**PROGRAMS FOR DATA MANIPULATION AND ANALYSIS
ON THE EBR-II SIGMA 5 COMPUTER**

by

H. A. Larson

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

**EBR-II Project
Argonne National Laboratory
Argonne, Illinois - Idaho Falls, Idaho**

February 1974

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

19

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT	7
I. INTRODUCTION	7
II. MATHEMATICAL BASES FOR PROGRAMS.	9
A. Data-smoothing Methods	9
1. Polynomial Filter.	10
2. Finite-difference Filter	13
B. Data-integration and Data-interpolation Methods.	15
1. Quadrature Integration Schemes	15
a. Using Fixed Intervals.	15
b. Using Weighted Intervals	16
2. Monte Carlo Integration.	18
3. Lagrange Interpolation	19
C. Data-analysis Methods.	20
1. Fast Fourier Transform	20
2. Crosscorrelation and Autocorrelation	21
3. Spectral Density and Convolution	21
4. Pseudorandom Binary Generator.	23
III. PROGRAM DESCRIPTIONS AND EXAMPLES.	24
A. Data-smoothing Programs.	24
1. SMOOTH, an Nth-order Polynomial Smoothing Routine.	24
2. FILTER, an Nth-order Digital Filter.	25
3. TCOEF, the FILTER Transfer-function Routine.	27
B. Data-integration and Data-interpolation Programs	31
1. Numerical Quadrature	31
a. TRAP, Trapezoidal-rule Integration	31
b. GLEG, Gauss-Legendre Integration	31
c. GLAG, Gauss-Laguerre Integration	32
2. MONTE, a Monte Carlo Integration Routine	32
3. YLAG, a Lagrangian Interpolation Routine	33

TABLE OF CONTENTS

	<u>Page</u>
C. Data-analysis Programs	34
1. FFT1, a Fast Fourier Transform Program	34
2. CORR, a Crosscorrelation and Autocorrelation Program.	34
3. PRB, a Pseudorandom Binary Generator	36
APPENDIX	
Program Listings	37
ACKNOWLEDGMENTS	49
REFERENCES	51

LIST OF FIGURES

<u>No.</u>	<u>Title</u>	<u>Page</u>
1.	Transfer Function for Zero-phase Polynomial Filter . . .	12
2.	Effect of SMOOTH Filtering of 20-Hz Signal Superposed on the Curve $f(x) = (1 + x^2)^{-1}$	26
3.	Effect of Removing 10-Hz Signal from Curve $f(t) = \sin(\omega t)$ + $(1/2) \sin(2\omega t/3)$ with Code FILTER.	28
4.	Relationship of Phase and Magnitude to Frequency for Notch Filter That Produced Results Shown in Fig. 3 . . .	30

LIST OF TABLES

<u>No.</u>	<u>Title</u>	<u>Page</u>
I.	Constants Defining First- and Second-order Digital Filters.	14
II.	Results of Monte Carlo Integration	35
III.	Time Required for YLAG Interpolation of Function $y = \sin x$	35

PROGRAMS FOR DATA MANIPULATION AND ANALYSIS
ON THE EBR-II SIGMA 5 COMPUTER

by

H. A. Larson.

ABSTRACT

This report provides mathematical description, input instructions, sample programs, and program listings for data-manipulation and data-analysis computer programs specifically designed for use as analysis aids with the EBR-II Sigma 5 computer system. Though not completely general and all-encompassing, this document may serve as a user's manual for EBR-II personnel. Included are digital filters, simple integration and interpolation schemes, frequency-domain-analysis schemes, and a generator for ~~pseudorandom~~ binary signals.

I. INTRODUCTION

Analog data associated with EBR-II are obtained from temperature sensors, ion chambers, and other instruments. These data contain two types of oscillations. One type, generally known as random noise, is the random displacement of the signal from an average value. The second is a flow-induced oscillation at frequencies in the range 10-12 Hz, which usually causes a beat with a period of about 2 sec to appear superposed on the signal.

These oscillations are objectionable primarily because they interfere with processing of data obtained by oscillator and rod-drop experiments.¹ For example, when the experimental perturbing device has small reactivity worth, the magnitude of the oscillations may be 30% of that of the signal. As a result: (1) a realistic value for the dependent variable is difficult to determine from observation of either the data

or a graph of the data; (2) further mathematical processing of the data is frustrated by rapid changes in the dependent variable with small changes in the independent variable.

Of first importance is the processing of analog data from the power chamber (the ion chamber measuring reactor power) during rod-drop experiments.² These data reflect both the random noise and the 10-Hz oscillations. Reactivity of the reactor is determined by applying an inverse-kinetics procedure.³ The inverse kinetics are applied to the data from both the power and zero-power rod-drop tests, and the resultant time-dependent reactivities are subtracted from one another to give the feedback reactivity.² Unfortunately, at this stage of the computations, the magnitude of the oscillations in the feedback is seldom less than 10% of the magnitude of the feedback reactivity.

Noise may be eliminated from a signal by an electronic (analog) filter, an electronic circuit that passes only the part of the signal in the desired frequency range. A more economical method, however, is the digital filter. The modern high-speed computer allows fast, efficient digital filtering to be applied to data while the computer is on line (i.e., while the data are being collected). Such a filter is known as a real-time digital filter.

Digital filters have several advantages over continuous analog filters. Since the accuracy of the digital filter depends on word length and sampling rate, the degree of filter accuracy can be controlled. The variables are controlled by the designer and may be made as accurate as desired; accuracy of analog filters, on the other hand, depends upon their components. Since a single computer may be used for a large number of digital filters, filter designs may be easily changed by programming, thus allowing a greater variety of filters. Lastly, since these filters are programmed, no special components are needed for time-varying filters.

Besides filtering procedures, interpolation programs and integration programs are often required. Interpolation procedures are usually polynomial-curve-fitting procedures used to evaluate a function (described by an array of ordered pairs) at some point not in the array. Integration programs refer either to an array or to an analytically expressed function; they provide an integration based on specified nodes (either equally spaced or optimally spaced) and corresponding weights. The integrations may or may not be used

in conjunction with interpolation routines. When multiple integrations are required, Monte Carlo integration schemes are sometimes easier to use.

This report describes the data-smoothing, interpolation, and integration schemes available in the EBR-II Sigma 5 computer system.⁴⁻⁷ It also lists program descriptions and gives example problems to show the capabilities of the various programs. Although not completely general and all-encompassing, the report may serve as a user's manual for EBR-II personnel.

Several programs for data analysis or signal generation are available on the Sigma 5 computer. These are a fast Fourier transform program, a pseudorandom binary generator program, and autocorrelation, crosscorrelation, and spectral-density programs for noise analysis. Input requirements for these programs, as well as examples of their use, are described in this report.

Many of the programs are modifications of those which already have been described in the literature. The explanations are well documented so that one may turn to the original source if background material is required. The programs are, however, self-contained and as versatile as possible, but simple enough so as not to put undue difficulty on the user. Every effort has been made to keep the programs simple, direct, and as internally compatible with each other as possible.

II. MATHEMATICAL BASES FOR PROGRAMS

This section discusses the mathematical bases for the data-smoothing, interpolation, and integration programs available for use in the Sigma 5 computer system. Transfer functions are derived, where applicable, so that users may easily determine the constants for their filtering (smoothing) programs.

A. Data-smoothing Methods

Data-smoothing programs on the Sigma 5 computer system are basically of two types: a filter routine that uses polynomial least-squares techniques, and a filter routine that uses finite-difference-equation techniques.

1. Polynomial Filter

One of the more elemental digital smoothing techniques is that which uses the least-squares fitting process.⁸ With this technique, a least-squares polynomial fit⁹ of order $M = 2m + 1$ (where m is an integer) is made to N equally spaced data points (with $N \geq M$) according to

$$f(x_i) = \sum_{k=0}^M b_k x_i^k = \sum_{k=\lfloor \frac{M}{2} \rfloor}^{\lfloor \frac{M}{2} \rfloor} b'_k y_i^k, \quad (1)$$

where $f(x_i)$ represents the fitted value of the function at the i th data point (x_i) , $i = 1, 2, \dots, N$, and b'_k and y_i are chosen such that b'_0 represents $f(x_i)$. The fitting process is repeated incrementing i and retaining M points for each curve fit. If sufficient smoothness is not attained with one pass across the N data points, a second pass (or more) may be used.

Although this procedure is efficient for high-frequency noise, it becomes less efficient with decreasing noise frequency. The reason is that a larger number of points, and a corresponding longer computer running time, must be used in the least-squares process when smoothing long-period oscillations. Therefore, a more selective smoothing technique should be used at the lower frequencies.

The transfer function associated with a least-squares smoothing procedure is discussed in Ref. 8. When we minimize the function

$$\frac{1}{\delta t} \int_{t-\delta t}^{t+\delta t} \left\{ f(\tau) - \left[A(t) (t-\tau)^2 + B(t) (t-\tau) + C(t) \right] \right\}^2 d\tau, \quad (2)$$

where $f(t)$ is the input function, $F(t) \equiv a(t)t^2 + b(t)t + c(t)$ is the filtered function, and $A(t) \equiv a(t)$, $B(t) \equiv b(t) + 2a(t)t$, $C(t) \equiv a(t)t^2 + b(t)t + c(t)$, then the frequency-response transfer function for a unit impulse at $t = \delta t/2$ becomes

$$G(j\omega) = -\frac{30}{\omega^2 \delta t^2} \cos\left(\frac{\omega \delta t}{2}\right) - \left(\frac{3}{\omega \delta t} - \frac{60}{\omega^3 \delta t^3}\right) \sin\left(\frac{\omega \delta t}{2}\right), \quad (3)$$

where

$$j = \sqrt{-1}$$

and

ω = frequency in radians.

This transfer function is for a zero-phase, low-pass filter with 3-dB attenuation at $\omega \delta t \approx 6.6$. After n passes have been made across the data, $G_n(j\omega) = [G(j\omega)]^n$. Figure 1 shows a plot of $|G^n|$ for $n = 1, 2$, and 3. There it is seen that this polynomial filter is a low-pass digital filter.

A point of interest to users is the fact that if the degree, M , of the polynomial in Eq. (1) and the number, N , of sampled points are increased indefinitely for each successive smoothed ordinate, the smoothed data approach those that would be obtained by using the successive values of $\text{sinc}(2Ni/m)$, where $\text{sinc}(x) = \sin(\pi x)/(\pi x)$.¹⁰ When $N < M$, the evaluated coefficient b'_0 of Eq. (1) is

$$b'_0 = \sum_{i=-\infty}^{\infty} y_i \frac{2N}{\pi m} \text{sinc}\left(\frac{2N}{\pi m} i\right).$$

Thus, as N and M are increased indefinitely, the smoothing obtained is equivalent to passing the data through an ideal low-pass filter.

It has been suggested¹⁰ that an alternative to polynomial smoothing is

$$b'_0 = \beta^{-1} \sum_{i=-q}^q y_i \text{sinc}(\alpha i),$$

Fig. 1. Transfer Function for Zero-phase Polynomial Filter

where

$$2/\pi < \alpha < 1$$

and

$$\beta = \sum_{i=-q}^q \text{sinc}(\alpha i)$$

for some appropriate value of q .

2. Finite-difference Filter

An alternative, and more selective, digital filter is the N th-order, recursive digital filter described by

$$Y(nT) = \sum_{i=0}^N a_i X(nT - iT) + \sum_{k=1}^M b_k Y(nT - kT), \quad (4)$$

where $n = 0, 1, \dots, I$ represents the data points equally spaced in units of sampling interval T , $X(nT)$ are the raw data, and $Y(nT)$ are the smoothed data. Table I lists the values of the constants a_i and b_k for several special cases of first- and second-order filters.¹¹ The table shows that several types of filters may be easily constructed, including bandpass and notch types. Judicious use of the constants will produce a satisfactory filtering process when they are properly evaluated.

As an aid to filter evaluation, the frequency-response function using constant coefficients a_i and b_i is given by

$$T(j\omega) = \frac{\sum_{r=0}^N \sum_{q=0}^M a_r b_q^* [\cos(q-r)\omega T + j \sin(q-r)\omega T]}{\sum_{r=0}^N \sum_{q=0}^M b_r^* b_q^* \cos(q-r)\omega T}, \quad (5)$$

TABLE I. Constants Defining First- and Second-order Digital Filters

Filter Type		a_0	a_1	a_2	b_1	b_2
First Order	Low Pass	$(k/2) (1 - r)$	a_0	0	r	0
	High Pass	$(k/2) (1 - r)$	$-a_0$	0	r	0
Second Order	Low Pass	$(k/2) (1 - 2r\cos\theta + r^2)$	$2a_0$	a_0	$2r\cos\theta$	$-r^2$
	High Pass	$(k/2) (1 + 2r\cos\theta + r^2)$	$-2a_0$	a_0	$2r\cos\theta$	$-r^2$
	Band-pass	$(k/2) (1 - r) \csc(\theta/2) \cdot \sqrt{1 - 2r\cos 2\theta + r^2}$	0	$-a_0$	$2r\cos\theta$	$-r^2$
	Notch	$(k/4) \csc^2(\theta_0/2) \cdot (1 - 2r\cos\theta + r^2)$	$-2a_0 \cos\theta_0$	a_0	$2r\cos\theta$	$-r^2$
	Moving Averager	$1/3$	$1/3$	$1/3$	0	0
Time-varying Averager		$1/n$	0	0	$\frac{n-1}{n}$	0
Time-varying Averager with Notch		$\frac{1}{2(1 - \cos\omega_0 T)n}$	$-2a_0 \cos\omega_0 T$	a_0	$\frac{n-1}{n}$	0

Definitions: $(1 - |r|)$ = damping factor $(-1 < r < 1)$
 T = sampling interval
 $\theta/(2\pi T)$ = cut-off or resonant frequency $(0 \leq \theta < \pi)$
 $\theta_0/(2\pi T)$ = ω_0 = notch frequency $(0 < \theta_0 < \pi)$
 k = arbitrary gain constant
 nT = sampling time

where $b_1^1 = -b_1$, and $b_0^1 = 1$.

This transfer function represents, in general a non-zero-phase digital filter that requires care when used, so that any undesired phase transformation is avoided. Thus, evaluation of the transfer function, by Eq. (5), may be necessary prior to using the digital filter; the parameters may be adjusted until the desired characteristic is obtained.

B. Data-integration and Data-interpolation Methods

The data-integration and data-interpolation programs discussed here are sufficient to cover most routine applications. The integration routines are of three general types: fixed interval, weighted interval, and probabilistic (Monte Carlo) in nature. The one interpolation program used allows use of any type of tabulated data in any of the integration routines.

1. Quadrature Integration Schemes

The two quadrature schemes -- fixed interval and weighted interval -- were selected so as to cover most routine applications of numerical integration.

a. Using Fixed Intervals

This scheme applies the trapezoidal rule to a data set of fixed intervals. For a table of $N + 1$ ordered pairs (x_i, y_i) , where $i = 0, 1, \dots, N$, the trapezoidal sum of an integral is given by

$$\sum_{i=1}^N y_i = \int_a^b y(x) dx, \quad (6)$$

where

$$y_i = \int_{x_{i-1}}^{x_i} y(x) dx = \frac{h}{2} (y_i + y_{i-1}),$$

with truncation error given by

$$R = - \frac{N-1}{12} h^3 y'''(\xi), \quad \xi \in (a,b),$$

$x_i = a + (i-1)h$, and h is the integration mesh spacing.

b. Using Weighted Intervals

The Gauss-Legendre quadrature integration uses weights determined from optimally placed nodes in the interval of integration (a,b) . The nodes are more closely spaced nearer the end points of the integral, and the function must be tabulated, or otherwise evaluated, at each of the nodes. A Lagrange interpolation scheme (see Sec. II.B.2 below) is used to evaluate tabulated functions at specified node points.

The integral to be evaluated,

$$Y = \int_a^b F(x) dx,$$

is approximated by

$$Y = \frac{b-a}{2} \sum_{i=1}^N A_i F\left[\frac{b-a}{2} x_i + \frac{b+a}{2}\right] + R_N, \quad (7)$$

where

$$A_i = 2 / \left\{ (1-x_i)^2 P_n'(x_i)^2 \right\},$$

x_i = the i th zero of the Legendre polynomial $P_n(x)$,

and

$$R_N = \frac{(b-a)^{(2N+1)} (N!)^4}{(2N+1) [(2N)!]^3} {}_2F^{(2N+1)}(\xi), \quad |\xi| < 1.$$

The values of A_i and x_i , with $i = 1, 2, \dots, N$, may be computed by the formulas shown, but they are tabulated quite extensively in the literature^{12,13} and thus may be conveniently expressed to any desired order of accuracy up to $N = 81$ (Ref. 12) or $N = 96$ (Ref. 13).

The Gauss-Laguerre quadrature is similar to the Gauss-Legendre quadrature, except that the weighting function in the infinite interval is e^{-x} rather than unity. The integral

$$Y = \int_a^\infty F(x) dx$$

is approximated by

$$Y = \sum_{i=1}^N W_i e^{y_i} F(y_i + a) + R_N, \quad (8)$$

where

$$W_i = \frac{(N!)^2 y_i}{(N+1)^2 [L_{N+1}(y_i)]^2},$$

y_i = the i th zero of the Laguerre orthogonal polynomial $L_N(y)$,

and

$$R_N = \frac{(N!)^2}{(2N)} F^{(2N)}(\xi), \quad 0 < \xi < \infty.$$

As before, the nodes and weights of Eq. (8) may be calculated by the formulas shown; however, these values are tabulated to $N = 15$ in Ref. 13. For more accuracy, a computer program provided in Ref. 14 may be used up to about $N = 30$ or modified in double precision to $N = 64$.

2. Monte Carlo Integration

Mathematical or scientific procedures sometimes require an approximate value for a multiple integration with several variables. Numerical integration schemes are restricted as to type of integrand allowed and may be quite involved. An alternative approach for these types of integrals is a Monte Carlo approach, a program which also is available on the Sigma 5 computer. The basis for the Monte Carlo approach is as follows:

Assume that it is required to evaluate

$$Y = \int_{a_N}^{b_N} \dots \int_{a_1}^{b_1} F(x_1, x_2, \dots, x_N) dx_1 dx_2 \dots dx_N, \quad (9)$$

where F is real, finite, and single-valued. Equation (9) is evaluated by the formula

$$V = \frac{S}{k} (\max F - \min F) \prod_{i=1}^N (b_i - a_i), \quad (10)$$

where V is the proportion of successes multiplied by the volume of the hypersolid defined by $[a_i, b_i]$, $\max F$ is the maximum value of F , and k is the number of trials.

To determine S , select a pseudorandom point $(x_1, x_2, \dots, x_N, Y)$ such that $x_i \in [a_i, b_i]$ and $Y \in [\min F, \max F]$. Then for

$$\left. \begin{array}{l} F(x_1, x_2, \dots, x_N) \leq 0, \\ Y \leq 0, \text{ and} \\ F \leq Y, \end{array} \right\} \text{ decrease } S \text{ by } 1;$$

and for

$$\left. \begin{array}{l} F(x_1, x_2, \dots, x_N) > 0, \\ Y > 0, \text{ and} \\ F > Y, \end{array} \right\} \text{ increase } S \text{ by } 1;$$

otherwise count one failure and do not increment S . As k increases without limit, $V \rightarrow Y$.

3. Lagrange Interpolation

For versatility with the integration routines, a Lagrangian interpolation routine is included in the Sigma 5 programs. Given a set of ordered pairs (X_i, Y_i) with or without equal spacing of the array of the independent variable X_i , a dependent variable $\bar{Y}$ is determined from any $\bar{X}$ by either interpolation or extrapolation according to the formula

$$F(\bar{X}) = \sum_{k=1}^K Y_k \prod_{\substack{j=1 \\ j \neq k}}^K \left[\frac{(\bar{X} - X_j)}{(X_k - X_j)} \right], \quad (11)$$

where K is the number of points to which one desires a fit.

The routine searches the independent-variable array until $\bar{X}$ is located between two adjacent X_i , or $\bar{X} < X_1$, or $\bar{X} > X_N$. The binary search method is used to optimize the location of an arbitrary $\bar{X}$ in the independent-variable set.

C. Data-analysis Methods

Several programs are available for frequency-domain analysis of data. This type of analysis is generally used in connection with noise analysis or with stochastic analyses¹⁵ of fluctuating signals. To maintain the integrity of the equations in these analyses, the signals are generally assumed to be stationary and ergodic.

1. Fast Fourier Transform

The fast Fourier transform (FFT) is an efficient way of computing a discrete Fourier transform (DFT), which is defined by¹⁶

$$A_r = \sum_{k=0}^{N-1} X_k e^{-2\pi jrk/N} \quad (12)$$

$$r = 0, 1, \dots, N-1,$$

where A_r is the rth coefficient of the DFT, X_k denotes the kth sample of the time series of N samples, $j = \sqrt{-1}$, and r is sometimes called the frequency of the DFT.

The inverse of Eq. 12, known as the IDFT, is given by

$$X_\ell = \frac{1}{N} \sum_{r=0}^{N-1} A_r e^{2\pi jr\ell/N} \quad (13)$$

$$\ell = 0, 1, \dots, N-1.$$

Its validity may be verified by the relation

$$\sum_{r=0}^{N-1} e^{2\pi j(n-m)r/N} = N$$

for $n = m \text{ Mod } N$; otherwise the sum is zero. It is assumed that $X(t)$ is known at N equally spaced points: $0, T/N, 2T/N, \dots, (N-1)T/N$, where T is the period. The frequency is given by $\omega = 2\pi/T$, and mesh size is given by $k = T/N$.

2. Crosscorrelation and Autocorrelation

Crosscorrelation and autocorrelation functions are of interest to reactor physicists and engineers primarily because of the significance of the functions in statistical control theory. Crosscorrelation is a calculation of the degree of interrelationship between two random variables; autocorrelation is a corollary of crosscorrelation and, as its name implies, is a measure of the degree of correlation within one function. The literature abounds with theories of these mathematical definitions, both discrete and continuous, so no more will be said here about their function.^{15,17}

Crosscorrelation and autocorrelation functions are given by

$$\phi_{fg}(k) = \frac{1}{N-kh} \sum_{i=0}^{N-kh} F_i G_{i+kh} \quad (14)$$

and

$$\phi_{ff}(k) = \frac{1}{N-kh} \sum_{i=0}^{N-kh} F_i F_{i+kh}, \quad (15)$$

respectively, where $k = 0, 1, \dots, N$, and h is an integer such that the lag is related to the time step in the fashion $t = h\delta t$.

3. Spectral Density and Convolution

Spectral density is a measure of the total spectral power, i.e., it is proportional to the power in the input signal. It is defined as the continuous Fourier transform of the correlation function, or

$$\phi_{xx}(j\omega) = \int_{-\infty}^{\infty} \phi_{xx}(t) e^{-j\omega t} dt$$

and

$$\phi_{xy}(j\omega) = \int_{-\infty}^{\infty} \phi_{xy}(t) e^{-j\omega t} dt,$$

where $\phi_{xx}(t)$ and $\phi_{xy}(t)$ are the continuous representations of Eqs. (14) and (15), respectively.

Since the continuous Fourier transform is approximated by the DFT, the power spectra can be obtained by use of the FFT, i.e.,

$$\phi_{fg}(\lambda) = \sum_{k=0}^{N-1} \phi_{fg}(k) e^{-2\pi j k \lambda / N} \quad (16)$$

and

$$\phi_{ff}(\lambda) = \sum_{k=0}^{N-1} \phi_{ff}(k) e^{-2\pi j k \lambda / N}, \quad (17)$$

where $\lambda = 0, 1, 2, \dots, N-1$. Thus the FFT, in the sense given above, represents the power spectra of one or more signals.

The utility of the FFT as a highly efficient method of performing a DFT also allows an efficient convolution calculation. A convolution given by

$$c(j) = \frac{1}{N} \sum_{k=0}^{N-1} a(k) b(j-k)$$

requires an evaluation time proportional to N^2 when lags are performed circularly, i.e., when those values shifted from one end of the summation

index are circulated into the other. If one designates the FFT of $c(j)$ by C_k , then, using $C_k = A_k B_k$, one may simply transform a and b , take their product, and inverse-transform this product to obtain $c(k)$; this involves a time given by $T = 3\alpha N \log_2 N + \beta N$, where the first term is the time to perform three fast Fourier transforms and βN is the time required to perform the product of N pairs of numbers.

4. Pseudorandom Binary Generator

In many situations where a random signal is called for, a pseudorandom signal that passes certain tests for randomness is preferable because it is reproducible and allows experiments to be repeated under identical conditions. Properties of purely random signals may not be measured exactly, since an infinite measurement time is required to reduce statistical fluctuations to zero.¹⁸

The impulse-response function of a linear system is determined by exciting the system with a pseudorandom noise input and crosscorrelating the input with the system output.¹⁹ Pseudorandom numbers may be used in a variety of situations. One such application is described by Balcomb *et al.*²⁰

A pseudorandom signal is designed to approximate a signal whose autocorrelation function is a delta function and whose power spectral density is a constant for all frequencies. These properties are achieved for a sequence $a_1, a_2, \dots, a_N$, with $a_i = \pm 1$, such that

$$\sum_{i=1}^N a_i a_{(i+j) \bmod N} = \begin{cases} 1 & \text{when } j=0. \\ -1/N & \text{when } j \neq 0. \end{cases} \quad (18)$$

This relation may be achieved in cases where N is prime and of the form $4k-1$, where k is an integer. For a sequence of the type of Eq. (18) and where T is the period, the autocorrelation function is given by

$$\phi_{11}(\tau) = \begin{cases} 1 - \frac{N+1}{T} \tau & \text{when } 0 \leq \tau \leq \frac{T}{N} . \\ -1/N & \text{when } \frac{T}{N} \leq \tau \leq (N-1) \frac{T}{N} . \\ -N + \frac{N+1}{T} \tau & \text{when } (N-1) \frac{T}{N} \leq \tau \leq T . \end{cases}$$

If the input is truly a series of binary pulses, crosscorrelation consists of merely multiplying the output by +1 or -1. In actual practice, a square wave is not attainable, so the individual multiplications must be carried out.

III. PROGRAM DESCRIPTIONS AND EXAMPLES

This section provides data-input instructions for the programs described above and listed in the Appendix. Only those programs of a more complicated nature or which require elaboration for full understanding are described by sample problems.

A. Data-smoothing Programs

1. SMOOTH, an Nth-order Polynomial Smoothing Routine

The SMOOTH subprogram is designed to least-squares fit a parabola of specified size to obtain a smoothed value for each successive point of an array. The calling statement is

SUBROUTINE SMOOTH (DT, NPT, INSP, NR, NSM, X),

where

DT = spacing of the points in array X

NPT = size of array X

INSP = location of the point, X(INSP), at which smoothing is to start

NR = number of smoothing passes to make

NSM = number of points to which a parabolic fit is made (an odd number)

X = array of equally spaced points to process

The number NSM is odd, which allows a translation of variables so that all sums with odd arguments in the determinant solution of Eq. (1) are zero. This saves a significant amount of computer computation time with no loss of generality.

As an example of the use of this program for smoothing purposes, Fig. 2 shows the effect of one and two passes of SMOOTH across a curve of the function $y = (1 + x^2)^{-1}$ with 20-Hz sinusoidal noise superposed on it. A 25-point least-squares parabolic fit was used. As predicted by the transfer function of Fig. 1, a second pass through the filter provides a smoother representation of the original data than one pass. As can be seen, SMOOTH is a low-pass filter.

2. FILTER, an Nth-order Digital Filter

The digital-filter code FILTER was written to allow smoothing of sampled data. One may use a high-pass, low-pass, notch, or moving-averager type of filter, as appropriate. Coefficients for this filter may be arbitrarily input or may be determined for first- or second-order filters from the program TCOEF, which is described in the following Section III.A.3. The calling statement for FILTER is

FUNCTION FILTER (S),

where S is the data point for which the smoothing operation is wanted.

As the code is written, several variables are required in common storage. The statement for access to that storage is

COMMON/FLTR/A, B, X, Y, N,

where

Fig. 2. Effect of SMOOTH Filtering of 20-Hz Signal Superposed on the Curve $f(x) = 1/(1 + x^2)$

A = an N-term vector of weighting coefficients applied to the raw data

B = an N-term vector of weighting coefficients applied to the smoothed data

X = working array of size N

Y = working array of size N

N = size of the arrays A and B; filter order is (N-1)

If the user desires, he may easily add these variables to the argument of the call statement for the program.

Figure 3 shows an example of the use of the digital filter as a smoothing function. A signal was constructed with a sampling rate of 200 Hz according to the formula

$$f(t) = \sin(\omega t) + (1/2) \sin(2\omega t/3),$$

where $\omega = 2\pi f$, and $f = 10$ Hz. This curve is represented in Fig. 3 by the simple solid line. It is desired to remove the unwanted 10-Hz signal from $f(t)$. To do so, a notch filter is generated by program TCOEF discussed immediately below. For a damping constant (R) of 0.9 and a resonant frequency (FCUT) of 10 Hz, the filter constants turn out to be

$a_0 = 1.00215866$	$b_0 = 1.0$
$a_1 = -1.90621905$	$b_1 = 1.71190174$
$a_2 = 1.00215866$	$b_2 = -0.81$

The fitted curve is represented by the starred line in Fig. 3. Clearly, following the transient part of the filter output, the undesired 10-Hz signal has been removed. The remaining frequency is two-thirds of 10 Hz.

3. TCOEF, the FILTER Transfer-function Routine

Before applying the digital filter, one must have the coefficients that will produce the desired smoothing operation. The program TCOEF was

Fig. 3. Effect of Removing 10-Hz Signal from Curve $f(t) = \sin(\omega t) + (1/2) \sin(2\omega t/3)$ with Code FILTER

designed to calculate these for a first- or second-order filter and produce a CALCOMP plot of the magnitude and phase of the filter's frequency dependence. The basis for this code is Eq. (5) along with the formulas of Table I. At the user's discretion are the damping factor, resonant frequency, and notch frequency. The judicious use of a combination of these determines the level of noise rejection and the amount of signal loss at frequencies near the critical-frequency boundary.

The calling statement for this program is

SUBROUTINE TCOEF (N, ITR, T, FCUT, FNOT, R, GAIN, A, B),

where

N = the order of the digital filter (see Table I)
ITR = the type of the filter (see Table I):
 ITR = 1, low-pass filter (N = 1, 2)
 ITR = 2, high-pass filter (N = 1, 2)
 ITR = 3, bandpass filter (N = 2)
 ITR = 4, notch filter (N = 2)
 ITR = 5, moving-averager filter
T = sampling interval in seconds
FCUT = resonant frequency in Hz
FNOT = notch frequency in Hz
R = damping constant; damping factor is $(1 - |R|)$
GAIN = multiplicative gain
A, B = weighting coefficients for the raw data and smoothed data,
 respectively, as output by this program, for input into program FILTER

Program TCOEF uses a call to a subprogram named TFLOG (N, A, B, T, ITR) to plot the magnitude and phase of the filter transfer function. The output of TFLOG has magnitude in dB for easy orientation with the present standard notation for transfer functions.

Figure 4 shows an example of the output of TCOEF. Shown are the magnitude and phase of the notch filter used to produce the filtered results of Fig. 3.

Fig. 4. Relationship of Phase and Magnitude to Frequency for Notch Filter That Produced Results Shown in Fig. 3

B. Data-integration and Data-interpolation Programs

1. Numerical Quadrature

a. TRAP, Trapezoidal-rule Integration

The TRAP subprogram is designed to provide an equally weighted, fixed-interval integration program that is simple to use. The calling statement is

```
SUBROUTINE TRAP (NP, h, Y, SUM),
```

where

```
NP  = size of array Y
h   = mesh spacing for the independent variable
Y   = dependent-variable array
SUM = resulting integral
```

The integration is performed according to Eq. (6), with error of the order of h^3 . Integration of the function $f(x) = x^2$ between 0 and 1, using $h = 0.1$, gives a result of $Y = 0.33500$.

b. GLEG, Gauss-Legendre Integration

The GLEG subprogram is designed to provide an optimally weighted integration program for finite intervals. The calling statement is

```
SUBROUTINE GLEG (B, A, AREA, FCN),
```

where

```
B   = upper limit of integration
A   = lower limit of integration
AREA = resulting integral
FCN = function f(x)
```

The integration is performed according to Eq. (7), using a 32-point Gauss-Legendre mesh. Integration of the function $f(x) = x^2$ between 0 and 1 gives a result of $Y = 0.33333$.

c. GLAG, Gauss-Laguerre Integration

The GLAG subprogram is designed to provide an optimally weighted integration program for infinite intervals. The calling statement is

SUBROUTINE GLAG (A, AREA, FCN),

where

A = lower limit of integration
AREA = resulting integral
FCN = function $f(x)$

The integration is performed according to Eq. (8), using a 30-point Gauss-Laguerre mesh. Integration of the function $f(x) = x^{-3}$ between 1.0 and ∞ gives a result of $Y = 0.49996$.

2. MONTE, a Monte Carlo Integration Routine

This routine is provided for estimating more-complicated integrals with a minimum of effort. The call statement for this program is

CALL MONTE (C, D, J, NPTS, ANS),

where

C = J-point array containing lower limits of integration
D = J-point array containing upper limits of integration
J = order of integration used
NPTS = number of trials to be made
ANS = result of integration

A requirement for this program is a subprogram for generating a pseudorandom number; that subprogram is called RANDU(S), where S is the random number generated.

Because MONTE is a statistical program, three sample problems were run to check its accuracy. Evaluations were made of the three integrations:

$$I_1 = 0.5 \int_0^1 dx = 0.5,$$

$$I_2 = \int_0^1 \sin x dx = 0.45970,$$

and

$$I_3 = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 e^u (v^2 + 1)wx(y - 1) \sqrt{z} dz dy dx dw dv du \\ = -0.19092.$$

Table II shows the results of evaluations of I_1 , I_2 , and I_3 with 10,000 and 50,000 sampled data points. Accuracy to the second or third decimal may be expected. Not much more accuracy is gained by using the averages of two trials with NPTS = 1,000,000. In this case, $I_2 = 0.459375$ and $I_3 = -0.189899$.

3. YLAG, a Lagrangian Interpolation Routine

The subprogram YLAG is provided to give the user more versatility in using the integration routines. The calling statement is

FUNCTION YLAG (XI, X, Y, N, IMAX),

where

XI = independent variable where function is to be evaluated
X = independent-variable array
Y = dependent-variable array
N = order of Lagrangian interpolation
IMAX = size of arrays X and Y

Several sample problems were run with YLAG. Table III indicates the relative speed of the code for interpolation. For each of the times shown, the function was evaluated 1000 times on the Sigma 5 computer.

C. Data-analysis Programs

1. FFT1, a Fast Fourier Transform Program

Subprogram FFT1 is provided to allow calculation of the discrete Fourier transform efficiently. The calling statement is

```
SUBROUTINE FFT1 (Z, ASIGN, IPOWR),
```

where

Z = complex-valued function to be transformed
ASIGN = indicator for transform and inverse transform calculations:
If -1, transform is taken as in Eq. (12); if 1, Eq. (13) is used
IPOWR = power of two describing array size of Z

Program FFT1 requires that the number of points be a power of two. A much more complex program that does not share this requirement is described in Ref. 21. That reference includes listings and input instructions for the program.

2. CORR, a Crosscorrelation and Autocorrelation Program

Subprogram CORR is provided to allow calculation of crosscorrelation and autocorrelation functions according to Eq. (14) and (15). Equal

TABLE II. Results of Monte Carlo Integration

Trial No.	NPTS = 10,000			NPTS = 50,000	
	I_1	I_2	I_3	I_2	I_3
1	0.497900	0.466006	-0.175601	0.458702	-0.184952
2	0.501000	0.463229	-0.210938	0.459594	-0.196151
3	0.498000	0.452374	-0.194085	0.458332	-0.190388
4	0.507800	0.464323	-0.173426	0.458012	-0.187561
5	0.498700	0.462893	-0.212569	0.459712	-0.194520
6	0.506300	0.460789	-0.189192	0.462068	-0.190062
7	0.502600	0.458853	-0.197347	0.458601	-0.190171
8	0.486900	0.461041	-0.188105	0.455084	-0.195716
9	0.500300	0.458180	-0.184299	0.457288	-0.189845
10	0.496600	0.457423	-0.197891	0.463549	-0.193324
Avg	0.499610	0.460511	-0.192345	0.459094	-0.191269
Time per Trial	9 sec	9 sec	24 sec	43 sec	120 sec

TABLE III. Time Required for YLAG Interpolation of
Function $Y = \sin X$

Order (N)	3	11	19
Time, sec	3	8	19

spacing of intervals is assumed, and the lag variable is a multiple of this spacing. The defining statement is

```
SUBROUTINE CORR (X, Y, NP, XY, NLP),
```

where

```
X   = array of size 2**IPOWR
Y   = array to be correlated with X
NP  = size of the arrays X and Y
XY  = array of results of correlation calculations; contains
      correlated function for each lag point
NLP = number of lag points for array XY
```

3. PRB, a Pseudorandom Binary Generator

Subprogram PRB is designed to provide pseudorandom signals. Only +1 and 0 are used in the sequence. The subprogram is defined by

```
SUBROUTINE PRB (NN, MPRB, M),
```

where

```
NN   = bit sequence length, which must be prime and satisfy the
      relation  $NN = 4k-1$  for an integer  $k$ ;  $NN < 8191$ 
MPRB = array of size 8200 containing pseudorandom sequence in
      units +1 and 0.
M     = array of size 300 containing hexadecimal representation
      of MPRB
```

APPENDIX

Program Listings

SMOOTH

```

1.      SUBROUTINE SMOOTH(DT,NPT,INSP,NR,NSM,X)
2.      IMPLICIT REAL*8 (A-H,O-Z)
3.      DIMENSION X(1)
4.      C      NR      = NUMBER OF SMOOTHING PASSES.
5.      C      NSM     = NUMBER OF SMOOTHING POINTS FOR THE PARABOLIC FIT
6.      C      INSP    = NUMBER OF POINT AT WHICH TO START THE SMOOTH
7.      C      DT      = SPACING BETWEEN POINTS
8.      C      NPT     = NUMBER OF POINTS TO PROCESS
9.      FNSM=NSM
10.     IP2=INSP+100
11.     NSM2=(NSM-1)/2
12.     FSQ=FNSM*FNSM
13.     SM2=FNSM*(FSQ-1.0)/24.0
14.     SM4=FNSM*(3.0*FSQ*FSQ-10.0*FSQ+7.0)/480.0
15.     DTSQ=DT*DT
16.     AK2=2.0*DTSQ*SM2
17.     AK4=2.0*DTSQ*DTSQ*SM4
18.     DTRM=FNSM*AK2*AK4-AK2*AK2*AK2
19.     PRINT 1000, INSP,SM2,SM4,DT,AK2,AK4,DTRM
20.     LM1=NPT-NSM2
21.     DO 900 NT=1,NR
22.     INSM=INSP
23.     750 SY=0.0
24.     SXY=0.0
25.     SXXY=0.0
26.     DO 800 I=1,NSM
27.     JK=I-1-NSM2
28.     JI=INSM+JK
29.     FJ=JK
30.     FT=FJ*DT
31.     SY=SY+X(JI)
32.     SXY=SXY+X(JI)*FT
33.     SXXY=SXXY+X(JI)*FT*FT
34.     800 CONTINUE
35.     X(INSM)=(AK2*AK4*SY-AK2*AK2*SXXY)/DTRM
36.     INSM=INSM+1
37.     IF (INSM-LT-LM1)GO TO 750
38.     900 CONTINUE
39.     PRINT 2000, (X(I),I=1,NPT)
40.     RETURN
41.     1000 FORMAT (1X,I5,1P6E15.6)
42.     2000 FORMAT (1X,16F7.0)
43.     END

```

FILTER

```

1      FUNCTION FILTER(S)
2      DIMENSION X(10),Y(10),A(10),B(10)
3      COMMON/FLTR/A,B,X,Y,N
4      SUM=A(1)*S
5      DO 10 I=2,N
6      J=N-I+1
7      SUM=SUM+A(I)*X(J)+B(I)*Y(J)
8      10 CONTINUE
9      X(N)=S
10     Y(N)=SUM
11     DO 20 I=2,N
12     X(I-1)=X(I)
13     Y(I-1)=Y(I)
14     20 CONTINUE
15     FILTER=SUM
16     RETURN
17     END

```

TCOEF

```

1.      SUBROUTINE TCOEF(N,ITR,T,FCUT,FNOT,R,GAIN,A,B)
2.      C
3.      C      THIS SUBPROGRAM CALCULATES THE WIEGHTING COEFFICIENTS FOR
4.      C      THIS SUBPROGRAM CALCULATES THE WEIGHTING COEFFICIENTS FOR
5.      C      THE NUMERATOR AND DENOMINATOR OF A Z-TRANSFORM TYPE DIFFERENCE
6.      C      EQUATION. CONSIDERATION IS GIVEN ONLY TO FIRST AND SECONDS
7.      C      ORDER DIGITAL FILTERS.
8.      C      N      = THE FILTER ORDER
9.      C      ITR    = THE FILTER TYPE:
10.     C          ITR=1, LOW PASS FILTER, N=1,2
11.     C          ITR=2, HIGH PASS FILTER, N=1,2
12.     C          ITR=3, BAND PASS FILTER, N=2
13.     C          ITR=4, NOTCH FILTER, N=2
14.     C          ITR=MOVING AVERAGER FILTER, N=2.
15.     C      T      = THE SAMPLING INTERVAL IN SECONDS.
16.     C      FCUT   = THE CRITICAL OR ROLL OFF FREQUENCY IN HZ.
17.     C      FNOT   = THE NOTCH FREQUENCY IN HZ
18.     C      R      = A DAMPING CONSTANT, DAMPING (U=ABS(R))
19.     C      GAIN   = THE GAIN CONSTANT
20.     C      A      = THE VECTOR OF NUMERATOR, OR INPUT, WEIGHTING CONSTANTS.
21.     C      B      = A VECTOR OF DENOMINATOR, OR SMOOTHED, WEIGHTING
22.     C               CONSTANTS.
23.     C
24.     DIMENSION A(3),B(3)
25.     THETA=2.0*3.14159*FCUT*T
26.     THETA0=2.0*3.14159*FNOT*T

```

27.	COSTH=COS(THETA)		
28.	COSTB=COS(THETA)		
29.	COST2=COS(2.0*THETA)		
30.	CSCHT=1.0/SIN(THETA*0.5)		
31.	CSCHTB=1.0/SIN(THETAB*0.5)		
32.	DEL1=3.14159/T		
33.	DEL2=2.0*DEL1		
34.	RA=R		
35.	IF(R.LT.0.0)RA=-RA		
36.	PRINT 150,N,GAIN,RA,T		
37.	PRINT 151, DEL1,DEL2		
38.	DO 3 I=1,3		
39.	B(I)=0.0		
40.	IF(1TR.EQ.5)GO TO 75		
41.	B(2)=R		
42.	GO TO (5,10),N		
43.		C	
44.	ORDER 1 --- LOW PASS	C	
45.		C	
46.	A(1)=0.5*GAIN*(1.-R)		
47.	A(2)=A(1)		
48.	A(3)=0.		
49.	IF(1TR.EQ.1)PRINT 156		
50.	IF(1TR.EQ.2)PRINT 161		
51.	IF(1TR.EQ.2)A(2)=-A(1)		
52.		C	
53.	ORDER 1 --- HIGH PASS	C	
54.		C	
55.	GO TO 100		
56.	B(2)=R*2.0*COSTH		
57.	B(3)=-R*R		
58.	GO TO (15,20,25,30),1TR		
59.		C	
60.	ORDER 2 --- LOW PASS	C	
61.		C	
62.	A(1)=0.25*GAIN*(1.0-2.0*R*COSTH+R*R)		
63.	A(2)=2.0*A(1)		
64.	A(3)=A(1)		
65.	PRINT 157,F,CUT		
66.	GO TO 100		
67.		C	
68.	ORDER 2 --- HIGH PASS	C	
69.		C	
70.	A(1)=0.25*GAIN*(1.0+2.0*R*COSTH+R*R)		
71.	A(2)=-2.0*A(1)		
72.	A(3)=A(1)		
73.	PRINT 162,F,CUT		
74.	GO TO 100		
75.		C	
76.	ORDER 2 --- BAND PASS	C	
77.		C	
78.	A(1)=0.5*GAIN*(1.0-R)*CSCHT*(1.-2.*R*COST2T+R*R)**0.5		
79.	A(2)=0.0		
80.	A(3)=-A(1)		

81.	PRINT 166,FCUT	
82.	GO TO 100	
83.		
84.	ORDER 2 --- NOTCH	C
85.		C
86.	A(1)=0.25*GAIN*CSCHT0*CSCHT0*(1.-2.*R*CSSTH+R*R)	
87.	A(2)=-2.0*A(1)*CSST0	
88.	A(3)=A(1)	
89.	PRINT 171,FCUT,FNOT	
90.	GO TO 100	
91.		C
92.	ORDER 2 --- MOVING AVERAGER	C
93.		C
94.	A(1)=0.333333	
95.	A(2)=0.333333	
96.	A(3)=0.333333	
97.	PRINT 176	
98.	B(1)=1.0	
99.	PRINT 201	
100.	PRINT 200,(I,A(I),I,B(I),I=1,3)	
101.	B(2)=-B(2)	
102.	B(3)=-B(3)	
103.	RETURN	
104.	FORMAT(1H, 'DIGITAL FILTER PARAMETERS',//, ' ORDER = ',I4,	
105.	' GAIN = ',F10.5,4X,DAMPING = (1.0-' F8.4,1), ' TIME STEP =	
106.	' F10.5,//)	
107.	FORMAT(1H, 'PI/DI = ',F10.3, ' 2*PI/DI = ',F10.3,//)	
108.	FORMAT(1H, 'LOW PASS',//)	
109.	FORMAT(1H, 'LOW PASS CUTOFF = ',F12.5, ' HZ',//)	
110.	FORMAT(1H, 'HIGH PASS',//)	
111.	FORMAT(1H, 'HIGH PASS CUTOFF = ',F12.5, ' HZ',//)	
112.	FORMAT(1H, 'BAND PASS CUTOFF = ',F12.5, ' HZ',//)	
113.	FORMAT(1H, 'CUTOFF = ',F12.5, ' NOTCH = ',F12.5, ' HZ',//)	
114.	FORMAT(1H, 'MOVING AVERAGER',//)	
115.	FORMAT(1H, 'A(1),F12.6,6X, 'B(1),F12.6)	
116.	FORMAT(1H, 'B(2),B(3) IN TRANSFER FUNCTION ARE NEG. OF THESE FOR	
117.	* FILTER ',//)	
118.	END	
1.	SUBROUTINE TFL08(N,A,B,T,ITR)	
2.	DIMENSION A(3),B(3),PH(1000)	
3.	DIMENSION P1(2),P2(2),P3(2),P4(2),P5(2)	
4.		C
5.	DB VS LINEAR FREQUENCY	C
6.		C
7.	Y0=-40	
8.	XLEN=5.	
9.	YLEN=5.	
10.	X0=0.	
11.	XS=(6.28318 / T)/XLEN	
12.	YS=50./YLEN	
13.	CALL OFFSET(X0,XS,Y0,YS)	
14.		C
15.	DRAW GRID	C

```

16.      CALL PLOT(0.0,YLEN,2)
17.      CALL PLOT(XLEN,YLEN,2)
18.      CALL PLOT(XLEN, 0.0,2)
19.      CALL PLOT( 0.0, 0.0,2)
20.      ENCODE (4,77,XBOT),XB
21.      TEMP=XS*XLEN*XB
22.      ENCODE (4,77,XTOP),TEMP
23.      ENCODE (4,77,YBOT),YB
24.      TEMP=YS*YLEN*YB
25.      ENCODE (4,77,YTOP),TEMP
26.      CALL SYMBOL (0.0,-0.2,0.1,XBOT,0.0,4)
27.      CALL SYMBOL (2.0,-0.4,0.15,'FREQ., HZ',0.0,9)
28.      CALL SYMBOL (4.7,-0.2,0.1,XTOP,0.0,4)
29.      CALL SYMBOL (5.120.0,0.1,'-200',0.0,4)
30.      CALL SYMBOL (5.1,1.0,0.1,'-150',0.0,4)
31.      CALL SYMBOL (5.122.0,0.1,'-100',0.0,4)
32.      CALL SYMBOL (5.6,2.0,0.15,'PHASE, DEG',90.0,10)
33.      CALL SYMBOL (5.123.0,0.1,'- 50',0.0,4)
34.      CALL SYMBOL (5.1,4.0,0.1,'  0',0.0,4)
35.      CALL SYMBOL (5.124.9,0.1,' 50',0.0,4)
36.      CALL SYMBOL (-0.45,4.9,0.1,YTOP,0.0,4)
37.      CALL SYMBOL (-0.50,1.5,0.15,'MAGNITUDE, ARB.UNITS',90.0,20)
38.      CALL SYMBOL (-0.45,0.0,0.1,YBOT,0.0,4)
39.      CALL PLOT (.0,0.0,13)
40.      IGC=0
41.      CGAIN=1.0
42.      N1=N+1
43.      DEL=.002*3.1415926/T
44.      5   WTF=0.0
45.      JL=0
46.      DO 50 KK=1,1000
47.      SUMD=0.0
48.      SUMNI=0.0
49.      SUMNR=0.0
50.      DO 10 I=1,N1
51.      II=I-1
52.      DO 10 J=1,N1
53.      JJ=J-1
54.      TRIG=(JJ-II)*WTF*T
55.      SCOS=COS(TRIG)
56.      COEF=A(I)*B(J)
57.      SUMNR=SUMNR+COEF*SCOS
58.      SUMNI=SUMNI+COEF*SIN(TRIG)
59.      SUMD =SUMD +B(I)*B(J)*SCOS
60.      10  CONTINUE
61.      WTR=SUMNR/SUMD
62.      WTI=SUMNI/SUMD
63.      PH(KK)=ATAN2(WTI,WTR)*57.29578
64.      IF(PH(KK).GT.0.0)PH(KK)=PH(KK)-180.
65.      IF(KK.GE.502)PH(KK)=-180.0-PH(KK)
66.      AMP=(WTR*WTR+WTI*WTI)**0.5
67.      IF(IGC.GT.0)GO TO 24
68.      GO TO(22,14,16,22,22),ITR
69.      14  IF(KK.EQ.500)GO TO 22
70.      GO TO 50
71.      16  IF(KK.EQ.250)GO TO 22

```

```

72.      GO TO 50
73.      22 CGAIN=1.0/AMP
74.      IGC=1
75.      GO TO 5
76.      24 CONTINUE
77.      AMP=AMP*CGAIN
78.      PMP=ALOG10(AMP)*20
79.      IF(PMP-GE.Y0)CALL PLOT(WTF,PMP,12)
80.      JL=JL+1
81.      P1(JL)=WTF
82.      P2(JL)=PMP
83.      P3(JL)=WTR*CGAIN
84.      P4(JL)=WTI*CGAIN
85.      P5(JL)= PH(KK)
86.      IF(JL.LT.2)GO TO 25
87.      JL=0
88.      25 CONTINUE
89.      WTF=KK*DEL
90.      50 CONTINUE
91.      PRINT 76,CGAIN
92.      CALL PLOT(0.0,YLEN,3)
93.      Y0=-200.
94.      YS=250./YLEN
95.      CALL OFFSET(X0,XS,Y0,YS)
96.      WTF=0.0
97.      DO 80 LI=1,1000
98.      CALL PLOT(WTF,PH(LI),12)
99.      80 WTF=LI*DEL
100.     CALL PLOT(XLEN,0.0,3)
101.     CALL NUPAGE(3.0,0.0)
102.     75 FORMAT(1H ,10E12.4)
103.     76 FORMAT(1H ,1COMPUTED GAIN(NORMALIZE TO UNITY AMP) =',E14.6,/)
104.     77 FORMAT (F)
105.     RETURN
      END

```

TRAP

```

1.      SUBROUTINE TRAP(NP,H,Y,SUM)
2.      DIMENSION Y(1)
3.      NP1=NP-1
4.      SUM=0.
5.      DO 10 I=1,NP1
6.      10 SUM=SUM+H/2.*(Y(I)+Y(I+1))
7.      RETURN
8.      END

```


GLEG

```

SUBROUTINE GLEG(B,A,AREA,FCN)
DIMENSION A16(32),X16(32)
DATA X16 /-.99726386,-.98561151,-.96476226,-.93490608, BL0C 370
*          -.89632116,-.84936761,-.79448380,-.73218212, BL0C 380
*          -.66304427,-.58771576,-.50689991,-.42135128, BL0C 390
*          -.33186860,-.23928736,-.14447196,-.04830766, BL0C 400
*          .04830767, .14447196, .23928736, .33186860, BL0C 410
*          .42135128, .50689991, .58771576, .66304427, BL0C 420
*          .73218212, .79448380, .84936761, .89632116, BL0C 430
*          .93490608, .96476226, .98561151, .99726386/, BL0C 440
*      A16 / .00701861,.016274395,.025392065,.034273864, BL0C 450
*          .042835898,.050998059,.058684093,.065822223, BL0C 460
*          .072345794,.078193896,.083311924,.087652093, BL0C 470
*          .091173879,.093844399,.095638720,.096540089, BL0C 480
*          .096540089,.095638721,.093844399,.091173879, BL0C 490
*          .087652093,.083311924,.078193896,.072345794, BL0C 500
*          .065822223,.058684093,.050998059,.042835898, BL0C 510
*          .034273863,.025392065,.016274395,.007018610/ BL0C 520
      CONST=(B-A)/2.
      CONST2=(B+A)/2.
      SUM=0.
      DO 10 I=1,32
10  SUM=SUM+A16(I)*FCN(CONST*X16(I)+CONST2)
      AREA=SUM*CONST
      RETURN
      END

```

GLAG

```

SUBROUTINE GLAG(A,SUM,FCN)
DIMENSION ETA(30),EL(30)
DATA ETA /.04740718054, .2499239168, .6148334544, BL0C 560
*          1.143195826, 1.836454555, 2.696521875, BL0C 560
*          3.725814508, 4.927293766, 6.304515591, BL0C 570
*          7.861693293, 9.603775985, 11.53654660, BL0C 580
*          13.66674469, 16.00222119, 18.55213484, BL0C 590
*          21.32720432, 24.34003576, 27.60555480, BL0C 600
*          31.14158670, 34.96965201, 39.11608495, BL0C 610
*          43.61365251, 48.50398616, 53.84138541, BL0C 620
*          59.69912186, 66.18061779, 73.44123860, BL0C 630
*          81.73681051, 91.55646652, 104.41575244/, BL0C 640
DATA EL /.1216778939, .2835568198, .4464323664, BL0C 650
*          .6105321301, .7763034787, .9442332894, BL0C 660
*          1.114844704, 1.288705436, 1.466439139, BL0C 670
*          1.648739458, 1.836387678, 2.030274257, BL0C 680
*          2.231427214, 2.441048113, 2.660560243, BL0C 690
*          2.891672641, 3.136468334, 3.397527594, BL0C 700
*          3.678104716, 3.982388550, 4.315898899, BL0C 710
*          4.686112561, 5.103502651, 5.583332982, BL0C 720
*          6.149044408, 6.839130544, 7.722947870, BL0C 730
*          8.943742448, 10.87187293, 15.02611124/, BL0C 740
      SUM=0.
      DO 10 I=1,30
10  SUM=SUM+EL(I)*FCN(ETA(I)+A)
      RETURN
      END

```

MONTE

```
1.      SUBROUTINE MONTE(C,D,NORD,NPASS,ANS)
2.      C
3.      C      MONTE CARLO INTEGRATOR ROUTINE
4.      C
5.      DIMENSION C(10),D(10),DMC(10),R(10)
6.      J=NORD
7.      S=0.0
8.      PROD=1.0
9.      NTOP=NORD+1
10.     5 DO 10 I=1,NTOP
11.       DMC(I)=D(I)-C(I)
12.       PROD=PROD*DMC(I)
13.     10 CONTINUE
14.     DO 80 L=1,NPASS
15.       DO 20 I=1,NTOP
16.         CALL RANDU(R(I))
17.         R(I)=R(I)*DMC(I)+C(I)
18.       20 CONTINUE
19.       30 Y=FUNCT(R,J)
20.       IF(Y.GT.0.0)GO TO 40
21.       IF(R(1).GE.0.0)GO TO 80
22.       IF(R(1).LT.Y)GO TO 80
23.       S=S-1.0
24.       GO TO 80
25.     40 IF(R(1).LE.0.0)GO TO 80
26.       IF(R(1).GT.Y)GO TO 80
27.       S=S+1.0
28.     80 CONTINUE
29.     ANS=S*PROD/NPASS
30.     RETURN
31.     END
```

```
1.      SUBROUTINE RANDU(S)
2.      C
3.      C      RANDOM NUMBER GENERATOR
4.      C
5.      DATA IX/123456789/
6.      IY=IX*65539
7.      IF(IY)5,6,6
8.     5 IY=IY+2147483647+1
9.     6 IX=IY
10.    YFL=IY
11.    S=YFL*0.4656613E-9
12.    RETURN
13.    END
```

YLAG

```
      FUNCTION YLAG(XI,X,Y,N ,IMAX,INX)
C *   LAGRANGIAN INTERPOLATION ROUTINE
C *   XI IS THE INTERPOLATED ENTRY IN X ARRAY
C *   N IS NUMBER OF POINTS TO BE USED IN INTERPOLATION--
C *   N IS ONE GREATER THAN DEGREE OF INTERPOLATION POLY-
C *   NOMIAL. IMAX IS MAX INDEX OF X AND Y ARRAYS
C *   Y IS ARRAY FROM WHICH YLAG IS OBTAINED BY INTERPOLA-
C *   TION. INX IS THE POINT AT WHICH SEARCH OF THE X
C *   ARRAY IS TO BEGIN; IF PASSED AS A 0, IT IS SET TO 1.
      DIMENSION X(1),Y(1)
      IF (N.GT.IMAX) N=IMAX
1     IF(INX.EQ.0) INX=1
      DO 2 J=INX,IMAX
        IF(XI-X(J)) 3,13,2
2     CONTINUE
      INX=IMAX
      GO TO 7
3     IND=J
      INX=J-1
      INL=IND-(N+1)/2
      IF(INL.LE.0) INL=1
      INU=INL+N-1
      IF(INU.LE.IMAX) GO TO 8
7     INU=IMAX
      INL=INU-N+1
8     S=0.
      P=1.
      DO 11 J=INL,INU
        P=P*(XI-X(J))
      D=1.
      DO 10 I=INL,INU
        IF(I.NE.J) GO TO 9
        XD=XI
      GO TO 10
9     XD=X(J)
10    D=D*(XD-X(I))
11    S=S+Y(J)/D
      YLAG=S*P
12    RETURN
13    YLAG=Y(J)
      INX=J
      GO TO 12

      END
```

FFT1

```

1.      SUBROUTINE FFT1(Z,ASIGN,IP0WR)
2.      C
3.      C      Z      = COMPLEX ARRAY
4.      C      ASIGN  = TYPE OF TRANSFORM, -1.0 FOR FWD TRANSFORM, +1.0 BACKWAF
5.      C      IP0WR  = POWER OF 2 FOR FFT NO OF PTS.
6.      C
7.      COMPLEX Z(1)
8.      N=2**IP0WR
9.      AN=N
10.     IF(ASIGN.LT.0.0)GO TO 5
11.     DO 4 I=1,N
12.     Z(I)=Z(I)/AN
13.     4 CONTINUE
14.     5 P=6.283185/AN
15.     50 DO 370 L=1,IP0WR
16.     IP0WR1=2** (IP0WR-L)
17.     M=0
18.     MKD=2** (L-1)
19.     DO 360 I=1,MKD
20.     K1=M/IP0WR1
21.     KC0DE=0
22.     GO TO 530
23.     240 A=P*K2
24.     Y1=COS(A)
25.     Y2=ASIGN*SIN(A)
26.     DO 340 J=1,IP0WR1
27.     K2=M+1
28.     K3=K2+IP0WR1
29.     Y3=REAL(Z(K3))*Y1-AIMAG(Z(K3))*Y2
30.     Y4=REAL(Z(K3))*Y2+AIMAG(Z(K3))*Y1
31.     Z(K3)=CMPLX(REAL(Z(K2))-Y3,AIMAG(Z(K2))-Y4)
32.     Z(K2)=CMPLX(REAL(Z(K2))+Y3,AIMAG(Z(K2))+Y4)
33.     M=M+1
34.     340 CONTINUE
35.     M=M+IP0WR1
36.     360 CONTINUE
37.     370 CONTINUE
38.     DO 480 I2=1,N
39.     K1=I2-1
40.     KC0DE=1
41.     GO TO 530
42.     410 IF(K2.GE.12) GO TO 480
43.     K2=K2+1
44.     AK3=REAL(Z(I2))
45.     AK4=AIMAG(Z(I2))
46.     Z(I2)=Z(K2)
47.     Z(K2)=CMPLX(AK3,AK4)
48.     480 CONTINUE
49.     RETURN
50.     530 K2=0
51.     DO 590 K=1,IP0WR
52.     K3=K1-2*(K1/2)

```

```

53.      K1=K1/2
54.      IF(K3.EQ.0) GO TO 590
55.      K2=K2+2** (IP0WR-K)
56.      590 CONTINUE
57.      IF(KCODE.EQ.0) GO TO 240
58.      GO TO 410
59.      END

```

CORR

```

1          SUBROUTINE CORR(X,Y,NP,XY,NLP)
2          C
3          C      THIS SUBROUTINE CALCULATES THE CORRELATION FUNCTIONS
4          C
5          C      X=ARRAY CORRELATES WITH Y=ARRAY.
6          C      NP IS THE MINIMUM SIZE OF THE ARRAYS X AND Y.
7          C      XY=ARRAY CONTAINS THE CORRELATED INFORMATION.
8          C      NLP IS THE NUMBER OF LAG POINTS CALCULATED AND PLACED
9          C      IN XY=ARRAY.
10         C
11         DIMENSION X(NP),Y(NP),XY(NLP)
12         DO 2 I=1,NLP
13         NPL=NP-(I-1)
14         XNPL=NPL
15         XY(I)=0.0
16         DO 1 J=1,NPL
17         XY(I)=XY(I)+X(J)*Y(J+I-1)
18         1 CONTINUE
19         XY(I)=XY(I)/XNPL
20         2 CONTINUE
21         RETURN
22         END

```

PRB

```
DIMENSION NP(4100),NPT(4100),MPRB(8200),M(300)
1 PRINT 205
  READ(105,210) NN
  LM=(NN-1)/2
  DO 10 I=1,LM
    ISQ=I*I
    NP(I)=ISQ-NN*(ISQ/NN)
10 CONTINUE
C SORT AND ARRANGE NP(I)
  ML=LM-1
  IF(ML.LE.0)GO TO 30
15 J=0
  DO 25 I=1,ML
    IF(NP(I+1).GT.NP(I))GO TO 20
    NPT(I)=NP(I)
    J=0
    NP(I)=NP(I+1)
    NP(I+1)=NPT(I)
20 J=J+1
25 CONTINUE
  IF((J-ML).NE.0)GO TO 15
C ASSIGN BINARY VALUES
C
30 DO 40 L=1,NN
  MPRB(L)=0
40 CONTINUE
  DO 50 I=1,LM
    J=NP(I)
    MPRB(J)=1
50 CONTINUE
  WRITE(108,220)(MPRB(I),I=1,NN)
  WRITE(106,220)(MPRB(I),I=1,NN)
  LL=NN/32
  JD=NN/32
  LL=JD+1
  LNXT=LL*32
  LST=NN+1
  DO 55 J=LST,LNXT
    MPRB(J)=0
55 CONTINUE
  DO 70 I=1,LL
    M(I)=0
    J=(I-1)*32
    DO 60 K=1,32
      M(I)=M(I)*2
      M(I)=M(I)+MPRB(J+K)
60 CONTINUE
70 CONTINUE
  CALL BUFFER OUT(1,1,M,LL,N1,N2)
  WRITE(108,240)(M(I),I=1,LL)
  GO TO 1
205 FORMAT(1H1)
210 FORMAT(I5)
220 FORMAT(25(1X,I2))
240 FORMAT(10(1X,Z10))
END
```

ACKNOWLEDGMENTS

This acknowledges the programming of the integration subprograms by Mary Campbell, a summer student appointee, and the programming of the digital filtering programs by Ivar A. Engen, former ANL employee. Further, many helpful discussions and comments were offered by R. W. Hyndman, EBR-II Project.

REFERENCES

1. R. W. Hyndman and R. B. Nicholson, *The EBR-II Feedback Function*, ANL-7476 (July 1968).
2. I. A. Engen, *A New Technique for Investigating Feedback and Stability of a Nuclear Reactor*, ANL-7686 (June 1970).
3. H. A. Larson, I. A. Engen, M. R. Tuck, and R. W. Hyndman, *Computer Programs for Describing the Dynamic Behavior of EBR-II*, ANL/EBR-076, (September 1973).
4. J. M. Allen, et al., *Functional Description of the EBR-II Digital Data Acquisition System*, ANL/EBR-029 (November 1970).
5. R. W. Hyndman, et al., *Description of Software for the EBR-II Digital Data Acquisition System*, ANL/EBR-033 (December 1970).
6. M. R. Tuck, I. A. Engen, and R. W. Hyndman, "Computer-aided Experimental Support," *Trans. Am. Nucl. Soc.* 14 (Suppl. 2), p. 34 (August 1971).
7. K. D. Tucker, E. W. Laird, and R. W. Hyndman, "Application of a Medium-scale Computer to Operational Support of EBR-II," *Trans. Am. Nucl. Soc.* 14 (Suppl. 2), p. 33 (August 1971).
8. B. J. Power, R. N. Hagen, and S. O. Johnson, *SPORT, A System for Processing Reactor Transient Data on the IBM-7040 Computer*, IDO-17078 (July 1965).
9. G. A. Mortensen and H. A. Larson, *Compensated Reactivity Error Analysis*, PTR-818, Phillips Petroleum Co. (October 1966).
10. M. J. E. Golay, "Smoothing of Data by Least Squares Procedures and by Filtering," *IEEE Transactions on Computers* C-21 (3) p. 299 (March 1972).
11. M. A. Soderstrand and J. K. Maulden, *Digital Filter Techniques for Mini-computers*, SCL-DR-710116 (March 1972).
12. A. S. Kronrod, *Nodes and Weights of Quadrature Formulas*, Consultants Bureau, New York (1965).
13. M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, AMS 55, National Bureau of Standards, Washington (1964).
14. A. H. Straud and Don Secrest, *Gaussian Quadrature Formulas*, Prentice Hall, New York (1966).
15. J. A. Thie, *Reactor Noise*, Rowman and Littlefield, New York (1963).

16. W. T. Cochran, *et al.*, "What Is the Fast Fourier Transform?", *Proceedings IEEE* 55 (10), p. 1664 (October 1967).
17. Lynn B. Weaver, *System Analysis of Nuclear Reactor Dynamics*, Rowman and Littlefield, New York (1963).
18. Anthony C. Davies, "Properties of Waveforms Obtained by Nonrecursive Digital Filtering of Pseudorandom Binary Sequences," *IEEE Transactions on Computers* C-20 (3), p. 270 (March 1971).
19. J. G. Truxal, *Control System Synthesis*, McGraw-Hill, New York (1955).
20. J. D. Balcomb, H. B. Demuth, and E. P. Gyftopoulos, "A Crosscorrelation Method for Measuring the Impulse Response of Reactor Systems," *Nucl. Sci. Eng.* 11 (2), p. 159 (October 1961).
21. N. M. Brenner, *Three FORTRAN Programs That Perform the Cooley-Tukey Fourier Transform*, Technical Note 1967-2, M.I.T. Lincoln Laboratory (July 1967).