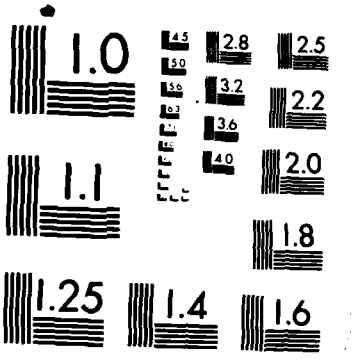
AD-A161 933 SIGNIFICANT FACTOR IDENTIFICATION USING DISCRETE SPECTRAL METHODS(U) CORNELL UNIV ITHACA NY P J SANCHEZ ET AL MAR 85 TR-654 N00014-81-K-0037 1/1

UNCLASSIFIED F/G 12/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

AD-A161 933

SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK

TECHNICAL REPORT NO. 654

March 1985

SIGNIFICANT FACTOR IDENTIFICATION USING DISCRETE SPECTRAL METHODS

by

Paul J. Sanchez and Lee W. Schruben[1]

DTIC
ELECTE
DEC 0 6 1985
S D
D

DTIC FILE COPY

[1]Research supported in part by the Office of Naval Research grant N00014-81-K-0037, and by a grant from Bethlehem Steel. The research was done at Cornell University.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

85 12 2 071

ABSTRACT

Discrete event simulations are computer models of complex systems. The accuracy of the model in reflecting the behavior of the real system is determined by, among other things, the values of parameters for distributions used in the model. The modeller could allocate experimental resources more effectively without loss of accuracy in the model if he or she could identify those parameters to which the response of interest has the greatest sensitivity.

One method of doing this is to try to model the response as a polynomial function of the model parameters. We are then interested in those terms in the polynomial which have non-zero coefficients.

Schruben and Cogliano developed a method whereby such polynomial models can be identified in fewer computer runs than previous methods allowed. Their concept was to do analysis in the frequency domain rather than the time domain. The virtual independence of frequency estimators in a spectrum means that many parameters can be tested independently within a single experiment using spectral methods.

This report attempts to extend the Schruben/Cogliano methodology to cover a more general class of models which includes discrete-valued parameters, such as policy decisions or capacities of queues. We evaluated the use of discrete-valued functions as a basis for spectral analysis. Several function sets were considered as possibilities, and Walsh

functions were selected as the best choice.

Our preliminary results indicate that Walsh analysis may present a promising method for identifying significant parameters. However, the method exhibits undesirable behavior when time lags are present in the model.

Accesion For	
NTIS CRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

QUALITY INSPECTED 3

TABLE OF CONTENTS

1) Introduction.....1

2) Orthogonal Representations.....4

2.1) General Background.....4

2.2) Rademacher Functions.....5

2.3) Walsh Functions.....7

2.4) Haar Functions.....10

2.5) Characteristics of Walsh Functions.....13

2.6) Generating Walsh Functions.....19

3) Design of a Spectral Experiment.....22

3.1) Distribution of the Spectrum Estimator...22

3.2) Sequency Selection.....24

3.3) Design of an Experiment.....28

4) Analysis and Conclusions.....32

4.1) Verification of Walsh Analysis.....32

4.2) A Polynomial Model.....36

5) Future Research.....39

6) Bibliography.....42

7) Appendix 1: programs.....43

8) Appendix 2: lag effects.....55

LIST OF FIGURES

1) Example of Rademacher functions.....6

2) Example of Walsh functions.....8

3) Example of Haar functions.....11

4) A nonlinear response surface.....18

5) Spectrum of signal run with noise component.....34

6) Spectrum of noise component.....34

7) Spectrum of signal without noise component.....35

8) Spectrum of signal/noise ratio.....35

9) Spectrum of signal/noise ratio for polynomial model...38

Section 1 Introduction

The identification of parameters or factors which affect performance is an important area of operations research and statistics. Until recently it has been too expensive for many computer simulation studies, which Hillier and Lieberman state is one of the major shortcomings of simulation[7]. The traditional approach to this problem is to make separate computer runs for each of many different factor values. The basic experimental unit is the computer run. Although the number of runs required may be reduced using screening designs[10], it increases geometrically with the number of factors. This approach becomes prohibitively expensive in terms of both user and computer time for all but the simplest of models.

Schruben and Cogliano[12] (S/C) addressed this problem utilizing an entirely different approach. Normally one views a parameter as a fixed, possibly unknown, attribute of the system. However, in a computer simulation the experimenter has complete control of the model and can alter parameter values during the run. Hence the terms parameter and factor can be used interchangeably in simulation. Schruben and Cogliano proposed varying the parameters sinusoidally during a run. Each parameter is assigned a unique frequency, and spectral estimators are used to analyze the system output at the different frequencies. After performing a suitable statistical test, if the power spectrum is not significantly

different from zero at a given frequency it is concluded that the system is insensitive to the parameter which was assigned that frequency. By analyzing the spectrum instead of just the assigned frequencies, one can detect non-linear response through a relatively simple set of relations specified in the S/C paper. Most of their paper addresses the technical aspects of implementing such a procedure.

The advantage of the S/C approach is that analysis is moved to the frequency domain. The output time series is represented using trigonometric functions as a linear algebraic basis. The experimental unit becomes a *frequency band*, and a single run of the simulation contains many almost independent frequency bands. The number of runs required is greatly reduced.

One limitation of the S/C procedure is that it can only be used to evaluate continuous parameters. This problem can be removed by choosing a different set of functions as a basis. Three alternative discrete-valued function sets are Rademacher, Walsh, and Haar functions, which are named after their inventors. It is the goal of this report to extend the S/C procedure by considering the use of these alternative bases for representing the time series.

Although this report concentrates on the application of Walsh functions to computer simulations, the methodology outlined here is equally applicable for any type of 2^n factorial experimental design situation.

The following section gives a summary of pertinent information available in the literature about discrete functions. The intention is to provide the reader with a sufficient background to understand the material in subsequent sections. The method proposed in section 2.6 for detecting system gain apparently has not appeared in previous literature. Section 3 contains new material about the statistical properties of the Walsh spectrum estimator, and tells how to design an experiment to identify significant parameters in a model. Section 4 gives an in depth description and an example of how to use the proposed methodology. In section 5 we propose a number of extensions to this work.

Section 2 Orthogonal Representations

2.1 General background

Our notation for trigonometric functions will be consistent with that used in the recent Ph.D. report by V. J. Cogliano[5]. The symbol ω will denote cycles/time.

We will use the following definition of orthogonality (Beauchamp[2]).

Defn - The series $\{S_n(t)\}$ and $\{S_m(t)\}$ are orthogonal on the interval [0,T] if

$$\frac{1}{T}\int_0^T S_n(t)\ S_m(t)\ dt = \begin{cases} k & n = m \\ 0 & n \neq m \end{cases} \quad n,m \in Z^+$$

and are orthonormal if $k = 1$. (Z^+ is the set of nonnegative integers.)

Any time series $y(t)$ can be approximated over a finite interval [0,T] by a weighted sum of terms in an orthogonal series:

$$y(t) \approx \sum_{n=0}^{N-1} c_n S_n(t)$$

$S_n(t)$ is term n of the orthogonal series
c_n is a weight.

The most common measure of precision in such an approximation is the mean squared error, abbreviated MSE.

$$MSE = \int_0^T \left[y(t) - \sum_{n=0}^{N-1} c_n S_n(t)\right]^2 dt$$

MSE can be minimized by setting

$$c_n = \frac{1}{T} \int_0^T y(t)\ s_n(t)\ dt.$$

It can be seen that if $e^{i2n\omega t}$ is substituted for $s_n(t)$ then this is the familiar formula for calculating Fourier coefficients.

A basis series should have the property of completeness[6]. One definition of completeness is that there should not exist any function which is orthogonal to every element of the basis. This implies

$$\lim_{N\to\infty} MSE = 0$$

for all y(t) such that y(t) contains at most a countable number of discontinuities.

The following sections present a brief summary of the properties of three sets of discrete-valued functions which might be considered as algebraic bases for y(t). Rademacher, Walsh, and Haar functions are discussed extensively in all the literature for discrete spectral methods. It turns out that only Walsh functions have all the properties needed for the type of analysis of interest to us.

2.2 Rademacher functions

For the purpose of evaluating a simulation which contains discrete-valued parameters we want to utilize discrete-valued functions as our input series. One such series is the set of Rademacher functions, which are illustrated in figure 1. These are block pulses which alternate regularly between 1 and -1. Each function R(k,t) is a function of the continuous index

variable $t \in [0,T]$. By convention T is assumed to be 1 unless otherwise stated, but any value can be used with appropriate scaling. The first Rademacher function, $R(0,t)$ $(0 \leq t \leq T)$, is just a line with value 1. Every subsequent function is constructed by changing the sign on half of each interval in the previous function, so that the function changes sign in a regular manner.

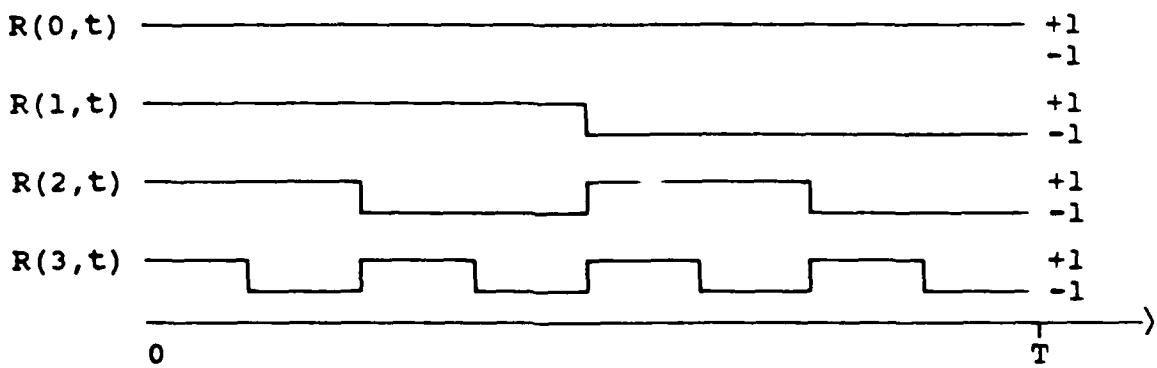


Figure 1. Example of Rademacher functions

These functions are easy to generate and are orthogonal, but do not form a complete basis. They therefore cannot be used to represent a general $y(t)$.

The orthogonality can be shown by considering any two distinct Rademacher functions $R(n,t)$ and $R(m,t)$, where n and m are sequence numbers of the Rademacher functions. By the construction of the series, the number of intervals for each function is a power of two. Each interval of constant value is of equal length for a given function. Let m be greater than n. Then each interval of fixed value for $R(n,t)$ will

correspond to an even number of equal length intervals with alternating signs for R(m,t), so the integral of the product of the two functions on each interval will be zero.

The incompleteness of a Rademacher basis can be seen by noticing that there is always an odd number of sign changes. Using Rademacher functions corresponds to restricting Fourier analysis solely to the use of unshifted sine functions. In fact, one way of defining Rademacher functions is

$$R(n,t) = \mathrm{sgn}[\ \sin(\ 2^n \pi \omega t\ )\ ],$$

where n is the sequence number of the Rademacher function

$$\mathrm{sgn}[\ x\ ] = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0. \end{cases}$$

Any cosine function of the form $\cos(2^n \omega t)$ is orthogonal to the basis. Attempting to complete the basis by supplementing it with terms $\mathrm{sgn}[\ \sin(\ n2\pi\omega t\ )\ ]$, where n is not a power of 2, and $\mathrm{sgn}[\ \cos(\ m2\pi\omega t\ )\ ]$, where m is any positive integer, destroys the orthogonal property.

Since Rademacher functions do not have the requisite properties of completeness and orthogonality, they cannot be used as a basis for spectral analysis.

2.3 Walsh functions

Walsh functions are a set of discrete-valued functions which assume only the values {-1,+1}. They are orthogonal and complete, having both even and odd symmetry. They can be constructed recursively, as products of Rademacher functions, or by constructing Hadamard matrices and sorting. Details of

construction will be presented later.

Each function is defined by convention on a fixed interval $t \in [0,T]$, and is written WAL(n,t). As with the Rademacher functions T is usually assumed to be 1 but can be any value if the function is scaled appropriately. The value n is an index which corresponds uniquely to the average number of zero crossings on the interval, which is called the sequency of the Walsh function in an analogy to frequency in trigonometric functions. Walsh functions are paired by even and odd symmetry and referred to as CAL and SAL functions, respectively. These are defined as follows:

$$\left.\begin{aligned} CAL(k,t) &= WAL(2k,t) \\ SAL(k,t) &= WAL(2k-1,t) \end{aligned}\right\} k = 1,\infty$$

where k is the sequency. Figure 2 illustrates four Walsh functions. The fifth and sixth Walsh functions are shown to illustrate that Walsh functions are not periodic in their variations.

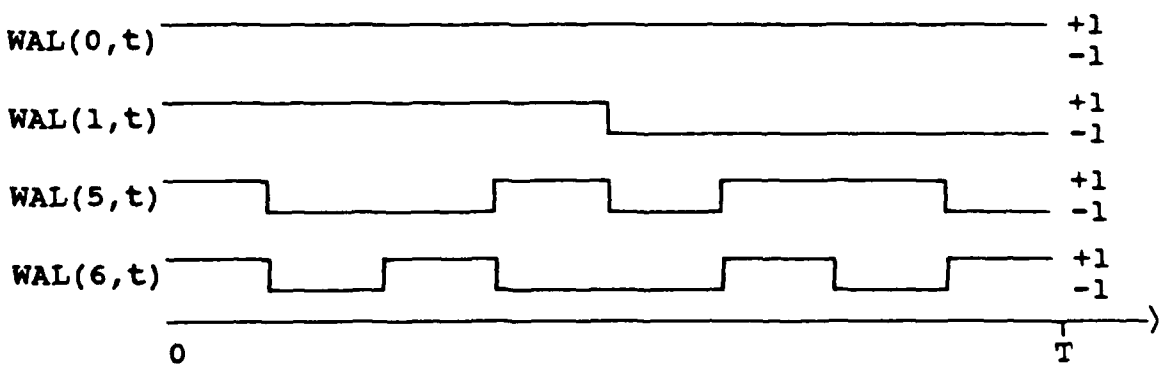


Figure 2. Example of Walsh functions

Walsh and Fourier spectra have a one to one correspondence. This can be seen from the following sets of relations, which are used to represent a time series y(t) in terms of a Fourier and a Walsh basis, respectively:

$$\text{(Fourier)}\quad y(t) = a_0 + \sum_{m=1}^{\infty} a_m \cos 2\pi mt + \sum_{m=1}^{\infty} b_m \sin 2\pi mt$$

$$\text{(Walsh)}\quad y(t) = A_0 + \sum_{n=1}^{\infty} A_n \mathrm{CAL}(n,t) + \sum_{n=1}^{\infty} B_n \mathrm{SAL}(n,t)$$

where a_i, b_i, A_i, and B_i are the Fourier and Walsh coefficients, respectively, denoted as c_n's in section 2.1. From this set of equalities it is shown by Maqusi[11] that

$$A_0 = a_0$$

and

$$\left.\begin{aligned} A_n &= \sum_{m=1}^{\infty} a_m \int_0^1 \cos 2\pi mt \ \mathrm{CAL}(n,t)\ dt \\ B_n &= \sum_{m=1}^{\infty} b_m \int_0^1 \sin 2\pi mt \ \mathrm{SAL}(n,t)\ dt \end{aligned}\right\} n = 1,\infty$$

or

$$\left.\begin{aligned} a_m &= \sum_{n=1}^{\infty} A_n \int_0^1 \mathrm{CAL}(n,t) \cos 2\pi mt\ dt \\ b_m &= \sum_{n=1}^{\infty} B_n \int_0^1 \mathrm{SAL}(n,t) \sin 2\pi mt\ dt \end{aligned}\right\} m = 1,\infty.$$

Clearly Walsh and Fourier spectra can be converted back and forth. We have found the linear change of basis transform, regardless of the fact that both bases are infinite. From linear algebra we know that for any complete series we consider we will be able to find a transformation to the familiar Fourier spectrum. By considering alternative bases we have neither gained nor lost information relative to a Fourier based spectral analysis. However, there are gains

to be had by using the Walsh representation.

The first advantage for Walsh analysis lies in the number of terms needed to approximate a discrete or discontinuous time series from the simulation output. Walsh functions are more efficient than trigonometric functions for this purpose, since it generally requires a large number of trigonometric terms to adequately approximate a discontinuity. For the purposes of altering the value of a discrete input parameter of the model, trigonometric functions are inappropriate.

The second advantage is computational. According to Beauchamp it takes $n \log_2 n$ complex <u>multiplications</u> <u>and additions</u> to evaluate a fast Fourier transform. Since the Walsh function can only have the values +1 or -1, evaluation requires only $n \log_2 n$ <u>additions</u> if the analogous fast transformation is used. Although the degree of improvement in efficiency is machine dependent, addition is much faster than multiplication on digital computers using current technology.

2.4 Haar functions

Haar functions are discrete-valued functions which are defined as

$$\mathrm{HAR}(0,t) = 1 \qquad 0 \le t \le 1$$

$$\mathrm{HAR}(1,t) = \begin{cases} 1 & 0 \le t < 1/2 \\ -1 & 1/2 \le t \le 1 \end{cases}$$

$$\mathrm{HAR}(2^p + n,t) = \begin{cases} \sqrt{2^p} & n/2^p \le t < (n + 1/2)/2^p \\ -\sqrt{2^p} & (n + 1/2)/2^p \le t < (n + 1)/2^p \\ 0 & \text{elsewhere.} \end{cases}$$

This definition allows for a sequential unique numbering system. The first seven Haar functions are illustrated in figure three. Notice that there is no unique sequency correspondence such as exists with the Walsh functions.

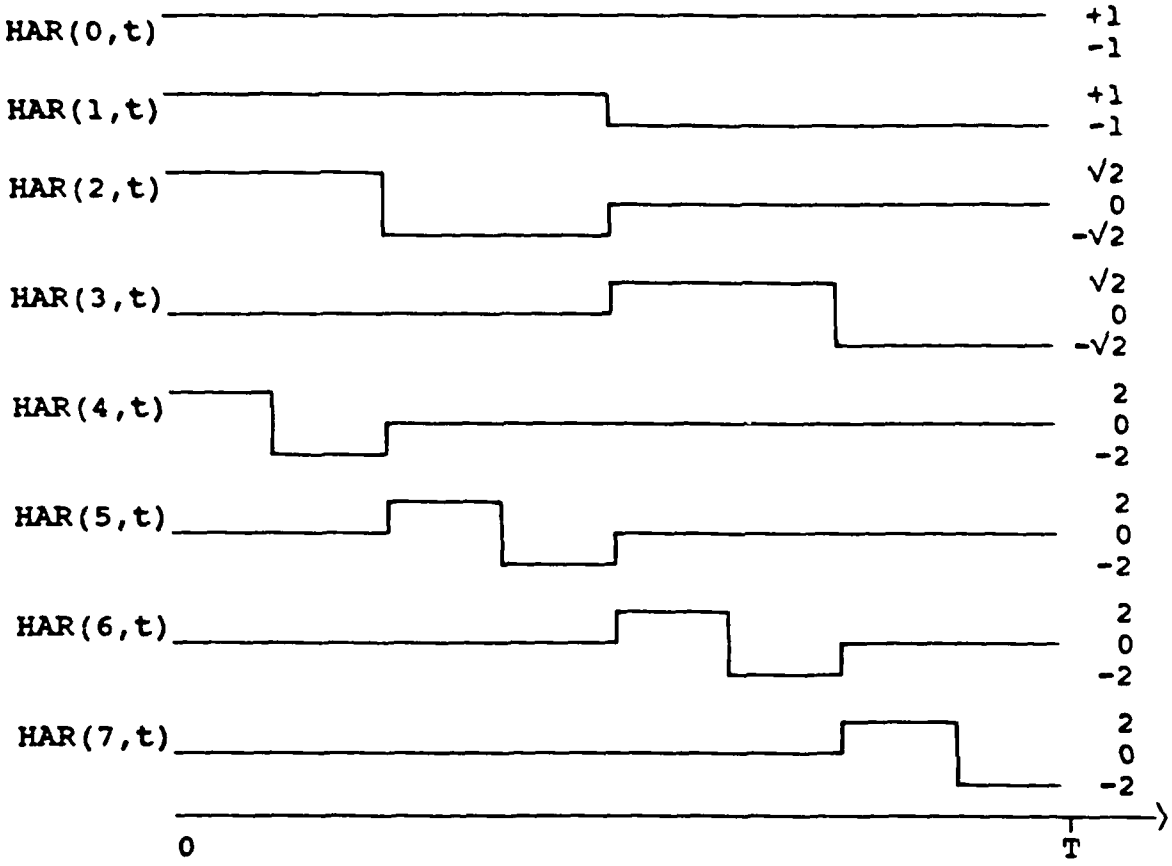


Figure 3. Example of Haar functions

A different indexing scheme groups Haar functions by degree i, the number of zero crossings in width 2^{-i}. The definition then becomes

$$\mathrm{HAR}(0,1,t) = 1 \qquad 0 \le t \le 1$$

$$\mathrm{HAR}(i,j,t) = \begin{cases} \sqrt{2^i} & (j-1)/2^i \le t \le (j-1/2)/2^i \\ -\sqrt{2^p} & (j-1/2)/2^i \le t < j/2^i \\ 0 & \text{elsewhere} \end{cases}$$

$$i = 0,1,2,\ldots \qquad j = 1,\ldots,2^i.$$

Only one term of each degree can be utilized if Haar functions are used for identification. Otherwise it would be impossible to distinguish whether an observed outcome was due to a specific parameter or due to a lag effect from a different parameter. This property could actually be advantageous for the purpose of identifying lagged models. Furthermore, there is a fast Haar transform which requires only $2(N - 1)$ additions.

These gains are offset by two disadvantages. The first is that Haar functions consist of three states and so cannot be used as input for binary parameters. The second and major disadvantage is that there is not a convenient product relationship for Haar functions such as exists for Walsh and trigonometric functions. The product of two Haar functions is either zero if there is no overlap of the non-zero intervals, or $\pm A_i$ HAR(k,t), where A_i is the amplitude of the Haar function with the larger non-zero interval, and HAR(k,t) is the function with the smaller non-zero interval. This makes the use of Haar spectra to identify interaction terms or higher order polynomial terms infeasible.

The remainder of this report will concentrate on Walsh functions since they appear to have the most desirable properties.

It should be noted that both Walsh and Haar analyses are computationally stable procedures, since they involve only addition operations. This is not true of Fourier analysis. Addition is order preserving for finite precision numbers, while multiplication is not.

2.5 Characteristics of Walsh Functions

At this point we must make the transition from continuous Walsh functions into the discrete domain. This is done by scaling the time axis relative to the highest order Walsh function of interest, and sampling the value of the continuous function over unit intervals. We will end up with a vector of N numbers which correspond to sampling WAL(k,t) at N equal intervals, i.e. at spacings of T/N. Thus WAL(k,i) ≡ WAL(k,t) where i is the integer portion of [(Nt/T)+1]. WAL(k,.) will be used to denote the vector consisting of WAL(k,i) for i = 1,...,N.

The value of N must be chosen so that each Walsh function has a unique vector associated with it. If k is the largest sequence number we wish to observe then we set $N = 2^{\lceil \log k \rceil}$, where the log is base 2 and $\lceil x \rceil$ is the smallest integer greater than x. As an example, if we were interested in Walsh functions up to order 5, we could represent them as vectors of length 8 whose elements assumed the values ±1.

$$\mathrm{WAL}(0,.) = (\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1)$$

$$\mathrm{WAL}(5,.) = (\ 1,-1,-1,\ 1,-1,\ 1,\ 1,-1)$$

Arithmetic operations on Walsh functions make extensive use of the ⊕ operator, which is called a dyadic sum. The dyadic sum is a bitwise XOR operation, where p XOR q is defined by the following table.

q \ p	0	1
0	0	1
1	1	0

examples of ⊕ operator:

$$\begin{array}{r} 7 \\ \oplus\ 5 \\ \hline 2 \end{array} \iff \begin{array}{r} 111 \\ \oplus\ 101 \\ \hline 010 \end{array} \qquad \begin{array}{r} 9 \\ \oplus\ 3 \\ \hline 10 \end{array} \iff \begin{array}{r} 1001 \\ \oplus\ 0011 \\ \hline 1010 \end{array}$$

We present a short list of interesting properties of Walsh functions. The list is not comprehensive. The following definitions will be used.

N is the number of observations

x_i is the ith parameter in the model

y_j is the jth term in the output time series

Y(k) is the kth term in the output series after transformation

$Y_c(k)$ and $Y_s(k)$ are the kth CAL and SAL terms, respectively

P(k) is the spectrum estimate of power at sequency k

It is worthwhile to note here that the Y's above are the estimates for the A's and B's used in the series representation in section 2.3. We are using the notation $y \mapsto Y$ to emphasize that the original observations and the estimated coefficients are a transform pair, either of which could be used to fully reconstruct the other.

PROPERTIES

symmetry	$WAL(k,i) = WAL(i,k)$
multiplication properties	$WAL(k,i)WAL(j,i) = WAL(j,i)WAL(k,i)$
	$WAL(k,i)WAL(j,i) = WAL(k \oplus j,i)$
	$CAL(k,i)CAL(j,i) = CAL(k \oplus j,i)$
	$SAL(k,i)CAL(j,i) = SAL(j \oplus [k-1],i)$
	$SAL(k,i)SAL(j,i) = CAL([k-1] \oplus [j-1],i)$
Walsh transformation	$Y(k) = \frac{1}{N}\sum_{i=0}^{N-1} y_i WAL(k,i)$
spectrum estimator	$P(0) = Y_c^2(0)$
	$P(k) = Y_c^2(k) + Y_s^2(k) \quad k=1,2,\ldots,(N/2)-1$
	$P(N/2) = Y_s^2(N/2)$

There is a shift theorem for trigonometric functions which states that shifting a trig function does not change the observed frequency. There is no comparable theorem for Walsh functions. Their shift behavior is discussed later.

Two immediate results for the $\oplus$ operator are that $k \oplus k = 0$ for all k, and $k \oplus 0 = k$ for all k. Applying these to the Walsh multiplication properties shows that if $x = WAL(k,i)$, then

$$x^2 = x * x = WAL(k \oplus k, i) = WAL(0, i)$$
$$x^3 = x * x^2 = WAL(k \oplus 0, i) = WAL(k, i)$$
$$x^4 = x * x^3 = WAL(k \oplus k, i) = WAL(0, i)$$
$$\vdots$$

By induction all even powers of x will have an indicator sequency of 0 and all odd powers will have sequency k/2. This means that we will be unable to estimate any but first order and interaction terms using a single Walsh function. If the problem is viewed geometrically this is reasonable, since using a Walsh function corresponds to sampling the data at only two points. With two points it is not possible to determine more than a first order model.

If we are only interested in significant parameter detection, this presents no difficulty. However, if we hope to use the data later to estimate a model there may be trouble. The situation is acceptable in the case of binary parameters, but there is a problem for continuous or p-state parameters where $p > 2$. We would prefer to have a single procedure which works for both continuous and discrete parameters. One way to overcome the problem is to use sums of Walsh functions as the input. A sum of two Walsh functions with amplitudes A_1 and A_2 will yield $\{\pm A_1 \pm A_2\}$, or up to four distinct sampling points. This should be sufficient to construct most non-linear models in practice.

example

$$x(i) = WAL(j,i) + WAL(k,i)$$

$$\begin{aligned} y(i) &= a_1 + a_2 x^2(i) \\ &= a_1 + a_2[WAL(j,i) + WAL(k,i)]^2 \\ &= a_1 + a_2[WAL(j \oplus j,i) + 2WAL(j \oplus k,i) \\ &\qquad + WAL(k \oplus k,i)] \\ &= (a_1 + 2a_2)WAL(0,i) + 2a_2 WAL(j \oplus k,i) \end{aligned}$$

We would observe an increase in the steady state term, sequency 0, and an interaction term at sequency $(j \oplus k)/2$ for the square term in a model of this form. A linear term in x would just have sequencies j/2 and k/2 appear in the spectrum.

This form of input may offer a viable approach to detecting system gain. Gain is the system's tendency to amplify or attenuate the response at different sequencies. If a parameter has a linear term, in the absence of system gain the amplitudes of WAL(j,i) and WAL(k,i) should be the same. By using a combination of high sequencies and low sequencies, and comparing the spectra of the two sequencies for a given run, any differences in power should be due to system gain or stochastic error. We can statistically test the spectrum for gain by performing an F-test as described later.

According to the literature Walsh functions are theoretically better than trigonometric functions for parameter detection in non-linear models. This can be explained by figure 4. Since input is held at a single value for each time interval, the system output has the characteristic of maintaining a single value over the same interval. The

amplitude of the output is changed, but the number of steps remains the same and will be detected by Walsh analysis.

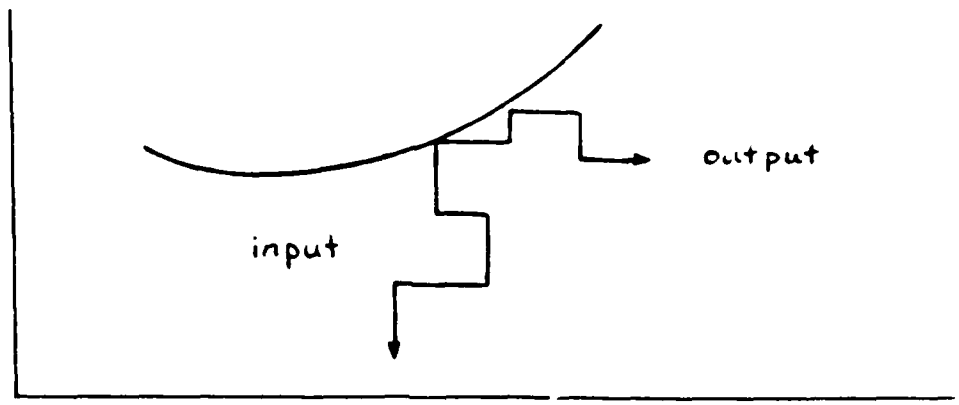


Figure 4. A nonlinear response surface

A problem in the use of Walsh analysis is that it is not invariant under phase shifts. According to Beauchamp the CAL and SAL functions of a given sequency vary inversely to each other in such a manner that the spectrum estimator is "relatively insensitive" to changes in phase. However, the practitioner should be cautious if the simulation model involves time lags. Our experiments indicate that the power spectrum is phase shift invariant if the sequency is N/2 or N/4, where N is the number of terms in the series. For other sequencies the spectrum will have a spike at the driving sequency, but there will be spikes of varying heights at other sequencies as well. The total power in the spectrum remains constant, and empirical evidence indicates that the bulk of the power is usually displayed at the original sequency. Spectra are usually plotted on a log scale, and it is

difficult to discern the primary from secondary spikes. Furthermore, there are cases where the power is uniformly distributed across all spikes, making identification impossible. We have been unable to find an analytical method of predicting the location of the extra spikes, other than by actually producing spectra of the sequency at a specified lag. See appendix 2 for details.

2.6 Generating Walsh functions

The vector notation introduced in the previous section allows us to use matrices to represent Walsh functions, which simplifies notation substantially. Throughout the remainder of this paper we will use W_N to denote an NxN matrix comprised of the first N Walsh functions.

example

$$W_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad W_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

Walsh functions can be generated recursively by the use of Hadamard matrices. One definition of a Hadamard matrix is

$$H_{2^{(n+1)}} = \begin{bmatrix} H_{2^n} & H_{2^n} \\ H_{2^n} & -H_{2^n} \end{bmatrix} \quad ; \quad H_1 = 1 \qquad (n \geq 0).$$

<u>example</u>

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Notice that H_N contains the first N Walsh functions as its rows and columns, but not in sequency order. After sorting we have W, which retains the symmetry property of H. However, while this is an interesting mathematical viewpoint it does not provide a convenient and efficient method for obtaining Walsh functions.

The discrete Walsh transformation can be viewed as a matrix transformation using W which performs a change of basis on a vector y of length $N = 2^n$.

$$Y = \frac{1}{N} W y$$

By the orthonormal property

$$W W^T = N I$$

where
T is the transpose operator
N is as defined above
I is the NxN identity matrix

$W = W^T$ since W is symmetric. W is invertible because it is composed of orthogonal vectors, and so must be of full rank. We can compute W^{-1} as follows:

$$W^{-1}W\,W^T = N\,W^{-1}I$$
$$W^T = N\,W^{-1}$$
$$W^{-1} = \frac{1}{N}\,W^T$$
$$W^{-1} = \frac{1}{N}\,W.$$

This means that the Walsh transformation is its own inverse after scaling by N. This provides us with perhaps the simplest method of generating Walsh functions. Just as with the Fourier transformation matrix, W has redundancies and can be reduced to a sparser matrix. Use of this sparse matrix results in a considerable savings in computational time and complexity. This is called the Fast Walsh Transform (FWT). By performing a FWT on a vector which has the value one in the location corresponding to the desired sequence number and which is zero elsewhere, we obtain the indicated Walsh function as output.

<u>example</u>

$$W_4 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \mathrm{WAL}(0,.)$$

$$W_4 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \mathrm{WAL}(2,.)$$

A computer algorithm for the FWT is included in appendix 1.

Section 3 Design of a Spectral Experiment

3.1 Distribution of the Spectrum Estimator

We assume that the output from a simulation of a stochastic system is composed of a deterministic signal term s and an error term ε.

$$y_i = s_i + \varepsilon_i$$

We will show that the spectrum estimator has a χ^2 distribution under the assumption that the additive error term has the distribution of discrete white noise as defined in Jenkins and Watts. This means that it is normally distributed with mean zero, variance σ_ε^2 , and $\mathrm{cov}(\varepsilon_i,\varepsilon_j) = 0$ for $i \neq j$. Using the terminology defined in section 2.5 we obtain

$$Y = \frac{1}{N} W y = \frac{1}{N} W (s + \varepsilon) = \frac{1}{N} W s + \frac{1}{N} W \varepsilon .$$

The random component associated with this is

$$Y_\varepsilon = \frac{1}{N} W \varepsilon .$$

Since the only variability in $Y_\varepsilon(i)$ is due to ε_i, clearly the variance of $Y_\varepsilon(i)$ is σ_ε^2 and $\mathrm{Cov}(Y_\varepsilon(i),Y_\varepsilon(j)) = 0$ for $i \neq j$.

Since $Y_\varepsilon(i)$ is a linear function of a normal random variable, it is distributed normally. Thus

$$\frac{Y_\varepsilon^2(i)}{\mathrm{Var}(Y_\varepsilon(i))} = \frac{Y_\varepsilon^2(k)}{\sigma_\varepsilon^2}$$

has a χ^2 distribution with υ, the degrees of freedom, equal to one. Since the covariance of the Y_ε's is zero the $Y_\varepsilon(i)$'s are independent.

The sum of two independent χ^2 random variables has a χ^2 distribution, so

$$\frac{P(i)}{\sigma_\varepsilon^2} = \left[Y_c^2(i) + Y_s^2(i) \right] \frac{1}{\sigma_\varepsilon^2} \sim \chi_2^2 \qquad i = 1,2,\ldots,\frac{N}{2}-1$$

and

$$\frac{P(i)}{\sigma_\varepsilon^2} \sim \chi_1^2 \qquad i = 0,\frac{N}{2}.$$

The variance of a χ^2 distribution with ν degrees of freedom is 2ν, so

$$\mathrm{Var}\left[\frac{P(i)}{\sigma_\varepsilon^2} \right] = \begin{cases} 4 & i = 1,2,\ldots,\frac{N}{2}-1 \\ 2 & i = 0,\frac{N}{2} \end{cases}$$

$$\mathrm{Var}[\ P(i)\ ] = \begin{cases} 4\ \sigma_\varepsilon^4 & i = 1,2,\ldots,\frac{N}{2}-1 \\ 2\ \sigma_\varepsilon^4 & i = 0,\frac{N}{2} \end{cases}$$

and the variance of P(i) is a constant, not dependent on N. This means that like the Fourier spectral estimator, the Walsh estimator is not consistent.

If we have two spectrum estimates $\hat{P}_1(i)$ and $\hat{P}_2(i)$ with the same number of observations, obtained from independent runs, then

$$\frac{\hat{P}_1(i)\ /\ P_1(i)}{\hat{P}_2(i)\ /\ P_2(i)} \sim \begin{cases} F_{2,2} & i = 1,\ldots,\frac{N}{2}-1 \\ F_{1,1} & i = 0,\frac{N}{2} \end{cases}$$

and under the hyporeport that $P_1(i) = P_2(i)$

$$\frac{\hat{P}_1(i)}{\hat{P}_2(i)} \sim \begin{cases} F_{2,2} & i = 1,\ldots,\frac{N}{2}-1 \\ F_{1,1} & i = 0,\frac{N}{2}. \end{cases}$$

This enables us to perform a statistical test of whether two independent spectrum estimates are equal. Hence we can test whether an observed spike in the spectrum is due to a factor effect or is due to error.

The next issue is how to increase the degrees of freedom for the estimator. Consider a run in which all of the sequencies assigned are less than or equal to k. Let $K = 2^{\lceil \log k \rceil}$, where the log is base 2. Then all crossterms will also be of order less than K, since the $\oplus$ operator works in a bitwise manner. We therefore can restrict the analysis to batches of K points. If we now let the simulation run for m batches of length K we will have m independent estimates of each of the K points under the assumption that the error is an additive discrete white noise term, since we are performing linear transformations on independent observations. We can now construct K new estimators from the sum of these m estimates. The new estimates will be distributed as χ^2 random variables with $\nu = 2m$ except at the endpoints, since they are constructed as a sum of m χ^2 variables with $\nu = 2$. The endpoint estimators are a sum of m χ^2 distributions with $\nu = 1$, and hence will have $\nu = m$.

3.2 Sequency Selection

The basic problem is to find a set of sequencies for n factors such that all of the original terms and the crossterms have a unique sequency. A straightforward solution is to

assign the factors sequencies which are powers of two, which actually assigns one bit to each factor. This will always yield a unique set of cross terms, and would be ideal if we were interested in all possible interactions. For example, for a three factor model with factor i assigned sequency 2^{i-1}, we might observe

sequency	→	term
1		x_1
2		x_2
4		x_3
3		x_1x_2
5		x_1x_3
6		x_2x_3
7		$x_1x_2x_3$

If we are interested only in two-term interactions this is inefficient. For a first order model there are n original terms and n choose 2 crossterms, or a total of $(n^2 + n)/2$ sequencies. We want to minimize the required number of observations, and the FWT needs 2^{k+1} observations to detect a sequency p in the range $2^{k-1} \leq p < 2^k$. The number is 2^{k+1} rather than 2^k because two Walsh functions are used for each sequency estimate. We can obtain substantial savings even for relatively small n if we can find $(n^2 + n)/2$ terms rather than 2^n terms. For example, if we have 10 factors we will generate 55 sequencies. If we select the original 10 in such a way that all 55 sequencies are less than 64, then we need only 128 observations. This is much more efficient than taking 2^{11}(=2048) observations to identify the same model.

Recall from section 2.6 that in order to detect a square term we need two Walsh sequencies. If the square term is present we will observe the crossterm of the two sequencies in the output. We can solve the selection problem as if we had a model with twice as many factors, and assign two unique sequencies per factor. Alternatively we can choose the sequencies as if there were one extra factor and make one sequency common to all factors being tested for a square term. Each of the two proposals has merit. The first one allows a pairing of high and low sequencies for every factor so that the model can be tested for system gain. The second has the desirable property of reducing the size of the problem we must solve to assign sequencies. The user must decide which of these considerations is more important for his specific application.

In order to solve the proposed problem we have written a computer program called SEQ which enumerates sets of n sequencies, generates the n choose 2 crossterms, sorts, and checks for uniqueness of all elements. It prints out the original n sequencies if they constitute a solution. The code can be found in appendix 1.

If we did this by explicit enumeration of all configurations the problem would rapidly become too large to solve. To find 3 input sequencies, for example, we have 6 terms. We want these to fall in the range 1 to 7. We can without loss of generality order them so that the first is smallest and the third is largest. Then to have uniqueness the

first can only range from 1 to 5, the second from 2 to 6, and the third from 3 to 7. If we call the number of parameters n and the highest sequency of interest is m, in our example $n = 3$ and $m = 7$. Then the number of configurations possible is m choose n. For our example this is 35. For $n = 10$ we need at least $(n^2 + n)/2 = 55$ sequencies. We might as well consider all sequencies up to the next power of two, i.e. sequencies 0 through 63, since they will all yield the same batch size. Then there are 63 choose 10 configurations, or more than 10^{11} possibilities to consider. While there may be several solutions, it could take a long time to find any of them.

We can drastically reduce the amount of work required by noting that for the set $\{x_1, x_2, \ldots, x_n\}$ to be a solution the subset $\{x_1, x_2, \ldots, x_{n-1}\}$ must also be a solution. Hence if we can find a solution to the problem at level k, we can add one additional element and range through the possible values until we have a solution at level k+1. If we fail to find any solutions we return to level k and search for a new solution there. By proceeding iteratively in this fashion we can build the solutions to fairly large problems quite easily. The program presented in appendix 1 can handle the case $n = 16$ in under three minutes on an IBM personal computer. Some solutions for various values of n are presented here.

n = 4	(1,2,4,8)
n = 5	(1,2,4,8,15)
n = 6	(1,2,4,8,15,16)
n = 16	(1,2,4,8,15,16,32, 51,64,85,106,128, 150,171,219,237)

A separate program called XTERM is included for predicting all the crossterms from a given set of input sequencies. Sample output from that program is shown here for the case n = 4.

```
enter the number of sequencies:        4
enter the sequencies:
       1        2        4        8
  1 xor    2:    3
  1 xor    4:    5
  1 xor    8:    9
  2 xor    4:    6
  2 xor    8:   10
  4 xor    8:   12
```

3.3 Design of an Experiment

We will now describe a general methodology for designing a Walsh spectral experiment for identification purposes.

The experimenter must first establish the number of factors to be identified, and the order of the model to be fit. If a model of greater than order 2 is desired we recommend using sequencies which are powers of two, as described in section 3.2. Otherwise the experimenter can use the programs included in appendix 1.

Next the experimenter should decide whether he needs two sequencies per factor or can utilize a common term approach. Two sequencies per factor rapidly increases the required number of sequencies, and hence increases the number of observations. For a minimum number of observations the common term approach is preferable, but it has two disadvantages. The first is that it may be more susceptible to system gain behavior, since not every factor can be assigned both a high and a low sequency. The second disadvantage is that it is not easy to discern the difference between crossterms and square terms in the model. For example, consider the following two factors

$$x_1 = a_1 \, [WAL(p,t) + WAL(q,t)]$$

$$x_2 = a_2 \, [WAL(p,t) + WAL(r,t)]$$

where a_1 and a_2 are the amplitudes of x_1 and x_2 respectively. Then

$$x_1^2 = a_1^2 \, [WAL(p \oplus p,t) + WAL(q \oplus q,t) + 2 \, WAL(p \oplus q,t)]$$

$$= 2 \, a_1^2 \, [WAL(0,t) + WAL(p \oplus q,t)]$$

$$x_1 x_2 = a_1 a_2 [WAL(p,t) + WAL(q,t)] [WAL(p,t) + WAL(r,t)]$$

$$= a_1 a_2 [WAL(p \oplus p,t) + WAL(p \oplus r,t) + WAL(p \oplus q,t) + WAL(q \oplus r,t)]$$

$$= a_1 a_2 [WAL(0,t) + WAL(p \oplus r,t) + WAL(p \oplus q,t) + WAL(q \oplus r,t)]$$

Both the x_1^2 and $x_1 x_2$ terms share $WAL(p \oplus q,t)$ as a common identifying sequency. In the absence of terms $WAL(p \oplus r)$ and $WAL(q \oplus r)$ we would conclude that the spike in the spectrum was

due to an x_1^2 effect, but if the other terms are present we cannot exclude the possibility that there is an x_1^2 effect as well as an x_1x_2 effect. Note, however, that all three spikes identifying an x_1x_2 term should have the same height. If the height at p⊕q is greater than the other two heights, it is probably due to the addition of an x_1^2 term, although it might be due to gain.

After deciding on the number of sequencies which will be needed, the experimenter should select a set such that crossterms are uniquely identifiable. The program SEQ in appendix 1 can be used for this purpose.

The user should then run the experiment by varying each factor according to the even numbered Walsh function which corresponds to its assigned sequency. The odd sequence number <u>cannot</u> be used because it does not preserve the ⊕ operator under which the sequencies were selected, while the even sequence numbers do. For example, if parameters x_1 and x_2 are assigned sequencies 4 and 8, respectively, then we have the following.

```
Seq(4) <=> WAL(7,t) , WAL(8,t)
Seq(8) <=> WAL(15,t) , WAL(16,t)
```

If the even terms are used

```
WAL(8,t) WAL(16,t) = WAL(8⊕16,t)
                   = WAL(24,t)
                   => seq(12)
                   = seq(8⊕4).
```

If the odd terms are used

$$\begin{aligned} WAL(7,t)\ WAL(15,t) &= WAL(7 \oplus 15,t) \\ &= WAL(8,t) \\ &\Rightarrow seq(4) \\ &\neq seq(8 \oplus 4). \end{aligned}$$

Thus we can only use the sequency based experimental design if we convert to even numbered Walsh functions.

Finally, the output from the experiment should be evaluated using a program such as SPECTRUM in appendix 1. The spectrum obtained from that program can be analyzed for sequency components corresponding to the factor inputs.

Section 4 Analysis and Conclusions

4.1 Verification of Walsh Analysis

Before proceeding further we wanted to verify the ability of Walsh analysis to extract the correct signal in the presence of noise. To this end we constructed a file of random numbers to be used as a signal with additive noise. A second file consisted of the fractional parts of the numbers from the first file, and was considered as the noise. A third file was constructed from the integer portion of the first file, and considered to be a noiseless signal. The three files are presented here.

Signal

4.1	3.9	4.1	3.9	3.8	4.7	5.2	6.4
4.1	3.9	4.1	3.9	3.8	4.7	5.2	6.4

Noise

0.1	0.9	0.1	0.9	0.8	0.7	0.2	0.4
0.1	0.9	0.1	0.9	0.8	0.7	0.2	0.4

Noiseless

4.0	3.0	4.0	3.0	3.0	4.0	5.0	6.0
4.0	3.0	4.0	3.0	3.0	4.0	5.0	6.0

All three files were subjected to spectral analysis using the program SPECTRUM in appendix 1. Each file was plotted individually, and the signal and noise files were used to form the signal to noise ratio. The spectral plots are presented as figures 5 through 8. The signal plot in figure 5 has a high sequency component which is not present in the noiseless

plot of figure 7. This noise component is successfully eliminated by taking the signal to noise ratio in figure 8. That plot produces spikes in the same location as those in the noiseless spectrum, indicating that we have identified the correct sequency components of the true signal in the presence of noise.

The noise component in this case does not meet the assumption of being normally distributed with mean zero. This indicates that the discriminating ability of Walsh spectra is not necessarily dependent on the distribution of the error terms. That distribution is important only in being able to construct a statistical test for identifying signficant terms. If the assumptions of section 3.1 hold, the values to the left of the signal to noise ratio plot are F values, and can be used directly after computing the appropriate degrees of freedom.

```
enter the name of the signal file:
signal

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       16


 2.0362650E+01    0+*****************************************
 0.0000000E+00     +
 4.1281260E-01     +******************
 0.0000000E+00     +
 1.5156250E-01     +************
 0.0000000E+00    5+
 9.9062500E-02     +**********
 0.0000000E+00     +
 4.5156280E-02     +*****
```

Figure 5. Spectrum of signal run with noise component

```
enter the name of the signal file:
noise

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       16


 2.6265620E-01    0+*****************************************
 0.0000000E+00     +
 1.2812500E-02     +**********
 0.0000000E+00     +
 1.4062500E-02     +***********
 0.0000000E+00    5+
 3.6562490E-02     +********************
 0.0000000E+00     +
 4.5156240E-02     +***********************
```

Figure 6. Spectrum of noise component

```
enter the name of the signal file:
noiseles

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       16


 1.6000000E+01    0+*****************************************
 0.0000000E+00     +
 5.0000000E-01     +**************
 0.0000000E+00     +
 2.5000000E-01     +*******
 0.0000000E+00    5+
 2.5000000E-01     +*******
 0.0000000E+00     +
 0.0000000E+00     +
```

Figure 7. Spectrum of signal without noise component

```
enter the name of the signal file:
signal

enter the name of the noise file or the word "nul":
noise

enter the number of batches and batch size of the files:
       1       16


 7.7525870E+01    0+*****************************************
 1.0000000E+00     +*******
 3.2219520E+01     +**********************************
 1.0000000E+00     +*******
 1.0777780E+01     +**************************
 1.0000000E+00    5+*******
 2.7094020E+00     +****************
 1.0000000E+00     +*******
 1.0000010E+00     +*******
```

Figure 8. Spectrum of signal/noise ratio

4.2 A Polynomial Model

We next tested a model with 4 factors. The model is as follows:

$$y_t = x_{1,t} + x_{1,t}^2 + \frac{x_{2,t}}{3} + \frac{x_{2,t}x_{3,t}}{3} + .001\, x_{4,t} + \text{error}_t$$

where x_1 through x_4 are factors

$$\text{error}_t = .8\ \text{error}_{t-1} + .6\ z$$

$$z \sim N(0,1).$$

The spectral plot included as figure 9 is based on 3 batches of 32 observations. Sequencies were assigned as follows, using the common term scheme.

x_1: seq(15) + seq(8)

x_2: seq(15) + seq(4)

x_3: seq(15) + seq(2)

x_4: seq(15) + seq(1)

We would predict the following sequencies to be observed.

x_1 : seq(8),seq(15)

x_1^2 : seq(0),seq(7)

x_1 : seq(4),seq(15)

x_2x_3: seq(0),seq(6),seq(11),seq(13)

x_4 : seq(1),seq(15)

The derivations of the linear terms should be evident. We will derive the x_2x_3 crossterm as an example.

$$x_2x_3 = [seq(15) + seq(4)]\ [seq(15) + seq(2)]$$

$$= [seq(15)^2] + [seq(15)\ seq(2)] + [seq(15)\ seq(4)] + [seq(4)\ seq(2)]$$

$$= seq(15 \oplus 15) + seq(15 \oplus 2) + seq(15 \oplus 4) + seq(4 \oplus 2)$$

$$= seq(0) + seq(13) + seq(11) + seq(6)$$

Spikes were observed at all predicted locations, and as expected the spike for x_4 was negligable. The spike for the $x_2/3$ term was small but noticable, and would probably be accepted as significant if more observations were taken ($F^{.05}_{6,6} = 4.28$).

```
enter the name of the signal file:
sig.run

enter the name of the noise file or the word "nul":
nse.run

enter the number of batches and batch size of the files:
       3       32


 4.4337260E+01     0+********************************
 9.9835490E-01      +******
 9.9999980E-01      +******
 9.9999940E-01      +******
 1.5021450E+00      +********
 1.0000000E+00     5+******
 1.1210890E+01      +**********************
 2.1937100E+02      +*****************************************
 4.2658360E+01      +******************************
 1.0000000E+00      +******
 1.0000010E+00    10+******
 4.9413060E+01      +*******************************
 9.9999920E-01      +******
 1.0608800E+01      +**********************
 9.9999960E-01      +******
 1.4179650E+02    15+***************************************
 9.9999980E-01      +******
```

Figure 9. Spectrum of signal/noise ratio for polynomial model

Section 5 Future Research

We present here several topics which are interesting extensions of the current work.

5.1 Generalized Walsh functions

While Walsh and traditional spectral analysis can be applied to any time series, it is preferable to tailor the analysis to the type of system being analyzed. One area for future research is to investigate the use of generalized Walsh functions for analyzing p-valued parameters. Elementary Walsh functions are binary, and can be used to evaluate binary parameters. Generalized Walsh functions are series which have the orthogonality and completeness properties while assuming p discrete values, where $p \geq 3$. Using these functions as the basis for spectral analysis, it would be theoretically possible to evaluate systems with multi-valued discrete parameters by spectral methods.

5.2 Parameter Estimation

Once the significant factors have been identified a logical next step is to attempt parameter estimation for our model of the system. This problem will potentially be complicated by non-linearities in the system. After some preliminary readings we would recommend an iterative least squares approach.

5.3 Minimal I/O System Estimation

One extension of parameter estimation would be to attempt to reduce the matrix of coefficients to find a minimal representation for the I/O behavior of the system. A representation of this sort would have many possible applications. It could be used for stability/sensitivity analysis of the system via analytic methods. It could also be used as an external control system for variance reduction of the computer simulation. It might be fruitful to investigate whether there is a relationship between minimal I/O systems and minimal representations of the simulation using event graphs.

5.4 Parameter Optimization

Optimization using computer simulation is generally regarded as a poor idea. It quickly becomes an expensive proposition if one employs the traditional approach of using each run as a single observation. However, using spectral methods it should be possible to use simulation for optimization at a substantially reduced cost. One possible approach would be to try to converge iteratively within a single run. Another approach would be to try to solve the problem analytically after estimating the system.

5.5 Aid to Variance Reduction

We have already mentioned the possible use of control variate variance reduction if we have estimated the I/O model

of the system. However, it is still possible to improve variance reduction without estimating a model by using term sensitivity to indicate which variance reduction technique is appropriate. For example, antithetic variates should only be used if the response surface is monotonic, so the presence of non-linear polynomial terms would contraindicate the use of this method.

5.6 Multiple-Input/Multiple-Output Models

Thus far all work done by Schruben, Cogliano, and Sanchez has focused on single output models. Work needs to be done so that the procedure can be applied to more general simulation models.

5.7 Statistical Design of Experiments

It is worth noting that the charts found in many experimental design texts for setting factor levels in a 2^k factorial experimental design are matrices of Walsh functions. One can conjecture that a spectral approach to experimental design has been overlooked prior to now because most statisticians are familiar only with the traditional Fourier spectrum. It may be possible to gain additional insights into design problems by using discrete spectra. For instance, generalized Walsh functions may be useful for designing and analyzing p^k factorial experiments.

Bibliography

[1] Askey - Orthogonal Polynomials and Special Functions, 1975, SIAM

[2] Beauchamp - Walsh Functions and Their Applications, 1975, Academic Press

[3] Chatfield - The Analysis of Time Series: Theory and Practice

[4] Churchill & Brown - Fourier Series & Boundary Value Problems, 1978, McGraw Hill

[5] Cogliano - Sensitivity Analysis and Model Identification in Simulation Studies: A Frequency Domain Approach, 1982 Ph.D. dissertation, SORIE, Cornell University.

[6] Higgins - Completeness & Basis Properties of Sets of Special Functions, 1977, Cambridge University Press

[7] Hillier & Lieberman - Introduction to Operations Research, 1967, Holden-Day, Inc.

[8] Jenkins & Watts - Spectral Analysis and its Applications, 1968, Holden-Day, Inc.

[9] Karpovsky - Finite Orthogonal Series in the design of Digital Devices, 1976, John Wile & Sons

[10] Kleijnan - Statistical Methods in Simulation, 1975, Marcel Dekker, Inc.

[11] Maqusi - Applied Walsh Analysis, 1981, Heyden

[12] Schruben & Cogliano - Sensitivity Analysis of Discrete Event Simulations: A Frequency Domain Approach, TR514, SORIE, Cornell University

[13] Strang - Linear Algebra, 1976, Academic Press, Inc.

Appendix 1

All programs were implemented in the Pascal language. Pascal was chosen for its readability and transportability. Standard features are used throughout with the exception of the program for enumerating sequencies. That program makes use of the IBM and Turbo Pascal compilers ability to do XOR operations on integers. Several programs were developed on the Purdue UNIX system and then implemented on an IBM personal computer.

The first program is a procedure which does a Fast Walsh Transform using the signal flow diagram in figure 3.3 of Beauchamp.

The second program generates permutations of sequencies and cross-terms for a given number of parameters. After generation the sequencies are sorted using a quicksort algorithm. They are checked for redundant values. If there are no redundancies the input sequencies are printed.

The third program will take a list of sequencies as input, and generate all the crossterms as output. This procedure is described in section 3.2.

The fourth program is a general procedure for doing Walsh analysis on an input data set. It prompts the user for the location of the data, number of batches, and size of each batch. It then does the Walsh analysis and prints the output as a barplot. If two input files are specified it produces a signal to noise ratio barplot using the first file as the numerator and the second as the denominator. Barplots are produced on a log scale so that smaller spikes will not be

excluded from visibility. The raw value is printed to the left of the plot, and in the case of a signal to noise plot can be regarded as an F ratio if the assumptions of section 3.1 are met.

```
procedure walsh(var v: vector;start,lgth_v: integer);
(*************************************************************
 procedure to perform fast walsh transform on a vector of
 length n.  Note that n must be a power of 2.  Also of
 interest is that by the symmetry of the FWT this algorithm
 is its own inverse, and can be used to generate walsh
 functions.
 *************************************************************)

var temp:vector;
    half_lgth, i, j, k: integer;

begin;
  half_lgth := lgth_v shr 1;
  j := start;
  for i := start to (start + half_lgth - 1) do
      begin;
        k := i + half_lgth;
        temp[i] := v[j] + v[j+1];
        temp[k] := v[j] - v[j+1];
        if ((i - start) mod 2) = 1 then temp[k] := -temp[k];
        j := j + 2;
      end;
  for i := start to (start + lgth_v - 1) do
      v[i] := temp[i];
  if (half_lgth > 1) then
      begin;
        walsh(v,start,half_lgth);
        walsh(v,start+half_lgth,half_lgth);
      end;
 end; {walsh}
```

```
program seq(input,output);
{*************************************************************
   Program for generating sequencies whose crossterms
   are unique.  Generates permutations, sorts, and
   compares adjacent elements for equality.
 *************************************************************}

type vector = array[1..210] of integer;

var v,p: vector;
    i,n_param,length,uplim: integer;


procedure permute(var v: vector; n: integer);
{generate the n choose 2 interaction terms for n parameters}
var i,j,index: integer;

begin;
index := n;

for i := 1 to (n - 1) do
  begin;
    for j := (i + 1) to n do
      begin;
        index := index + 1;
        v[index] := v[i] xor v[j];
      end;
  end;
end;{permute}


procedure sort(var v: vector; lower,upper: integer);
{sort of a vector v of length n using recursive quicksort}

var pivot: integer;

  procedure partition(var v:vector;lower,upper:integer;
                      var pivot:integer);

        procedure swap(var v:vector;j,k:integer);
        {swaps the contents of two vector locations}

          var temp: integer;

          begin;
            temp := v[j];
            v[j] := v[k];
            v[k] := temp;
          end; {swap}{
```

```
    begin;
      pivot := lower;

      repeat

        repeat
          lower := lower + 1;
        until ((v[lower] > v[pivot])  or  (lower >= upper));

        while ((lower < upper)  and  (v[pivot] < v[upper]))
          do   upper := upper - 1;

        if (lower < upper) then    swap(v,lower,upper);

      until (lower >= upper);

      if v[lower] < v[pivot] then
        begin;
          swap(v,lower,pivot);
          pivot := lower;
        end
      else
        begin;
          swap(v,lower-1,pivot);
          pivot := lower -1;
        end;

    end; {partition}

  begin; {quicksort}
    partition(v,lower,upper,pivot);
    if (pivot - lower) < (upper - pivot) then
       begin;
         if ((pivot - lower) >= 2) then sort(v,lower,pivot-
1);
         if ((upper - pivot) >= 2) then
sort(v,pivot+1,upper);
       end
    else
       begin;
         if ((upper - pivot) >= 2) then
sort(v,pivot+1,upper);
         if ((pivot - lower) >= 2) then sort(v,lower,pivot-
1);
       end;

end; {sort}{
```

```
function accept(var v: vector; n,length: integer): boolean;
{accept or reject a set of parameter sequencies as having
 unique elts}

var i: integer;
    result: boolean;

begin;
permute(v,n);
sort(v,1,length);
result := true;
i := 1;

repeat
  if v[i] = v[i+1] then
      result := false;
  i := i + 1;
until (result = false) or (i >= length);
accept := result;
end;{accept}


procedure add_lvl(var v,p: vector;
current_lvl,n_param,tot_lgth: integer);
{given unique elts at level n, add elts at level n+1}
var i,j,length: integer;

begin;
length := (sqr(current_lvl) + current_lvl) div 2;

for i := (p[current_lvl-1]+1) to (tot_lgth-(n_param-
current_lvl)) do
  begin;
    for j := 1 to (current_lvl - 1) do v[j] := p[j];
    v[current_lvl] := i;
    if accept(v,current_lvl,length) then
      begin;
        p[current_lvl] := i;
        if current_lvl < n_param then
          add_lvl(v,p,current_lvl+1,n_param,tot_lgth)
        else
          begin;
            for j := 1 to current_lvl do write(p[j]:4);
            writeln;
          end;{write}
      end;{accept block}
  end;{loop}
 end;{add_lvl}{
```

```
begin;
write('enter # sequencies needed:');
readln(n_param);
length := (sqr(n_param) + n_param) div 2;
uplim := 1;
while uplim < length do uplim := uplim * 2;

for i := 1 to (uplim - (n_param - 1)) do
  begin;
    p[1] := i;
    add_lvl(v,p,2,n_param,uplim);
  end;

end. {SEQ}
```

```
program xterm(input,output);
(*************************************************************
   Program to calculate the crossterm sequency locations
 *************************************************************)

type vector = array[1..20] of integer;

var v: vector;
    i,n: integer;
    result: text;
    fname: string[14];

procedure permute(var v: vector; n: integer);
{generate the n choose 2 interaction terms for n parameters}
var i,j,index: integer;

begin;
for i := 1 to (n - 1) do
    for j := (i + 1) to n do
        writeln(result,v[i]:3,' xor ',v[j]:3,':', v[i] xor
v[j]:4);
end;{permute}


begin;
    writeln;
    write('enter the output filename:');
    read(fname);
    assign(result,fname);
    rewrite(result);
    writeln(result);
    write(result,'enter the number of sequencies:');
    writeln;
    write('enter the number of sequencies:');
    readln(n);
    writeln(result,n);
    writeln('enter the sequencies:');
    writeln(result,'enter the sequencies:');
    for i := 1 to n do read(Trm,v[i]);
    for i := 1 to n do write(result,' ',v[i]);
    writeln(result);
    permute(v,n);
    close(result);
 end.
```

```
program spectrum(input,output);
(*****************************************************************
 This program reads in the simulation output from a signal
run
 and a noise run, or from just a signal run.  It prompts the
 user for which type of input he has, the number of batches,
 and size of each batch.  It then calculates the sequency
 spectrum if there is only one input, or calculates the
signal
 to noise ratio if there are two.  Plots are on a log scale,
 and data values are printed to the left of the plot.  Output
 is written to a file of the users choosing.
*****************************************************************)

const size = 1024;
      half_size = 512;

type runtype = (signal,noise);
     vector = array[1..size] of real;
     sequency = array[0..half_size] of real;

var data: array[runtype] of vector;
    run,uplim: runtype;
    seq: array[runtype] of sequency;
    plot: text;
    infile: array[runtype] of text;
    signal_file,noise_file,plot_file: string[14];
    b_size,n_batches,n_seq,i,j: integer;(
```

```
procedure initialize;
  begin
  write('enter the name of the signal file:');
  readln(signal_file);
  assign(infile[signal],signal_file);
  reset(infile[signal]);
  writeln;
  write(
   'enter the name of the noise file or the word "nul":');
  readln(noise_file);
  if (noise_file = 'nul')
     then
       uplim := signal
     else
       begin
         assign(infile[noise],noise_file);
         reset(infile[noise]);
         uplim := noise;
       end;
  writeln;
  write(
   'enter the number of batches and batch size of the
 files:');
  readln(n_batches,b_size);
  writeln;
  n_seq := b_size shr 1;
  for i := 0 to n_seq do
      for run := signal to uplim do
          seq[run][i] := 0.0;
  write('enter the name of the output file:');
  readln(plot_file);
  assign(plot, plot_file);
  rewrite(plot);
  writeln;
  write(plot,'enter the name of the signal file:');
  writeln(plot,signal_file);
  writeln(plot);
  write(plot,
   'enter the name of the noise file or the word "nul":');
  writeln(plot,noise_file);
  writeln(plot);
  write(plot,
   'enter the number of batches and batch size of the
 files:');
  writeln(plot,n_batches,' ',b_size);
  writeln(plot);
  writeln(plot);
  end;{initialize}{
```

```
procedure logplot(var v: sequency; lgth: integer);
(A procedure to produce a log scale barplot from vector v.
 The plot is scaled by the largest difference in v.)

var i,j: integer;
    max,min,range: real;

begin
max := 0.0;
min := 1E37;
for i := 0 to lgth do
  begin
    if v[i] > max then max := v[i];
    if (v[i] < min) and (v[i] > 0.0) then min := v[i];
  end;
max := ln(max);
min := ln(min/2.8);
if min >= max then min := 0;
range := max - min;

for i := 0 to lgth do
  begin (loop)
    write(plot,v[i]);
    if (i mod 5) = 0
      then
        write(plot,i:5,'+')
      else
        write(plot,'     +');
    if v[i] > 0.0 then
      for j := 1 to trunc(40*((ln(v[i])-min)/range)) do
        write(plot,'*');
    writeln(plot);
  end;{loop}
end;{logplot}

procedure crunch(var v: vector; var seq: sequency;
                 lgth,s_lgth: integer);
{procedure to calculate spectrum estimator for vector v,
 result placed in vector seq.}

var i: integer;

begin
seq[0] := seq[0] + sqr(v[1]/lgth);
seq[s_lgth] := seq[s_lgth] + sqr(v[lgth]/lgth);

for i := 1 to s_lgth-1 do
  begin
    seq[i] := seq[i] + sqr(v[2*i]/lgth) +
sqr(v[(2*i)+1]/lgth);
  end;
end;{crunch}{
```

```
(*i walsh.pas)

begin
initialize;
for j := 1 to n_batches do
  for run := signal to uplim do
    begin
      for i := 1 to b_size do read(infile[run],data[run][i]);
      walsh(data[run],1,b_size);
      crunch(data[run],seq[run],b_size,n_seq);
    end;
if uplim = noise then
  for i := 0 to n_seq do
    if (seq[noise][i] > 0.0) then
      seq[signal][i] := seq[signal][i] / seq[noise][i]
    else if((seq[noise][i] = 0.0) and (seq[signal][i] = 0.0))
then
      seq[signal][i] := 1.0;
logplot(seq[signal],n_seq);
close(plot);
end.
```

Appendix 2

Since there is no analytical method for evaluating the impact of a lag upon the Walsh spectrum, we resorted to empirical methods. It was hoped that we would observe a clear spike at the original sequency, and at worst a small amount of noise elsewhere. What we actually observed was that while the original sequency usually has the largest spike, at times it does not have any spike at all. We present here 18 spectral plots of different sequencies with different lags. All plots were constructed from 64 observations. The filename for input is descriptive of the original sequency and lag. For example, LAG602 contains Walsh sequence number 60 lagged 2 periods. Recall that:

$$\text{sequency} = \text{integer}[(\text{sequence \#} + 1) / 2]$$

Notice that sequence numbers 63, 31, and 15 are completely periodic with periods 1, 2 and 4, respectively. We consequently need not lag them more than their period lengths to have completely identified their behavior.

We include the program TESTLAG, which was used to generate the lagged Walsh functions. The output was plotted using SPECTRUM.

```
program lageffect(input,output);
(*****************************************************
  Generate Walsh functions of a specified lag
 *****************************************************)

type vector = array[0..1023] of real;

var v : vector;
    seq,size,lag,i : integer;
    outfile : text;
    fname : string[14];

{$i walsh.pas}

begin;{lageffect}
    write('enter size, sequence #, and lag:');
    readln(trm,size,seq,lag);
    write('enter the output filename:');
    readln(trm,fname);
    assign(outfile, fname);
    rewrite(outfile);
    for i := 0 to (size - 1) do v[i] := 0;
    v[seq] := 1;
    walsh(v,0,size);
    for i := 0 to (size - 1) do
      writeln(outfile,v[(i-lag) mod size]:3);
    close(outfile);
 end.{lageffect}
```

```
enter the name of the signal file:
a:lag51

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 7.8125000E-03      +*******
 0.0000000E+00      +
 6.6406250E-01      +*****************************************
 0.0000000E+00      +
 3.9062500E-02     5+********************
 0.0000000E+00      +
 7.8125000E-03      +*******
 0.0000000E+00      +
 7.8125000E-03      +*******
 0.0000000E+00    10+
 3.9062500E-02      +********************
 0.0000000E+00      +
 3.9062500E-02      +********************
 0.0000000E+00      +
 7.8125000E-03    15+*******
 0.0000000E+00      +
 7.8125000E-03      +*******
 0.0000000E+00      +
 3.9062500E-02      +********************
 0.0000000E+00    20+
 3.9062500E-02      +********************
 0.0000000E+00      +
 7.8125000E-03      +*******
 0.0000000E+00      +
 7.8125000E-03    25+*******
 0.0000000E+00      +
 3.9062500E-02      +********************
 0.0000000E+00      +
 3.9062500E-02      +********************
 0.0000000E+00    30+
 7.8125000E-03      +*******
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag52

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 3.1250000E-02      +************
 0.0000000E+00      +
 4.0625000E-01      +*****************************************
 0.0000000E+00      +
 1.5625000E-01     5+*******************************
 0.0000000E+00      +
 3.1250000E-02      +************
 0.0000000E+00      +
 3.1250000E-02      +************
 0.0000000E+00    10+
 1.5625000E-01      +*******************************
 0.0000000E+00      +
 1.5625000E-01      +*******************************
 0.0000000E+00      +
 3.1250000E-02    15+************
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    30+
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag53

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 7.0312500E-02      +***************************
 0.0000000E+00      +
 2.2656250E-01      +*************************************
 0.0000000E+00      +
 3.5156250E-01     5+*******************************************
 0.0000000E+00      +
 7.0312500E-02      +***************************
 0.0000000E+00      +
 7.8125000E-03      +********
 0.0000000E+00    10+
 3.9062500E-02      +**********************
 0.0000000E+00      +
 3.9062500E-02      +**********************
 0.0000000E+00      +
 7.8125000E-03    15+********
 0.0000000E+00      +
 7.8125000E-03      +********
 0.0000000E+00      +
 3.9062500E-02      +**********************
 0.0000000E+00    20+
 3.9062500E-02      +**********************
 0.0000000E+00      +
 7.8125000E-03      +********
 0.0000000E+00      +
 7.8125000E-03    25+********
 0.0000000E+00      +
 3.9062500E-02      +**********************
 0.0000000E+00      +
 3.9062500E-02      +**********************
 0.0000000E+00    30+
 7.8125000E-03      +********
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag54

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1      64


 0.0000000E+00     0+
 1.2500000E-01      +****************
 0.0000000E+00      +
 1.2500000E-01      +****************
 0.0000000E+00      +
 6.2500000E-01     5+*******************************************
 0.0000000E+00      +
 1.2500000E-01      +****************
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    30+
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag55

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1      64


 0.0000000E+00     0+
 1.9531250E-01      +************************************
 0.0000000E+00      +
 1.0156250E-01      +******************************
 0.0000000E+00      +
 3.5156250E-01     5+*****************************************
 0.0000000E+00      +
 7.0312500E-02      +***************************
 0.0000000E+00      +
 7.8125000E-03      +********
 0.0000000E+00    10+
 3.9062500E-02      +**********************
 0.0000000E+00      +
 3.9062500E-02      +**********************
 0.0000000E+00      +
 7.8125000E-03    15+********
 0.0000000E+00      +
 7.8125000E-03      +********
 0.0000000E+00      +
 3.9062500E-02      +**********************
 0.0000000E+00    20+
 3.9062500E-02      +**********************
 0.0000000E+00      +
 7.8125000E-03      +********
 0.0000000E+00      +
 7.8125000E-03    25+********
 0.0000000E+00      +
 3.9062500E-02      +**********************
 0.0000000E+00      +
 3.9062500E-02      +**********************
 0.0000000E+00    30+
 7.8125000E-03      +********
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag601

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 0.0000000E+00      +
 3.1250000E-02      +*********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 3.1250000E-02      +*********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 3.1250000E-02    10+*********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 3.1250000E-02      +*********
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 3.1250000E-02      +*********
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 3.1250000E-02      +*********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 3.1250000E-02      +*********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 7.8125000E-01    30+*****************************************
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag602

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 1.2500000E-01      +****************
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 1.2500000E-01      +****************
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 1.2500000E-01      +****************
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 6.2500000E-01    30+****************************************
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag603

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 0.0000000E+00      +
 3.1250000E-02      +***********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 3.1250000E-02      +***********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 3.1250000E-02    10+***********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 3.1250000E-02      +***********
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 3.1250000E-02      +***********
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 3.1250000E-02      +***********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 2.8125000E-01      +*********************************
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 5.3125000E-01    30+*****************************************
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag604

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1      64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 5.0000000E-01      +****************************************
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 5.0000000E-01    30+****************************************
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag605

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1      64


 0.0000000E+00     0+
 0.0000000E+00      +
 3.1250000E-02      +***********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 3.1250000E-02      +***********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 3.1250000E-02    10+***********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 3.1250000E-02      +***********
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 3.1250000E-02      +***********
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 3.1250000E-02      +***********
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 2.8125000E-01      +*********************************
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 5.3125000E-01    30+*****************************************
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag608

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 1.0000000E+00    30+****************************************
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag6016

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 1.0000000E+00    30+****************************************
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag631

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    30+
 0.0000000E+00      +
 1.0000000E+00      +****************************************
```

```
enter the name of the signal file:
a:lag311

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 1.0000000E+00      +****************************************
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    30+
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag312

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1      64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 1.0000000E+00      +****************************************
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    30+
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag151

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 5.0000000E-01      +*****************************************
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 5.0000000E-01      +*****************************************
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    30+
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag152

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1      64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 1.0000000E+00      +*****************************************
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    30+
 0.0000000E+00      +
 0.0000000E+00      +
```

```
enter the name of the signal file:
a:lag153

enter the name of the noise file or the word "nul":
nul

enter the number of batches and batch size of the files:
       1       64


 0.0000000E+00     0+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00     5+
 0.0000000E+00      +
 0.0000000E+00      +
 5.0000000E-01      +****************************************
 0.0000000E+00      +
 0.0000000E+00    10+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    15+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    20+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 5.0000000E-01      +****************************************
 0.0000000E+00    25+
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00      +
 0.0000000E+00    30+
 0.0000000E+00      +
 0.0000000E+00      +
```

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS Unrestricted
2a. SECURITY CLASSIFICATION AUTHORITY Office of Naval Research	3. DISTRIBUTION/AVAILABILITY OF REPORT Unlimited Distribution
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE Not applicable	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Technical Report No. 654	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Cornell University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State and ZIP Code) Ithaca, NY 14853	7b. ADDRESS (City, State and ZIP Code) Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-81-K-0037
8c. ADDRESS (City, State and ZIP Code) Arlington, VA 22217	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UN NO.
11. TITLE (Include Security Classification) Significant factor identification using discrete spectral methods (unclassified)	
12. PERSONAL AUTHOR(S) Paul J. Sanchez and Lee Schruben	
13a. TYPE OF REPORT Interim 13b. TIME COVERED FROM Mar 31 84 TO Aug 25 84	14. DATE OF REPORT (Yr., Mo., Day) 1985 March 15. PAGE COUNT 74
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Discrete event simulations are computer models of complex systems. The accuracy of the model in reflecting the behavior of the real system is determined by, among other things, the values of parameters for distributions used in the model. The modeller could allocate experimental resources more effectively without loss of accuracy in the model if he or she could

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Lee W. Schruben	22b. TELEPHONE NUMBER (Include Area Code) (607) 256-4856 22c. OFFICE SYMBOL

identify those parameters to which the response of interest has the greatest sensitivity.

One method of doing this is to try to model the response as a polynomial function of the model parameters. We are then interested in those terms in the polynomial which have non-zero coefficients.

Schruben and Cogliano developed a method whereby such polynomial models can be identified in fewer computer runs than previous methods allowed. Their concept was to do analysis in the frequency domain rather than the time domain. The virtual independence of frequency estimators in a spectrum means that many parameters can be tested independently within a single experiment using spectral methods.

This report attempts to extend the Schruben/Cogliano methodology to cover a more general class of models which includes discrete-valued parameters, such as policy decisions or capacities of queues. We evaluated the use of discrete-valued functions as a basis for spectral analysis. Several function sets were considered as possibilities, and Walsh functions were selected as the best choice.

Our preliminary results indicate that Walsh analysis may present a promising method for identifying significant parameters. However, the method exhibits undesirable behavior when time lags are present in the model.

END

FILMED

1-86

DTIC