Behavior Research Methods, Instruments, & Computers
1992, 24 (2), 238-247

Analysis of periodic data using Walsh functions

HILARY A. BROADBENT and YORK A. MAKSIK
Brown University, Providence, Rhode Island

(Student Award Winning Paper for 1991)

Data often contain periodic components plus random variability. Walsh analysis reveals periodicities by fitting rectangular functions to data. It is analogous to Fourier analysis, which represents data as sine and cosine functions. For many behavioral measures, Fourier transforms can produce spurious peaks in power spectra and fail to resolve separable components. Walsh analysis is superior for strongly discontinuous data. The strengths and weaknesses of each transform are discussed, and specific algorithms are given for the newer Walsh technique.

There are many methods for analyzing periodicities in data. In the time domain, autocorrelations provide a convenient way to look for cyclic patterns. However, such time-domain analyses often fail to resolve multiple periodicities superimposed in a single data set, particularly if some frequency components have little amplitude relative to others. Frequency-domain analyses typically provide greater resolution and are therefore often a better approach than their time-based counterparts; but many frequency-based methods exist, and the choice of the best method for a given data set is not always straightforward. The purpose of this paper is to compare two methods, the familiar Fourier transform power spectrum and the less familiar Walsh transform power spectrum, and to demonstrate that for some types of data, the Walsh spectrum provides a much clearer and more detailed understanding of the data than does the Fourier spectrum. Algorithms for computing a Walsh spectrum will be presented and explained, and applications of the analysis method will be considered.

Figure 1 shows a single trial from a rat pressing a lever for food on a 48-sec fixed-interval schedule in which food was delivered on a random half of the trials. Responses are shown as +1s, and time bins in which no responses occurred are shown as −1s. The trial illustrated is a typical example of responding following omission of food in this procedure; responses occurred in bursts throughout the interval. To determine whether these response bursts were periodically or randomly spaced, it is necessary to choose a method of periodic analysis that will provide an accurate representation of these data.

This research was supported in part by grants from the National Institute of Mental Health (RO1-MH44234) and the National Science Foundation (BNS 9110158). We would like to thank Russell M. Church for his help in exploring these analysis techniques and for aiding in the writing of this manuscipt. We would also like to thank Howard Kaplan for his comments on an earlier draft. Our thanks also to the anonymous reviewers for their comments and suggestions. Correspondence should be addressed to Hilary A. Broadbent, Department of Psychology, Box 1853, Brown University, Providence, RI 02912.

The Fourier Series

Fourier analysis is one common method of periodic analysis. It consists of fitting a series of orthogonal sine and cosine functions to the data and computing the magnitude of each fitted function. The left panel of Figure 2 shows the first eight nonzero frequency components of a Fourier series. A power spectrum provides a convenient quantitative and graphical summary of the transform. It is derived from the transform by combining the magnitude coefficients of the sine and cosine functions at each frequency. The value thus derived for each frequency is the power at that frequency; a plot of the power as a function of the frequency is called a power spectrum (Karl, 1989).

Fourier's theorem states that any periodic function can be represented as a sum of sines and cosines (Karl, 1989), but representing a rectangular wave requires an infinite number of sines and cosines. The data shown in Figure 1 take only two values, +1 or −1, so any periodicity that is present in these data must be in the form of a rectangular wave, and therefore analysis by Fourier transform is not optimal for these data.

The Walsh Series

The Walsh transform is directly analogous to the Fourier transform, but it uses rectangular waves instead of sine and cosine functions. The right panel of Figure 2 shows the first eight nonzero sequency components of a Walsh series. Note that the raised parts of these functions are not all of equal length; for this reason, it is not accurate to speak of the *frequency* of a Walsh function. The term *sequency* is used instead, either to denote the number of zero crossings that the function makes in the course of a unit of time, or to denote the number of raised portions or *lumps* in the function in a unit of time (Beauchamp, 1984). In this paper, *sequency* will be used to refer to the number of raised portions. In terms of the data in Figure 1, *sequency* refers to the number of response bursts in an interval. Also note the similarity in shape between the Walsh functions of a given sequency and the Fourier functions of a given frequency. For this reason, certain Walsh functions in Figure 2 have been labeled SAL for *sine*

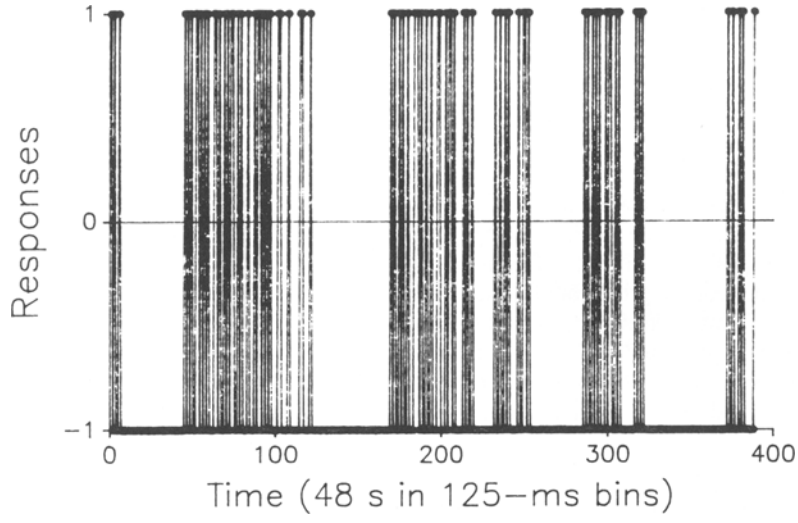

Figure 1. Data from a single omission trial of a rat responding in a 48-sec fixed-interval experiment. Responses are shown as +1s, and time bins with no responses are shown as −1s.

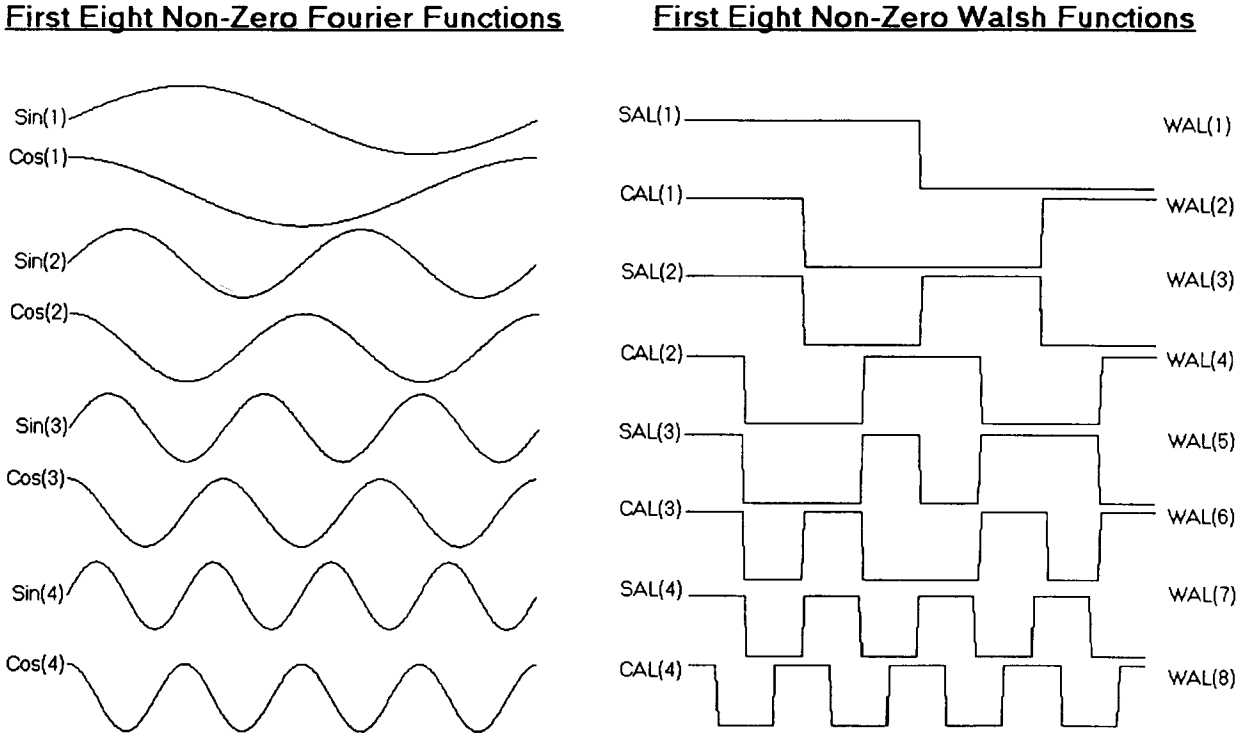

Figure 2. The first eight components of the Fourier series (left panel) and the corresponding components of the Walsh series (right panel).

analogous function, and other Walsh functions have been labeled CAL for *cosine analogous* function.

Walsh series share a number of properties with Fourier series. Both comprise functions that are orthogonal and form a complete set. The property of orthogonality means that the correlation of any two functions in the series is zero. When the transform is calculated, orthogonality ensures that information is not duplicated in the various coefficients of the component functions. This property, in turn, ensures that the power at different frequencies or sequencies is independent of the power at all other frequencies or sequencies. Completeness means that every frequency (or sequency) is represented in the series so that no information is lost when the transform is computed. The data can thus be recreated exactly from the transform. Both Walsh and Fourier spectra are shift-invariant, meaning that the same spectrum powers result regardless of the initial phase of the entire complex wave form being analyzed. Thus, in both cases, phase information of the periodic components relative to each other is lost when a power spectrum is derived from the coefficients of the transform. Walsh and Fourier transforms and their spectra are directly analogous to one another. The question is whether the rectangular waves of the Walsh transform can provide a better representation of the data than do the sines and cosines of the Fourier transform.

Transformation of a Square Wave

To examine this question, it is easiest to work with simple, simulated wave forms. An example is shown in the top panel of Figure 3. This is a single square wave with a frequency (and sequency) of two cycles in the arbitrary length of time represented by the 128 bins along the horizontal axis.

The middle panel shows the power spectrum from the Fourier transform of the square wave in the top panel. The vertical axis is scaled to show relative power, so that the highest peak in the spectrum has the value 1. The horizontal axis is scaled from frequency 0 to 64 cycles for the 128 arbitrary bins used in the simulation. The highest frequency that can be represented in a power spectrum is always equal to one half the number of data bins in the series being analyzed, which represents single alternation of high and low states. Although the greatest power in the middle panel of Figure 3 is present at frequency 2 (2 cycles in 128 arbitrary time bins), several other peaks are present in the spectrum, occurring at every fourth frequency and declining in magnitude with increasing frequency. This representation is mathematically accurate to the extent that many sines and cosines can be summed to make up a square wave. However, it is not a succinct representation of the function shown in the top panel.

The bottom panel of Figure 3 shows the equivalent Walsh transform power spectrum. Again, the vertical axis is scaled to show relative power, and the horizontal axis runs from sequency 0 to sequency 64, which, as in the Fourier power spectrum, represents single alternation and is the highest sequency rectangular wave that can be represented in the power spectrum. The graph shows maximal power at sequency 2 and no power at any other sequency. Clearly, the Walsh transform gives a simpler representation of the periodicity in the square wave in the top panel of the figure. For a single-component square wave, then, the Walsh transform is the analysis method of choice.

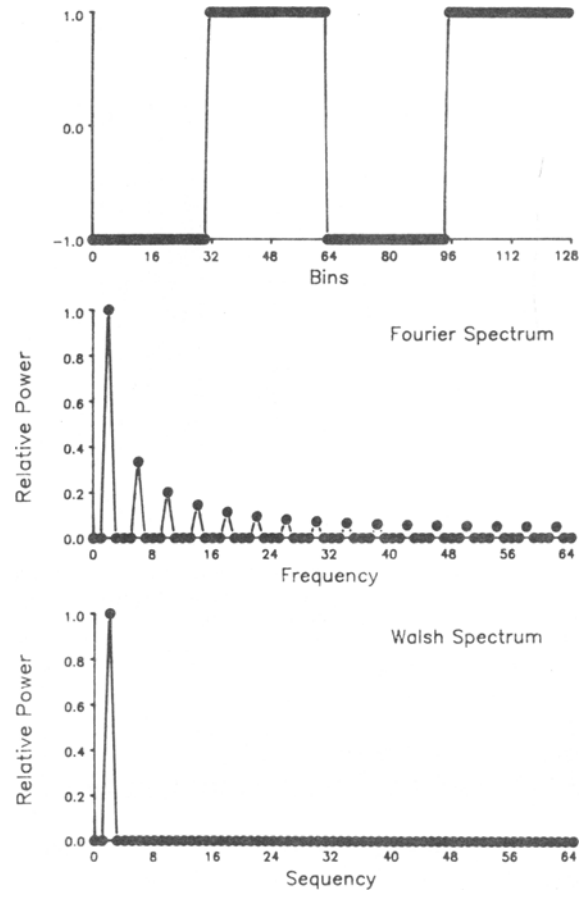

Figure 3. A synthesized square wave (top panel) with its Fourier spectrum (center panel) and Walsh spectrum (bottom panel).

Transformation of a Sine Wave

Is a Walsh transform always superior to a Fourier transform? The top panel of Figure 4 shows a sine wave of frequency 2, analogous to the square wave of Figure 3. The Fourier power spectrum of this sine wave is shown in the middle panel of Figure 4. All the power is concentrated at frequency 2 with no power elsewhere in the spectrum, which is the desirable result, given the input function in the top panel.

In contrast, the Walsh power spectrum, shown in the bottom panel of the figure, does not capture the sine wave as well as does the Fourier spectrum. As with the Fourier representation of the square wave, extra components are seen in the Walsh representation of the sine wave. The reason for this is analogous to the case of the Fourier transform of a square wave: many rectangular waves

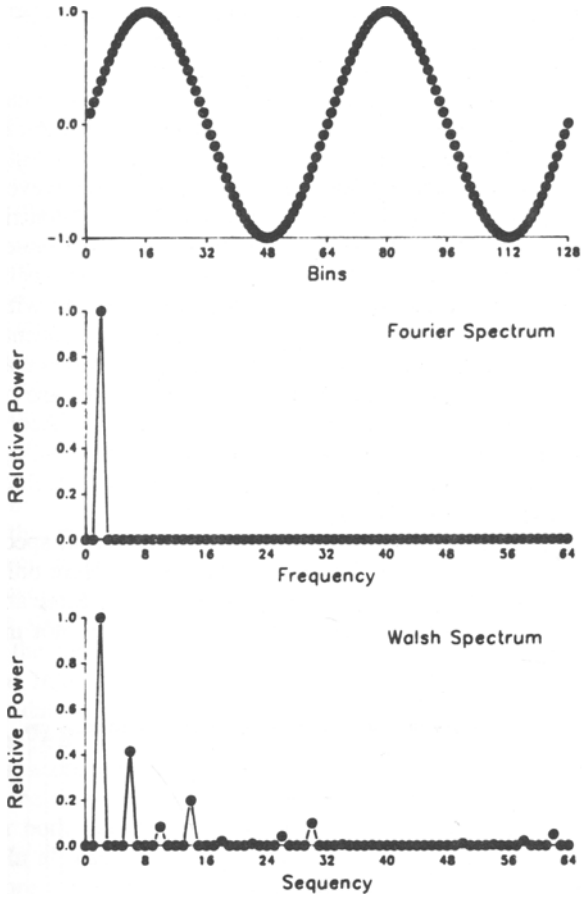

Figure 4. A synthesized sine wave (top panel) with its Fourier spectrum (center panel) and Walsh spectrum (bottom panel).

must be summed to compose a sine wave. The Walsh power spectrum gives as mathematically accurate a representation of the sine wave as the Fourier power spectrum does of the square wave, and it suffers equally from a lack of succinctness.

Transformation of a Summed Composite Rectangular Wave

It has been shown that the Walsh transform is superior to the Fourier transform for a single-component square wave. A composite wave form can be created by adding rectangular waves together. Figure 5 shows three such wave forms. One rectangular wave cycles 4 times in the course of the interval shown, one cycles 12 times, and one cycles 16 times. The top panel of Figure 6 shows their additive composite. Instead of taking the values -1 and $+1$ as in the data and square-wave examples, the additive composite can take integer values ranging from -3 to $+3$. One may ask whether Walsh functions, which are binary in nature, can provide a good representation of a wave form that is not binary and that contains more than a single rectangular component. The Fourier transform might be expected to do better than the Walsh transform in this case, because sine and cosine functions are not binary.

The middle panel of Figure 6 shows the Fourier spectrum of the additive composite in the top panel. While the highest peaks in the spectrum are at appropriate frequencies, many more frequency components show up in this spectrum than there are square waves that make up the data series in the top panel. The Fourier spectrum does not provide a succinct representation of the data.

The bottom panel of Figure 6 shows the corresponding Walsh spectrum. Here the picture is much clearer. There are peaks at the sequencies 2, 12, and 16, as there should be, and all peaks are at nearly equal, maximal height. This spectrum is a much better representation of the components than is the Fourier spectrum.

In general, Walsh analysis is superior for any data set that contains relatively few values, as in the case of the data of Figure 1 and the square wave of Figure 4, where the data are constrained to be $+1$ or -1. Fourier analysis is superior if the function takes on many values, as in the case of the sine wave in Figure 3, where many values between -1 and $+1$ occur. For intermediate cases, the

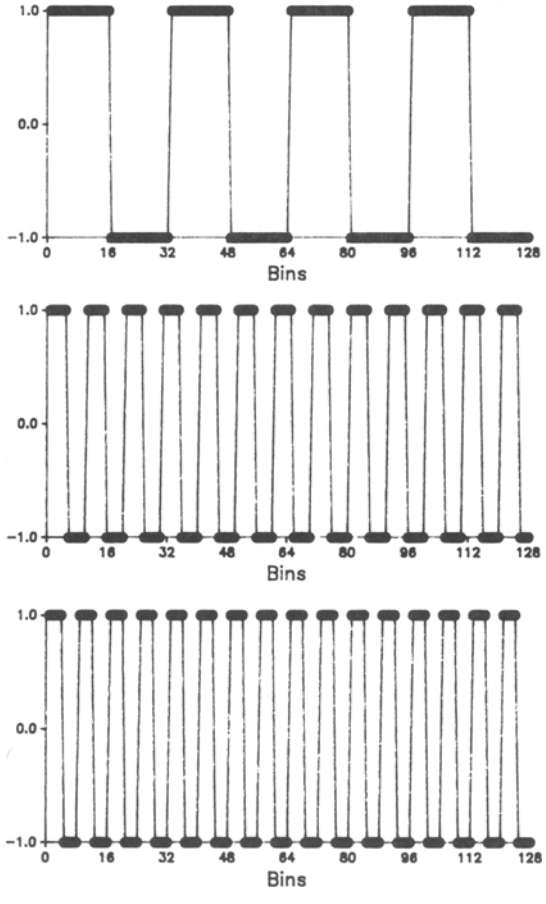

Figure 5. Three square waves. The first (top panel) has a frequency of 4 cycles, the second (center panel) has a frequency of approximately 12 cycles, and the third (bottom panel) has a frequency of 16 cycles in the series of bins shown.

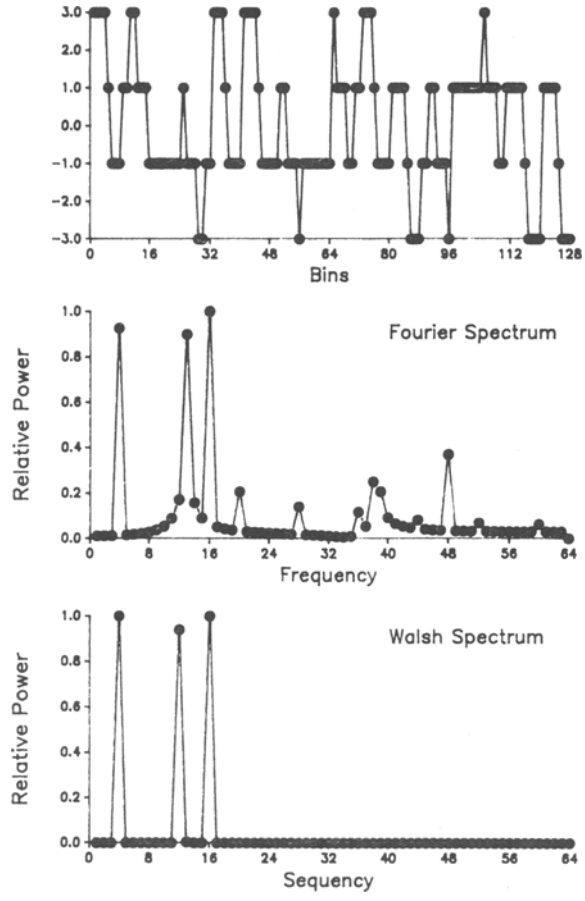

Figure 6. A composite wave (top panel), formed by adding the three square waves shown in Figure 5, with its Fourier spectrum (center panel) and its Walsh spectrum (bottom panel).

performance of the two analyses is similar. The characteristics that are most relevant in choosing an analysis method are (1) the number of different values taken, relative to the number of time bins (75 different data values could be considered a small number if the data series were 1,000 points long), and (2) the number of plateaus, or runs of any single value, in the data set (Beauchamp, 1984). As the sequency or frequency of the periodicity increases, Walsh and Fourier transforms converge in how well they represent the data.

Transformation of a Binary Composite Rectangular Wave

Periodic processes can be combined in ways other than additively. For some systems, if any of several periodic processes is active at any given time, a unitary response will be produced. An example might consist of a neuron that has several units projecting to it. Activation of any of those units might cause the neuron to fire, but the spike is not larger if two or more of the projecting units are active than if only one is. Another example might be an animal pressing a lever for food. Several periodic physiological or experimental processes might each lead to pressing the lever, but many hardware systems do not permit measuring the strength of the response, only its presence or absence, so that binary data result.

The mathematical equivalent of this type of combination of processes into unitary responses is the logical "or." The top panel of Figure 7 shows the composite waveform created by "or-ing" together the three waves of Figure 5. An analysis that can reveal the equal contribution of each of the three periodic processes is desirable.

The middle panel of Figure 6 shows the Fourier spectrum of the composite shown in the top panel. As with the additive composite, many more frequency components show up in this spectrum than there are square waves that make up the data series in the top panel. In the case of this binary composite, it is even difficult to discern whether the primary peaks are at appropriate frequencies. The Fourier spectrum again does not provide a succinct representation of the data.

Compare the Fourier spectrum with the Walsh spectrum shown in the bottom panel of Figure 7. Here only three peaks in the spectrum are evident, and all three are of equal magnitude. The Walsh spectrum does not in-

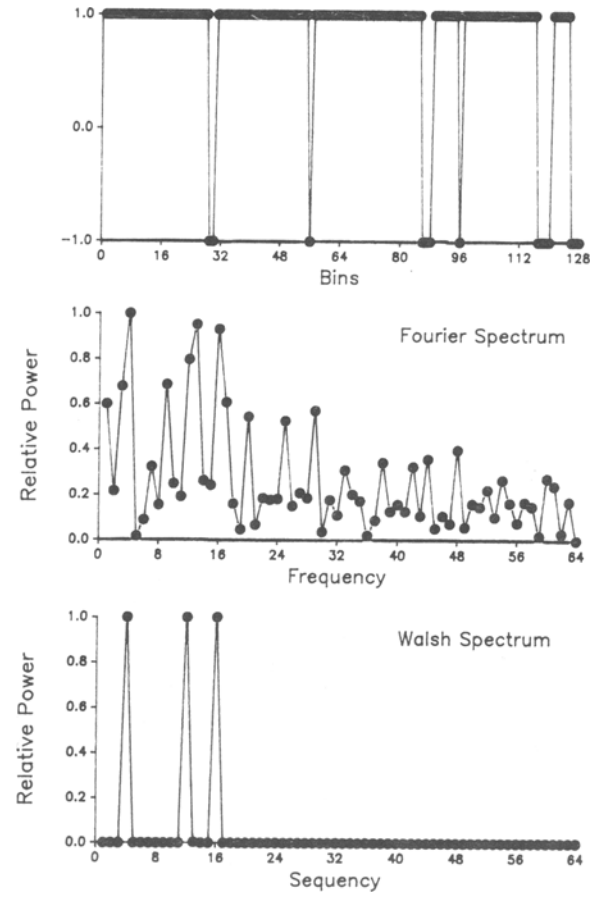

Figure 7. A composite wave (top panel), formed by combining the three square waves shown in Figure 5 with an "or" operation. The Fourier spectrum is shown in the center panel, and the Walsh spectrum is shown in the bottom panel.

troduce spurious sequencies, and it correctly identifies the magnitude of the three sequencies that were combined in the binary composite.

Reconstruction of Simulated, Random Data Through Inverse Transforms

Thus far it has been shown graphically that Walsh transforms perform better than Fourier transforms for data composed of step functions. The best way to quantify the improvement that Walsh transforms offer is to compare analyses of random data. It is intuitively evident that the rectangular waves of Figures 3 and 5 should be represented better by a technique that fits rectangular waves to data, but it is not so obvious that the same will be true of data that are not obviously rectangular in form, or that are not known to contain any periodicities.

We compared the Fourier and Walsh transforms of random data by generating 101 128-bin-long binary series with a probability of .5 of getting a +1 or a −1 in any given bin. We then computed the Fourier and Walsh power spectra of each series. As a measure of how well the spectra summarized the data, we took the top 25% of the frequency or sequency components in the spectra (the 16 highest points in the spectra), and we regenerated the data series from those components of the transform. The percentage of the variance in the original series that was accounted for in each of the reconstructions was used as the measure of goodness of fit, and this was calculated for both transforms for each of the 101 random series.

In all cases, the Walsh reconstruction accounted for more variance than did the Fourier analysis. The median difference was 18.8%; the highest 16 peaks in the Walsh spectrum accounted for 73.3% of the variance, whereas the highest 16 peaks in the Fourier spectrum accounted for 54.5% of the variance. The case that gave the median difference in the percentages of variance accounted for by the two procedures is illustrated in Figure 8. The top panel shows the fit of the Fourier reconstruction (open circles) to the first 50 points of the random data (closed circles). The reconstructed function has roughly the same shape as that of the original, but it fails to fit the spikiness of the random data and does not reach sufficient magnitude. The bottom panel of Figure 8 shows the fit of the Walsh reconstruction to the same random data set. Again, the functions have roughly the same shape, but the Walsh reconstruction follows the peaks and valleys of the original set, particularly in places where the positive and negative values alternate. It is clear that the Walsh transform contains more information about the data set in fewer components than does the Fourier.

Reconstruction of Actual Data Through Inverse Transforms

The original question posed in this paper was, Which analysis technique is most appropriate for the data shown in Figure 1? To answer this question, we subjected the Fourier and Walsh transforms to the same test that was employed for random data. Both transforms were computed, power spectra were derived, and the highest 25% of the peaks in the power spectrum were used to reconstruct the original data. The Walsh reconstruction accounted for 76.8% of the variance in the original data, whereas the Fourier reconstruction accounted for only 49.8% of the variance. Figure 9 shows the reconstruc-

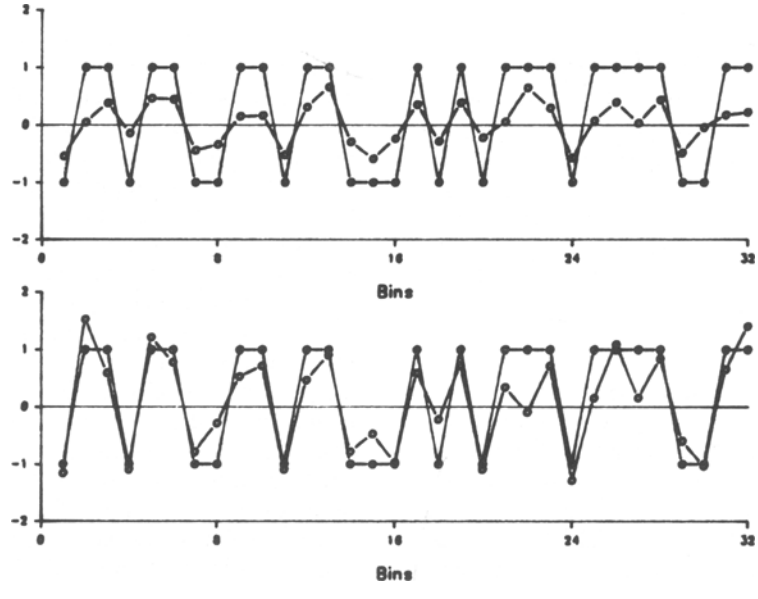

Figure 8. Partial reconstructions of a synthesized random data series. In both panels, the data are shown as filled circles. The top panel shows the Fourier reconstruction (open circles), and the bottom panel shows the Walsh reconstruction (open circles). See text for description.

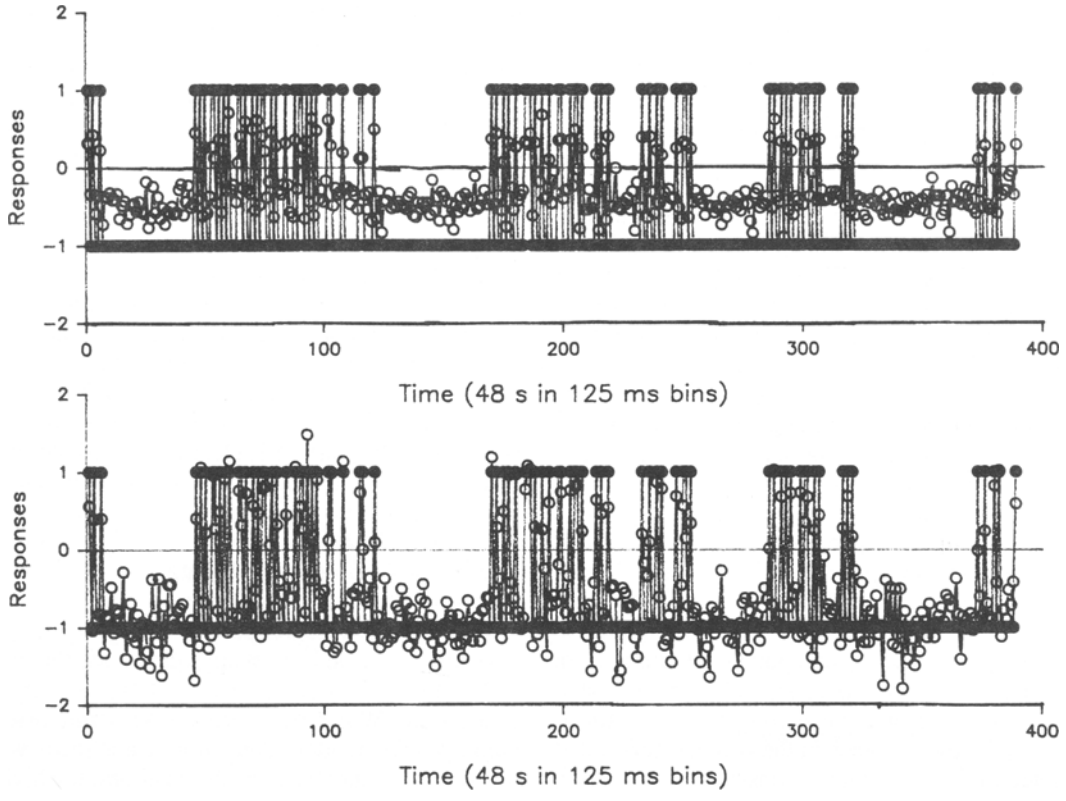

Figure 9. Partial reconstructions of the data series shown in Figure 1. In both panels, the data are shown as filled circles. The top panel shows the Fourier reconstruction (open circles), and the bottom panel shows the Walsh reconstruction (open circles). See text for description.

tions, with the data represented as closed circles and the reconstructions as open circles. The Fourier reconstruction is illustrated in the top panel, and the Walsh in the bottom panel.

In this case, there was a 27% improvement of Walsh over Fourier, indicating that Walsh analysis is indeed the method of choice for these data. Many psychological data, including measures of operant performance, biological rhythms, neural activity, and so on, are frequently recorded in a form that contains two values (an event occurs at a given time or does not occur). For all such data, if a periodic analysis is sought, a Walsh transform can often be expected to be superior to a Fourier transform.

Algorithms and Computational Methods

Transforms are analogous to computing correlations by calculating inner products of data with digitized versions of orthogonal functions. In other words, each sine or cosine function (in the case of Fourier transforms) or each rectangular function (in the case of Walsh transforms) is fit to the data in exactly the same way as any single-parameter curve fitting is done. Because the functions are orthogonal, there is no overlap of information between the fits of wave forms of different frequencies or sequencies (unlike, for example, a polynomial fit, where higher order terms share information with lower order terms).

While sine and cosine functions are easier to express algebraically than Walsh functions, the latter are easier to generate computationally because of their binary nature. They can be generated with the use of entirely binary operations such as "and," "or," and "not." Walsh functions of any order can be derived from any single Walsh function by simply shifting and reflecting parts of the function in a systematic way (Beauchamp, 1984). This means that some Walsh functions contain parts of other Walsh functions and that, for computational purposes, pieces of an inner product calculated with one Walsh function may be saved and simply repeated for many of the other functions. (This principle does not contradict the independence mentioned above that stems from the orthogonality of the Walsh functions.) Linear algebra provides a tool for conducting a thorough examination of which calculations can be reused and which cannot, leading to a computationally efficient method of calculating Walsh transforms that is very similar to the fast Fourier transform (Elliott & Rao, 1982). The binary nature of Walsh functions eliminates the need for use of floating point arithmetic and makes it easy for a digital computer to do these calculations very quickly and accurately. The

redundancy of calculation and the binary nature of Walsh functions lead to the algorithm (and C code) for the fast Walsh transform listed in the Appendix. This code represents a redundancy-free method of calculating all of the inner products necessary for a Walsh transform. Its only limitation is that the number of bins in the data set being transformed must be a power of two.

Examination of the code for the function walsh() shows that the main part of the function contains only very simple binary arithmetic operations for each pair of numbers, as well as several loops and counters. This can be translated to a very small series of binary "and," "or," and "not" operations, and therefore can be easily implemented in hardware for use in real-time data compression, filtering, and spectral analysis. Due to their binary nature, it is easy to use standard sparse matrix techniques (Press, Flannery, Teukolsky, & Vetterling, 1989) to store the Walsh functions themselves.

By contrast, Fourier transforms typically require heavy use of floating point processing. They tend to suffer from rounding errors because of the limited representation available for floating point numbers in many current computers, and they are difficult even for the technically sophisticated to implement in hardware for real-time applications.

For the purpose of examining a data series for periodicities, it is often necessary to calculate a power spectrum from either the Walsh or the Fourier transform, as we have done in this paper. A power spectrum can be defined for our purposes as a relative measure of the amount that a particular frequency or sequency contributes to making up a given wave form. In other words, it helps to give a picture of how well a particular frequency is represented in the data stream. A spectrum is often useful, because while the transform of a data set is unique to that data set, the spectrum of a data set is the same for all circular shifts of that data set—that is, for all realizations of the same process independent of the initial value. For example, the data set $\{1, -1, 1, 1\}$ and the data set $\{1, 1, -1, 1\}$ would produce different transforms but would produce the same power spectrum because they are circular shifts of one another. Calculating a power spectrum loses information about the phases of the frequencies within a particular data set as well as the phase of the data set as a whole, but it allows data sets from the same process, but with different overall phases, to be combined, averaged, and manipulated together in many ways that would be difficult or impossible otherwise. It is also possible to take successive samples of a large data set and calculate a power spectrum for each sample; many short samples with a comparatively small signal-to-noise ratio may be combined by averaging the spectra for each sample. The signal is then enhanced relative to the noise, which often tends to average out to a flat line.

Power spectra may also be used to explore the noise distribution in data. White, or Gaussian, noise appears as a straight line, while pink noise and other noise distributions have other distinct shapes in power spectra (Blackman & Tukey, 1959).

A spectrum may be calculated from a Walsh or Fourier transform in several ways, yielding different information and giving different amounts of translational invariance. The most common way to calculate a power spectrum is to take the coefficients of each sine and cosine wave of a given frequency or SAL and CAL wave of a given sequency, square their values, add the two squared values, and take the square root of the sum. This is essentially an application of the Pythagorean theorem (Lynn, 1989).

The C code in the Appendix may be used to calculate a Walsh transform from any one-dimensional array of data. Two-dimensional or N-dimensional transforms are also possible (Elliott & Rao, 1982). The code includes two functions. The first function, called walsh(), takes three arguments: (1) a C pointer to an array of data, which starts filling the array at index equal to zero; (2) the length of the data series to transform; and (3) the base-2 logarithm of the length of data to transform. It should be noted that this fast Walsh transform can deal correctly only with data arrays whose lengths are powers of two. Valid function calls would look like walsh(data_array, 256, 8) and walsh(other_array, 1024, 10) but *not* walsh(some_array, 145, 7.4). The function walsh() is an in-place transform, which means that the data in the input array are replaced by the transform of the data starting at array index zero and continuing the entire length of the array. The transformed data must then be scaled by dividing every element of the output array by the length of the input array. This can be added to the program, but it has been omitted here because it is sometimes desirable to examine the unscaled version.

When the function "walsh ()" is called with an input data array, it outputs the transform to the same array, replacing each data value with a coefficient. It is important to note that the coefficients, or terms, of the transformed array are put in a special order called inverse Gray code order (Beauchamp, 1984). Gray code is a reordering of the binary numbers that minimizes the number of bit changes from one number to the next successive number. Inverse Gray code order is simply Gray code order listed backwards. An example of the first four binary numbers is 00, 01, 10, 11. Notice that in the transition from 01 to 10, two bit positions have changed place. The first four binary numbers in Gray code are 00, 01, 11, 10, because in every transition there is only one bit that changes. Inverse Gray code order is the same list, only reversed as follows: 10, 11, 01, 00; in decimal, this would be 2, 3, 1, 0. This means that for a four-point Walsh transform using the supplied algorithm, the zero'th coefficient is located at the fourth position, the first coefficient is located at the third position, the second coefficient is located at the first position, and the third coefficient is located at the second position.

This ordering is used because it is much easier to code the fast Walsh transform to output coefficients in reverse

Gray code order than in sequency order. It also allows the algorithm to run much faster than the sequency-ordered fast Walsh transform. However, this is not a very useful order for the purposes of data analysis, and we have included a routine called invgray() that will correctly index the output array of the walsh() function.

If the input and output data array is called data[], then to correctly access the fourth element of the output array, one must write something like this: x = data[invgray(4, 10)], where 4 is the index of the number to be accessed, and 10 is the base-2 logarithm of the length of the transformed data series. For example, if one wanted to find out what the 10th coefficient of a Walsh transform of 256 data points was, then the call x = data[invgray(10, 8)] would be appropriate, because 10 is the index one would want and 8 is the base-2 logarithm of 256.

To obtain the inverse Walsh transform, one simply calls the function walsh() with the input array filled with data from a previous transform; the input array will be replaced with the inverse transform of the data. Notice that the fast Walsh transform is its own inverse.

If the data in the input array for an inverse transform are not directly from a previous transform, the data may have to be reordered in inverse Gray code order in the input array before an inverse Walsh transform can be performed correctly. The reordering can be done as follows: The invgray() function may be used to find out where each transform coefficient should be placed in the input array to get an inverse transform to come out in time sequence order; one simply gives the invgray() function the location of where the index would be in sequency or time series order along with the usual base-2 logarithm of the length of the data series, and it will return the location in the input array that a data element should be placed.

To calculate a Walsh power spectrum from the output of the fast Walsh transform, one must write a simple loop that will read in successive pairs of coefficients from the output array of the fast Walsh transform, and take the square root of the sum of the squares, using the invgray() function to find the appropriate index of each coefficient as needed. For example, the first magnitude of a Walsh power spectrum resulting from a Walsh transform on an array of data named "data" with a length of 1,024 points would be: sqrt((data[invgray(1,10)] * data[invgray(1,10)]) + (data[invgray(2,10)] * data[invgray(2,10)])). This operation would then be repeated for elements 3 and 4, 5 and 6, and so on, until the last coefficient was reached. The zero'th coefficient in the output array is the DC component, which is a measure of the mean of the data series and is typically excluded from the power spectrum or simply squared and treated as the zero'th frequency/sequency component. The final element in the transform array is not combined with any other element because, at this point, there will be no other uncombined elements. Typically, one takes its absolute value and includes this value in the power spectrum as the last magnitude in the power spectrum without further modification.

Code for the fast Fourier transform is available from many sources. We have used the algorithm in *Numerical Recipes* (Press et al., 1989). (However, there are some minor errors in this algorithm in the first and second printings of the version for C programmers, and the third printing [1989 edition] should be used.)

Sparse data can present problems for both the fast Fourier transform and the fast Walsh transform, because both techniques produce the most useful transforms and spectra when the input data have a mean of zero and very little or no padding. This means that neither method is very useful for investigating periodicities in data where there are few pulses in a very long time. Fourier methods can be supplemented with filters and windowing techniques to deal with sparse data and spike data. The Haar-Slant filter is used with the Walsh method (Beauchamp, 1984; Elliott & Rao 1982). It is a method of running through an array of sparsely distributed data and filling gaps between spikes with +1s and −1s, producing a data series with local half-wave symmetry and a zero mean. This filter renders the Walsh transform analogous to a Fourier transform on data that has been filtered with a Hanning, Hamming, or triangular window. If results are difficult to interpret from either transform and the data are fairly sparse, a filter may be in order. One should consider filtering any data that do not have a mean of zero before a transform is performed.

Conclusions

The Walsh series of orthogonal functions has been in existence since 1923 (see Elliott & Rao, 1982), and the techniques to compute a shift-invariant spectrum from a Walsh transform were developed in 1971 (see Beauchamp, 1984). They have not been widely used for data analysis, however, probably because they are cumbersome to express algebraically. With computers to do the analysis, they offer computational advantages over Fourier analysis in addition to their obvious advantages for characterizing many kinds of data. To our knowledge, no software to compute Walsh transforms is currently available to the public. The algorithms presented here have been written in C and run on an IBM AT, but they can be implemented in any language on any computer.

For discontinuous data sets, Walsh transforms are often a superior analysis tool for revealing periodic components that are separable from each other and from random variability. Many behavioral measures that are likely to contain periodic components are discontinuous, including measures of operant behavior (as in the data example presented above), recordings of neural activity, and measures of motor activity and biological rhythms. For these and other kinds of data, Walsh transforms often surpass Fourier transforms in computation ease, resolution, and succinctness of data representation.

REFERENCES

BEAUCHAMP, K. G. (1984). *Applications of Walsh and related functions with an introduction to sequency theory.* New York: Academic Press.

BLACKMAN, R. B., & TUKEY, J. W. (1959). *The measurement of power spectra.* New York: Dover.

ELLIOTT, D. F., & RAO, K. R. (1982). *Fast transforms: Algorithms, analyses, applications*. New York: Academic Press.
KARL, J. H. (1989). *An introduction to digital signal processing*. San Diego: Academic Press.
LYNN, P. A. (1989). *An introduction to the analysis and processing of signals*. New York: Hemisphere.
PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., & VETTERLING, W. T. (1989). *Numerical recipes: The art of scientific computing*. Cambridge: Cambridge University Press.

APPENDIX

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

/* The invgray function takes 2 arguments:   */
/* "binary", which is the number that needs */
/* to be converted to inverse Gray code      */
/* order, and "digits" which is the base-2   */
/* logarithm of the number of bins in the    */
/* input data array. The function invgray    */
/* converts a desired sequency-ordered       */
/* decimal index into the correct index      */
/* into the output of the function "walsh"   */

long invgray(binary,digits)
long binary,digits;
{
char bits[32];
char gray_list[32];
long gray_code=0;
long start = 1;
int c;
for(c = 0; c < digits; c++, start <<= 1)
        {
        if (start & binary)
            bits[c] = 1;
        else
            bits[c] = 0;
}
bits[digits] = 0;
for(c = digits; c > 0; c--)
    gray_list[c-1] = (bits[c] + bits[c-1]) % 2;

for(c = 0; c < digits; c++)
   {
   gray_code <<= 1;
   gray_code = gray_code + gray_list[c];


}

return(gray_code);
}

/* The function "walsh" takes three arguments:  */
/* "data", which is a C pointer to an array of  */
/* floating point numbers to be Walsh trans-    */
/* formed; "len", which is the number of bins   */
/* to transform in the array that contains the  */
/* data to be transformed (it must be a power   */
/* of two in length); "log2_dat", which is the  */
/* base-2 logarithm of the length of the data   */
/* to be transformed in the array "data".       */
/* The function "walsh" performs a Walsh trans- */
/* form on the data in the array supplied as    */
/* the first argument to the function.          */

walsh(data,len,log2_dat)
float data[];
long len;
long log2_dat;
{
long count, x,y,z,c2,c3;
long half;
float low, high, temp1, temp2, sw1, sw2;
long step1, step2, groups, bflys;

half = (len / 2) - 1;
step1 = len;
step2 = half+1;
groups = 1;
bflys = half+1;

for(y = 0; y < log2_dat; y++)
    {
    for(x = 0; x < groups; x++)
        {
        for(z = 0; z < bflys; z++)
            {

            low = data[z + x * step1];
            high = data[z + step2 + x * step1];
            data[z + x * step1] = low + high;
            data[z + step2 + x * step1] = low - high;
        }
    }
    step2 = step2 / 2;
    step1 = step1 / 2;
    groups = groups * 2;
    bflys = bflys / 2;

}
}
```