  PHSCOR(J,JNUM) - PHSCOR(J,JDENOM)      !GET PHASE
      PHSDIF = PHSTMP - PHSOLD
10    IF (PHSDIF .LE. PI) GOTO 20
      PHSDIF = PHSDIF -TWOPI
      GOTO 10
20    IF (PHSDIF .GE. -PI) GOTO 30
      PHSDIF = PHSDIF + TWOPI
      GOTO 20
30    PHSTMP = PHSOLD + PHSDIF
      PHASE(J) = PHSTMP
C
      CRFLAG(J) = STARS                !SET C/R FLAG IF C/R LOW
      IF (CDIVR(J,  JNUM) .LT. SIXDB) GOTO 40
      IF (CDIVR(J,JDENOM) .LT. SIXDB) GOTO 40
      CRFLAG(J) = BLANK
      PHSOLD = PHSTMP
40    CONTINUE
C
      RETURN
      END
```

APPENDIX C
OTHER MAJOR FORTRAN ROUTINES

This Appendix contains documentation for the FORTRAN subprograms TITLER, RWHEAD, and RWDATA, which are common to the VERRUN and VERNAL software systems.

subroutine TITLER

FUNCTION: Reads and writes title information

OPERATION: Title information may be read from or written to either a file or the terminal. Title information includes the file name (if relevant), run number, date, time, and user-defined commentary.

The flag IRW indicates whether TITLER reads or writes (1=read, 2=write). If information is to be specified interactively (indicated by the value of LUNIT), the current date and time are determined by calls to the FORTRAN subroutines DATE and TIME, and date, time, and run number are displayed to the user. If the program is in the "run" mode (indicated by the value 'R', for the flag MODE), the user is provided the option to change the run number. Finally, the user is given the opportunity to specify up to six lines of commentary.

If title information is being written to the terminal, display of the commentary will be suppressed if TITLER is called with MODE set to 'S'. A call to FILIN (FILOUT) is made to transfer commentary when title information is being read from (written to) a file.

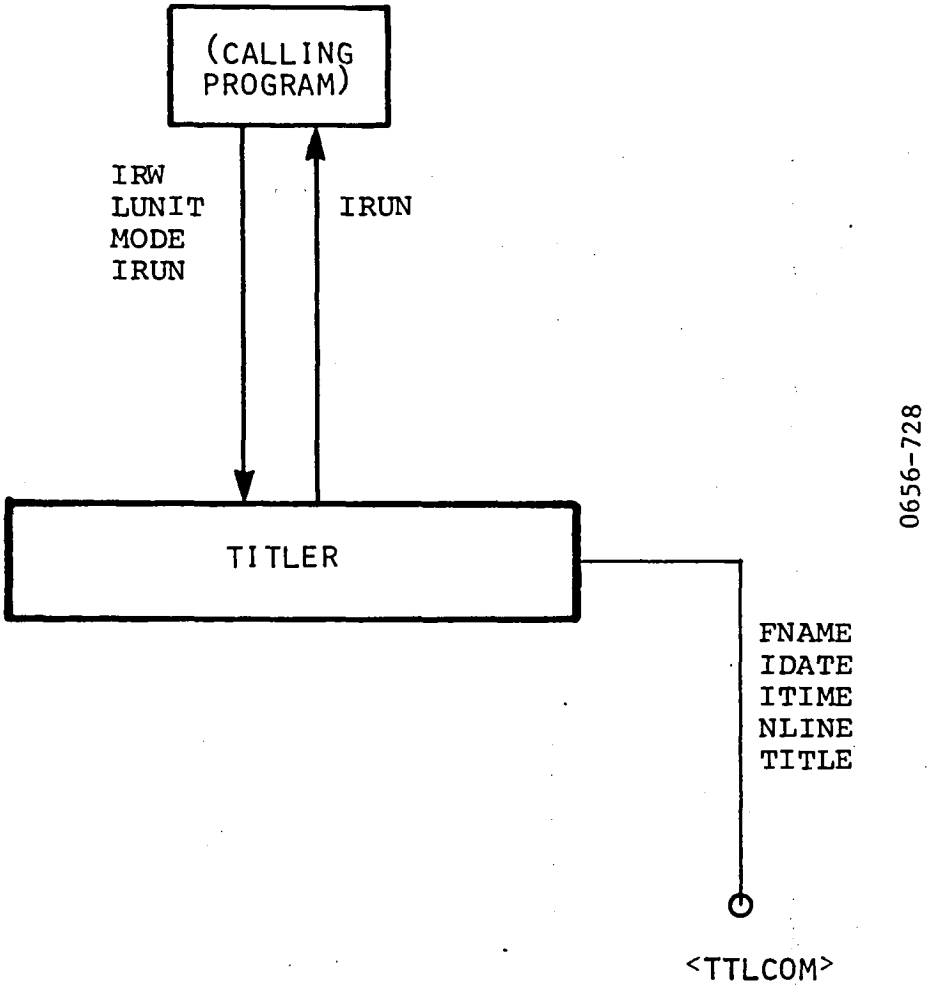
INPUTS: ARGLST: IRW, LUNIT, MODE, IRUN

OUTPUTS: ARGLST: IRUN

I/O: <TTLCOM>: FNAME, IDATE, ITIME, NLINE, TITLE

CALLER: VERRUN, RWHEAD (VERRUN software system)
VERNAL, RWHEAD (VERNAL software system)

subroutine TITLER



```
      SUBROUTINE TITLER (IRW, LUNIT, MODE, IRUN)
C
C     TITLER READS/WRITES THE TITLE FROM/TO A FILE OR TTY
C     THE TITLE INCLUDES FILE NAME, DATE, TIME, AND COMMENTS
C
C           INPUTS  (VIA ARGLST)    IRW (1 = READ, 2 = WRITE)
C                   (VIA ARGLST)    LUNIT,MODE
C           OUTPUTS:(VIA ARGLST)    IRUN
C                   (VIA TTLCOM)    IDATE, ITIME, NLINE, TITLE
C
      COMMON /TTLCOM/ FNAME, IDATE, ITIME, NLINE, TITLE
C
      LOGICAL*1 LASK,MODE,ITIME(8),IDATE(9),TITLE(255),FNAME(11)
      INTEGER HOURS, SECONS
C
      DATA NDIM /255/
      DATA LUNTTY /5/
C
      GOTO (100,200) IRW
C
C           READ-IN SECTION
 100  IF (LUNIT .NE. LUNTTY)  GOTO 130
C
C           READ IN FROM TTY
 110  CALL DATE (IDATE)
      CALL TIME (ITIME)
      WRITE (LUNIT, 3000) IRUN, IDATE, ITIME
 3000 FORMAT (1X,'RUN NUMBER: ',I4,4X,'DATE: ',9A1,4X,'TIME: ',
      9A1,/)
      IF (MODE .EQ. 'P') GOTO 120
      CALL TTYOUT(' ')
      IF (LASK('CHANGING THE RUN NUMBER? ') .EQ. 'N') GOTO 120
      CALL TTYOUT ('NEW RUN NUMBER: $')
      IRUN = IANS (0, 100)
      GOTO 110
 120  CALL TTYOUT ('NUMBER OF COMMENT LINES: $')
      NLINE = IANS (0, 6)
      IF (NLINE .EQ. 0) RETURN
      CALL TTYIN (NLINE, NDIM, TITLE)
      RETURN
C
C           READ IN FROM FILE
 130  READ (LUNIT, 1000) FNAME, IRUN, IDATE, ITIME
 1000 FORMAT (7X,11A1,15X,I4,11X,9A1,10X,9A1)
      READ (LUNIT, 1010) NLINE
 1010 FORMAT (19X,I4)
      CALL FILIN (NLINE, NDIM, TITLE, LUNIT)
      RETURN
C
C           WRITE-OUT SECTION
```

```
 200    IF (LUNIT .NE. LUNTTY) GOTO 210
C
C               WRITE OUT ONTO TTY
       WRITE (LUNIT, 2000) FNAME, IRUN, IDATE, ITIME
 2000  FORMAT (1X,'FILE: ',11A1,6X,' RUN NO: ',I4,4X,' DATE: ',
       1 9A1,3X,' TIME: ',9A1)
       IF (MODE .EQ. 'S') RETURN       !SUPPRESS TITLE WRITEOUT
       IF (NLINE .NE. 0) CALL TTYOUT (TITLE)
       RETURN
C
C               WRITE OUT ONTO FILE
 210   WRITE (LUNIT, 2000) FNAME, IRUN, IDATE, ITIME
       WRITE (LUNIT, 2010) NLINE
 2010  FORMAT (1X,'TITLE LINE COUNT: ',I4)
       IF (NLINE .NE. 0) CALL FILOUT (TITLE, LUNIT)
       RETURN
       END
```

subroutine RWHEAD

FUNCTION: Reads and writes header information

OPERATION: Information may be written to or read from a data file, or written to (but not read from) the terminal. If RWHEAD is called with LUNIT set to the terminal device number, header information is displayed on the terminal, and control returns to the calling program. If information exchange with a data file is indicated, the following operations are performed:

a. The user specifies the name of the data file.

b. If currently open, the data file is closed.

c. The data file is opened, and the flag IOPEN is set to 'Y'.

d. Header information is written/read. This information consists of a program version number, title information (via a call to TITLER), time base parameters, and SOS parameters.

If the flag ICLOSE is set to 1, the data file is closed and IOPEN is set to 'N'; otherwise, the file remains open. The file will be closed if program VERRUN is being run in the parameter setup mode; it will remain open if program VERRUN is operating in the "run" mode, or if program VERNAL is being run.

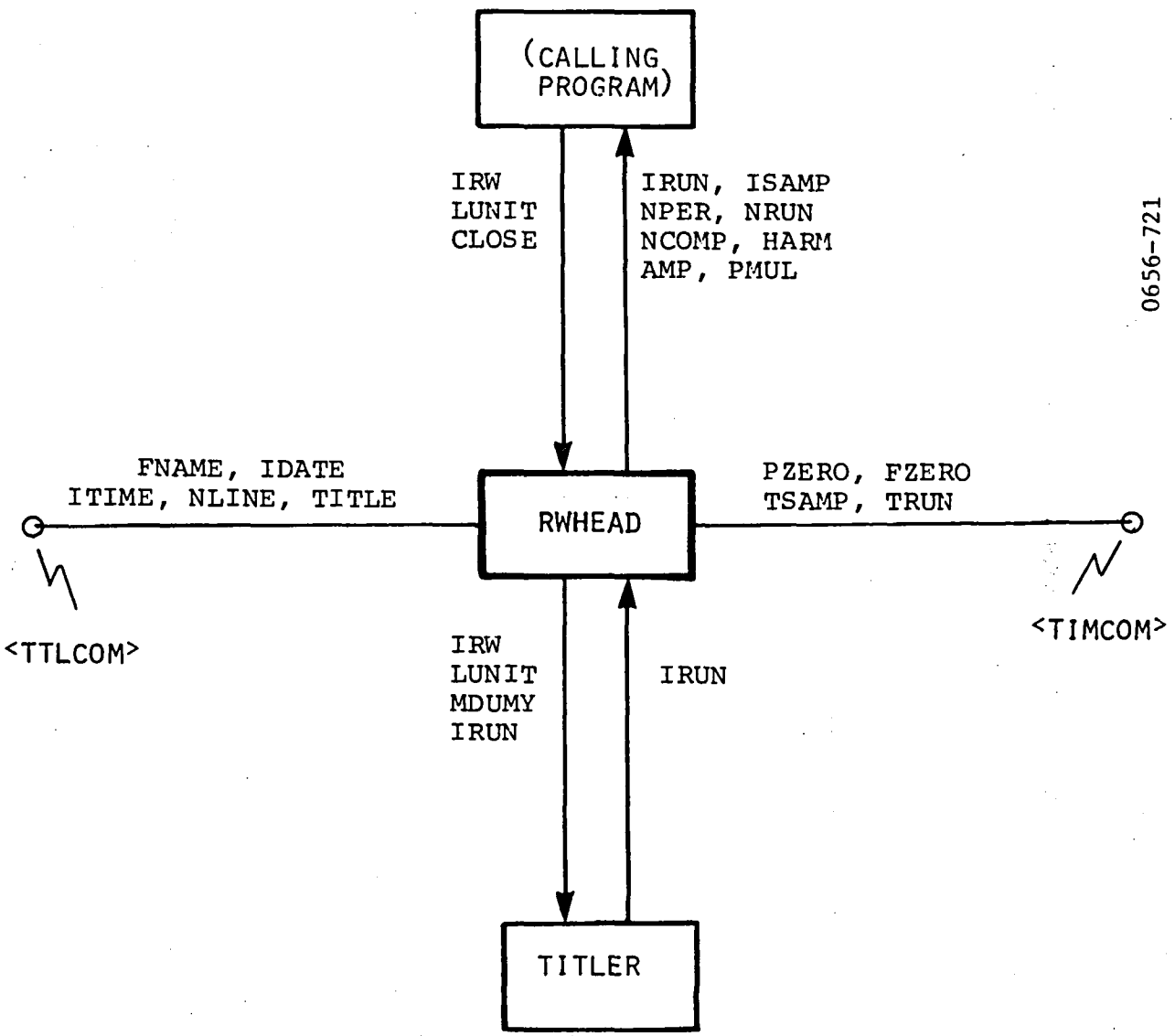
INPUTS: ARGLST: IRW, LUNIT, ICLOSE

I/O: ARGLST: IRUN, ISAMP, NPER, NRUN, NCOMP, HARM, AMP, PMUL
<TIMCOM>: PZERO, FZERO, TSAMP, TRUN
<TTLCOM>: FNAME, IDATE, ITIME, NLINE, TITLE

CALLER: VERRUN, PARSET (VERRUN software system)
VERNAL (VERNAL software system)

CALLS: TITLER

subroutine RWHEAD



```
      SUBROUTINE RWHEAD(IRW,LUNIT,ICLOSE,IRUN,ISAMP,NPER,
     1                  NRUN,NCOMP,HARM,AMP,PMUL)
C
C     CHANGES BY W.H. LEVISON,  12/9/83
C     1.  INITIALIZE MDUMY TO BE 'P'
C     2.  ELIMINATE READ/WRITE OF ISEED
C
C     READS/WRITES HEADER FROM/TO A DATA FILE
C     ALSO WRITES HEADER TO TTY
C
C     INPUTS: (VIA ARGLST)    IRW    (1=READ HEADER,2=WRITE HEADER)
C             (     "    )    LUNIT
C             (VIA ARGLST)    ICLOSE (1=CLOSE FILE, 2=LEAVE FILE)
                              OPEN
C     I/O:    (VIA ARGLST)    IRUN, ISAMP
C             (     "    )    NPER, NRUN, NCOMP
C             (     "    )    HARM, AMP, PMUL
C             (VIA TIMCOM)    PZERO, FZERO, TSAMP, TRUN
C
      COMMON /TIMCOM/ PZERO, FZERO, TSAMP, TRUN
      COMMON /TTLCOM/ FNAME, IDATE, ITIME, NLINE, TITLE
C
      LOGICAL*1 IOPEN,MDUMY,FNAME(11), IDATE (9), TITLE (255)
      INTEGER HARM(1), PMUL(1), HOURS, SECONS
      DIMENSION AMP(1)
C
      DATA NVERS /2/
      DATA LUNTTY /5/
      DATA IOPEN /'N'/
      DATA MDUMY/'P'/
C
      IF (LUNIT .EQ. LUNTTY) GOTO 201
C
    5 CALL FILNAM (IRW, FNAME, NCHAR)          !GET FILE NAME
      IF (IOPEN .EQ. 'N') GOTO 10
      CLOSE (UNIT = LUNIT, DISPOSE = 'SAVE')   !AND CLOSE IT IF OPEN
      IOPEN = 'N'                              !AND INDICATE IT'S
                                               CLOSED
   10 GOTO (100, 200) IRW
C
C             READ FROM FILE
C
  100 CONTINUE
      OPEN(UNIT=LUNIT,NAME=FNAME,CARRIAGECONTROL='LIST',TYPE='OLD')!
      IOPEN = 'Y'
      READ (LUNIT, 105) NVERS
  105 FORMAT (17X, I1,/,/)
      CALL TITLER (IRW, LUNIT, MDUMY, IRUN)
      READ (LUNIT, 110) ISAMP
  110 FORMAT (/, /, 25X, I4)
```

```
        READ (LUNIT, 115) FZERO, PZERO
 115    FORMAT (17X, 1PE12.3, 21X, 1PE12.3)
        READ (LUNIT, 120) TEMP, NPER
 120    FORMAT (17X, 1PE12.3, 28X, I5)
        READ (LUNIT, 125) TRUN, NRUN
 125    FORMAT (17X, 1PE12.3, 28X, I5)
        READ (LUNIT, 130) NCOMP
 130    FORMAT (/,/, 22X, I4,/)
        READ (LUNIT, 135)  (HARM(I), AMP(I), PMUL(I), I=1,NCOMP)
 135    FORMAT (10X, I5, 16X, F6.3, 5X, I6)
        GOTO 300
C
C               WRITE TO FILE (OR TTY)
C
 200    CONTINUE
        OPEN(UNIT=LUNIT,NAME=FNAME,CARRIAGECONTROL='LIST',TYPE='NEW')!
        IOPEN = 'Y'
 201    WRITE (LUNIT, 205) NVERS
 205    FORMAT (1X, 'VERSION NUMBER: ', I1)
        WRITE (LUNIT, 206)
 206    FORMAT (/, 1X, '***RUN IDENTIFICATION***')
        CALL TITLER (IRW, LUNIT, MDUMY, IRUN)
        WRITE (LUNIT, 209)
 209    FORMAT (/, 1X, '***TIME BASE PARAMETERS***')
        WRITE (LUNIT, 210) ISAMP
 210    FORMAT (1X, 'SAMPLE PERIOD:  ', 8X, I4,' MSEC')
        WRITE (LUNIT, 215) FZERO, PZERO
 215    FORMAT (1X, 'BASE FREQUENCY: ',1PE12.3, '   HZ',
     1          4X, 'BASE PHASE: ',    1PE12.3, ' DEG')
        WRITE (LUNIT, 220) NPER*(ISAMP/1000.), NPER
 220    FORMAT (1X, 'SOS PERIOD:     ',1PE12.3, '  SEC',
     1          4X ,'WITH: ', 13X, I5, ' PTS')
        WRITE (LUNIT, 225) TRUN, NRUN
 225    FORMAT (1X, 'RUN LENGTH:     ',1PE12.3, '  SEC',
     1          4X, 'WITH: ', 13X, I5, ' PTS')
        WRITE (LUNIT, 229)
 229    FORMAT (/, 1X, '***SOS SIGNAL PARAMETERS***')
        WRITE (LUNIT, 230) NCOMP
 230    FORMAT (1X, '# OF SOS COMPONENTS: ', I4)
        WRITE (LUNIT, 234)
 234    FORMAT (2X, 'COMP', 5X, 'HARM', 7X, 'FREQ', 7X, 'AMP',
     1          8X, 'PMUL', 7X, 'PHS')
        WRITE (LUNIT, 235)  (J,HARM(J),FZERO*HARM(J),AMP(J),
     1                       PMUL(J),PZERO*PMUL(J), J=1,NCOMP)
 235    FORMAT (I5,5X,I5,5X,F6.2,5X,F6.3,5X,I6,5X,F6.1)
C
        IF (LUNIT .EQ. LUNTTY) RETURN   !RETURN IF JUST DONE TTY WRITE
C
 300    IF (ICLOSE .NE. 1)  RETURN
        CLOSE (UNIT = LUNIT, DISPOSE = 'SAVE')  !CLOSE FILE
```

```
IOPEN = 'N'                                    !AND INDICATE CLOSED
RETURN
END
```

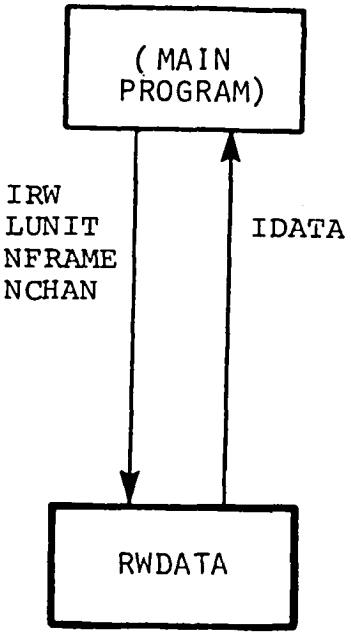
subroutine RWDATA

FUNCTION:	Reads and writes time history data
OPERATION:	The data array IDATA is written to or read from a data file. IDATA may also be displayed on the user's terminal. Data are stored in IDATA in an interleaved format: the first data sample from the first channel, followed by the first sample from the second channel, etc. The data file, which has been opened previously by a call to RWHEAD, is closed upon completion of data transfer.
INPUTS:	ARGLST: IRW, LUNIT, NFRAME, NCHAN
OUTPUTS:	ARGLST: IDATA
CALLER:	(main program)
CALLS:	----

```
      SUBROUTINE RWDATA (IRW, LUNIT, NFRAME, NCHAN, IDATA)
C
C     RWDATA READS/WRITES THE DATA ARRAY IDATA FROM/TO FILE
C
C            INPUTS: (VIA ARGLST)    IRW (1 = READ, 2 = WRITE)
C                                    LUNIT, NFRAME, NCHAN, IDATA
C            OUTPUT: (VIA ARGLST)    IDATA
C
      DIMENSION IDATA (1), ITEMP (10)
C
      DATA LUNTTY/5/
      DATA NCMAX/4/
C
      IF(NCHAN .LE. NCMAX) GOTO 10
      CALL TTYOUT('******RWDATA: NCHAN .GT. NCMAX******')
      STOP
C
 10   GOTO (100,200) IRW
C
C     READ DATA FROM FILE & LOAD IDATA
 100  IF (LUNIT .NE. LUNTTY) GOTO 105
      CALL TTYOUT ('******RWDATA: TRYING TO READ FROM TTY******')
      STOP
C
 105  READ (LUNIT, 999)
 999  FORMAT(/,/)
C
      INDEX = 0
      DO 120 I = 1, NFRAME
      READ (LUNIT,1000) IDUMMY, (ITEMP(J), J=1,NCHAN)
 1000 FORMAT (1X,5I5)
      DO 110 J = 1, NCHAN
 110  IDATA(INDEX+J) = ITEMP(J)
 120  INDEX = INDEX + NCHAN
      GOTO 300
C
C     WRITE ALL CHANNELS OF DATA FROM IDATA TO FILE (OR TTY)
 200  WRITE (LUNIT, 2000) NCHAN
 2000 FORMAT (/, 1X, '***RECORDED DATA OF ', I3, ' CHANNELS***')
      WRITE (LUNIT, 2001)
 2001 FORMAT (2X, 'IFRM','  C1 ','  C2 ','  C3 ','  C4 ')
      INDEX = 0
      DO 220 I=1,NFRAME
      DO 210 J=1,NCHAN
 210  ITEMP(J) = IDATA(INDEX+J)
      WRITE (LUNIT,1000) I, (ITEMP(J),J=1,NCHAN)
 220  INDEX = INDEX + NCHAN
      IF (LUNIT .EQ. LUNTTY) RETURN
C
 300  CLOSE (UNIT = LUNIT, DISPOSE = 'SAVE')
```

```
      RETURN
      END
```

subroutine RWDATA



APPENDIX D
FORTRAN I/O LIBRARY ROUTINES

A list of the I/O library routines, along with brief descriptions of their functions, are included in Table D.1. Listings of each routine follow.

TABLE D.1 IOLIB ROUTINES

FILIN	Reads multiple lines of text from a file unit specified by the calling program, stores in an array specified by the calling program.
FILNAM	Reads in a character string from the TTY, to be used in specifying a file name for I/O on the system disk
FILOUT	Outputs a text string onto a file unit specified in the calling sequence.
FILSTR	Reads a text string from a file unit specified by the calling program and stores it in an array specified by the calling program.
GETSTR	Reads a single line of text from the TTY and stores it in an array specified in the calling sequence.
IANS	Reads an integer value from the TTY and checks that the value is within bounds specified by the calling routine
LANS	Reads a single character from the TTY and checks that it is valid according to the calling routine's specifications
LASK	Writes out a character string onto the TTY, reads back a single Y/N character, and checks that the character is Y or N.
PUTSTR	Outputs a single line of text with carriage control at the end of the line.
RANS	Reads a value from the TTY and checks that the value is within bounds specified by the calling routine.

RGET	Reads a real value from the TTY.
STRING	Same as GETSTR, except a character count is returned to the calling routine.
TTYIN	Reads in multiple lines of text from the TTY and stores it in an array supplied by the called routine
TTYOUT	Writes out onto the TTY a character array supplied by the calling routine, with carriage control at both the beginning and the end of the text.
VECTIN	Loads a real vector, component by component from TTY input, providing a range check on the component value, and an opportunity for user corrections.
VECVAL	Prompts the user to specify for a real number. Used by VECTIN.

```
      SUBROUTINE FILIN (NLINE,NDIM,TITLE,LUN)
C
      LOGICAL*1 TEMP (255), TITLE (1), CRTURN ,LNFEED
C
      ISAVE=0
      NTEMP=NDIM-2
C
      IF (NLINE .EQ. 0)  RETURN
      CALL FILSTR (TEMP, LUN)     ! READ ONE LINE FROM FILE
      DO 10 I=1,71           !LOAD TEMP INTO TITLE
      ITEMP=I+ISAVE
      IF((ITEMP.GE.NTEMP).OR.(TEMP(I).EQ.0))GO TO 15
C                                !CHECK FOR FULL TITLE VECTOR OR
C                                !NULL CHARACTER IN TEMP INPUT
   10 TITLE(ITEMP)=TEMP(I)
   15 CONTINUE
      IF (ITEMP.GE.NTEMP) GO TO 20    !QUIT IF TITLE IS FILLED
      ISAVE=ITEMP                   !SAVE LAST LOADED POSN
    5 CONTINUE                      !BOTTOM OF LINE LOOP
   20 IF (L .GT. NLINE) GOTO 25
      DO 30 J = L, NLINE
   30 CONTINUE
   25 ITEMP=ITEMP+1
      TITLE(ITEMP)=0
      RETURN
C
      DATA CRTURN, LNFEED /13, 10/
C
      END
```

```
      SUBROUTINE FILNAM(IOCHAN,NAME,NCHAR)
C
C     This subroutine acepts file names, checking them for legality
C     (all alphanumeric characters, etc...)
C
C     Input is IOCHAN.  1 for Input filename, 2 for Output filename.
C     NAME is the array containing the name of the file.
C     NCHAR is the number of characters in the filename.
C
      LOGICAL*1 NAME(10), DOT
      LOGICAL*1 UPCSA, UPCSZ, ASCII0, ASCII9
      CALL TTYOUT('ENTER FILENAME FOR $', 5)
      GOTO (5,10) IOCHAN                ! Check for legitimate IOCHAN
      STOP'****FILNAM:ILLEGAL IOCHAN VALUE****'
C
C           Print appropriate prompt and read filename.
C
   5  CALL TTYOUT('$INPUT: $', 5)
      GOTO 15
  10  CALL TTYOUT('$OUTPUT: $', 5)
  15  READ (5, 20) NAME
  20  FORMAT(10A1)
C
C           Is the first character a letter? (not <a or >b)
C
      I = 1
      IF ((NAME(I) .LT. UPCSA) .OR. (NAME(I) .GT. UPCSZ)) GOTO 100
C
C     Now check the rest of the name to see if it is all
      alphanumeric
C     characters, and set NCHAR = to 3 places after the '.'
C
      DO 200 I = 2,10
      IF (NAME(I) .EQ. DOT) GOTO 50
      IFLAG = -1
      IF ((NAME(I).LT.UPCSA).OR.(NAME(I).GT.UPCSZ)) IFLAG=IFLAG+1
      IF ((NAME(I).LT.ASCII0).OR.(NAME(I).GT.ASCII9))IFLAG=IFLAG+1
      IF (IFLAG) 200, 200, 100
  50  IF ((I .EQ. 1) .OR. (I .GE. 8)) GOTO 100
      NCHAR = I + 3
      DO 110 J = I+1, NCHAR
 110  IF (NAME(J) .EQ. DOT) GOTO 100
      RETURN            ! Legal File Name.  Return.
 200  CONTINUE
C
C           Bad filename: deal with it...
C
```

```
 100    CALL TTYOUT('INVALID FILENAME.  TRY AGAIN: $', 5)
        GOTO 15
C
        DATA UPCSA, UPCSZ, ASCII0, ASCII9 /65, 90, 48, 57/
        DATA DOT /'.'/
        END
```

```
      SUBROUTINE FILOUT (MSG, LUN)
C
C
C     This subroutine outputs the string MSG onto the file unit LUN.
C
C     Last Modification Date:  12-July-83
C
      LOGICAL*1 MSG(1), EOF
C
      ISTART = 1
C
C     This next loop goes through the string until it encounters an
C     End Of File indicator in order to find the terminating
      position
C     in the string.
C
      ISTOP = ISTART
   15 ISTOP = ISTOP + 1
      IF (MSG(ISTOP) .NE. EOF) GOTO 15
C
C
      INUM = ISTOP - ISTART
      INUM = INUM + 2
      WRITE (LUN, 200) (MSG (I), I = ISTART, ISTOP)
  200 FORMAT (1X, 255A1)
C
      RETURN
C
      DATA EOF /0/
      END
```

```
        SUBROUTINE FILSTR (CHAR, LUN)
        LOGICAL*1 CHAR(1)

C  GETS UP TO 255 CHARACTERS FROM THE FILE
C  THE TEXT STRING IS TERMINATED BY A NULL BYTE.
C  <CR> IS NOT INCLUDED IN THE TEXT STRING
C
        READ (LUN, 101)  (CHAR (I), I = 1, 255)
101     FORMAT(255A1)
C  THE STRING WILL BE PADDED WITH SPACES (32)
C  FIND THE FIRST NON SPACE AND SET THE BYTE
C  AFTER IT TO 0.
        DO 20 I=70,1,-1
20      IF(CHAR(I).NE.32)GOTO 30
        CHAR(1)=0
        RETURN
30      CHAR(I+1)=0
        RETURN
        END
```

```
        SUBROUTINE GETSTR(CHAR,MAX)
        LOGICAL*1 CHAR(1)

C  GETS UP TO 'MAX' CHARACTERS FROM THE TTY:
C  THE TEXT STRING IS TERMINATED BY A NULL BYTE.
C  <CR> IS NOT INCLUDED IN THE TEXT STRING
C
        ACCEPT 101,(CHAR(I),I=1,MAX)
101     FORMAT(100A1)
C  THE STRING WILL BE PADDED WITH SPACES (32)
C  FIND THE FIRST NON SPACE AND SET THE BYTE
C  AFTER IT TO 0.
        DO 20 I=MAX,1,-1
20      IF(CHAR(I).NE.32)GOTO 30
        CHAR(1)=0
        RETURN
30      CHAR(I+1)=0
        RETURN
        END
```

```
      FUNCTION LANS(ANS1,ANS2)
C
      LOGICAL*1 LANS,ANS1,ANS2
    5 READ(5,100)LANS
  100 FORMAT($, A1)
      IF((LANS.EQ.ANS1).OR.(LANS.EQ.ANS2))RETURN
      WRITE(5,200)ANS1,ANS2
  200 FORMAT(' PLEASE ANSWER ',A1,' OR ',A1,':',$)
      CALL TTYOUT ('$ ')
      GO TO 5
      END
C
      FUNCTION IANS(MIN,MAX)
    5 READ(5,100)IANS
  100 FORMAT(I6)
      IF((IANS.GE.MIN).AND.(IANS.LE.MAX))RETURN
      WRITE(5,200)MIN,MAX
  200 FORMAT(1X,'MIN=',I6,' AND MAX=',I6,'   TRY AGAIN:'$)
      GO TO 5
      END
```

```
      FUNCTION LASK (MSG)
C     THIS PRINTS MSG AS A PROMPT OF UP TO 70 CHARACTERS, THEN
C     ACCEPTS EITHER Y OR N AS A RESPONSE.
C
      LOGICAL*1 LANS, LASK, MSG(1)
C
      DO 10 I = 1, 70
      IF (MSG (I) .EQ. 0) GOTO 20
   10 WRITE (5, 100) MSG (I)
  100 FORMAT ($, 1H+, A1, $)
   20 LASK = LANS ('Y', 'N')
      TYPE 200
  200 FORMAT (/)
      RETURN
      END
```

```
      SUBROUTINE PUTSTR(CHAR,CEND)
      LOGICAL*1 CHAR(1),CEND

C  OUTPUT UP TO 70 CHARACTERS ON THE TTY:
C  IF CEND=$ THEN SURPRESS THE FINAL <CR>.
C
      DO 5 IC=1,70
5     IF(CHAR(IC).EQ.0)GOTO 6
      IC=71
6     IC=IC-1
      IF(CEND.EQ.'$')GOTO 8
      TYPE 1000,(CHAR(I),I=1,IC)
1000  FORMAT('+',70A1)
      RETURN
8     TYPE 1001,(CHAR(I),I=1,IC)
1001  FORMAT('+',70A1,$)
      RETURN
      END
```

```
      FUNCTION RANS(RMIN,RMAX)
C
C     FUNCTION TAKES A REAL NUMBER IN A SPECIFIC RANGE AS INPUT
C
 100  RANS = RGET()
      IF ((RANS .LT. RMIN) .OR. (RANS .GT. RMAX)) GOTO 200
      RETURN
 200  WRITE (5, 10) RMIN, RMAX
  10  FORMAT ('$MIN= ',1PE15.5,' AND MAX= ',1PE15.5,'  TRY AGAIN: ')
      GOTO 100
      END
```

```
        FUNCTION RGET()
C
C       FUNCTION TAKES A REAL NUMBER AS INPUT, CHECKING FOR VALIDITY
C
        LOGICAL*1 CHAR(25), ERR, STRING
  10    CALL STRING(CHAR,15,I)          !TAKE UP TO 15 CHARACTERS
        IF (CHAR(I) .EQ. 0) GOTO 20     !IMMEDIATE CR/LF NOT ALLOWED
        DECODE (I, 100, CHAR, ERR = 20) RGET
 100    FORMAT(F15.0)
        RETURN
C
C       ERROR IN INPUT...DEAL WITH IT
C
  20    TYPE 200
        GOTO 10
 200    FORMAT('0NOT A VALID REAL NUMBER.  TRY AGAIN: ', $)
        END
```

```
      SUBROUTINE STRING(CHAR,MAX,I)
C
C     STRING TAKES UP TO "MAX" CHARACTERS FROM THE TTY
C     END OF TEXT IS A NULL BYTE
C     THE CR/LF ISN'T INCLUDED IN THE TEXT
C
      LOGICAL*1 CHAR(1)
      ACCEPT 101, (CHAR(I), I=1,MAX)
 101  FORMAT(100A1)
C
C     THE STRING WAS AUTOMATICALLY PADDED WITH SPACES, SO NOW WE
C     GET TO GET RID OF THEM...
C
      DO 20 I=MAX,1,-1
  20  IF (CHAR(I) .NE. 32) GOTO 30
      CHAR(1) = 0
      RETURN
  30  CHAR(I+1) = 0
      RETURN
      END
```

```
      SUBROUTINE TTYIN(NLINE,NDIM,TITLE,LAST)
C
C     READS NLINE LINES OF TTY INPUT, CHARACTER BY CHARACTER,
C     AND STRINGS IT TOGETHER IN TITLE, SEPARATING EACH LINE
C     WITH A CARRIAGE RETURN & LINE FEED
C
C     ROUTINE READS A MAX OF (NDIM-2*NLINE-1) CHARACTERS,
C     WHERE NDIM IS DIMENSION OF TITLE
C
C     END OF TTY INPUT IS INDICATED BY A NULL CHARACTER
C     LAST POSITION IS RETURNED IN "LAST".
C
      LOGICAL*1 TEMP (71), TITLE (1), CRTURN, LNFEED, BLANK
C
      ISAVE=0
      NTEMP=NDIM-2
C
      DO 5 L=1,NLINE          !READ NLINE LINES
      WRITE(5,200)            !WRITE PROMPT CHARACTER
  200 FORMAT(/, '+!'$)
      CALL GETSTR (TEMP, 70)     !READ ONE TTY LINE OF UP TO 70
C                             !CHARACTERS; TERMINATE WITH NULL
      DO 10 I=1,71            !LOAD TEMP INTO TITLE
      ITEMP=I+ISAVE
      IF((ITEMP.GE.NTEMP).OR.(TEMP(I).EQ.0))GO TO 15
C                                !CHECK FOR FULL TITLE VECTOR OR
C                                !NULL CHARACTER IN TEMP INPUT
   10 TITLE(ITEMP)=TEMP(I)
   15 TITLE(ITEMP)=CRTURN
      ITEMP=ITEMP+1
      TITLE(ITEMP)=LNFEED
      ITEMP = ITEMP + 1
      TITLE(ITEMP)=BLANK
      IF(ITEMP.GE.NTEMP)GO TO 20    !QUIT IF TITLE IS FILLED
      ISAVE=ITEMP                   !SAVE LAST LOADED POSN
    5 CONTINUE                   !BOTTOM OF LINE LOOP
   20 ITEMP=ITEMP+1
      TITLE(ITEMP)=0
      LAST = ITEMP
      RETURN
C
      DATA CRTURN, LNFEED, BLANK /13, 10, 32/
C
      END
```

```
      SUBROUTINE TTYOUT (MSG)
C
C
C     This subroutine outputs the string MSG onto the user's
      terminal.
C     if there is a leading dollar sign in the string, the initial
C     carrige return/line feed is supressed.  A trailing dollar sign
C     supressed the CR/LF.
C
C     Last Modification Date:  12-July-83
C
      LOGICAL*1 MSG(1), DOLLAR, CRTURN, LNFEED, EOF
C
      ISTART = 1
      IF (MSG(ISTART) .NE. DOLLAR) GOTO 5  ! Check if user wants
       CR/LF
      ISTART = ISTART + 1  ! $ is there, message begins at next
       character
      GOTO 10
C
    5 WRITE (5, 100) CRTURN, LNFEED  ! No $, print CR/LF
   10 IF (MSG(ISTART) .EQ. EOF) RETURN  ! Null msg.
        Returns to main prog.
C
C     This next loop goes through the string until it encounters an
C     End Of File indicator in order to find the terminating
     position
C     in the string.
C
      ISTOP = ISTART
   15 ISTOP = ISTOP + 1
      IF (MSG(ISTOP) .NE. EOF) GOTO 15
C
      ISTOP = ISTOP - 1  ! Get index of the last character
      IF (MSG(ISTOP) .EQ. DOLLAR) ISTOP = ISTOP - 1
      IF (ISTART .GT. ISTOP) RETURN  ! Quit if double dollar sign
C
      DO 25 I = ISTART, ISTOP
   25 WRITE (5, 100) MSG(I)
C
      IF (MSG(ISTOP+1) .EQ. DOLLAR) RETURN
      WRITE (5, 100) CRTURN, LNFEED  ! Do CR/LF if no ending $
      RETURN
C
  100 FORMAT ($,1H+,A1,$)
      DATA DOLLAR, CRTURN, LNFEED, EOF /'$', 13, 10, 0/
      END
```

```
      SUBROUTINE VECTIN(MODE,VECNAM,VECDIM,VECTOR,VECMIN,VECMAX)
C
C     LOADS A VECTOR VARIABLE (VECTOR) COMPONENT BY
C     COMPONENT FROM THE TTY, IN A PROMPTING MODE,
C     CHECKING THAT THE TTY INPUT VALUE IS BETWEEN VECMIN
C     AND VECMAX.
C     VECNAM IS A ONE-CHARACTER LITERAL ASSOCIATED WITH THE
C     VECTOR, AND VECDIM IS THE VECTOR'S DIMENSION; BOTH
C     ARE ASSUMED SUPPLIED BY THE CALLING ROUTINE.
C     WHEN MODE=1 SEQUENTIAL ENTRY & CORRECTION ARE DONE
C               =2 CORRECTION ONLY IS DONE
C
      LOGICAL*1 LASK,VECNAM
      INTEGER VECDIM
      DIMENSION VECTOR(1)
C
      GO TO(5,15)MODE
      STOP'*****VECTIN:ILLEGAL VALUE FOR MODE*****'
C
    5 DO 10 J=1,VECDIM          !READ-IN SECTION
      I = J
   10 VECTOR (I) = VECVAL (I, VECNAM, VECMIN, VECMAX)
C
      CALL TTYOUT (' ')
      IF (LASK ('ANY CHANGES? ') .EQ. 'N') RETURN
C
   15 CALL TTYOUT('ENTER COMPONENT INDEX') !CORRECTION SECTION
   20 CALL TTYOUT('I=$')
      I=IANS(1,VECDIM)
      VECTOR (I) = VECVAL (I, VECNAM, VECMIN, VECMAX)
      CALL TTYOUT (' ')
      IF (LASK ('MORE? ') .EQ. 'Y') GOTO 20
      RETURN
C
      END
C
C
C
      FUNCTION VECVAL (I, VECNAM, VECMIN, VECMAX)
C
      LOGICAL*1 VECNAM
C
    5 WRITE (5,100) VECNAM, I
  100 FORMAT(1X,A1,'(',I2,')=$)
      VECVAL = RANS(VECMIN, VECMAX)
      END
```

APPENDIX E
MACRO LIBRARY

A list of the assembly-language routines, along with brief descriptions of their functions, are included in Table E.1. Listings follow.

TABLE E.1 UTLLIB ROUTINES

CLSTOP	Stops the Clock
CLSTRT	Sets clock A to repeated interval mode, presets the buffer to an integer value set by the calling routine, and starts the clock
CLWAIT	Determines clock status upon enter. If the clock has already timed out, a flag is set to indicate a "bad interval", and control is returned to the calling routine. Otherwise, the flag is set of a "good interval", and a wait loop is continued until the clock times out.
ATOD	Samples a single A/D channel as specified in the calling routine, converts the sampled voltage into an integer between 0 and 4095, and returns this integer to the calling routine.
DTOA	Accepts integer value between 0 and 4095 from calling routine and does D/A conversion for a single channel (specified by calling routine)
RNUM	Generates and returns to the calling routine a vector of N random integers
RNSEED	Accepts from or returns to the calling routine the seed number used by RNUM

```
        .TITLE ATOD
;       SUBROUTINE ATOD(ICHAN,IDATA)
;
;       IN FILE ATOD.MAC
;
;       ICHAN SPECIFIES CHANNEL NO. FROM 0 TO 15
;       IDATA IS DATA WORD, BETWEEN 0 AND 4095,
;       INCLUSIVE
;
;
        .GLOBL  ATOD

;       HPL/SAT DEFINITION    (!!!COMMENT OUT FOR MNC!!!)
        LPSADS = 170400          ;A/D CONVERTER STATUS

;       MNC     DEFINITION    (!!!COMMENT OUT FOR HPL/SAT!!!)
;       LPSADS = 171000          ;A/D CONVERTER STATUS

;       COMMON DEFINITION
        LPSADB = LPSADS+2        ;A/D CONVERTER BUFFER

ATOD:   TST     (R5)+            ; SKIP PAST PARAMETER COUNT
        MOV     (R5)+,R0         ; GET ADDRESS OF BUFFER
        MOV     (R5),R1          ; GET DATA BUFFER ADDRESS
        CLR     @#LPSADS        ; INITIALIZE CONVERTER
        MOV     (R0), R2         ; GET CHANNEL NUMBER
        ASH     #10,R2           ; SHIFT TO LEFT BYTE
        MOV     R2,@#LPSADS
        INC     @#LPSADS        ; START CONVERSION

1$:     TSTB    @#LPSADS        ; WAIT FOR CONVERSION
        BPL     1$               ; TO FINISH

        TSTB    @#LPSADS+1      ; CHECK FOR CONVERSION
        BMI     2$               ; ERROR

        MOV     @#LPSADB,(R1)   ; SAVE DATA IN BUFFER
        RTS     PC

2$:     MOV     #-1,(R1)         ; FORCE ERRONEOUS DATA TO -1
        RTS     PC

        .END
```

```
        .TITLE DTOA
;       SUBROUTINE DTOA(ICHAN,IDATA)
;
;
;       ICHAN SPECIFIES CHANNEL NO. FROM 0 TO 5
;       IDATA IS DATA WORD, ASSUMED BETWEEN ZERO AND
;       4095 INCLUSIVE
;

        .GLOBL DTOA

;       HPL/SAT DEFINITION          (!!!COMMENT OUT FOR MNC!!!)
        EXTDA=170420

;       MNC DEFINITION              (!!!COMMENT OUT FOR HPL/SAT!!!)
;       EXTDA=171060

DTOA:   TST     (R5)+               ;SKIP ARGUMENT COUNT
        MOV     @(R5)+,R0          ;GET CHANNEL NUMBER
        ASL     R0                  ;AND MPY BY 2
        MOV     @(R5)+,EXTDA(R0);LOAD DA
        RTS     PC

        .END
```

```
        .TITLE  CLOCK
;SIMPLE MSEC CLOCK ROUTINES FOR LPS-11 ON HPL, SAT, MNC

;HPL/SAT DEFINITIONS     (!!!COMMENT OUT FOR MNC!!!)
STATUS= 170404
MODE1=  400
RATSHF= 1       ;NUMBER OF BITS RATE MUST BE LEFT-SHIFTED

;MNC DEFINITIONS         (!!!COMMENT OUT FOR HPL/SAT!!!)
;STATUS=171020
;MODE1= 2
;RATSHF=3       ;NUMBER OF BITS RATE MUST BE LEFT-SHIFTED

;COMMON DEFINITIONS
PRESET= STATUS+2  ;NO INTERRUPT VECTORS USED
RUN=    1
DONEFL= 200

;CLSTRT(IRATE,NTICKS): SET CLOCK FOR NTICKS AT IRATE, MULTIPLE
    INTERVAL MODE
;IRATE: 1=1MHZ, 2=100KHZ, 3=10KHZ, 4=1KHZ, 5=100HZ, 6=SCHMITT-
    TRIGGERED, 7=LINE
CLSTRT::CLR     STATUS                  ;CLEAR ANY EXISTING STATE
        TST     (R5)+           ;SKIP ARG COUNT
        MOV     @(R5)+,R1       ;GET RATE
        ASH     #RATSHF,R1      ;SHIFT TO REQUIRED POSITION

        BIS     #MODE1+RUN,R1   ;SET MODE, RUN BITS
        MOV     @(R5)+,R0       ;GET NO OF CLOCK TICKS IN PERIOD
        BEQ     CLSX            ;DO NOWT IF NO TICKS..
        NEG     R0
        MOV     R0, PRESET      ;SET COUNTER
STMOD1: MOV     R1, STATUS
CLSX:   RTS     PC

;LOGICAL FUNCTION CLWAIT() RETURNS R0 .FALSE. IF TIMED-OUT ON ARRIVAL,
;  ELSE, WAITS TILL CLOCK TIMES OUT, RETURNS R0 .TRUE. FOR GOOD
   INTERVAL
CLWAIT::CLR     R0              ;SET FLAG FOR BAD INTERVAL
        BIT     #DONEFL, STATUS         ;ARE WE DONE?
        BNE     WAITX           ;YES
        BIT     #RUN, STATUS    ;IS AN INTERVAL SET UP?
        BEQ     WAITX                   ;NO: ABORT
WTLOOP: BIT     #DONEFL, STATUS         ;YES: WAIT FOR DONE FLAG
        BEQ     WTLOOP
        COM     R0              ;FLAG GOOD INTERVAL
WAITX:  BIC     #DONEFL, STATUS
        RTS     PC
```

```
;CLSTOP() STOPS CLOCK DEAD
CLSTOP::CLR      STATUS
        RTS      PC
        .END
```

```
        .TITLE  RANDOM

        .GLOBL  RNUM,RNSEED

;       SUBROUTINE RNUM(IRAN,N)
;
;       ROUTINE TO GENERATE A VECTOR OF N RANDOM INTEGERS: IRAN.

RNUM:   TST     (R5)+           ; SKIP ARG COUNT
        MOV     (R5)+,R3        ; GET ADDRESS OF DATA VECTOR
        MOV     @(R5)+,R4       ; GET VECTOR LENGTH
NEXT:   MOV     RNLOW,R0        ; GET COPY OF LOW RN
        MOV     RNHIGH,R1       ; GET COPY OF HIGH RN
        MOV     R0,RNHIGH       ; NEW HIGH RN FORMED
        ASL     R0              ; SHIFT LOW RN LEFT 2
        ASL     R0
        XOR     R0,R1           ; EXCLUSIVE OR OF 31 & 13
        CLR     R0
        ASHC    #1,R0           ; SHIFT R0,R1 LEFT 1
        BIS     R0,RNHIGH       ; MOVE (31!13) INTO BIT 0
        MOV     R1,RNLOW        ; MOVE (30!12) (16!29) INTO RNLOW
        ASHC    #2,R0           ; SHIFT R0,R1 LEFT 2
        ASL     R0              ; SHIFT TO ZERO BIT 0
        XOR     R0,RNLOW        ; EXCLUSIVE OR FOR LOW-ORDER BITS
        MOV     RNHIGH,(R3)+    ; STORE RANDOM INTEGER
        SOB     R4,NEXT         ; DONE YET?
        RTS     PC              ; YES, RETURN.

RNLOW:  .BLKW
RNHIGH: .BLKW

;       SUBROUTINE RNSEED(ILOW,IHIGH)
;
;       ROUTINE TO SET OR RETRIEVE SEED NUMBER FOR RANDOM INTEGER
;       GENERATOR. IF BOTH ILOW=IHIGH=0, THE CURRENT SEED VALUES ARE
;       RETURNED IN ILOW AND IHIGH. OTHERWISE THE VALUES OF ILOW AND
;       HIGH ARE USED TO SET THE SEED.
;
;       NOTE: ILOW AND IHIGH CAN BE POSITIVE OR NEGATIVE INTEGERS, BUT
;       ILOW MUST BE EVEN.
;

RNSEED: TST     (R5)+           ; SKIP ARG CNT
        MOV     @(R5)+,R1       ; GET ILOW
        MOV     @(R5)+,R2       ; GET IHIGH
        BNE     SETSD           ; CHECT FOR ZERO ARG
```

```
        TST     R1
        BNE     SETSD
        MOV     RNHIGH,     ; BOTH Ø, SO RETRIEVE CURRENT SEED
                    R5
        MOV     RNLOW,
                    R5
        RTS     PC
SETSD:  MOV     R1,RNLOW        ; STORE NEW SEED
        MOV     R2,RNHIGH
        RTS     PC

        .END
```

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-172311		

4. Title and Subtitle	5. Report Date
The VERRUN and VERNAL Software Systems for Steady-State Visual Evoked Response Experimentation	March 1984
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
William H. Levison and Greg L. Zacharias	5486
	10. Work Unit No.
9. Performing Organization Name and Address Bolt Beranek and Newman Inc. 10 Moulton Street Cambridge, MA 02238	11. Contract or Grant No. NAS1-16982
	13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code 505-35-33-01

15. Supplementary Notes

Langley Technical Monitor: Alan T. Pope
Final Report

16. Abstract

Two digital computer programs have been developed for use in experiments involving steady-state visual evoked response (VER): VERRUN, whose primary functions are to generate a sum-of-sines (SOS) stimulus and to digitize and store electro-cortical responses; and VERNAL, which provides both time- and frequency-domain metrics of the evoked response. These programs have been coded in FORTRAN for operation on the Digital Equipment Corporation PDP-11/34, using the RSX-11 Operating System, and the PDP-11/23, using the RT-11 Operating System. Users' and programmers' guides to these programs are provided, and guidelines for model analysis of VER data are suggested.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Bioinstrumentation (physiological) Biotechnology Man-machine interface	Unclassified - Unlimited Subject Category 54

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	221	A10