Data Acquisition, Processing and Analysis for Distributed Decision Support

Broad Contents

Chapter 4

Data Analysis

The analysis of data (either in the form of column vectors - "signal" - or in the form of 2-d data – "image") and their interpretation form the core, important tasks in this exercise. Unlike the pre-processing methods discussed in Chapter 3, the techniques of data analysis and interpretation are closely related to the underlying physical processes that generate the data that is analysed and interpreted. In this Chapter, we present a set of data analysis methods that are predominantly used by us for NDE in our laboratories, and those that are a part of the DESKPACK Software System (DSS). Most of these methods are equally applicable in domains other than NDE, tempered of course, by the appropriate domain knowledge.

While the pre-processing methods generally treat the data as a set of numbers, data analysis and interpretation must take into account the underlying domain knowledge to arrive at the right conclusions. Hence, after the pre-processing of the data, the domain expert(s) must decide on the appropriate methods of data analysis to be employed, their sequence, the formats in which results must be studied and their interpretation. This is the subject matter of this Chapter. In this Chapter, we assume that the data has been acquired in adequate number, and it has been pre-processed as per the suitable methods described in Chapter 3, and that the data is ready for analysis.

The next Section (4.1) offers an insight into the various possible objectives of data analysis. For these different objectives, Section 4.2 highlights the concepts and methods involved. Analysis methods specific to Signals and Images in their respective native domains are dealt with in Sections 4.3 and 4.4 respectively. Specific methods of analysis in the Frequency & Power domains and those in the Correlation domains are described in Sections 4.5 and 4.6 respectively. Analysis in domains other than time and frequency are detailed in Section 4.7. The types of outcomes of these analyses are listed in Section 4.8, whereas Section 4.9 elucidates the methods in interpreting this analysed data / results. The final Section, 4.10, presents a few Questions for further thought and discussions.

Wherever appropriate, the computer code listing pertaining to a data analysis function is given in this Monograph. Some analysis functions use very large data arrays. The code listing is in the language 'C', which requires special allocation and de-allocation of memory, for handling large array data, specifically in Personal Computers. Special functions that perform memory allocation and de-allocation alone are sourced from the book "Numerical Recipes in 'C'". Many of the analysis functions listed here will not work on Personal Computer based 'C' compilers, without these specialised memory allocation and de-allocation functions.

While every code listing given here has been tested, readers are advised to check them thoroughly using known, test data for bugs and errors, before using them particularly in mission critical applications with unknown data.

The data analysis methods discussed in this Chapter should at best be taken as indicative rather than comprehensive. In each of the methods described, the reader must understand its underlying physical basis, identify the extra information that method

provides, the pre-requisites for using that method and if that method would solve his / her problem at hand. The ability to choose the right data analysis method would evolve with increasing experience in handling a wide variety of problems. The short bibliography (and the references therein) given at the end of this Chapter (with special reference to non-destructive evaluation) should serve as a starting point to analyse the efficiencies of the various data analysis methods described herein. Pay particular attention to the various case studies discussed in the bibliography, *vis-a-vis* the analysis methods used. For a good overview of the DESKPACK Software System (DSS), on which this Monograph is based, see http://deskpack.tripod.com/thesis/KBS-DSS-Thesis.pdf

This Chapter also includes a Glossary of Terms, in Appendix – I, commonly used in data pre-processing and analysis.

4.1 Objectives of Data Analysis

The two broad objectives of data analysis could be either data classification or prediction of a certain data value. In data classification, a set of data samples (either a set of signals or a set of images, or a set of features extracted from these signal/image data) are to be classified into two or more classes, based on their characteristic properties. These properties or features as they are normally called, might originate from either the time domain representation of the signal or from the transformation of the signal to other domains, say, frequency domain. The chosen feature set could also be a combination computed from multiple domains, such as time, frequency, power, correlation, etc. We shall see more on these in Sections 4.2 and 4.7.

If the objective is prediction, then a set of input data is fed to a system to predict a physical value that may otherwise be difficult to measure directly. If no established, analytical relationships exist between the input set and the physical value(s) that need to be predicted, then one might still predict the value, rather accurately, by using an artificial neural network (ANN), provided a number of samples (input-output value pairs) exist in pairs for training the ANN. In order to accomplish that, this Chapter describes with source code, the feed-forward, error back-propagation artificial neural network, a popular variant of the multi-layered perceptron (MLP).

4.2 Concepts and Methods of Data Analysis

This Section describes the introductory general methods of data analysis – those issues that are common to both signals and images.

While selecting a particular method or a set of methods (to be used in a particular sequence) for data analysis, one must clearly keep in mind the objective – whether it is for data classification or for data prediction.

If the objective is data classification, based on the physical basis of generation of the data (the underlying principles of physics that led to the generation of this data), one must estimate the domain in which the data can be separated well. At this stage, the physical basis will also provide insight into the methods required to reduce noise, if present in the

data. The type and extent of noise present in the data will also have a major say in the way the analysis methods are chosen.

For example, if two types of acoustic signals were being analysed to distinguish noise and leak signatures, a good approach would be to look at the data in the frequency domain. One would have broadband characteristics while the other will have unique, narrow band signature features determined by the topology of the leak. In this case, frequency domain analysis could be faster and effective.

As another example, if we consider a task to distinguish between ultrasonic signals from a defective plane and from a rugged crack, one should look for single reflection signature and signatures that arise from multiple reflections (leading to constructive and destructive interferences). The autopower spectra have been found effective in such cases.

As a third example, if we wish to distinguish between two sets of nearly identical noisy data (signals having poor signal-to-noise ratio), use of statistical methods or cluster generation and analysis principles would be appropriate. These methods can be applied by directly operating upon transformed data (raw data transformed from one domain to another, say, from time domain to power domain), or by extracting features (or properties) from the transformed data. Care must be taken while extracting features and using them. More on these can be seen in Section 4.7.

The type of analysis method that will be effective can be identified through practice in handling a number of cases.

4.3 The Time Domain - Signals

In this Section, we shall see the methods of analysis of signals (i.e., time vs. measured physical value), in their native domain, which is Time. A number of important aspects about the signal can be obtained from their native domain.

Quite a few branches of Nondestructive Evaluation use the native time domain itself to obtain as much information as possible about the underlying physical process, particularly if the signal-to-noise ratio (SNR) of the data is very good. Acoustic Emission Testing, Ultrasonic A-Scan and Eddy Current figure-of-eight curves are good examples of signals that are analysed / interpreted in their native domain.

A number of time-domain properties can be extracted from a given signal, many of which could be effective discriminators. In addition, it is easier to associate physical processes with many of the time-domain properties. Some important time-domain properties could be:

* First Highest Positive Peak Value - Max 1
* Second Highest Positive Peak Value - Max 2
* Third Highest Positive Peak Value - Max 3
* First Highest Negative Peak Value - Min 1
* Second Highest Negative Peak Value - Min 2
* Third Highest Negative Peak Value - Min 3

* Inter-peak Distance Between Max 1 and Max 2
* Inter-peak Distance Between Max 2 and Max 3
* Inter-peak Distance Between Min 1 and Min 2
* Inter-peak Distance Between Min 2 and Min 3
* First Peak's P-P Value
* Second Peak's P-P Value
* Third Peak's P-P Value
* Ratio of I to II P-P Values
* Ratio of II to III P-P Values
* Ratio of I to III P-P Values
* Number of Zero Crossings in the Signal
* Ratio of Inter-Positive-Peak Distances
* Ratio of Inter-Negative-Peak Distances
* Ratio of Inter-Positive-Peak (I-II) Distance to the Record Length
* Ratio of Inter-Positive-Peak (II-III) Distance to the Record Length
* Ratio of Inter-Negative-Peak (I-II) Distance to the Record Length
* Ratio of Inter-Negative-Peak (II-III) Distance to the Record Length
* Location of First Maximum Positive Peak
* Location of Second Maximum Positive Peak
* Location of Third Maximum Positive Peak
* Location of First Maximum Negative Peak
* Location of Second Maximum Negative Peak
* Location of Third Maximum Negative Peak
* Rise-Time of the First Maximum Positive Peak
* Rise-Time of the Second Maximum Positive Peak
* Rise-Time of the Third Maximum Positive Peak
* Fall-Time of the First Maximum Positive Peak
* Fall-Time of the Second Maximum Positive Peak
* Fall-Time of the Third Maximum Positive Peak
* Width of the First Maximum Positive Peak Pulse
* Width of the Second Maximum Positive Peak Pulse
* Width of the Third Maximum Positive Peak Pulse

A full list of such properties is shown in the Code Listing 4.15.

If the underlying physical process (ultrasonic, eddy-current, acoustic emission, etc.) is well understood, all or at least most of the properties listed above can be understood with a conceptual basis. This understanding would help in choosing the right set of properties to data analysis, either data classification or data prediction.

4.4 The Space Domain - Images

In this Section, we shall review the concepts of analysing images in their native domain, viz. in the X-Y space. Images obtained from Penetrant Testing and Magnetic Particle Testing are good examples where the 2-d data is analysed in their native X-Y space domain.

There are significant aspects that differentiate data (or a column vector signal) analysis in its native time domain and the analysis of images in their native x-y domain (could be viewed either as a time or space domain). These aspects can be listed briefly as follows:

- The analysis of an image by an human being and that of a machine
- Parallel versus serial processing of images by a machine
- Colour, Pseudo-colour, grey scale or black-and-white analysis

The first aspect (human being vs. a machine) is the most interesting of the three. This aspect is accentuated by the fact that the human being can analyse the image in full colour, in parallel (need not worry about pixel-level analysis), understand and make decisions about the full picture even when it is some what incomplete and blurred (low resolution). The way a human being stores the image is also very much different than how a machine does. A human being "recognises" an image whereas a machine "analyses" the image. On the other hand, a machine almost always does a pixel level analysis and offers pixel level processed information. It again calls for the attention of the human being to get the "full pictures" from the resulting processed data. The machine never "understands" the image as a human being does, in spite of its "faster" data processing capability.

The second aspect (parallel versus serial processing) is more algorithmic as it is academic. Here again, the human being always analyses the image in a "parallel processing fashion". Whereas a machine can do both serial and parallel processing, depending upon the algorithm and the hardware employed for analysis.

The third aspect has more to do with machine analysis - Colour, Pseudo-colour, grey scale or black-and-white – all of which make things difficult both the machine and the human being who writes the algorithm to interpret the processed data. Care must be taken while interpreting data that is presented using different formats, while essentially containing the same information.

As in the case of column vector signals, the underlying physical process that resulted in the input image must be taken into account while interpreting the image. For example, it would be worthwhile to note if the image that is analysed was captured as it is from a physical process (e.g., explosion of a super nova, penetrant testing results), or was assembled / synthesised based on a number of physical measurements (e.g., a cross-section temperature profile of a room, represented in temperature calibrated pseudo-colour).

Once these aspects are studied and a standard method of image representation, analysis and interpretation is agreed upon, the procedures themselves are similar to those presented in the native time domain described in the previous Section 4.3.

4.5 The Frequency and Power Domain

Next to the native domains time (for signals) and X-Y space (for images), the frequency and power domains assume great importance in the analysis of data. The quantities 'frequency' and 'power' are directed related to the measurable and actionable properties of the physical world. The methods for obtaining the frequency spectrum and the power spectrum are remarkably similar for signals and images.

The most famous transform that takes a time-domain signal to the frequency domain is the Fourier transform. The algorithm devised by Cooley and Tukey for this purpose is the Fast Fourier transform. The following code listing, which can be found in any standard textbook, shows a variant of this famous algorithm.

In the following code listing 4.1, the time-domain data having a length N, which needs to be converted to frequency domain, resides in the vector 'data'. At the end of the transform the N points of the vector 'data' contain the Fourier transform in the order [Re(0), Re(1),...,Re(N/2),Im(N/2-1),...,Im(1)]. The value Re(n) represents the magnitude of the frequency spectrum at the point 'n' (the point 'n' itself represents a unique frequency value, determined by the sampling frequency of the original time record and its record length). The value Im(m) represents the phase of the frequency spectrum at the point 'm'. Note that due to the nature of this Fast Fourier transform algorithm, the phase values are stored in reverse sequence. This must be remembered while using the Code Listing 4.1.

```
/*    Implementation of the Module that finds the forward Fourier
      Transform using FFT.
//   Result is stored in working_array.
//   Real-valued, in-place, Cooley-Tukey radix-2 FFT Module
//   Real input and output data in working_array
//   Length N = 2**M
//   Decimation-in-time, cos/sin in innermost loop
//   Output in order [Re(0), Re(1),...,Re(N/2),Im(N/2-1),...Im(1)]
*/

void findfftdata(float *data, int N)
{
int j;
int i;
int i1;
int i2;
int i3;
int i4;
int k;
int N1;
int N2;
int N4;
int M;
float *x;
float temp;
float E;
float A;
float cc;
float ss;
float t1;
float t2;

x = vector(0,N);

M = gettwospower(N);

for(i=0; i<N; i++)
   {
   j = i + 1;
   x[j] = data[i];
```

```
   }

/* Digit reverse counter */

j  = 1;
N1 = N - 1;
for(i=1; i<(N1+1); i++)
   {
   if(i<j)
     {
     temp = x[j];
     x[j] = x[i];
     x[i] = temp;
     k = N/2;
     }
   else if(i>=j) k = N/2;
   while(k<j)
        {
        j = j - k;
        k = k/2;
        }
   j = j + k;
   }

/* Length two butterflies */
for(i=1; i<(N+1); i++)
   {
   temp = x[i];
   x[i] = temp + x[i+1];
   x[i+1] = temp - x[i+1];
   i++;
   }

/*Other butterflies */
N2 = 1;
for(k=2; k<(M+1); k++)
   {
   N4 = N2;
   N2 = 2*N4;
   N1 = 2*N2;
   E  = 6.2831853079586/N1;
   for(i=1; i<(N+1); i++)
      {
      temp = x[i];
      x[i] = temp + x[i+N2];
      x[i+N2] = temp - x[i+N2];
      x[i+N4+N2] = -x[i+N4+N2];
      A = E;
      /* Note that in the first run, N4 = 1, so the next
         statement, could cause some problem. */
      for(j=1; j<N4; j++)
         {
         i1 = i+j;
         i2 = i - j + N2;
         i3 = i + j + N2;
         i4 = i - j + N1;
         cc = cos(A);
         ss = sin(A);
         A = A + E;
         t1 = x[i3]*cc + x[i4]*ss;
         t2 = x[i3]*ss - x[i4]*cc;
```

```
            x[i4] = x[i2] - t2;
            x[i3] = -x[i2] - t2;
            x[i2] = x[i1] - t1;
            x[i1] = x[i1] + t1;
            }
         i = i + (N1-1);
         }
    }
for(i=0; i<N; i++)
    {
    j = i + 1;
    data[i] = x[j];
    }
free_vector(x,0,N);
}
```

Code Listing 4.1 The Fourier Transform

Figure 4.1 shows a time domain signal that has a sine wave superposed with random noise.

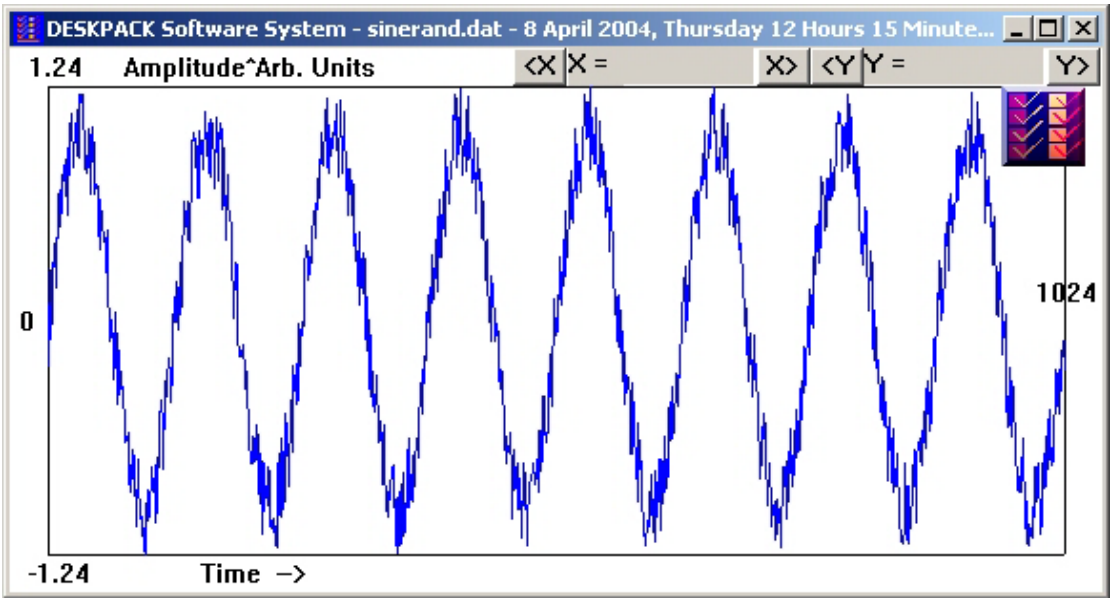


Figure 4.1 Sinusoidal wave superposed with random noise

If we use the code listing 4.1 and take a Fourier transform, the resultant data is shown in Figure 4.2.

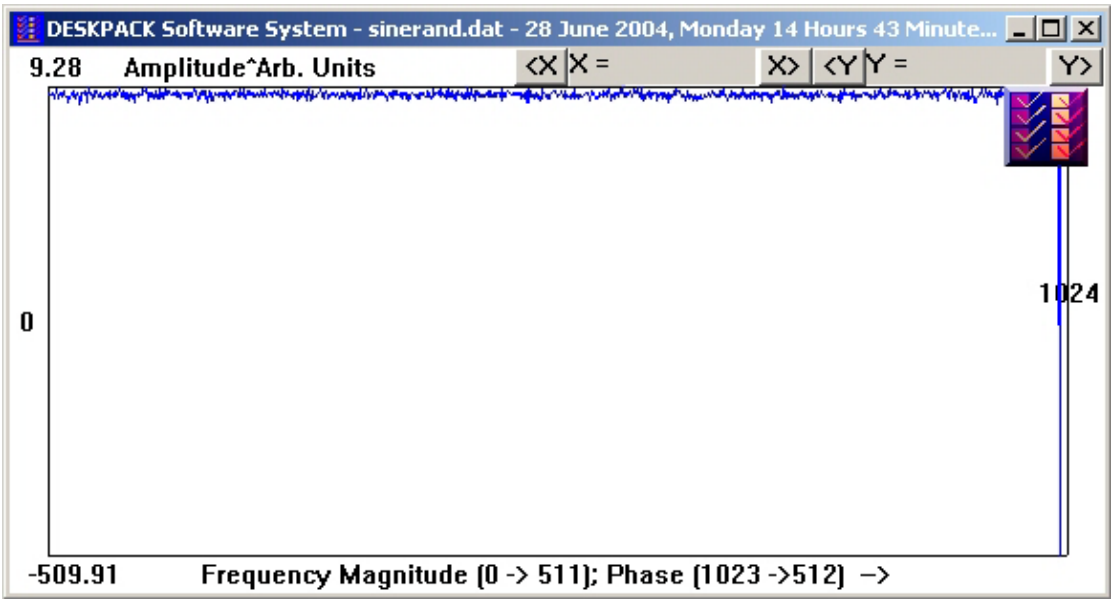


Figure 4.2 Fourier transform of Figure 4.1

In Figure 4.2, note that the first 512 points (0 to 511) represent the Frequency Magnitude values and the next 512 points (1023 to 512) represent the Frequency Phase in reverse order.

If we apply the inverse Fourier transform as shown in Code listing 4.2 to the Figure 4.2, we will get the original time domain signal as shown in Figure 4.1.

```
/*    Implementation of the Module that finds the Inverse Fourier
      Transform.
//   Real-valued, in-place, split-radix IFFT Module
//   Hermitian Symmetric input and Real output
//   Length N = 2**M
//   Decimation-in-frequency, cos/sin in second loop
//   Input order [Re(0), Re(1),...,Re(N/2),Im(N/2-1),...Im(1)]


void findifftdata(float *data, int N)
{
float *x, xt, t1, t2, t3, t4, t5, r1;
int M, N1, N2, N4, N8, is, id, k, j, i, i0, i1, i2, i3, i4, i5, i6,
i7, i8;
double a, a3, e, cc1, ss1, cc3, ss3;

x = vector(0,N);

M = gettwospower(N);

for(i=0; i<N; i++)
   {
   j = i + 1;
   x[j] = data[i];
   }

/* Implementation from the Paper Starts Here */
```

```
/* L Shaped Butterflies */

N2 = 2*N;
for(k=1; k<=M-1; k++)
   {
   is = 0;
   id = N2;
   N2 = N2/2;
   N4 = N2/4;
   N8 = N4/2;
   e = 6.283185307179586/N2;
   do {
      for(i=is; i<=N-1; i= i+id)
         {
         i1 = i + 1;
         i2 = i1 + N4;
         i3 = i2 + N4;
         i4 = i3 + N4;
         t1 = x[i1] - x[i3];
         x[i1] = x[i1] + x[i3];
         x[i2] = 2.0*x[i2];
         x[i3] = t1 - 2.0*x[i4];
         x[i4] = t1 + 2.0*x[i4];
         if(N4!=1)
           {
           i1 = i1 + N8;
           i2 = i2 + N8;
           i3 = i3 + N8;
           i4 = i4 + N8;
           t1 = (x[i2] - x[i1])/sqrt(2.0);
           t2 = (x[i4] + x[i3])/sqrt(2.0);
           x[i1] = x[i1] + x[i2];
           x[i2] = x[i4] - x[i3];
           x[i3] = 2.0*(-1.0*t2 - t1);
           x[i4] = 2.0*(-1.0*t2 + t1);
           }
         }
         is = 2*id - N2;
         id = 4*id;
      } while (is < N-1);
   a = e;
   for(j=2; j<=N8; j++)
      {
      a3 = 3.0*a;
      cc1 = cos(a);
      ss1 = sin(a);
      cc3 = cos(a3);
      ss3 = sin(a3);
      a = j*e;
      is = 0;
      id = 2*N2;
      do {
         for(i=is; i<=N-1; i = i + id)
            {
            i1 = i + j;
            i2 = i1 + N4;
            i3 = i2 + N4;
            i4 = i3 + N4;
            i5 = i + N4 -j + 2;
            i6 = i5 + N4;
            i7 = i6 + N4;
```

```
            i8 = i7 + N4;
            t1 = x[i1] - x[i6];
            x[i1] = x[i1] + x[i6];
            t2 = x[i5] - x[i2];
            x[i5] = x[i2] + x[i5];
            t3 = x[i8] + x[i3];
            x[i6] = x[i8] - x[i3];
            t4 = x[i4] + x[i7];
            x[i2] = x[i4] - x[i7];
            t5 = t1 - t4;
            t1 = t1 + t4;
            t4 = t2 - t3;
            t2 = t2 + t3;
            x[i3] = t5*cc1 + t4*ss1;
            x[i7] = -1.0*t4*cc1 + t5*ss1;
            x[i4] = t1*cc3 - t2*ss3;
            x[i8] = t2*cc3 + t1*ss3;
            }
         is = 2*id - N2;
         id = 4*id;
         } while(is < N-1);
      }
   }
/* Length two butterflies */
is = 1;
id = 4;
do
  {
  for(i0=is; i0<=N; i0 = i0 + id)
     {
     i1 = i0 + 1;
     r1 = x[i0];
     x[i0] = r1 + x[i1];
     x[i1] = r1 - x[i1];
     }
  is = 2*id - 1;
  id = 4*id;
  } while(is < N);

/* Digit Reverse Counter */
j = 1;
N1 = N - 1;
for(i=1; i<=N1; i++)
   {
   if(i<j)
     {
     xt = x[j];
     x[j] = x[i];
     x[i] = xt;
     }
   k = N/2;
   while(k<j)
        {
        j = j - k;
        k = k/2;
        }
   j = j + k;
   }

for(i=1; i<=N; i++)
   {
```

```
   x[i] = x[i]/N;
   }

/* Implementation from the Paper Ends Here */

for(i=0; i<N; i++)
   {
   j = i + 1;
   data[i] = x[j];
   }
free_vector(x,0,N);
}
```

Code Listing 4.2 The Inverse Fourier Transform

On closer scrutiny, one would find that Figure 4.2 (the Fourier transform of Figure 4.1) is technically correct, but offers little insight into the frequency spectrum of the input signal, as a sum total of the frequency magnitude and phase. A nicer way to know the frequency spectrum would be to know the extent of contribution, or power, of each frequency component. The autopower spectrum offers this information.

If the n^{th} frequency magnitude and phase component in a Fourier transform is represented as $F(n) = x(n) + iy(n)$, where 'x' and 'y' are the magnitude and phase, then the n^{th} component of the autopower is given as

$$APW(n) = [x(n) + iy(n)]\,[x(n) + iy(n)]^* \qquad \text{Equation 4.1}$$

Where $[x(n) + iy(n)]^*$ is the complex conjugate of $[x(n) + iy(n)]$.

This operation is listed in the form of source code in the following code listing 4.3.

```
/* Function to find the Auto-Power of an array */
void autopwrdata(float *data, int rlength)
{
int i,j,k;
k = rlength/2;

findfftdata(data,rlength);
data[0] = pow(data[0],4.0);
data[k] = pow(data[k],4.0);
for(i=1; i<k; i++)
   {
   j = rlength - i;
   data[i] = data[i]*data[i] + data[j]*data[j];
   data[j] = 0.0;
   }
}
```

Code Listing 4.3 The Auto-Power Spectrum

So, how does an autopower look ? Figure 4.3 shows the autopower of the sinusoidal superposed with noise signal (Figure 4.1).

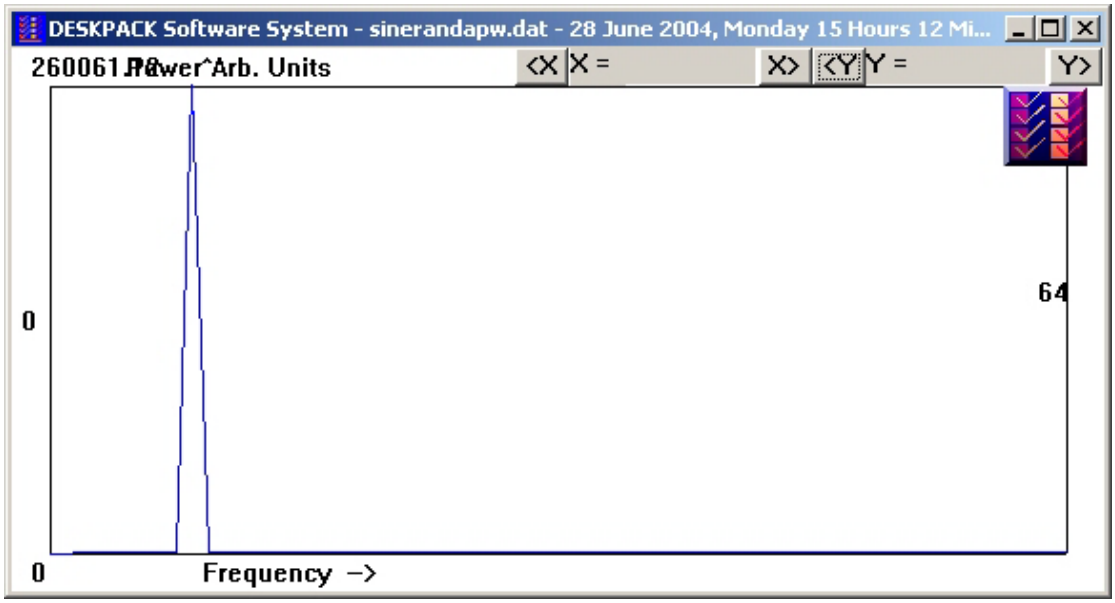


Figure 4.3 AutoPower of the Figure 4.1

If we study the auto power shown in Figure 4.3 closely, we observe the following points:

- There is a single peak corresponding to the single sinusoidal wave
- The peak has a certain width
- The magnitude and phase information are now fused
- All the Power values (for each frequency component) are positive

In view of observation 3 mentioned above, we cannot get back the time domain signal from the autopower spectrum.

Also note that the spectrum is called "autopower" because the Fourier transform of the *same* signal is complex conjugate multiplied to get the result (See Equation 4.1).

If we use the Fourier transform of two different signals to obtain the complex conjugate result, as shown in Equation 4.2, then the resulting waveform is called Cross Power.

$$CPW(n) = [x(n) + iy(n)]\,[a(n) + ib(n)]^* \qquad \text{Equation 4.2}$$

Equation 4.2 shows the complex conjugate multiplication of the Fourier transform of two different input data, one represented by x(n) + iy(n) and the other a(n) + ib(n). The Cross Power Spectrum highlights the common power features of each of the input waveforms. Unlike the auto-power spectrum, the cross-power spectrum could be either positive or negative, for a given frequency component. The following Code Listing 4.4 shows the method to obtain the Cross Power of two input waveforms.

```
/* Implementation to find the cross-power between Test Data and the
//Ref. Data, both IN ARRAYS. Result is stored in the TEST ARRAY,
'tesdata' */
```

```
void crospwrdata(float *refdata, float *tesdata, int rlength)
{
float *cpwrdata;
int i,j,k;

k = rlength/2;

cpwrdata = vector(0,rlength-1);
findfftdata(refdata,rlength);
findfftdata(tesdata,rlength);
cpwrdata[0] = pow(refdata[0],2.0) + pow(tesdata[0],2.0);
cpwrdata[k] = pow(refdata[k],2.0) + pow(tesdata[k],2.0);
for(i=1; i<k; i++)
   {
   j = rlength - i;
   cpwrdata[i] = refdata[i]*tesdata[i] + refdata[j]*tesdata[j];
   cpwrdata[j] = refdata[j]*tesdata[i] - refdata[i]*tesdata[j];
   }
for(i=0; i<rlength; i++)
   {
   tesdata[i] = cpwrdata[i];
   }
free_vector(cpwrdata,0,rlength-1);
}
```

Code Listing 4.4 The Cross-Power Spectrum

Figure 4.4 shows the cross-power spectrum of a sine wave and a random wave.

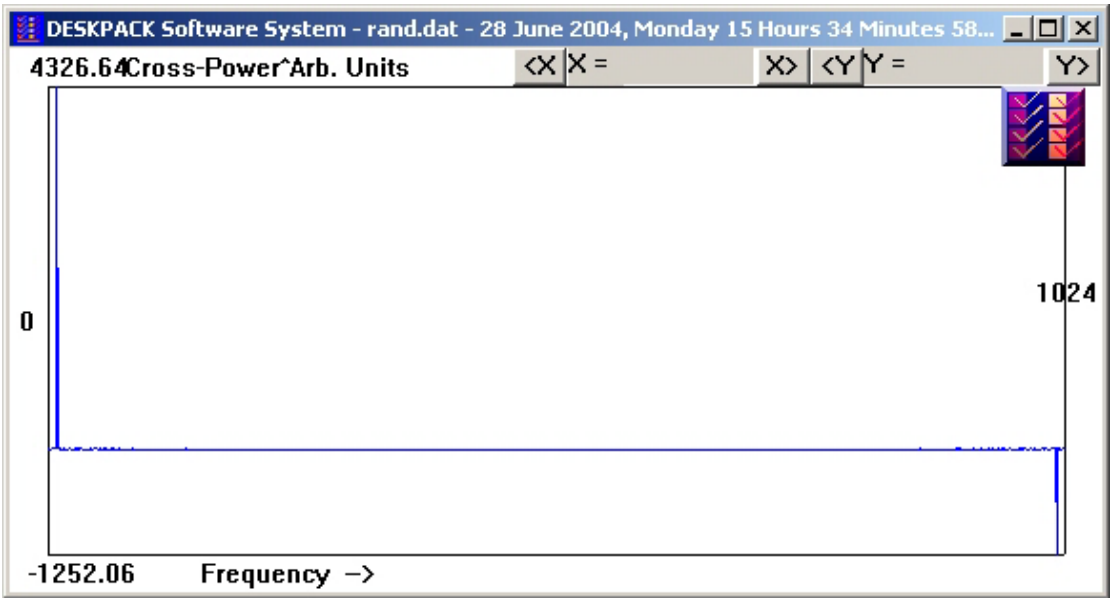


Figure 4.4 Cross Power Spectrum between a Sine and a Random Waveform

Some times, we are interested in finding only the magnitude of the frequency spectrum (magnitude of each frequency component) or the phase of the frequency spectrum (phase of each frequency component). The following two Code Listings (4.5 and 4.6) show the method used to obtain this information for a given signal input.

```
/* Function to find ONLY the Frequency Magnitude Values of an array
*/
void findfreqmagdata(float *data, int rlength)
{
int i,j,k;
k = rlength/2;

findfftdata(data,rlength);
data[0] = pow(data[0],4.0);
for(i=1; i<k; i++)
   {
   j = rlength - i;
   data[j] = 0.0;
   }
}
```

Code Listing 4.5 To find the Magnitude of the Frequency Spectrum

```
/* Function to find ONLY the Frequency Phase Values of an array */
void findfreqphasedata(float *data, int rlength)
{
int i,j,k;
k = rlength/2;

findfftdata(data,rlength);
data[0] = pow(data[k],4.0);
for(i=1; i<k; i++)
   {
   j = rlength - i;
   data[i] = data[j];
   data[j] = 0.0;
   }
}
```

Code Listing 4.6 To find the Phase of the Frequency Spectrum

Figures 4.5 and 4.6 show the magnitude only and phase only values of a given input signal. In this case, the input signal is the sinusoidal signal superposed with random noise (same as Figure 4.1).

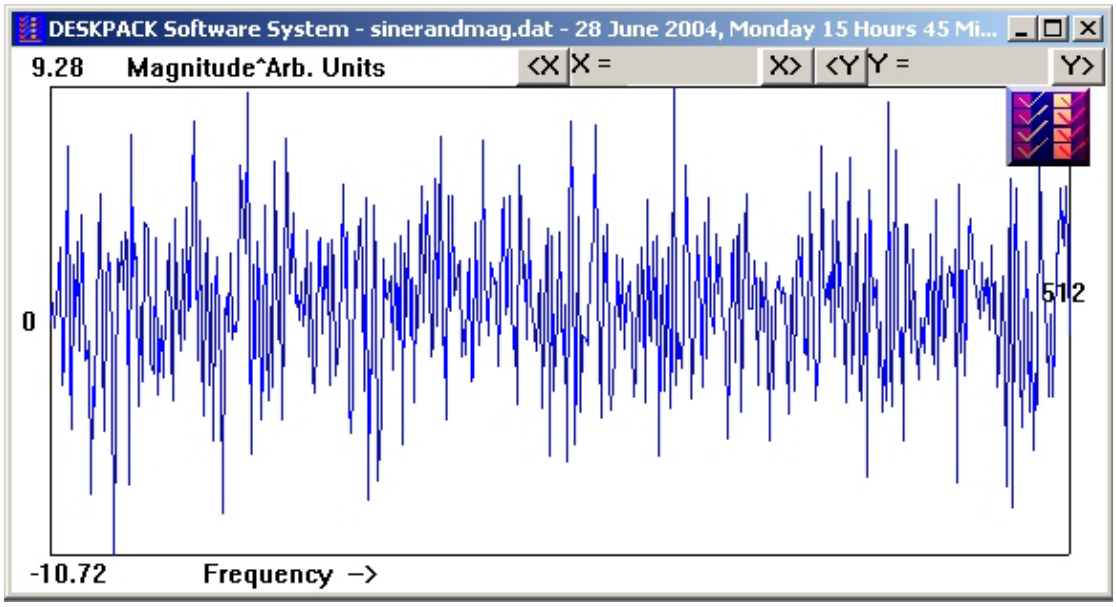


Figure 4.5 Magnitude of the Frequency Spectrum of Input shown in Figure 4.1

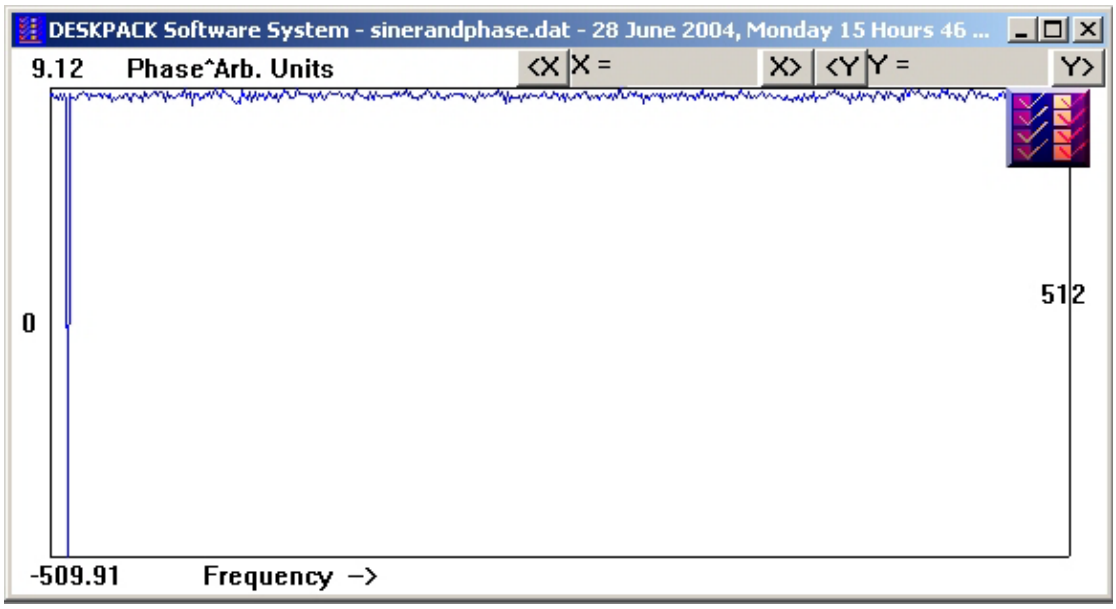


Figure 4.6 Phase of the Frequency Spectrum of Input shown in Figure 4.1

It would be worthwhile to compare the Figures 4.5 and 4.6 with the combined figure shown in Figure 4.2. Figure 4.2 is a combination of Figures 4.5 and 4.6 (the latter being combined in the reverse order).

Such separate values (separate Magnitude and Phase values) might be useful in generating Clusters to separate data having poor signal-to-noise ratio.

Another type of input that could be useful in cluster generation is the magnitude of the cross power spectrum. In an Argand diagram sense, the magnitude of a cross power value say [k(n) + ij(n)], can be represented as

$$Mag(n) = \{[k(n)]^2 + j(n)^2\}^{0.5} \qquad \text{Equation 4.3}$$

The following code (Code Listing 4.7) shows this approach to get the magnitude values of a cross-power spectrum.

```
/* Function to find the Magnitude of a Cross-Power Spectrum given as
data - Begins Here */
void findcpwrtomagdata(float *data, int rlength)
{
int i,j,k;
float temp;

k = rlength/2;

for(i=0; i<k; i++)
   {
   j = (rlength - 1) - i;
   temp = data[i]*data[i] + data[j]*data[j];
   data[i] = pow(temp, 0.5);
   }
}
/* Function to find the Magnitude of a Cross-Power Spectrum given as
data - Ends Here */
```

Code Listing 4.7 To find the Magnitude from an input Cross-Power Data

For a typical cross-power spectrum, say as shown in Figure 4.4, its magnitude spectrum as computed by the Code Listing 4.7 is shown in Figure 4.7.

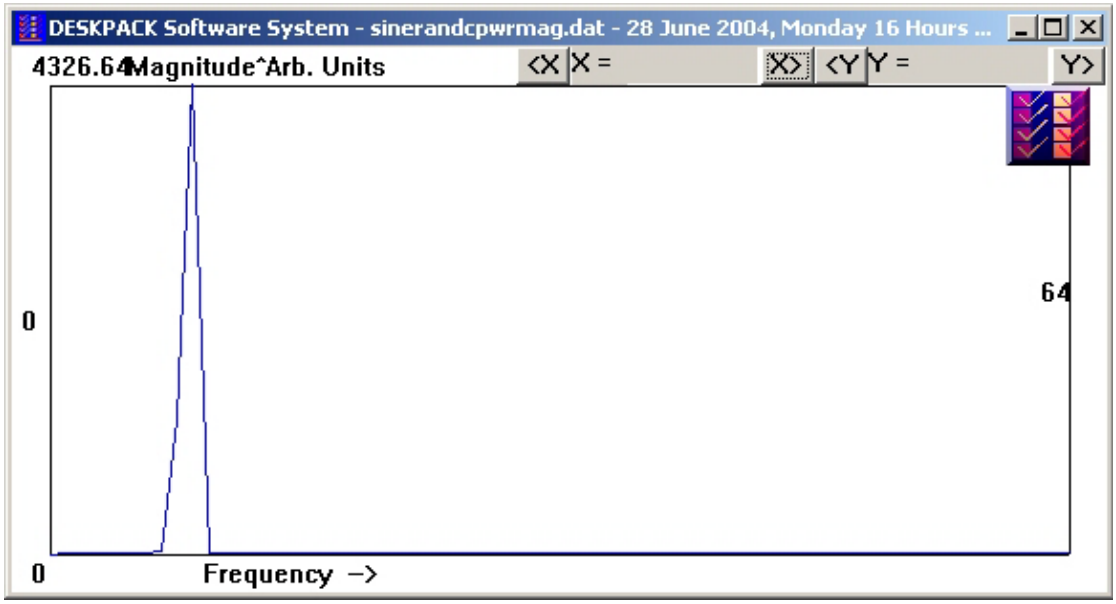


Figure 4.7 Magnitude values of a Cross Power Data (Cross Power between a sine wave and a random wave)

Question: Compare Figures 4.3 and 4.7. Write down your observations.

If we wish to find an estimate of the variations present in a frequency spectrum how do we go about? How to know if the frequency spectrum is constant or if it has undulations? If undulations are present in a frequency spectrum how to quantify these? These are some of the issues addressed by the quantity known as Cepstrum. One can think of this as a spectrum of a spectrum. The term spectrum has degenerated to Cepstrum, whose frequency equivalent is called Quefrency. The following Code Listing 4.8 shows the method to obtain the Cepstrum.

```
/* Function to find the Cepstrum of a Waveform in an Array - Begins
Here*/
void findcepsdata(float *data, int rlength)
{
autopwrdata(data,rlength);
findlogdata(data,rlength/2);
multsigscalardata(data,rlength/2,0.5);
autopwrdata(data,rlength/2);
}
/* Function to find the Cepstrum of a Waveform in an Array - Ends
Here */
```

Code Listing 4.8 To find the Cepstrum of a Signal Data

4.6 The Correlation Domain

Relationships between points within a signal or image, either in time or space, can be aggregated to give very useful information. Such information in time gives insight into the behaviour of the data over time, e.g., its periodicity. If space is used as the basis to obtain correlation, then the resulting information offers insight into the self-similarity of the data. Such aggregate correlation relationships can be obtained for the same data set (auto-correlation) represented by R_{xx} or for two different data sets (cross-correlation) represented by R_{xy}.

Code Listing 4.9 shows the procedure to obtain the auto-correlation function for a column vector of data.

```
/* Implementation of the module that finds the auto-corr. of a signal
IN AN ARRAY */
void autocordata(float *data, int rlength)
{
float *rxx;
int m;
int i;

rxx = vector(0,rlength-1);
```

```
for(m=0; m<rlength; m++)
   {
   rxx[m] = 0.0;
   for(i=0; i<(rlength-m); i++)
      {
      rxx[m] += data[m+i]*data[i];
      }
   }

for(m=0; m<rlength; m++)
   {
   data[m] = rxx[m];
   }
free_vector(rxx,0,rlength-1);
}
```

Code Listing 4.9 The auto-correlation function

The following Figure 4.8 shows the auto-correlation function plot of the input signal data shown in Figure 4.1.

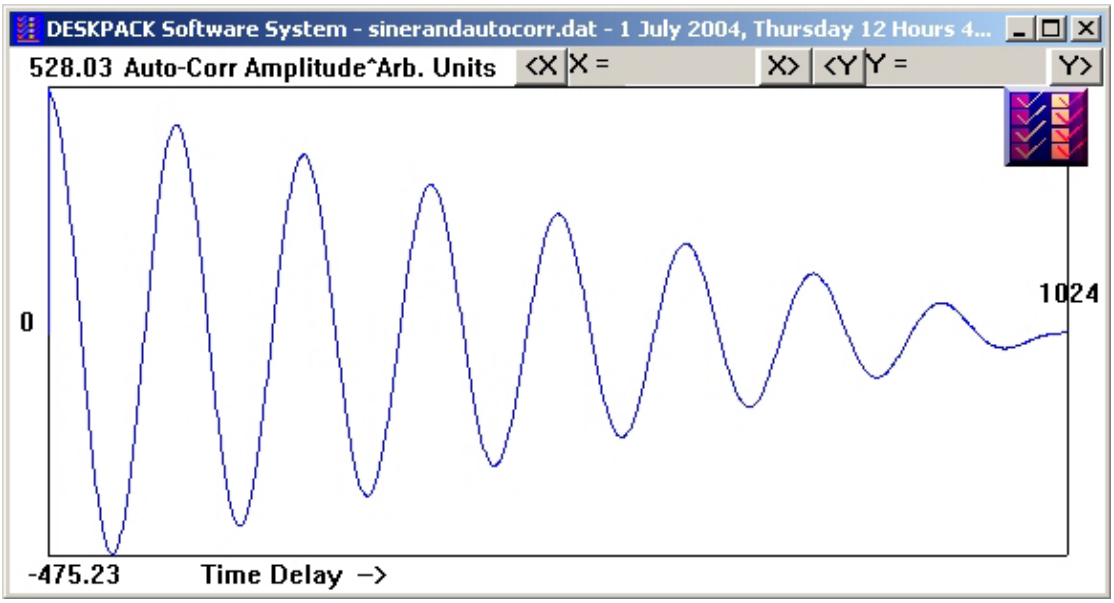


Figure 4.8 Auto-Correlation Plot of a Sine + Random Input (Figure 4.1)

Figure 4.9 below shows the same auto-correlation function plot for a completely random signal. The input random signal is shown in Figure 4.10.

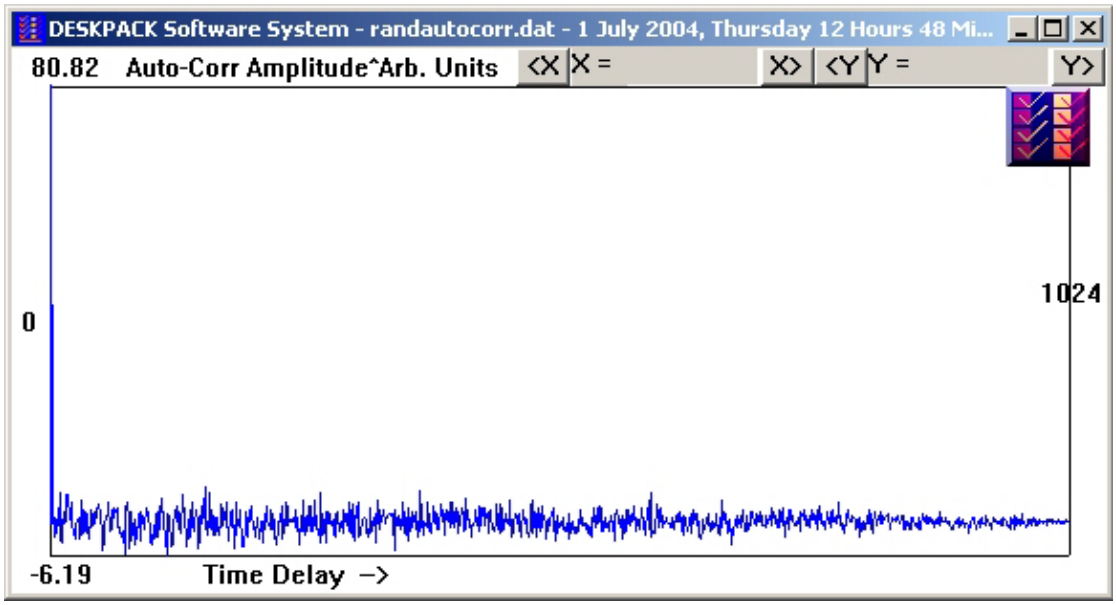


Figure 4.9 Auto-Correlation Plot of a Random Input Signal

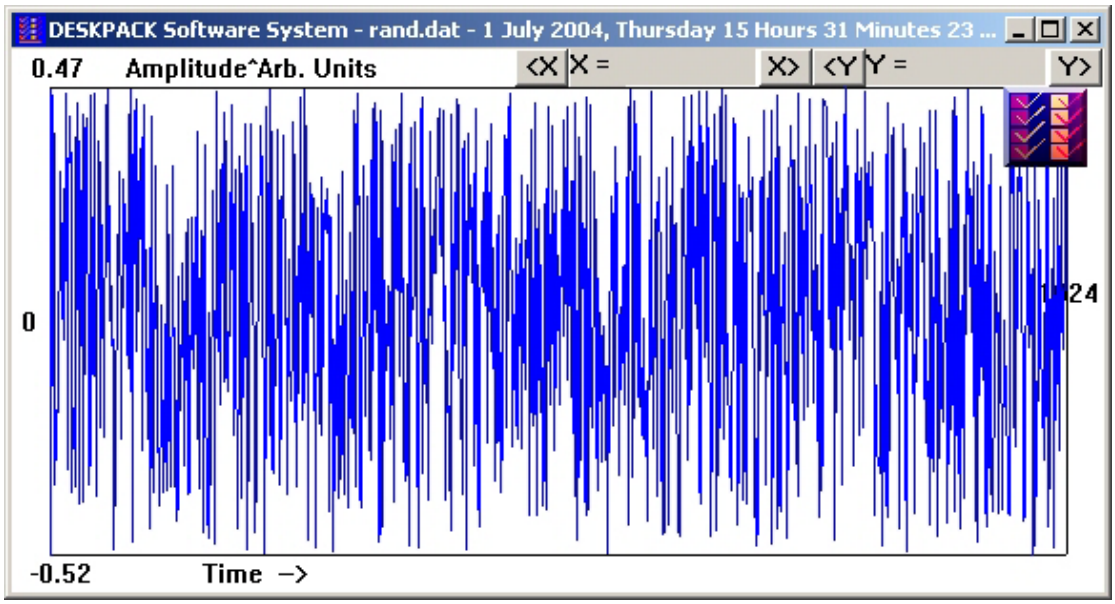


Figure 4.10 A Typical Random Signal

It would be worthwhile comparing the two figures, Figure 4.8 and Figure 4.9. The x-axis of both the figures represent time-delay, while y-axis represent the amplitude of the auto-correlation function as time delay increases. The first point (x-axis origin) represents the auto-correlation function value when there is **no** time delay (or, zero time delay). Also note that these figures show just one half of the auto-correlation function (either the positive or negative delay portions). The negative-x-axis will show the other half.

Figure 4.8 shows periodicity even as the time-delay increases. This is natural since the input (shown in Figure 4.1) sinusoidal signal is basically periodic, even though corrupted by a small amount of random noise. In a sine wave as we travel from the left to the right

(i.e., as we change the time delay) its periodicity is generic, which is also reflected in the autocorrelation function.

This is not the case in a random signal. In a truly random signal, there will be no periodicity and hence if we wish to correlate one portion of the random signal with another portion (of the same random signal, since this is **auto**-correlation function), we would discover that there is no correlation indeed. This is what is represented by the auto-correlation function of a random signal, as shown in Figure 4.9. The initial peak (x=0; no time delay) represents the correlation without any delay. The moment we introduce a delay (x>1; seeking correlation between one portion of the signal with another portion of the signal), the value of the correlation function comes down drastically.

There are some other interesting observations about the auto-correlation functions:

- They are even – i.e., $R_{xx}(n) = R_{xx}(-n)$, where 'n' represents positive delay, while '-n' represents equal negative delay
- The auto-correlation function has the highest value when the time delay is zero, i.e., when there is no time delay or when n = 0; $R_{xx}(0) >= R_{xx}(n)$, where $n \neq 0$.
- The value $R_{xx}(0)$ is related to the root means square value of the data record

The next Code Listing 4.10 shows the procedure to obtain the *Cross*-Correlation function, obtained using two different data records.

```
/* Implementation to find the cross-correlation between Test Data and
the
//Ref. Data, both IN FILES. Result is stored in the TEST FILE */
void croscordata(float *refdata, float *tesdata, int N)
{
float *rxy;
int i, m;

rxy = vector(0,N-1);

for(m=0; m<N; m++)
   {
   rxy[m] = 0.0;
   for(i=0; i<(N-m); i++)
      {
      rxy[m] += tesdata[m+i]*refdata[i];
      }
   }

for(m=0; m<N; m++)
   {
   tesdata[m] = rxy[m];
   }
free_vector(rxy,0,N-1);
}
```

Code Listing 4.10 The cross-correlation function

Figure 4.11 shows the cross-correlation plot for two different input signals, one a sinusoidal signal corrupted with noise (Figure 4.1) and the other a random noise signal (Figure 4.10).

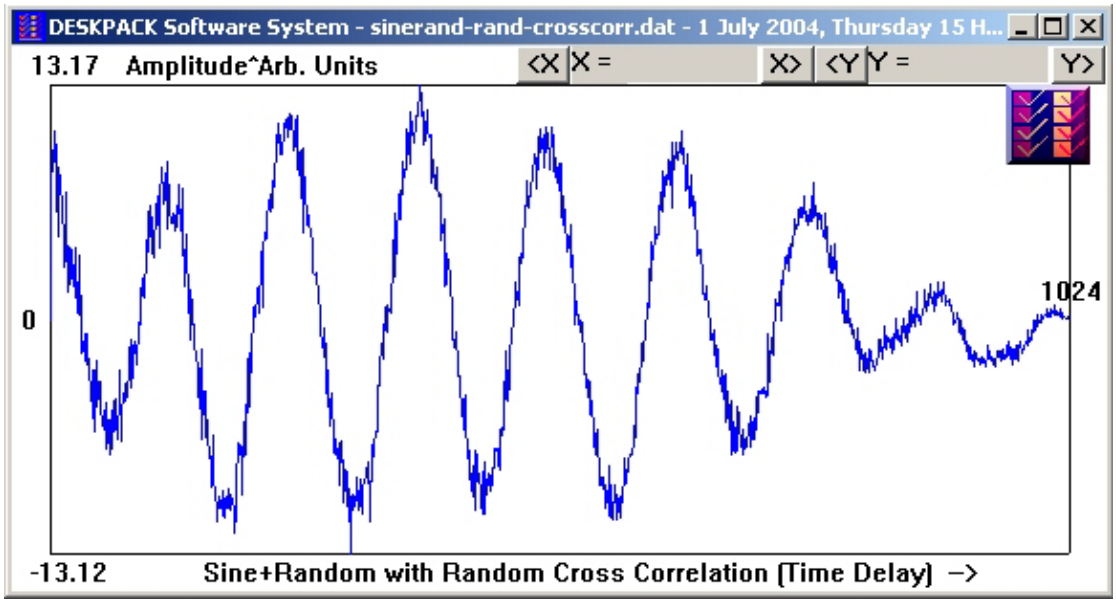


Figure 4.11 Cross Correlation between a Sine+Random Signal and a Random Signal

A comparison between Figure 4.8 (auto-correlation function of a Sine+Random signal) and 4.11 (cross-correlation function between a Sine+Random Signal and a Random Signal) provides a good insight into the differences between the auto- and cross-correlation functions.

Some of these are listed below:

- In Figure 4.8, even though the input signal (figure 4.1) is a Sine+Random signal, when the auto-correlation function is found, the randomness has reduced in the resultant signal shown in Figure 4.8.
- In Figure 4.11, since the cross-correlation function highlights a *commonality* among input signals (with respect to time or delay), the random nature which is predominant in one of the inputs (Figure 4.10), shows up in the resultant cross-correlation function in Figure 4.11.
- Figure 4.8 (auto-correlation function) has only positive values; Figure 4.11 (cross-correlation function) has both positive and negative values, depending upon the time delay
- Even though this is not shown, the cross-correlation function need not be an even function like the auto-correlation function. That is, for an auto-correlation function $R_{xx}(n) = R_{xx}(-n)$; for a cross-correlation function, it could be $R_{xy}(n) \neq R_{xy}(-n)$.
- Also, note that the highest value that appears in the cross-correlation plot need not be when the time-delay is zero. It can occur anywhere depending upon the input signals. That is $R_{xx}(n) >= R_{xx}(0)$, where 'n' is **any** time delay value.

The last property listed above is interesting; this means that if the same type of activity occurs in the two input signal records, but at different locations within the records (i.e., if the same activity occurs with a certain time delay), this difference is highlighted by the cross-correlation function. This property is used in some ultrasonic measurements to measure liquid flow speed or in assessing the wave velocity.

The following Code Listing 4.11 shows the method to obtain a special pattern called the Demodulated Auto-Correlogram pattern which is a derivative of the auto-correlation function and is useful in discriminating / classifying data.

```
/* Function to find the DMAC Pattern of a given data in an array  */
void finddmacdata(float *data, int rlength)
{
autocordata(data,rlength);
powerdata(data,rlength,2.0);
findlogdata(data,rlength);
}
```

Code Listing 4.11 To find the Demodulated Auto-Correlogram (DMAC) Pattern

The DMAC pattern generation can be explained as follows: It involves finding the auto-correlation function of a given data, and taking its square and finally the log of the squared function. The logarithmic operation is a crucial step in this method. It is well known that the logarithm is directly related to the amount of information present. The log function, in this case, brings out not merely the information lucidly, but also helps unfold the various convolution factors, which results while the signal travels from one medium to the other, characterised by their individual transfer functions. It would be worthwhile to note that, in simple terms, the log function converts the multiplication (in signal terms, convolution) terms into additive ones.

The outline of the resulting DMAC pattern forms an envelope, which is distinct and unique for a given class of signal. The envelope of the pattern is extracted in a normal way by successive maximum technique. This method is a type of digital low pass filtering, where the maximum among a five-value window is set as the value, and the window is moved successively from one end of the function to the other end. Since the autocorrelation function is an even function ie., $x(t) = x(-t)$, it is enough to find the envelope till $N/2$, where N is the total number of points in the auto-correlation function, and the rest can be traced by reflection.

The success of this powerful technique lies in the ability to interpret the resulting envelope pattern. Experiments with natural and artificial defects signals and noise signals indicate that the patterns are indeed different for these signals. These patterns can be characterised, thereby quantifying this approach, by the following parameters:

- The width of the pattern
- The shape of the covering envelope of the pattern (ie., the rate and the mode of fall) (curvature)
- The number of lobes present in the pattern

- Lobe periodicity
- Width of the central lobe
- No.of lobelets (tiny lobes) present in the central lobe, etc.

Figures 4.12 and 4.13 show the DMAC patterns for Sine+Random and Random signal inputs, respectively.

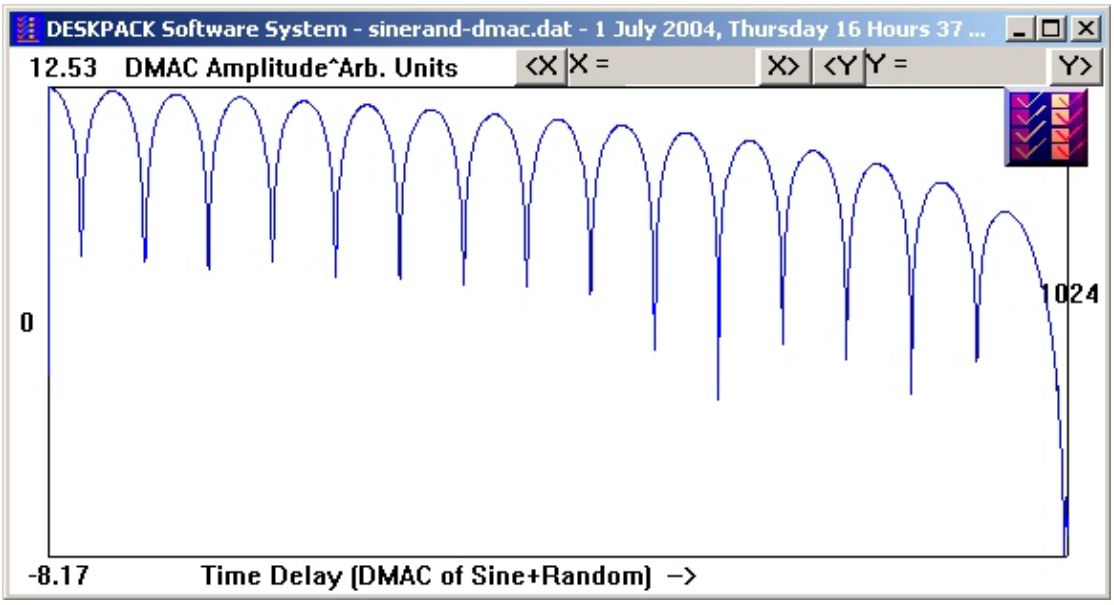


Figure 4.12 DMAC Pattern for a Sine+Random Input Signal (Figure 4.1)

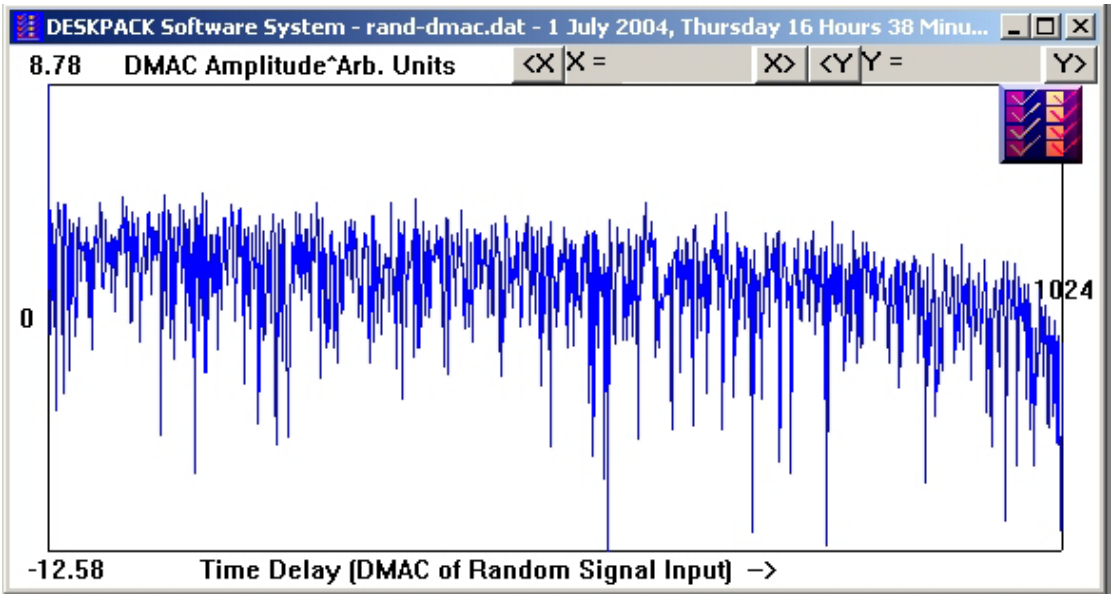


Figure 4.13 DMAC Pattern for a Random Input Signal

Note that even though amplitude-wise there is not much difference between Figures 4.12 and 4.13, the patterns are entirely different. By studying the DMAC properties listed above, it is possible to qualitatively classify data.

The DMAC patterns of real life signals, could be anywhere in between. For example, Figure 4.14 shows the DMAC pattern for an ultrasonic data record, obtained from bubbles present in a column of distilled water.

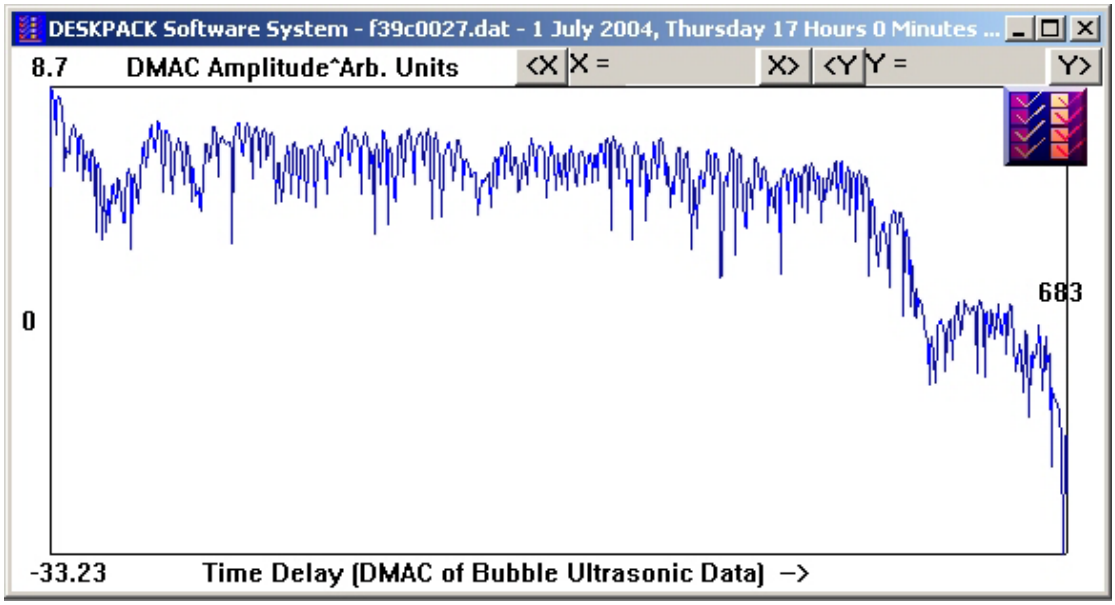


Figure 4.14 DMAC Pattern of Ultrasonic reflection data from Bubbles in Distilled Water

4.7 Data Analysis in Miscellaneous Domains

Apart from the native (time & space), frequency and correlation domains, there are other equally important domains of analysis. Some of them are the Haar domain, Hilbert transform domain, eigenspace domain, feature domain, etc.

```
/* Implementation of the module that finds the Haar Transform of a
signal IN AN ARRAY */
void haardata(float *data, int rlength)
{
float *s, *a;
int m;
int i;
int t;

s = vector(0,rlength-1);
a = vector(0,rlength-1);

t = rlength;

for(i=0; i<rlength; i++)
```

```
    {
     s[i] = data[i];
    }

while(t>1)
{
m = t/2;
for(i=0; i<m; i++)
    {
    a[i]   = (s[2*i] + s[2*i + 1])/2.0;
    a[m+i] = (s[2*i] - s[2*i + 1])/2.0;
    }
for(i=0; i<m; i++)
    {
    s[i]   = a[i];
    s[m+i] = a[m+i];
    }
t = t/2;
}

for(i=0; i<rlength; i++)
    {
    data[i] = s[i];
    }

free_vector(s,0,rlength-1);
free_vector(a,0,rlength-1);
}
```

Code Listing 4.12 The Haar Transform

Figure 4.15 and Figure 4.17 show a typical Sine wave and a typical ultrasonic pulse waveform respectively. Their respective Haar transforms are shown in Figures 4.16 and 4.18 respectively.

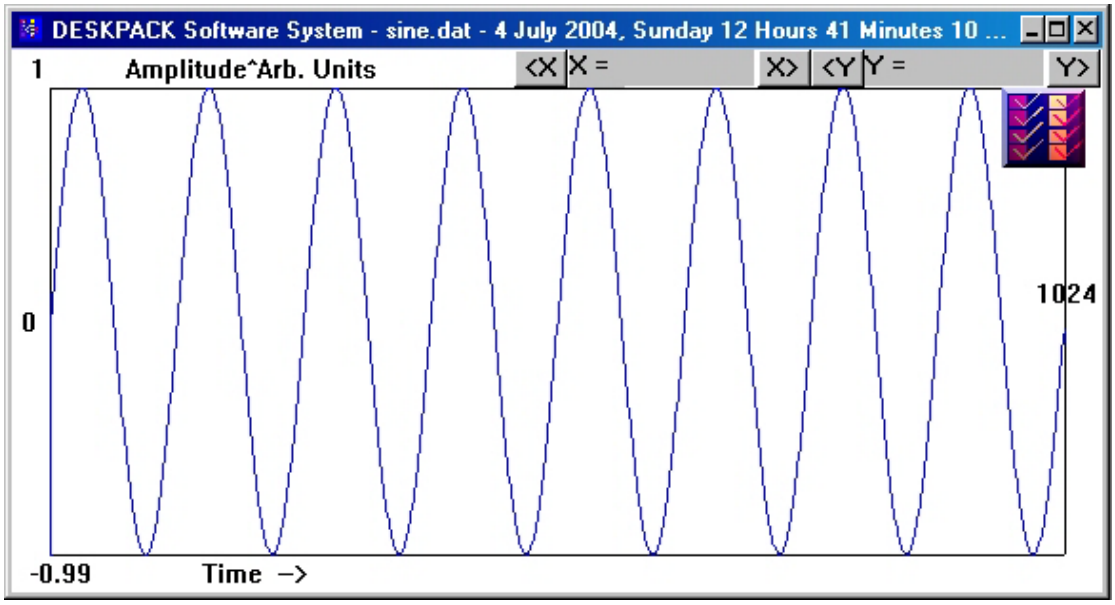


Figure 4.15 Typical Sine Wave

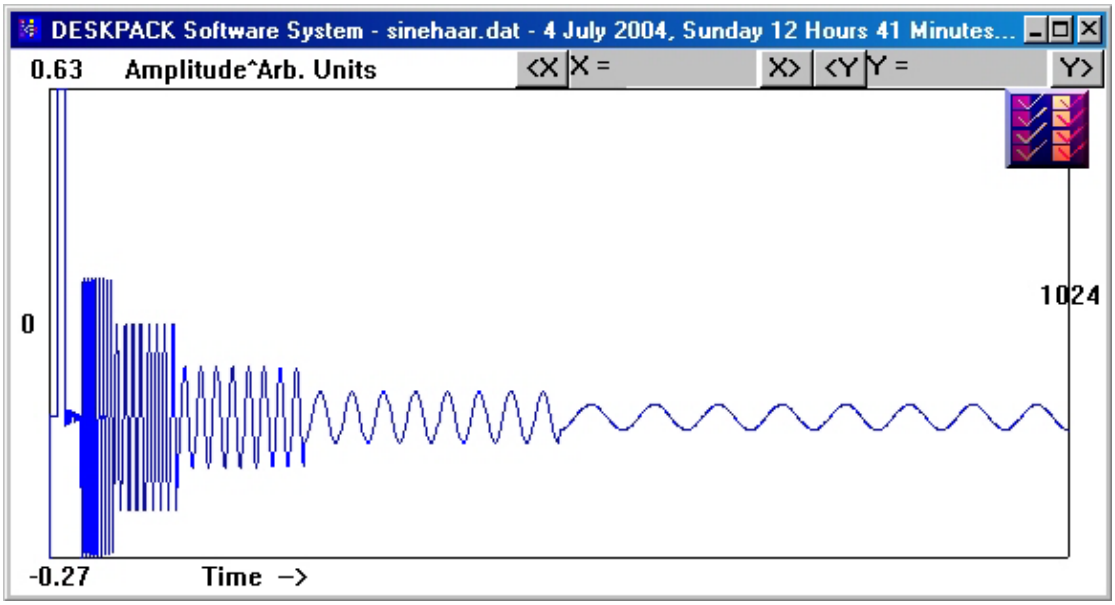


Figure 4.16 Haar Transform of the Sine wave shown in Figure 4.15

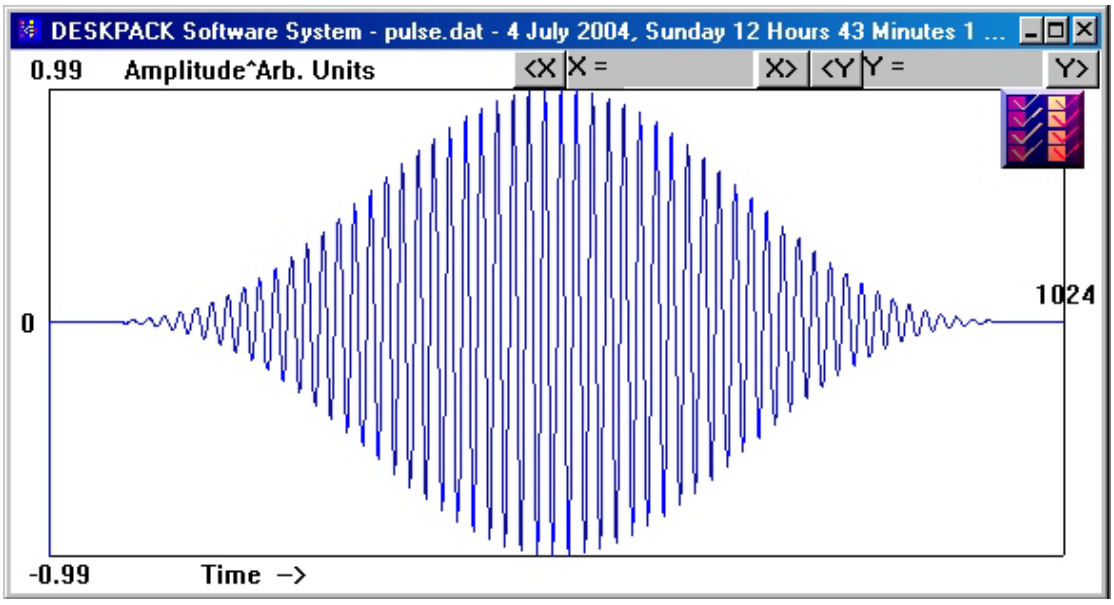


Figure 4.17 Typical Pulse waveform

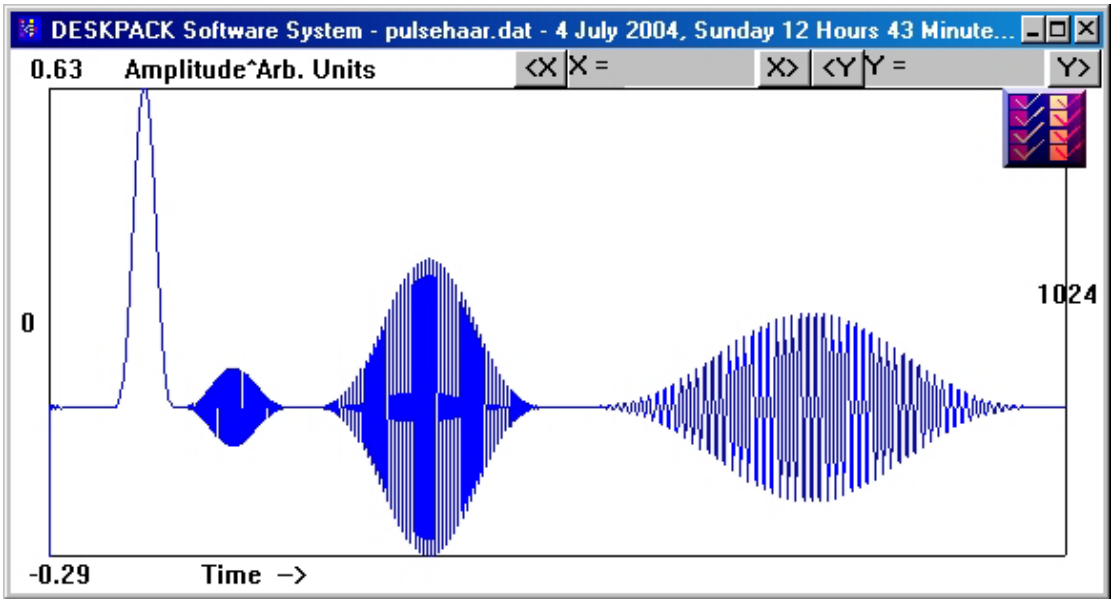


Figure 4.18 Haar Transform of a typical Pulse shown in Figure 4.17

The Hilbert transform is another important transform that represents the convolution of the input data with the function $1/\pi x$. It can be obtained by finding the Fourier transform of the given column data vector, resetting the phase values to zero and then taking the inverse Fourier transform as shown in the Code Listing 4.13.

```
/* Function to find the Hilbert Transform of an array */
void findhilbertdata(float *data, int rlength)
{
int i,j;
j = (rlength/2) + 1;

findfftdata(data,rlength);
for(i=j; i<rlength; i++)
   {
   data[i] = 0.0;
   }
findifftdata(data,rlength);
}
```

Code Listing 4.13 The Hilbert Transform

Figures 4.19 and 4.20 show the Hilbert transforms of a typical sine wave (Figure 4.15) and a typical ultrasonic pulse (Figure 4.17) respectively.

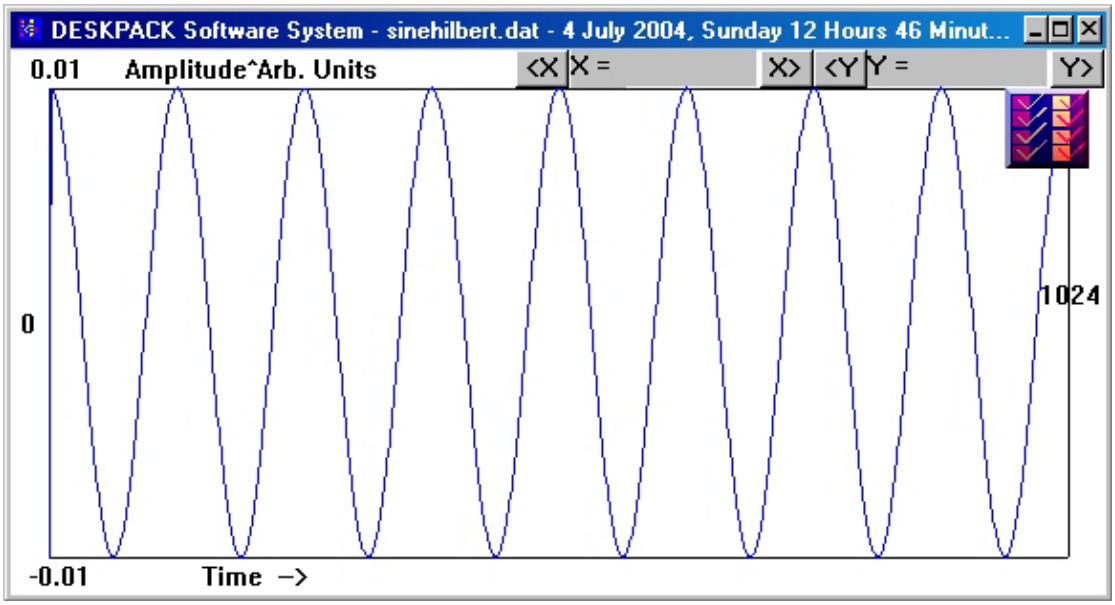


Figure 4.19 Hilbert Transform of a typical Sine wave shown in Figure 4.15

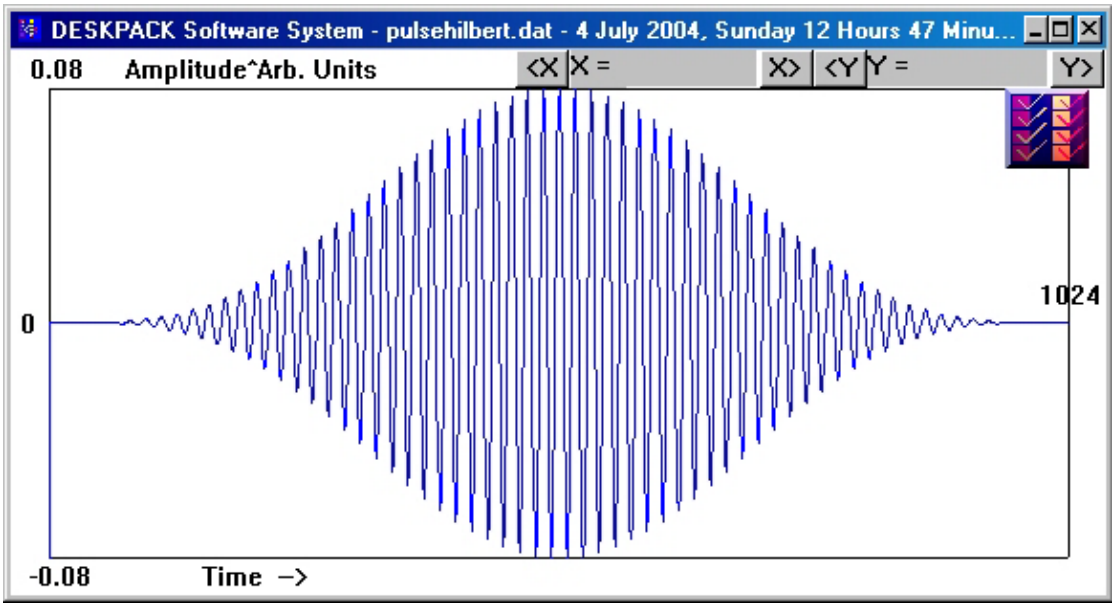


Figure 4.20 Hilbert Transform of a typical Pulse waveform shown in Figure 4.17

The split-spectrum processing has been widely used in reducing noise in ultrasonic waveforms that have poor signal-to-noise ratio (SNR), such as those obtained while testing weld overlays.

The principle behind this technique is to slice (or split) the input waveform's Fourier transform using a number of Gaussian windows, and then take the inverse Fourier transform of the resultant Gaussian slices. If 10 numbers of Gaussian windows are used for splitting the Fourier transform, we would then obtain 10 time-domain signals, each pertaining to one slicing Gaussian window. These 10 resultant time domain signals are then subjected to two noise-reducing algorithms namely minimisation and polarity

thresholding, to obtain a single, noise-reduced final waveform, which is much more easier to interpret because of its superior signal-to-noise ratio.

The following Code Listing 4.14 codifies the split spectrum processing method described above.

```
void splitspectrum(char Infile[128],char Outfile[128],char
LF[128],int *cerr)
     {
      int i,j,rlength,N,sd=5000 ;
      float *tmp,*split,**v;
      float min;
      FILE *fp;

     if((fp=fopen(LF,"a"))==NULL) {*cerr= 37 ;exit(1);}
     fprintf(fp,"%s\n","Opened logfile from within 'Split Spectrum
processing' C Module");
     fprintf(fp,"%s\n","Ready to execute the program");
     fclose(fp);

     rlength=getfflength(Infile);
  /*** find the frequency range ***/
     N=rlength/2;
     N=N/10;

     v=matrix(0,9,0,rlength-1);
  /*** load the values in separate arrays ***/
     for(i=0;i<10;++i)
      {
       load_data(v[i],rlength,Infile);
      }

     for(i=0;i<10;++i)
      {
       gauss("gaus.dat",LF,rlength,sd, N*(i+1),cerr);
       tmp=vector(0,rlength-1);
       load_data(tmp,rlength,"gaus.dat");

       findfftdata(v[i],rlength);

 /*** if prob is greater than 0 then multiply the array value
        with the prob else make the array value as zero ***/
       for(j=0;j<rlength;++j)
        {
         if(tmp[i] > 0)
          {
           v[i][j] = v[i][j]*tmp[i];
          }
         else
          {
           v[i][j]=0;
          }
        }

       findifftdata(v[i],rlength);
       free_vector(tmp,0,rlength-1);
      }

      split=vector(0,rlength-1);
```

```
/*** minimization technique ***/
/*** take the min value and store in the array ***/
   for(i=0;i<rlength;++i)
    {
     min= 35000.0;
     for(j=0;j<10;++j)
      {
       if(v[j][i] < min )
        {  min=v[j][i];  }
       }
      split[i]=min;
    }

   save_data(split,rlength,Outfile);
   free_vector(split,0,rlength-1);

  if((fp=fopen(LF,"a"))==NULL) {*cerr= 37 ;exit(1);}
  fprintf(fp,"%s\n","Completed Split Spectrum processing");
  fclose(fp);
  }
```

Code Listing 4.14 The Split-Spectrum Processing

For further discussions on split spectrum processing and typical results, the readers are referred to the bibliography section where relevant papers are listed.

4.8 Fruits of Data Analysis

A clear outcome of data analysis on the basis of the methods discussed above is that the initial data is "transformed" into a state that is much more palatable for human interpretation.

Even at this stage, if we wish to use a machine to interpret the resultant data for us, we need to provide the machine inputs that are numerical. Processed data in its "raw" form may be suitable for human interpretation, but for a machine, the processed data must be presented in a more refined numerical form, may be as a set of measurable properties of the processed data itself.

Properties of column-vector data or images are also called features. These features are normally fed to machine-learning algorithms to obtain a final decision.

This procedure has three distinct stages, namely:

- Feature Extraction from the processed data
- Feature Selection and Ranking
- Feature Input to the machine-learning algorithm, to obtain a decision

Let us examine each of these three stages briefly:

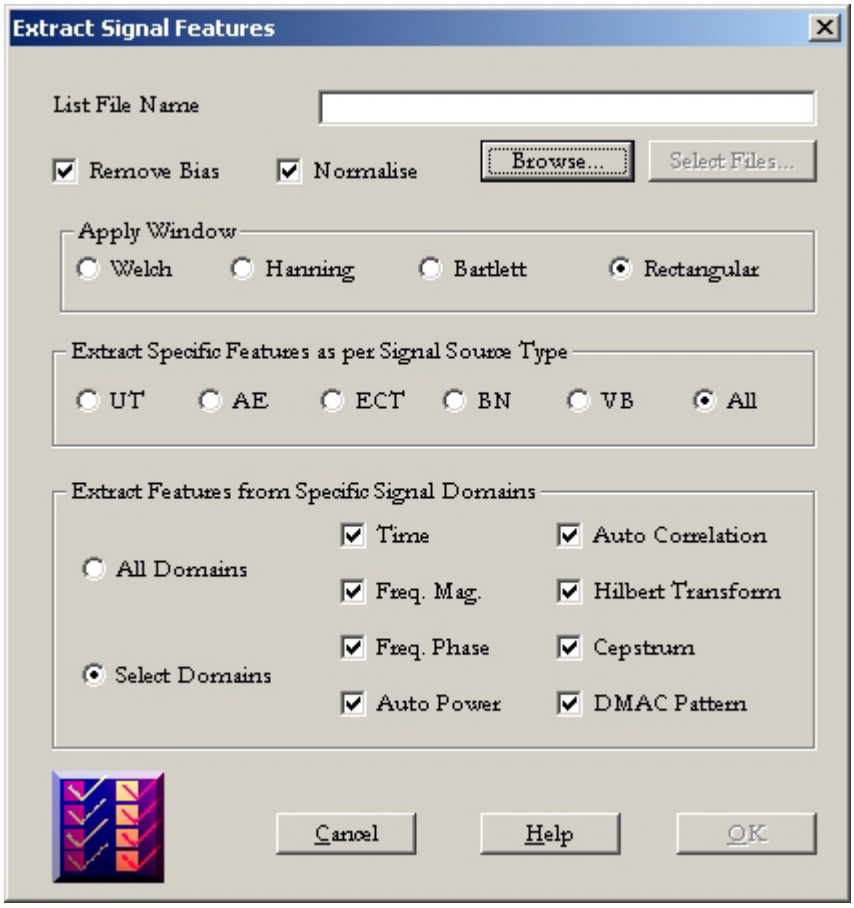
The extraction of features from a column-vector of data (signal) or an image is nothing but the process of "measuring various properties" of the data / image. There are no limits to the number of properties (or features) that can be measured. The following Code Listing 4.15 shows a typical set of properties that are measured for a 1-d bipolar

signal. There is however, an important aspect that one needs to remember while extracting features. The feature extraction algorithm or method, as one shown in the following Code Listing 4.15, extracts these features irrespective of the domain to which the input column-vector data belongs. For example, if a property, say, the "distance" between the first two positive peaks in a column-vector data is measured; this property is just a "number" until the domain of the input data is factored in. If the input data belongs to the time domain this "number" would mean something; and if the input data is from the frequency domain, say, a Fourier transform, then this "number" would mean a physical property that is entirely different! Hence it is very essential to understand the domain of discourse, while both extracting and interpreting features.

The second important aspect is the selection of right features and ranking them. As we saw earlier, there is no limit on the number of features or properties that can be extracted from a given data record. Selection of right features then means that to consider those features, for which a physical interpretation is both possible and plausible. Yet, there are situations where one would be forced to use features that may not have a direct, established physical meaning. Among the selected features, it would be prudent to use a minimal set of features for either classification or prediction. It is not true that the accuracy of classification or prediction improves with increase in the number of features used. In fact, for every classification problem, there are a set of features which are most effective, a set whose numbers are usually much less than the total set of features extracted. One of the most popular algorithms used for ranking features (for their effectiveness) for a given classification problem, is the Fisher feature ranking criteria.

Once the most effective subset of features is identified, these fed to either a classification or prediction algorithm for solving respective tasks.

The following dialog shows the various options available while extracting features from a set of signals (1-d column vectors):



Dialog 4.1 Dialog that shows the Various Input Parameters while extracting features

This dialog, once the User clicks 'OK', feeds the various input parameters to the feature extracting algorithm whose code is shown in Code Listing 4.15.

```
/* The 'C' function that extracts features from a single, bipolar
signal in an array - Begins Here */
void getbipolsigfeadata(float *data, float *features, int r_length)

/************ Features found by this Module *****************

0. First Highest Positive Peak Value - Max 1
1. Second Highest Positive Peak Value - Max 2
2. Third Highest Positive Peak Value - Max 3
3. First Highest Negative Peak Value - Min 1
4. Second Highest Negative Peak Value - Min 2
5. Third Highest Negative Peak Value - Min 3
6. Inter-peak Distance Between Max 1 and Max 2
7. Inter-peak Distance Between Max 2 and Max 3
8. Inter-peak Distance Between Min 1 and Min 2
9. Inter-peak Distance Between Min 2 and Min 3
10.First Peak's P-P Value
11.Second Peak's P-P Value
12.Third Peak's P-P Value
```

```
13.Ratio of I to II P-P Values
14.Ratio of II to III P-P Values
15.Ratio of I to III P-P Values
16.Total Spectral Energy (TSE)
17.Energy above 10% of Max. Peak
18.Energy above 20% of Max. Peak
19.Energy above 30% of Max. Peak
20.Energy above 50% of Max. Peak
21.Ratio of TSE to features[17]
22.Ratio of TSE to features[18]
23.Ratio of TSE to features[19]
24.Ratio of TSE to features[20]
25.Net Positive Area
26.Net Negative Area
27.Ratio of Positive to Negative Area
28.Ratio of Positive to Total Energy
29.Ratio of Negative to Total Energy
30.Energy in the first 1/8th Time Slot
31.Energy in the second 1/8th Time Slot
32.Energy in the third 1/8th Time Slot
33.Energy in the fourth 1/8th Time Slot
34.Energy in the fifth 1/8th Time Slot
35.Energy in the sixth 1/8th Time Slot
36.Energy in the seventh 1/8th Time Slot
37.Energy in the eighth 1/8th Time Slot
38.Ratio of first Slot to Total Energy
39.Ratio of second Slot to Total Energy
40.Ratio of third Slot to Total Energy
41.Ratio of fourth Slot to Total Energy
42.Ratio of fifth Slot to Total Energy
43.Ratio of sixth Slot to Total Energy
44.Ratio of seventh Slot to Total Energy
45.Ratio of eighth Slot to Total Energy
46.Number of Zero Crossings in the Signal
47.Ratio of Inter-Positive-Peak Distances
48.Ratio of Inter-Negative-Peak Distances
49.Ratio of Inter-Positive-Peak (I-II) Distance to the Record Length
50.Ratio of Inter-Positive-Peak (II-III) Distance to the Record
Length
51.Ratio of Inter-Negative_Peak (I-II) Distance to the Record Length
52.Ratio of Inter-Negative-Peak (II-III) Distance to the Record
Length
53.Location of First Maximum Positive Peak
54.Location of Second Maximum Positive Peak
55.Location of Third Maximum Positive Peak
56.Location of First Maximum Negative Peak
57.Location of Second Maximum Negative Peak
58.Location of Third Maximum Negative Peak
59.Rise-Time of the First Maximum Positive Peak
60.Rise-Time of the Second Maximum Positive Peak
61.Rise-Time of the Third Maximum Positive Peak
62.Fall-Time of the First Maximum Positive Peak
63.Fall-Time of the Second Maximum Positive Peak
64.Fall-Time of the Third Maximum Positive Peak
65.Width of the First Maximum Positive Peak Pulse
66.Width of the Second Maximum Positive Peak Pulse
67.Width of the Third Maximum Positive Peak Pulse

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

{
```

```
int i,j,k;
int slot,zerocross,currpol,prevpol;
int maxpos[3], minpos[3];

float maxpeak[3], minpeak[3], slottse[8];
float tse, tenpcmaxpeak, twepcmaxpeak, thipcmaxpeak, fifpcmaxpeak,
netpos, netneg, netposen, netnegen;
float max10pc, max90pc, temp;

/* The following segment finds the first three maximum and minimum
values and positions */

for(i=0; i<3; i++)
   {
   maxpeak[i] = -650000.00;
   minpeak[i] = 650000.00;
   maxpos[i] = -1;
   minpos[i] = -1;
   for(j=0; j<r_length; j++)
      {
      if(data[j]>=maxpeak[i]) {maxpeak[i] = data[j]; maxpos[i] = j;}
      if(data[j]<minpeak[i]) {minpeak[i] = data[j]; minpos[i] = j;}
      }
   data[maxpos[i]] = 0.0;
   data[minpos[i]] = 0.0;
   }

for(i=0; i<3; i++)
   {
   data[maxpos[i]] = maxpeak[i];
   data[minpos[i]] = minpeak[i];
   features[i] = maxpeak[i];        /* First Three Highest Peaks */
   features[i+3] = minpeak[i];            /* First Three Lowest Peaks
*/
   }

features[6]  = fabs(maxpos[1] - maxpos[0]);     /* Inter-peak
Distance Between Max 1 and Max 2 */
features[7]  = fabs(maxpos[2] - maxpos[1]);     /* Inter-peak
Distance Between Max 2 and Max 3 */
features[8]  = fabs(minpos[1] - minpos[0]);     /* Inter-peak
Distance Between Min 1 and Min 2 */
features[9]  = fabs(minpos[2] - minpos[1]);     /* Inter-peak
Distance Between Min 2 and Min 3 */
features[10] = maxpeak[0] - minpeak[0];         /* First Peak's P-P
Value */
features[11] = maxpeak[1] - minpeak[1];         /* Second Peak's P-P
Value */
features[12] = maxpeak[2] - minpeak[2];         /* Third Peak's P-P
Value */
if(features[11]!=0.0) {features[13] = features[10]/features[11];} /*
Ratio of I to II P-P Values */
if(features[12]!=0.0) {features[14] = features[11]/features[12];} /*
Ratio of II to III P-P Values */
if(features[12]!=0.0) {features[15] = features[10]/features[12];} /*
Ratio of I to III P-P Values */

/* The following segment finds the total spectral energy and energies
at various threshold levels */
tenpcmaxpeak = 0.0;
twepcmaxpeak = 0.0;
```

```
thipcmaxpeak = 0.0;
fifpcmaxpeak = 0.0;
tse = 0.0;
netpos = 0.0;
netneg = 0.0;
netposen = 0.0;
netnegen = 0.0;
slot = r_length/8;
currpol = 0;
prevpol = 0;
zerocross = 0;
for(i=0; i<8; i++) {slottse[i] = 0.0;}
for(j=0; j<r_length; j++)
   {
   if(data[j]>=0.0) {currpol = 1; zerocross = zerocross + currpol -
currpol*prevpol; prevpol = 1;}
   if(data[j]<0.0)  {currpol = 0; zerocross = zerocross + prevpol -
currpol*prevpol; prevpol = 0;}
   tse = tse + data[j]*data[j];
   if(j<=slot) {slottse[0] = slottse[0] + data[j]*data[j];}
   if((j>slot) && (j<=slot*2)) {slottse[1] = slottse[1] +
data[j]*data[j];}
   if((j>slot*2) && (j<=slot*3)) {slottse[2] = slottse[2] +
data[j]*data[j];}
   if((j>slot*3) && (j<=slot*4)) {slottse[3] = slottse[3] +
data[j]*data[j];}
   if((j>slot*4) && (j<=slot*5)) {slottse[4] = slottse[4] +
data[j]*data[j];}
   if((j>slot*5) && (j<=slot*6)) {slottse[5] = slottse[5] +
data[j]*data[j];}
   if((j>slot*6) && (j<=slot*7)) {slottse[6] = slottse[6] +
data[j]*data[j];}
   if(j>slot*7) {slottse[7] = slottse[7] + data[j]*data[j];}
   if(fabs(data[j])>(maxpeak[0]*0.1)) {tenpcmaxpeak = tenpcmaxpeak +
data[j]*data[j];}
   if(fabs(data[j])>(maxpeak[0]*0.2)) {twepcmaxpeak = twepcmaxpeak +
data[j]*data[j];}
   if(fabs(data[j])>(maxpeak[0]*0.3)) {thipcmaxpeak = thipcmaxpeak +
data[j]*data[j];}
   if(fabs(data[j])>(maxpeak[0]*0.5)) {fifpcmaxpeak = fifpcmaxpeak +
data[j]*data[j];}
   if(data[j]>=0.0) {netpos = netpos + data[j]; netposen = netposen +
data[j]*data[j];}
   if(data[j]<0.0) {netneg = netneg + fabs(data[j]); netnegen =
netnegen + data[j]*data[j];}
   }
features[16] = tse;           /* Total Spectral Energy */
features[17] = tenpcmaxpeak;  /* Energy above 10% of Max. Peak */
features[18] = twepcmaxpeak;  /* Energy above 20% of Max. Peak */
features[19] = thipcmaxpeak;  /* Energy above 30% of Max. Peak */
features[20] = fifpcmaxpeak;  /* Energy above 50% of Max. Peak */
if(tse!=0.0) {features[21] = tenpcmaxpeak/tse;} /* Ratio of tse to
features[17] */
if(tse!=0.0) {features[22] = twepcmaxpeak/tse;} /* Ratio of tse to
features[18] */
if(tse!=0.0) {features[23] = thipcmaxpeak/tse;} /* Ratio of tse to
features[19] */
if(tse!=0.0) {features[24] = fifpcmaxpeak/tse;} /* Ratio of tse to
features[20] */
features[25] = netpos;        /* Net Positive Area */
features[26] = netneg;        /* Net Negative Area */
```

```
if(netneg!=0.0) {features[27] = netpos/netneg;} /* Ratio of Positive
to Negative Area */
if(tse!=0.0) {features[28] = netposen/tse;}     /* Ratio of Positive
to Total Energy */
if(tse!=0.0) {features[29] = netnegen/tse;}     /* Ratio of Negative
to Total Energy */
features[30] = slottse[0];    /* Energy in the first 1/8th Time Slot
*/
features[31] = slottse[1];    /* Energy in the second 1/8th Time Slot
*/
features[32] = slottse[2];    /* Energy in the third 1/8th Time Slot
*/
features[33] = slottse[3];    /* Energy in the fourth 1/8th Time Slot
*/
features[34] = slottse[4];    /* Energy in the fifth 1/8th Time Slot
*/
features[35] = slottse[5];    /* Energy in the sixth 1/8th Time Slot
*/
features[36] = slottse[6];    /* Energy in the seventh 1/8th Time
Slot */
features[37] = slottse[7];    /* Energy in the eighth 1/8th Time Slot
*/
if(tse!=0.0) {features[38] = slottse[0]/tse;}   /* Ratio of first
Slot to Total Energy */
if(tse!=0.0) {features[39] = slottse[1]/tse;}   /* Ratio of second
Slot to Total Energy */
if(tse!=0.0) {features[40] = slottse[2]/tse;}   /* Ratio of third
Slot to Total Energy */
if(tse!=0.0) {features[41] = slottse[3]/tse;}   /* Ratio of fourth
Slot to Total Energy */
if(tse!=0.0) {features[42] = slottse[4]/tse;}   /* Ratio of fifth
Slot to Total Energy */
if(tse!=0.0) {features[43] = slottse[5]/tse;}   /* Ratio of sixth
Slot to Total Energy */
if(tse!=0.0) {features[44] = slottse[6]/tse;}   /* Ratio of seventh
Slot to Total Energy */
if(tse!=0.0) {features[45] = slottse[7]/tse;}   /* Ratio of eighth
Slot to Total Energy */
features[46] = zerocross;     /* Number of Zero Crossings in the
Signal */
if(features[7]!=0.0) {features[47] = features[6]/features[7];}    /*
Ratio of Inter-Peak Distances */
if(features[9]!=0.0) {features[48] = features[8]/features[9];}    /*
Ratio of Inter-Peak Distances */
features[49] = features[6]/r_length;      /* Ratio of Inter-Peak
Distance to the Record Length */
features[50] = features[7]/r_length;      /* Ratio of Inter-Peak
Distance to the Record Length */
features[51] = features[8]/r_length;      /* Ratio of Inter-Peak
Distance to the Record Length */
features[52] = features[9]/r_length;      /* Ratio of Inter-Peak
Distance to the Record Length */
features[53] = maxpos[0];     /* Location of First Maximum Peak */
features[54] = maxpos[1];     /* Location of Second Maximum Peak */
features[55] = maxpos[2];     /* Location of Third Maximum Peak */
features[56] = minpos[0];     /* Location of First Minimum Peak */
features[57] = minpos[1];     /* Location of Second Minimum Peak */
features[58] = minpos[2];     /* Location of Third Minimum Peak */

i = maxpos[0];
max10pc = maxpeak[0]*0.1;
```

```
max90pc = maxpeak[0]*0.9;
k = 0;
do {
   temp = data[i];
   if(temp<=max90pc) {k = k + 1;}
   i = i - 1;
   } while((temp>max10pc) && (i>=0));
features[59] = k;             /* Rise-Time of the First Maximum Peak
*/

i = maxpos[1];
max10pc = maxpeak[1]*0.1;
max90pc = maxpeak[1]*0.9;
k = 0;
do {
   temp = data[i];
   if(temp<=max90pc) {k = k + 1;}
   i = i - 1;
   } while((temp>max10pc) && (i>=0));
features[60] = k;             /* Rise-Time of the Second Maximum Peak
*/

i = maxpos[2];
max10pc = maxpeak[2]*0.1;
max90pc = maxpeak[2]*0.9;
k = 0;
do {
   temp = data[i];
   if(temp<=max90pc) {k = k + 1;}
   i = i - 1;
   } while((temp>max10pc) && (i>=0));
features[61] = k;             /* Rise-Time of the Third Maximum Peak
*/

i = maxpos[0];
max10pc = maxpeak[0]*0.1;
max90pc = maxpeak[0]*0.9;
k = 0;
do {
   temp = data[i];
   if(temp<=max90pc) {k = k + 1;}
   i = i + 1;
   } while((temp>max10pc) && (i<=r_length));
features[62] = k*1.0;              /* Fall-Time of the First Maximum
Peak */

i = maxpos[1];
max10pc = maxpeak[1]*0.1;
max90pc = maxpeak[1]*0.9;
k = 0;
do {
   temp = data[i];
   if(temp<=max90pc) {k = k + 1;}
   i = i + 1;
   } while((temp>max10pc) && (i<=r_length));
features[63] = k*1.0;              /* Fall-Time of the Second
Maximum Peak */

i = maxpos[2];
max10pc = maxpeak[2]*0.1;
max90pc = maxpeak[2]*0.9;
```

```
k = 0;
do {
   temp = data[i];
   if(temp<=max90pc) {k = k + 1;}
   i = i + 1;
   } while((temp>max10pc) && (i<=r_length));
features[64] = k*1.0;              /* Fall-Time of the Third Maximum
Peak */

features[65] = features[59] + features[62];     /* Width of the First
Maximum Peak Pulse */
features[66] = features[60] + features[63];     /* Width of the
Second Maximum Peak Pulse */
features[67] = features[61] + features[64];     /* Width of the Third
Maximum Peak Pulse */

}
/* The 'C' function that extracts features from a single, bipolar
signal in an array - Ends Here */
```

Code Listing 4.15 Feature Extraction Module for a single, bi-polar signal

Code Listing 4.16 (a-c) show the extraction of 52 invariant moments from a given image. Invariant moments are very effective in characterising an image, since these properties are translation, rotation and magnification independent, and hence truly characterise the image.

While using the invariant features, one must remember that these features are "global" features of the image. That is, these features do not capture the local variations within two images so very effectively. For example, if we have two images having a number of smaller objects within them, one set of objects having smaller radii, then, the invariant features of these two images may not be distinct enough for classification. In such cases, one must use object analysis functions that truly characterise the objects present inside an image. If the objective is the global classification of images (for example, images having different textures), then these invariant features can be fed to a classification algorithm directly.

```
void imoments(char TD[128], char IN[128], char LS[128], char FI[128],
int size, char OPID[128], char LF[128], int *cerr)
{
float *x;                        /* Storage for Pixel Values Read from
a File */
float **xv;                        /* Stores the pixel values for ONE
file at a time */
float *xmean, *ymean;            /* Stores the Mean X and Y co-orinate
values, for various
                                    rows  and  columns,  in  floating
point */
float *pvalue;                   /* Pixel Value holder for a Column or
Row */
double meanx, meany;              /* Stores the actual mean values of
'x' and 'y' co-ords. */
int i_nodes;                        /* Total number of nodes in the
Hopfield NN = size*size */
```

```
int n_files;                  /*   Number   of   files   listed   in
filename.lst */
int index, skip, count;         /* Just Counters ! */
char f_name[128];       /* Name of the individual files */
char fnames[128];       /* Name of the individual files */
char noname[128] = "";          /* Empty String */
char ffname[128] = "";          /* Name of the file that contains the
features */
char xyname[128] = "";        /*  Name  of  the  file  that  stores  the
names of the 'xyval*.fea' filename */
char xyfid[128] = "";         /*  Name  of  the  File  that  holds  the
Feature ID Values */
char exten1[] = "DAT";        /* File extension */
char exten2[] = "FEA.LST";    /* File extension */
char exten3[] = ".FEA";         /* File extension */
char exten4[] = "FEA.FID";    /* File extension */
char buffer[5];                       /* Temporary Storage space for
current Session Number */
float temp;                         /* Temporary floating-point value
holder */
double mu[10][10];            /* Stores moments of m[p][q] */
double a[12][12],b[12][12];   /*  Stores  the  intermediary  values  of
the invariant moments */
double m[53];                   /* Stores the final invariant moments
*/
int j,k,l,p,q;                /* Just Counters */
int sl, opl, tdl;               /* Length of a string */
double p10,p01;
float maxpos, maxneg, range,pp,qq;

FILE *fpi;
FILE *fpf;
FILE *fpt;
FILE *fp;
FILE *fpl;

if((fpl=fopen(LF,"a")) == NULL) {*cerr = 43; exit(1);}
fprintf(fpl,"%s\n","Opened   Log   file   from   within   'imoments'   C
Module");
fprintf(fpl,"%s\n","Ready to Execute the Program");
fclose(fpl);

tdl = strlen(TD);
sl = strlen(LS);
sl = sl - 7;
strncat(xyfid,LS,sl);
strncat(xyname,LS,sl);
strncat(xyfid,exten4,7);
strncat(xyname,exten2,7);

if((fpl=fopen(LF,"a")) == NULL) {*cerr = 43; exit(1);}
fprintf(fpl,"%s\n","This Segment is Image Feature Extraction");
fprintf(fpl,"%s\n","The  following  pairs  of  XY  files  /  images  were
used in this Session");
fclose(fpl);

n_files = getsflength(LS);
i_nodes = size * size;

/* Memory Allocation for various arrays */
x=vector(0,i_nodes-1);
```

```
xv=matrix(0,size-1,0,size-1);
xmean=vector(0,size-1);
ymean=vector(0,size-1);
pvalue=vector(0,size-1);
/* End of Memory Allocation */

if((fpl=fopen(LF,"a")) == NULL) {*cerr = 43; exit(1);}
fprintf(fpl,"%s%d\n","Size = ",size);
fprintf(fpl,"%s%s\n","Input Filename.lst = ",FI);
fprintf(fpl,"%s%d\n","Total Number of Files = ",n_files);
fprintf(fpl,"%s%d\n","Total Number of Nodes = ",i_nodes);
fprintf(fpl,"%s\n","The following (*.bmp or XY) files were used, to
get the feature values (*.fea) :");
fprintf(fpl,"%s%s\n","These  *.FEA  files  are  listed  for  further
processing in ",xyname);
fclose(fpl);

fp  = fopen(IN,"r");
fpi = fopen(FI,"r");
fpt = fopen(xyname,"w");
for(index=0; index<n_files; index++)      /* index is the counter for
the file number */
   {  /* step is used as the file counter in the later parts of the
Program */
   fscanf(fpi,"%s",&f_name);
   fscanf(fp,"%s",&fnames);
   sl = strlen(fnames);
   sl = sl - 4;
   strcpy(ffname,noname);
   strncat(ffname,TD,tdl);
   strncat(ffname,fnames,sl);
   strncat(ffname,OPID,opl);
   strncat(ffname,exten3,4);
   if((fpl=fopen(LF,"a")) == NULL) {*cerr = 43; exit(1);}
   fprintf(fpl,"%s %c%c%c %s\n",fnames,'-','-','>',ffname);
   fclose(fpl);
   if(index!=n_files-1) {fprintf(fpt,"%s\n",ffname);}
   else {fprintf(fpt,"%s",ffname);}
   fpf = fopen(f_name,"r");

   maxpos = 0.0;
   maxneg = 0.0;

   for(count=0; count<i_nodes; count++)   /* count is the counter for
the data points in a file */
      { /* index is the counter used for referring to data points
later in the Program */
      fscanf(fpf,"%d",&skip);
      x[count] = skip;
      if(skip>maxpos) {maxpos = skip;}
      }

   range  = maxpos - maxneg;

   for(count=0; count<i_nodes; count++)   /* count is the counter for
the data points in a file */
      { /* index is the counter used for referring to data points
later in the Program */
      k=(count%size);
      if(count<=size-1) {l = 0;}
      if(count>size-1) {l = (count-k)/size;}
```

```
   xv[l][k] = (maxpos - x[count])/range;
   }
fclose(fpf);

   mu[0][0] = moment(0.0,0.0,size,0.0,0.0,xv);
   mu[0][1] = moment(0.0,1.0,size,0.0,0.0,xv);
   mu[1][0] = moment(1.0,0.0,size,0.0,0.0,xv);

   meanx = mu[1][0]/mu[0][0];
   meany = mu[0][1]/mu[0][0];

   p10 = moment(1.0,0.0,size,meanx,meany,xv);
   p01 = moment(0.0,1.0,size,meanx,meany,xv);

   for(p=0; p<10; p++)
      {
      for(q=0; q<10; q++)
         {
         if((p+q)>1 && (p+q)<10)
           {
           pp = p;
           qq = q;
           mu[p][q] = moment(pp,qq,size,meanx,meany,xv);
           mu[p][q] = norm(pp,qq,mu[p][q],mu[0][0]);
           }
         }
      }

   for(p=0; p<12; p++)
      {
      for(q=0; q<12; q++)
         {
         a[p][q] = 0.0;
         b[p][q] = 0.0;
         }
      }

   /* Intermediate Values are Calculated here */

   a[4][0] = mu[2][0] + mu[0][2];
   a[4][2] = mu[2][0] - mu[0][2];
   b[4][2] = 2.0*mu[1][1];
   a[5][1] = mu[3][0] + mu[1][2];
   b[5][1] = mu[2][1] + mu[0][3];
   a[5][3] = mu[3][0] - 3.0*mu[1][2];
   b[5][3] = 3.0*mu[2][1] - mu[0][3];
   a[6][0] = mu[4][0] + 2.0*mu[2][2] + mu[0][4];
   a[6][2] = mu[4][0] - mu[0][4];
   b[6][2] = 2.0*(mu[3][1] + mu[1][3]);
   a[6][4] = mu[4][0] - 6.0*mu[2][2] + mu[0][4];
   b[6][4] = 4.0*(mu[3][1] - mu[1][3]);
   a[7][1] = mu[5][0] + 2.0*mu[3][2] + mu[1][4];
   b[7][1] = mu[4][1] + 2.0*mu[2][3] + mu[0][5];
   a[7][3] = mu[5][0] - 2.0*mu[3][2] - 3.0*mu[1][4];
   b[7][3] = 3.0*mu[4][1] + 2.0*mu[2][3] - mu[0][5];
   a[7][5] = mu[5][0] - 10.0*mu[3][2] + 5.0*mu[1][4];
   b[7][5] = 5.0*mu[4][1] - 10.0*mu[2][3] + mu[0][5];
   a[8][0] = mu[6][0] + 3.0*mu[4][2] + 3.0*mu[2][4] + mu[0][6];
   a[8][2] = mu[6][0] + mu[4][2] - mu[2][4] - mu[0][6];
   b[8][2] = 2.0*(mu[5][1] + 2.0*mu[3][3] + mu[1][5]);
   a[8][4] = mu[6][0] - 5.0*mu[4][2] - 5.0*mu[2][4] + mu[0][6];
```

```
      b[8][4] = 4.0*(mu[5][1] - mu[1][5]);
      a[8][6] = mu[6][0] - 15.0*mu[4][2] + 15.0*mu[2][4] - mu[0][6];
      b[8][6] = 2.0*(3.0*mu[5][1] - 10.0*mu[3][3] + 3.0*mu[1][5]);
      a[9][1] = mu[7][0] + 3.0*mu[5][2] + 3.0*mu[3][4] + mu[1][6];
      b[9][1] = mu[6][1] + 3.0*mu[4][3] + 3.0*mu[2][5] + mu[0][7];
      a[9][3] = mu[7][0] - mu[5][2] - 5.0*mu[3][4] - 3.0*mu[1][6];
      b[9][3] = 3.0*mu[6][1] + 5.0*mu[4][3] + mu[2][5] - mu[0][7];
      a[9][5]  =  mu[7][0]  -  9.0*mu[5][2]  -  5.0*mu[3][4]  +
5.0*mu[1][6];
      b[9][5]  =  5.0*mu[6][1]  -  5.0*mu[4][3]  -  9.0*mu[2][5]  +
mu[0][7];
      a[9][7]  =  mu[7][0]  -  21.0*mu[5][2]  +  35.0*mu[3][4]  -
7.0*mu[1][6];
      b[9][7]  =  7.0*mu[6][1]  -  35.0*mu[4][3]  +  21.0*mu[2][5]  -
mu[0][7];
     a[10][0] = mu[8][0] + 4.0*mu[6][2] + 6.0*mu[4][4] + 4.0*mu[2][6]
+ mu[0][8];
     a[10][2] = mu[8][0] + 2.0*mu[6][2] - 2.0*mu[2][6] - mu[0][8];
     b[10][2]  =  2.0*(mu[7][1]  +  3.0*mu[5][3]  +  3.0*mu[3][5]  +
mu[1][7]);
     a[10][4]  =  mu[8][0]  -  4.0*mu[6][2]  -  10.0*mu[4][4]  -
4.0*mu[2][6] + mu[0][8];
     b[10][4] = 4.0*(mu[7][1] + mu[5][3] - mu[3][5] - mu[1][7]);
     a[10][6] = mu[8][0] - 14.0*mu[6][2] + 14.0*mu[2][6] - mu[0][8];
     b[10][6]  =  2.0*(3.0*mu[7][1]  -  7.0*mu[5][3]  -  7.0*mu[3][5]  +
3.0*mu[1][7]);
     a[10][8]  =  mu[8][0]  -  28.0*mu[6][2]  +  70.0*mu[4][4]  -
28.0*mu[2][6] + mu[0][8];
     b[10][8]  =  8.0*(mu[7][1]  -  7.0*mu[5][3]  +  7.0*mu[3][5]  -
mu[1][7]);
     a[11][1] = mu[9][0] + 4.0*mu[7][2] + 6.0*mu[5][4] + 4.0*mu[3][6]
+ mu[1][8];
     b[11][1] = mu[8][1] + 4.0*mu[6][3] + 6.0*mu[4][5] + 4.0*mu[2][7]
+ mu[0][9];
     a[11][3]  =  mu[9][0]  -  6.0*mu[5][4]  -  8.0*mu[3][6]  -
3.0*mu[1][8];
     b[11][3]  =  3.0*mu[8][1]  +  8.0*mu[6][3]  +  6.0*mu[4][5]  -
mu[0][9];
     a[11][5]  =  mu[9][0]  -  8.0*mu[7][2]  -  14.0*mu[5][4]  +
5.0*mu[1][8];
     b[11][5]  =  5.0*mu[8][1]  -  14.0*mu[4][5]  -  8.0*mu[2][7]  +
mu[0][9];
     a[11][7]  =  mu[9][0]  -  20.0*mu[7][2]  +  14.0*mu[5][4]  +
28.0*mu[3][6] - 7.0*mu[1][8];
     b[11][7]  =  7.0*mu[8][1]  -  28.0*mu[6][3]  -  14.0*mu[4][5]  +
20.0*mu[2][7] - mu[0][9];
     a[11][9]  =  mu[9][0]  -  36.0*mu[7][2]  +  126.0*mu[5][4]  -
84.0*mu[3][6] + 9.0*mu[1][8];
     b[11][9]  =  9.0*mu[8][1]  -  84.0*mu[6][3]  +  126.0*mu[4][5]  -
36.0*mu[2][7] + mu[0][9];


     /* Final Invariant Moments are Calculated here */

     m[1] = a[4][0];
     m[2] = pow(a[4][2],2.0) + pow(b[4][2],2.0);
     m[3]  =  a[4][2]*(pow(a[5][1],2.0)  -  pow(b[5][1],2.0))  +
2.0*a[5][1]*b[5][1]*b[4][2];
     m[4] = pow(a[5][1],2.0) + pow(b[5][1],2.0);
     m[5] = pow(a[5][3],2.0) + pow(b[5][3],2.0);
```

```
     m[6] = a[5][1]*a[5][3]*(pow(a[5][1],2.0) - 3.0*pow(b[5][1],2.0))
+ b[5][1]*b[5][3]*(3.0*pow(a[5][1],2.0) - pow(b[5][1],2.0));
     m[7] = a[5][1]*b[5][3]*(pow(a[5][1],2.0) - 3.0*pow(b[5][1],2.0))
- b[5][1]*a[5][3]*(3.0*pow(a[5][1],2.0) - pow(b[5][1],2.0));
     m[8] = a[6][0];
     m[9] = pow(a[6][2],2.0) + pow(b[6][2],2.0);
    m[10] = pow(a[6][4],2.0) + pow(b[6][4],2.0);
    m[11]   =   a[6][4]*(pow(a[6][2],2.0)   -   pow(b[6][2],2.0))   +
2.0*a[6][2]*b[6][2]*b[6][4];
    m[12]   =   b[6][4]*(pow(a[6][2],2.0)   -   pow(b[6][2],2.0))   -
2.0*a[6][2]*b[6][2]*a[6][4];
    m[13]   =   a[6][2]*(pow(a[7][1],2.0)   -   pow(b[7][1],2.0))   +
2.0*a[7][1]*b[7][1]*b[6][2];
    m[14] = pow(a[7][1],2.0) + pow(b[7][1],2.0);
    m[15] = pow(a[7][3],2.0) + pow(b[7][3],2.0);
    m[16] = pow(a[7][5],2.0) + pow(b[7][5],2.0);
    m[17] = a[7][1]*a[7][3]*(pow(a[7][1],2.0) - 3.0*pow(b[7][1],2.0))
+ b[7][1]*b[7][3]*(3.0*pow(a[7][1],2.0) - pow(b[7][1],2.0));
    m[18] = a[7][1]*b[7][3]*(pow(a[7][1],2.0) - 3.0*pow(b[7][1],2.0))
- b[7][1]*a[7][3]*(3.0*pow(a[7][1],2.0) - pow(b[7][1],2.0));
    m[19] = a[8][0];
    m[20] = pow(a[8][2],2.0) + pow(b[8][2],2.0);
    m[21] = pow(a[8][4],2.0) + pow(b[8][4],2.0);
    m[22] = pow(a[8][6],2.0) + pow(b[8][6],2.0);
    m[23]   =   a[8][4]*(pow(a[8][2],2.0)   -   pow(b[8][2],2.0))   +
2.0*a[8][2]*b[8][2]*b[8][4];
    m[24]   =   b[8][4]*(pow(a[8][2],2.0)   -   pow(b[8][2],2.0))   -
2.0*a[8][2]*b[8][2]*a[8][4];
    m[25]   =   a[8][2]*(pow(a[9][1],2.0)   -   pow(b[9][1],2.0))   +
2.0*a[9][1]*b[9][1]*b[8][2];
    m[26] = pow(a[9][1],2.0) + pow(b[9][1],2.0);
    m[27] = pow(a[9][3],2.0) + pow(b[9][3],2.0);
    m[28] = pow(a[9][5],2.0) + pow(b[9][5],2.0);
    m[29] = pow(a[9][7],2.0) + pow(b[9][7],2.0);
    m[30] = a[9][1]*a[9][3]*(pow(a[9][1],2.0) - 3.0*pow(b[9][1],2.0))
+ b[9][1]*b[9][3]*(3.0*pow(a[9][1],2.0) - pow(b[9][1],2.0));
    m[31] = a[9][1]*b[9][3]*(pow(a[9][1],2.0) - 3.0*pow(b[9][1],2.0))
- b[9][1]*a[9][3]*(3.0*pow(a[9][1],2.0) - pow(b[9][1],2.0));
    m[32]         =         a[9][1]*a[9][5]*(pow(a[9][1],4.0)         -
10.0*pow(a[9][1],2.0)*pow(b[9][1],2.0)   +   5.0*pow(b[9][1],4.0))   +
b[9][1]*b[9][5]*(5.0*pow(a[9][1],4.0)                                 -
10.0*pow(a[9][1],2.0)*pow(b[9][1],2.0) + pow(b[9][1],4.0));
    m[33]         =         a[9][1]*b[9][5]*(pow(a[9][1],4.0)         -
10.0*pow(a[9][1],2.0)*pow(b[9][1],2.0)   +   5.0*pow(b[9][1],4.0))   +
b[9][1]*a[9][5]*(5.0*pow(a[9][1],4.0)                                 -
10.0*pow(a[9][1],2.0)*pow(b[9][1],2.0) + pow(b[9][1],4.0));
    m[34] = a[10][0];
    m[35] = pow(a[10][2],2.0) + pow(b[10][2],2.0);
    m[36] = pow(a[10][4],2.0) + pow(b[10][4],2.0);
    m[37] = pow(a[10][6],2.0) + pow(b[10][6],2.0);
    m[38] = pow(a[10][8],2.0) + pow(b[10][8],2.0);
    m[39]   =   a[10][4]*(pow(a[10][2],2.0)   -   pow(b[10][2],2.0))   +
2.0*a[10][2]*b[10][2]*b[10][4];
    m[40]   =   b[10][4]*(pow(a[10][2],2.0)   -   pow(b[10][2],2.0))   -
2.0*a[10][2]*b[10][2]*a[10][4];
    m[41]   =   a[10][8]*(pow(a[10][4],2.0)   -   pow(b[10][4],2.0))   +
2.0*a[10][4]*b[10][4]*b[10][8];
    m[42]   =   b[10][8]*(pow(a[10][4],2.0)   -   pow(b[10][4],2.0))   -
2.0*a[10][4]*b[10][4]*a[10][8];
    m[43]   =   a[10][2]*(pow(a[11][1],2.0)   -   pow(b[11][1],2.0))   +
2.0*a[11][1]*b[11][1]*b[10][2];
```

```
    m[44] = pow(a[11][1],2.0) + pow(b[11][1],2.0);
    m[45] = pow(a[11][3],2.0) + pow(b[11][3],2.0);
    m[46] = pow(a[11][5],2.0) + pow(b[11][5],2.0);
    m[47] = pow(a[11][7],2.0) + pow(b[11][7],2.0);
    m[48] = pow(a[11][9],2.0) + pow(b[11][9],2.0);
    m[49]       =       a[11][1]*a[11][3]*(pow(a[11][1],2.0)       -
3.0*pow(b[11][1],2.0))  +  b[11][1]*b[11][3]*(3.0*pow(a[11][1],2.0)  -
pow(b[11][1],2.0));
    m[50]       =       a[11][1]*b[11][3]*(pow(a[11][1],2.0)       -
3.0*pow(b[11][1],2.0))  -  b[11][1]*a[11][3]*(3.0*pow(a[11][1],2.0)  -
pow(b[11][1],2.0));
    m[51]       =       a[11][1]*a[11][5]*(pow(a[11][1],4.0)       -
10.0*pow(a[11][1],2.0)*pow(b[11][1],2.0)  +  5.0*pow(b[11][1],4.0))  +
b[11][1]*b[11][5]*(5.0*pow(a[11][1],4.0)                            -
10.0*pow(a[11][1],2.0)*pow(b[11][1],2.0) + pow(b[11][1],4.0));
    m[52]       =       a[11][1]*b[11][5]*(pow(a[11][1],4.0)       -
10.0*pow(a[11][1],2.0)*pow(b[11][1],2.0)  +  5.0*pow(b[11][1],4.0))  -
b[11][1]*a[11][5]*(5.0*pow(a[11][1],4.0)                            -
10.0*pow(a[11][1],2.0)*pow(b[11][1],2.0) + pow(b[11][1],4.0));


      remove(f_name);
      fpf = fopen(ffname,"w");
      for(j=1; j<53; j++)
      {
      if(j!=52) {fprintf(fpf,"%f\n",m[j]);}
      else {fprintf(fpf,"%f",m[j]);}
      }
      fclose(fpf);

   } /* This bracket closes operation of the file counter index,
i.e., end of processing
       for one file */


/* The following segment registers the feature ID values in the
corresponding file */
fpf = fopen(xyfid,"w");
for(j=1; j<53; j++)
   {
   if(j!=52) {fprintf(fpf,"%d\n",j+1000);}
   else {fprintf(fpf,"%d",j+1000);}
   }
fclose(fpf);
/*  The  above  segment  registers  the  feature  ID  values  in  the
corresponding file */

fclose(fpi);
fclose(fpt);
fclose(fp);

/* Freeing Memory Allocated for Various Arrays - Begin */
free_vector(x,0,i_nodes-1);
free_matrix(xv,0,size-1,0,size-1);
free_vector(xmean,0,size-1);
free_vector(ymean,0,size-1);
free_vector(pvalue,0,size-1);
/* Freeing Memory Allocated for Various Arrays - End */

if((fpl=fopen(LF,"a")) == NULL) {*cerr = 43; exit(1);}
fprintf(fpl,"%s\n","Completed 'imoments' C Module");
```

```
fclose(fpl);
*cerr = 0;
} /* End of the function imoments */
```

Code Listing 4.16a To find the 52 invariant moments of an image

```
double moment(float p, float q, int size, double meanx, double meany,
/*float totpix,*/ float **xv)
{
int m,n;
double mf,nf;
double dx, dy;
double mpq;

mpq = 0.0;

for(m=0; m<size; m++)
   {
   for(n=0; n<size; n++)
      {
      mf = m;
      nf = n;
      dx = mf - meanx;
      dy = nf - meany;
      mpq = mpq + pow(dx,p)*pow(dy,q)*xv[m][n];
      }
   }
return mpq;
}
```

Code Listing 4.16b To find the moment (a sub-function of imoments)

```
double norm(float p, float q, double m, double b)
{
double tmp1;
double tmp2;

tmp1 = ((p+q)/2.0) + 1.0;
tmp2 = pow(b,tmp1);
return m/tmp2;
}
```

Code Listing 4.16c To find the norm (a sub-function of imoments)

Figure 4.21 (a-c) show three different regions in a weld, and their corresponding invariant image features. In each case, only a partial list of features are shown (among the total 52 image invariant features available), just to highlight the differences in their values.

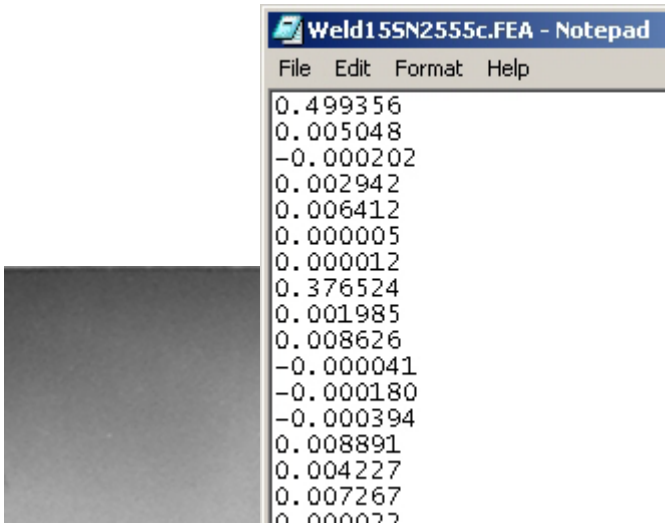


Figure 4.21a Raw Image of a Weld (Weld15.bmp) and some of its Invariant Features

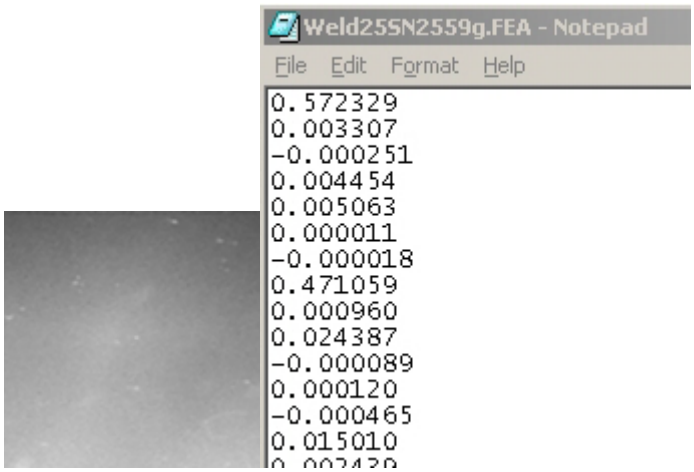


Figure 4.21b Raw Image of a Weld (Weld25.bmp) and some of its Invariant Features

Figure 4.21c Raw Image of a Weld (Weld39.bmp) and some of its Invariant Features

As one can see from Figures 4.21 (a-c), even minor changes in the image under analysis, generates different invariant image features. These image features can then be fed to a classification algorithm for data partitioning.

Among the many classification algorithms we present the feed-forward, error back-propagation multi-layered perceptron method below. The reader would be best advised to see the Frequently Asked Questions for this Neural Network, posted in the web, for up to date, authentic information.

The corresponding code for this algorithm is shown in the Code Listing 4.17.

```
/* Implementation of Backpropagation NN */
void backprop(int mode, int nnm, char FI[128], char WF[128], int
i_nodes, int h_layers, int o_nodes, int h1n, int h2n, int h3n, int
h4n, int h5n, int h6n, int rate, int max_epoch, int dcbr, int norm,
int sesnum, char LF[128], int *cerr)
{
int index,count,step,loop;    /* Just counters ! */
int r_length;                 /* Length of a single record */
int n_files;                  /* Number of files listed in
filename.lst */
char f_name[128];       /* Name of the individual files */
char ocname[13] = "CLAS";     /* Name of the file that stores the
classification result */
char efname[13] = "EVAL";     /* Name of the file that stores the
epoch, patt_err and other results */
char exten[] = ".DAT";  /* File extension */
float sum;        /* This finds the total of each record */
float mean;       /* This finds the mean of each record */
int skip;         /* No. of points to skip after reading a point in
the input file */
float high, low;  /* Stores the max. and min. values among all the
files */
float fact;       /* Used in normalisation and linear mapping */
float learnrate,lrate;  /* Variable for holding the learning rate */
float **rawdata;        /* This holds the raw signal data (initially)
*/
float ***wt;            /* Stores the weights between nodes of any
two successive layers */
float ***nw;            /* Stores the weights when patt_err is the
lowest */
float **ot;             /* Stores the output of a given node in a
given layer */
float **ft;             /* Stores the factored output (x*(1-x)) of a
given node in a given layer */
float **et;             /* Stores the error of a given node in of a
given layer */
int *des_out;           /* Desired Output Value (Class Number) for
each file - Classification */
float *des;       /* Desired Output Value (Class Number) for each
file - Prediction */
int *h_nodes;           /* Number of nodes in the given layer; +2:
one for i/p and one for o/p layer */

float temp;       /* A temporary storage variable ! */
float flash;            /* A temporary storage variable ! */
int epoch = 0;          /* Stores the Epoch Number */
```

```
float patt_err;         /* Pattern Error (Desrd. O/P - Actual O/P) */
float patt_low;         /* Stores the lowest pattern error */
char buffer[5];         /* Temporary Storage space for current
Session Number */
int hln[6];       /* Temporary Storage for No. of Nodes in Hidden
Layers */
float fepoch, fmax_epoch; /* The floating point equivalents of epoch
and max_epoch */

FILE *fpi;
FILE *fpw;
FILE *fpf;
FILE *fpt;
FILE *fpl;

if((fpl=fopen(LF,"a")) == NULL) {*cerr = 40; exit(1);}
fprintf(fpl,"%s\n","Opened Log file from within 'backprop' C
Module");
fprintf(fpl,"%s\n","Ready to Execute the Program");

n_files = getsflength(FI); /* Get the length of the file FI */


fpi = fopen(FI,"r");
fscanf(fpi,"%s",&f_name);
fclose(fpi);
r_length = getfflength(f_name); /* Get the length of the file f_name
*/


/*

Dynamic Memory Allocation Begins Here */
rawdata=matrix(0,n_files-1,0,i_nodes-1);        /* Holds the raw
signal */
wt=f3tensor(0,h_layers+2,0,i_nodes-1,0,i_nodes);      /* Stores the
weights between nodes of any two successive layers */
nw=f3tensor(0,h_layers+2,0,i_nodes-1,0,i_nodes);      /* Stores the
weights when patt_err is the lowest */
ot=matrix(0,h_layers+2,0,i_nodes);                    /* Stores the
output of a given node in a given layer */
ft=matrix(0,h_layers+2,0,i_nodes-1);                  /* Stores the
factored output (x*(1-x)) of a given node in a given layer */
et=matrix(0,h_layers+2,0,i_nodes);                    /* Stores the
error of a given node in of a given layer */

if(nnm == 53) {des_out=ivector(0,n_files-1);}                     /*
Desired Output Value (Class Number) for each file */
if(nnm == 54) {fprintf(fpl,"%s\n","Assigning Memory for desired
values..."); des=vector(0,n_files-1);}

h_nodes=ivector(0,h_layers+2);                        /* Number of
nodes in the given layer; +2: one for i/p and one for o/p layer */
/* End of Dynamic Memory Allocation */

temp = sesnum;
_gcvt(temp,4,buffer);
strncat(ocname,buffer,4);
strncat(efname,buffer,4);
strncat(ocname,exten,4);
strncat(efname,exten,4);
```

```
skip = r_length/i_nodes;
h_nodes[0] = i_nodes;
hln[1] = h1n;
hln[2] = h2n;
hln[3] = h3n;
hln[4] = h4n;
hln[5] = h5n;
hln[6] = h6n;
h_nodes[h_layers+1] = o_nodes;
learnrate = rate/10.0;

for (step=1;step<=h_layers; step++)
    {
    h_nodes[step] = hln[step];
    }

fprintf(fpl,"%s\n","This Segment is BackProp Neural Network");
fprintf(fpl,"%s%d\n","Skip = ",skip);
fprintf(fpl,"%s%d\n","Input Nodes = ",i_nodes);
fprintf(fpl,"%s%d\n","Hidden Layers = ",h_layers);
fprintf(fpl,"%s%d\n","Output Nodes ",o_nodes);
fprintf(fpl,"%s%d\n","Nodes in Hidden Layer 1 = ",h1n);
fprintf(fpl,"%s%d\n","Nodes in Hidden Layer 2 = ",h2n);
fprintf(fpl,"%s%d\n","Nodes in Hidden Layer 3 = ",h3n);
fprintf(fpl,"%s%d\n","Nodes in Hidden Layer 4 = ",h4n);
fprintf(fpl,"%s%d\n","Nodes in Hidden Layer 5 = ",h5n);
fprintf(fpl,"%s%d\n","Nodes in Hidden Layer 6 = ",h6n);
fprintf(fpl,"%s%d\n","D.C.Bias Removal = ",dcbr);
fprintf(fpl,"%s%d\n","Normalisation = ",norm);
fprintf(fpl,"%s%s\n","Input Filename.lst = ",FI);
fprintf(fpl,"%s%d\n","Total Number of Files = ",n_files);
fprintf(fpl,"%s%d\n","Record Length of Each Signal = ",r_length);
fprintf(fpl,"%s%d\n","Mode (Train / Test) = ",mode);
fprintf(fpl,"%s%f%s\n","Learning Rate = ",learnrate," (Decreases with
Epoch)");
fprintf(fpl,"%s%s\n","Weight Values Used / Stored = ",WF);
fprintf(fpl,"%s\n","The following files were used in this Session:");

fpi = fopen(FI,"r");
for(index=0; index<n_files; index++)      /* index is the counter for
the file number */
   {  /* step is used as the file counter in the later parts of the
Program */
   fscanf(fpi,"%s",&f_name);
   fprintf(fpl,"%s\n",f_name);
   fpf = fopen(f_name,"r");
   for(count=0; count<i_nodes; count++)   /* count is the counter for
the data points in a file */
      { /* index is the counter used for referring to data points
later in the Program */
      if (skip==1)
       {
       fscanf(fpf,"%f",&rawdata[index][count]);
       }
      else
       {
       fscanf(fpf,"%f",&rawdata[index][count]);
       for(step=1; step<skip; step++)
          {
          fscanf(fpf,"%f",&sum);
```

```
          }
          sum = 0.0;
       }
      }
   fclose(fpf);
   }
fclose(fpi);

if (dcbr == 1)   {removebias(rawdata, n_files, i_nodes);}

if (norm == 1)   {normalise(rawdata, n_files, i_nodes, 0.9, 0.1);}

/* 0.9 & 0.1 represent the upper & lower limits, to which data is
normalised */

switch (mode)  // Start of switch mode - mode 1 is Training and mode
2 is Testing
       {
       case (49):
            { /* Start of switch mode 1 - Start of Backpropagation
Training Phase */

            /*    The following program fragment reads the desired
output from
            the file "desrdout.dat" */
            fpt = fopen("desrdout.dat","r");
            if(nnm == 53)
              {
              for (step=0; step<n_files; step++)
              {
              fscanf(fpt,"%d",&des_out[step]);
              }
            }
          if(nnm == 54)
            {
            fprintf(fpl,"%s\n","Reading Values to be Predicted
(Desired Values)...");
            for (step=0; step<n_files; step++)
              {
              fscanf(fpt,"%f",&des[step]);
              }
            }
            fclose(fpt);

             /*   The following loop initialises the weight matrix to
random values */
            for (index=1; index<=h_layers+1; index++) /* index spans
each layer from the i/p +1 to the o/p layer */
            {
            for (count=0; count<h_nodes[index]; count++) /* count
spans each node the I layer */
                  {
                  for (loop=0; loop<=h_nodes[index-1]; loop++) /*
loop spans each node the previous layer */
                        {
                  wt[index][count][loop] = rand()/32767.0;
                        }
                  }
               }

fpt = fopen(efname,"w");
```

```
patt_low = 35000.0;
fmax_epoch = max_epoch;
do { /* 'do' loop starts here */
   epoch = epoch + 1; /* epoch counts each cycle */
   fepoch = epoch;
   lrate = learnrate*((fmax_epoch - fepoch + 1.0)/fmax_epoch);
   patt_err = 0.0;
   for (step=0; step<n_files; step++)     /* step spans all the files
for each epoch */
       {
       for (loop=0; loop<=i_nodes; loop++)
           {
           if(loop==i_nodes)
             {
             ot[0][loop] = 1.0;
             }
           else
              {
              ot[0][loop] = rawdata[step][loop]; /* each file is
stored in ot[0][loop] to facilitate computation later */
              }
           }
           for (index=1; index<=h_layers+1; index++) /* index spans
all the layers from the i/p +1 to the o/p layer for each step (file)
*/
               { /* start of the first hidden layer */
               for (count=0; count<h_nodes[index]; count++) /* count
spans all the nodes for each index (layer) */
                   { /* starts for each layer */
                   ot[index][count] = 0;
                   for (loop=0; loop<=h_nodes[index-1]; loop++) /*
loop spans nodes in the previous layer */
                       { /* starts for each node */
                       ot[index][count] += ot[index-
1][loop]*wt[index][count][loop];
                       }  /* ends for each node */
                       ot[index][count] = 1/(1 + exp(-
1.0*ot[index][count]));
                       ft[index][count] = ot[index][count]*(1.0 -
ot[index][count]);
                   } /* end for each layer */
               } /* end of the last output layer */

               for (loop=0; loop<h_nodes[h_layers+1]; loop++)
                   { /* Determines the Desired Output */
                   if(nnm == 53)
                     {
                     if (des_out[step] == loop)
                        {
                        flash = 0.95;
                        }
                     else
                        {
                        fprintf(fpl,"%s\n","Setting 'flash' values to
be the desired values...(Classify)");
                        flash = 0.05;
                        }
                     }
                   if(nnm == 54)
                     {
                     flash = des[step];
```

```
                    }
                  patt_err += (flash - ot[h_layers+1][loop])*(flash
- ot[h_layers+1][loop]);
                  et[h_layers+1][loop] = (flash -
ot[h_layers+1][loop])*ft[h_layers+1][loop];
                  }
                  for (index=h_layers; index>=1; index--) /*
Considering layers backwards, starting from the last hidden layer */
                      {
                      for (count=0; count<=h_nodes[index]; count++)
/* count spans the nodes in the current layer */
                          {
                          et[index][count] = 0.0;
                          for (loop=0; loop<h_nodes[index+1];
loop++) /* loop spans the nodes in the NEXT layer here only ! */
                              {
                              et[index][count] = et[index][count] +
wt[index+1][loop][count]*et[index+1][loop];
                              }
                          et[index][count] =
et[index][count]*ft[index][count];
                          }
                      }

/*
At the end of this stage, we have 'et' (the error in the o/p) for all
the nodes, in
all the hidden layers and the output layer, for the file 'step'
The following lines update the weights, in all the layers, for the
file 'step'
*/
                  for (index=h_layers+1; index>=1; index--) /*
Considering layers backwards, starting from the output layer */
                      {
                      for (count=0; count<h_nodes[index]; count++)
/* count spans the nodes in the current layer */
                          {
                          for (loop=0; loop<=h_nodes[index-1];
loop++) /* loop spans the nodes in the previous layer */
                              {
                              wt[index][count][loop] +=
lrate*et[index][count]*ot[index-1][loop];
                              }
                          }
                      }
/*
At the end of this stage, all the weights connecting all the layers
are updated for
file 'step'
*/

       } /* End of one epoch - weights are updated ONCE for all the
files */

       patt_err = sqrt(patt_err);
       fprintf(fpt,"%f\n",patt_err);
       if (patt_err < patt_low)
          {
          patt_low = patt_err;
          for (index=1; index<=h_layers+1; index++) /* index spans
each layer from the i/p +1 to the o/p layer */
```

```
              {
              for (count=0; count<h_nodes[index]; count++) /* count
spans each node the I layer */
                  {
                  for (loop=0; loop<=h_nodes[index-1]; loop++) /*
loop spans each node the previous layer */
                      {
                      nw[index][count][loop] =
wt[index][count][loop];
                      }
                  }
              }
          }
   } while ((patt_err > 0.01) && (epoch < max_epoch)); /* End of the
do loop */
fclose(fpt);

fprintf(fpl,"%s%d%s%f\n","Completed Max Epochs...",epoch,"
",patt_err);

/* The following lines write the final weights in a file called
"owname" */

fpw = fopen(WF,"w");
for(index=1; index<=h_layers+1; index++) /* index spans each layer
from the i/p +1 to the o/p layer */
   {
   for (count=0; count<h_nodes[index]; count++) /* count spans each
node the I layer */
       {
       for (loop=0; loop<=h_nodes[index-1]; loop++) /* loop spans
each node the previous layer */
           {
           fprintf(fpw,"%f\n",nw[index][count][loop]);
           }
       }
   }
fclose(fpw);

fprintf(fpl,"%s\n","Completed Writing the Weight Matrix...");

      break;
} /* End of switch mode 1 - End of Backpropagation Training Phase */

case (50):
      { /* Start of switch mode 2 - Start of Backpropagation Testing
Phase */
      fpw = fopen(WF,"r");
      fpt = fopen(ocname,"w");
        for (index=1; index<=h_layers+1; index++) /* index spans each
layer from the i/p +1 to the o/p layer */
            {
            for (count=0; count<h_nodes[index]; count++) /* count
spans each node the I layer */
                {
                for (loop=0; loop<=h_nodes[index-1]; loop++) /* loop
spans each node the previous layer */
                    {
                    fscanf(fpw,"%f",&wt[index][count][loop]);
                    }
                }
```

```
            }
        fclose(fpw);


        for (step=0; step<n_files; step++) /* step spans all the
files for each epoch */
            {
            for (loop=0; loop<=i_nodes; loop++)
                {
                if(loop==i_nodes)
                  {
                  ot[0][loop] = 1.0;
                  }
                else
                   {
                   ot[0][loop] = rawdata[step][loop]; /* each file is
stored in ot[0][loop] to facilitate computation later */
                   }
                }
            for (index=1; index<=h_layers+1; index++) /* index spans
all the layers from the i/p +1 to the o/p layer for each step (file)
*/
                {  /* start of the first hidden layer */
                for (count=0; count<h_nodes[index]; count++) /* count
spans all the nodes for each index (layer) */
                    { /* starts for each layer */
                    ot[index][count] = 0;
                    for (loop=0; loop<=h_nodes[index-1]; loop++)
                        { /* starts for each node */
                        ot[index][count] += ot[index-
1][loop]*wt[index][count][loop];
                        } /* ends for each node */
                    ot[index][count] = 1/(1 + exp(-
1.0*ot[index][count]));
                    if (index==h_layers+1)
                       {
                       if(nnm == 53)
{fprintf(fpt,"%s%d%s%d%s%f\n","File Number = ",step," Node  =
",count," Output = ",ot[index][count]);}
                       if(nnm == 54)
{fprintf(fpt,"%f\n",ot[index][count]);}
                       }
                    }
                }
            }
        fclose(fpt);
        break;
      } /* End of switch mode 2 - End of Backpropagation Testing
Phase */
    } /* Closing bracket of switch statement */

fprintf(fpl,"%s\n","Freeing Memory....");

/* Freeing Various Memory Allocations - Begin Here */
free_matrix(rawdata,0,n_files-1,0,i_nodes-1);
free_f3tensor(wt,0,h_layers+2,0,i_nodes-1,0,i_nodes);
free_f3tensor(nw,0,h_layers+2,0,i_nodes-1,0,i_nodes);
free_matrix(ot,0,h_layers+2,0,i_nodes);
free_matrix(ft,0,h_layers+2,0,i_nodes-1);
free_matrix(et,0,h_layers+2,0,i_nodes);
if(nnm == 53) {free_ivector(des_out,0,n_files-1);}
```

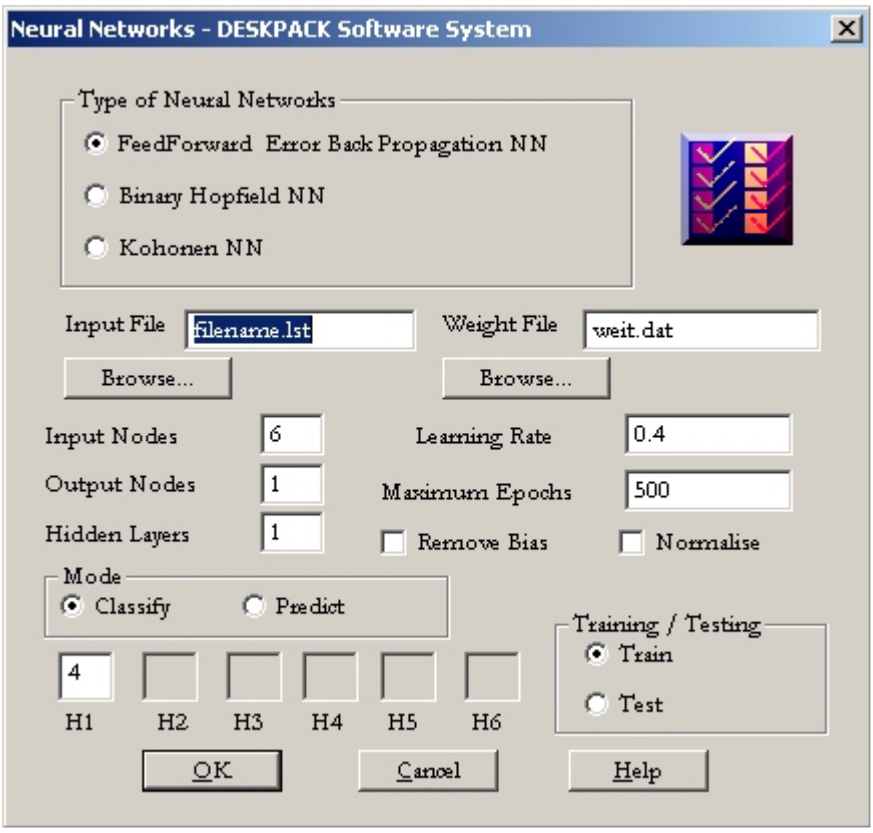
```
if(nnm == 54) {free_vector(des,0,n_files-1);}
free_ivector(h_nodes,0,h_layers+2);
/* Freeing Various Memory Allocations - Ends Here */

fprintf(fpl,"%s\n","Completed Freeing Memory...");

fprintf(fpl,"%s\n","Completed 'backprop' C Module");
fclose(fpl);
*cerr = 0;
} /* Closing bracket of the function backprop */
```

Code Listing 4.17 The feed-forward, error-backpropagation Neural Network

The following dialog screen capture shows the various parameters of the MLP Code Input:



Dialog 4.1 The Neural Networks Input Parameters Dialog

After the User fills up the details, this dialog passes on the input parameters to MLP algorithm shown in the Code Listing 4.17.

Some of the salient features of this algorithm as coded in the Code Listing 4.17 can be summarised as follows:

- This variant of the algorithm uses the online learning method
- The code can take up to six hidden units, each of which can take any number of nodes

- The code can be used for either classification or prediction
- The given code uses the sigmoidal function as the squashing function
- The number of nodes in the input layer and the number of nodes in the output layer can be fixed arbitrarily by the User
- Both the Learning Rate and the Number of Epochs (how many times the data set – training set – passed through the network) can be fixed by the User. The code has been tested even for 75000 epochs
- Since the MLP is a supervised learning algorithm, the correct outputs (or the desired outputs) must be fed to the code using an external file called "desrdout.dat". If the number of nodes in the output layer is N, then the "desrdout.dat" will have a column vector of N components (during the training stage)
- The code is called from Prolog; the results are stored in files in the current directory
- The code stops execution (during the training stage) after the specified maximum number of epochs
- If the maximum number of epochs is fixed as N, then resultant weight matrix (during the training stage) will have those set of values for which the actual network outputs are closest to the desired outputs, i.e., when the Pattern Error is minimum

4.9 Interpretation of Analysed Data

While most of the steps in pre-processing of data and established analysis methods can be automated, the interpretation of results stage cannot be automated, but for a straightforward go-no-go situation. Also, unlike pre-processing and analysis methods, there are no established methods for interpreting the results, which flows directly from one's experience.

Clear understanding of (a) the analysis "path" taken by the data sets to arrive at the results, (b) the objective of the task in hand, and (c) the underlying physical basis of the data sets are essential for the correct interpretation of the results obtained. Particular attention must be paid to the axes and their units.

Interpretation however, can be aided by decision support tools, that guide the User / Expert away from known pitfalls, taking into consideration the pillars of established concepts.

This forms the content of our next Chapter.

4.10 Questions

- Is there any difference between the way we human beings analyse "data" and the way a Computer analyses data? List the differences.

- In the case of the human beings what constitutes a data?
- What are the types of data analysis methods in which the human being is superior to a Computer? Give Examples. Why is it so?
- What are the differences between a human being and a machine in the way data is stored? Which is better? Why?
- Is it possible to find a physical meaning for every feature extracted from a signal data? Under what circumstances is it not possible to find a physical meaning?
- When a time-domain data is projected onto the frequency domain, the data decomposes into sinusoidal components of different types. Are there other ways (or "bases") to represent time-domain data?
- Why is it that the sinusoidal representation is more popular?
- How to visualise partitioning of data in dimensions more than three?
- How similar are the artificial neural networks (ANN) described in textbooks with biological neural networks (BNN)? What are computational neural networks (CNN)? How do they differ with respect to BNN?

Bibliography

1. P.Kalyanasundaram, C.Rajagopalan, C.V.Subramanian, M.Thavasimuthu and Baldev Raj, "Ultrasonic Signal Analysis for Defect Characterisation in Composite Materials", British Journal of NDT, Vol. 33, No. 5, May 1991, pp.221-226.

2. P.Kalyanasundaram, C.Rajagopalan, Baldev Raj, O.Prabhakar and D.G.R. Sharma, "High Sensitivity Detection and Classification of Defects in Acoustic Weldments using Cluster Analysis and Pattern Recognition", Brit. J. of NDT, Vol. 33, No. 6, June 1991, pp.290-297.

3. P.Kalyanasundaram, C.Rajagopalan, Baldev Raj, O.Prabhakar and D.G.R.Sharma, "Detection of Small Defects in Austenitic Stainless Steel Weldments Using Pattern Recognition and Cluster Analysis Approaches", J. of the Acoustical Society of India, Vol. XVIII, Nos.3 & 4, November 1990, pp.244 - 250.

4. P.Kalyanasundaram, C.Rajagopalan and Baldev Raj, "Ultrasonic Signal Analysis for defect characterisation in composite materials", Journal of the Acoustical Society of India, Vol. XVII, Nos. 3 & 4, 1989, pp. 359-363.

5. P.Kalyanasundaram, C.Rajagopalan and Baldev Raj, "Some concepts in elucidating characterisation of weak acoustic emission signals", Journal of the Acoustical Society of India, Vol. XVII, Nos. 3 & 4, 1989, pp.364-369.

6. P.Kalyanasundaram, C.Rajagopalan, C.V.Subramanian, M.Thavasimuthu and Baldev Raj, "Reliable applications of signal analysis methods for ultrasonic evaluation of materials and components", J. of Pure and Applied Ultrasonics, Vol:14, No:1, 1992, pp.13 - 20.

7. M.T.Shyamsunder, C.Rajagopalan, K.K.Ray and Baldev Raj, "A comparative study of conventional and artificial neural network classifiers for eddy current signal classification", INSIGHT, Vol. 37, No. 1, January 1995, pp. 26 - 30.

8. M.Thavasimuthu, C.Rajagopalan, P.Kalyanasundaram and Baldev Raj, "Improving the Evaluation Sensitivity of Ultrasonic Pulse Echo Technique using a Neural Network Classifier", NDT&E International, Vol. 29, No. 3, pp. 175 - 179, 1996.

9. C.V.Subramanian, M.Thavasimuthu, C.Rajagopalan, P.Kalyanasundaram and Baldev Raj, "Ultrasonic Test Procedure for Evaluating Fuel Clad EndCap Weld Joints of PHWRs", Materials Evaluation, Vol. 53, No. 11, November 1995, pp. 1290 - 1295.

10. C.Rajagopalan, Baldev Raj and P.Kalyanasundaram, "The Role of Artificial Intelligence in Nondestructive Testing and Evaluation", INSIGHT, Vol. 38, No. 2, February 1996, pp. 118 - 123.

11. P.Mukherjee, P.Barat, T.Jayakumar, P.Kalyanasundaram, C.Rajagopalan, Baldev Raj, "Acoustic Emission Studies On Welded And Thermally Treated Aisi 304 Stainless Steel During Tensile Deformation", Scripta Materialia, Vol.37, No.8, pp. 1193-1198, 1997.

12. M.Thavasimuthu, C.Rajagopalan, T.Jayakumar and Baldev Raj, "Effect of Front Surface Roughness on Ultrasonic Contact Testing : A Few Practical Observations", Materials Evaluation, November 1998, pp. 1302 - 1309.

13. M.T.Shyamsunder, C.Rajagopalan, B.Raj, S.K.Dewangan, B.P.C. Rao and K.K.Ray, "Pattern Recognition Approaches for the Detection and Characterisation of Discontinuities by Eddy Current Testing", Materials Evaluation, Vol. 58, No. 1, January 2000, pp. 93 - 101.

14. P. Kalyanasundaram, C. K. Mukhopadhyay, C.Rajagopalan and Baldev Raj, "On-line Prediction of Quality and Shear Strength of Spacer Pad Welds of Nuclear Fuel Pins by Applying Cluster and Neural Network Analysis of Acoustic Emission Signals", Accepted for publication in the Journal, Science & Technology of Welding and Joining. (December 2003)

15. C.Rajagopalan, Baldev Raj and P.Kalyanasundaram, "A Soft Computing framework for Fault Diagnosis", Information Sciences, Vol. 127, pp. 87 - 100, 2000. (Special Issue on Soft Computing).

16. P.Kalyanasundaram, C.Rajagopalan, Baldev Raj, "DMAC - A Versatile Tool for 1-D Pattern Analysis for Ultrasonic Signals", Insight, Vol. 46, No. 1, January 2004, pp. 37 - 43.

17. B.Venkataraman, C.Rajagopalan and Baldev Raj, Multilayered, Error-backpropagated, Feed-forward Artificial Neural Network for Prediction of Temperature and Strain Rate Generated, During Tensile Deformation, NDT & E International.

18. Baldev Raj, P.Kalyanasundaram, C.Rajagopalan, G.Vaidyanathan and K.Swaminathan, "Detection and Characterisation of Bubbles, Suspensions and Colloids in Water using Ultrasonic Signal Analysis", Proc. of 13th World Conference on NDT, Brazil, October 1992, pp.1058-1064.

19. Baldev Raj, M.Thavasimuthu, C.V.Subramanian, P.Kalyanasundaram and C.Rajagopalan, "Ultrasonic Evaluation of End Cap Weld Joints of Fuel Elements of PHWRs using Signal Analysis methods", ibid., pp.1065-1070.

20. C.V.Subramanian, M.Thavasimuthu, C.Rajagopalan, "Ultrasonic evaluation of longitudinal seam welded thin walled Hasteloy tube", Proc. of NDE 92, Vol. II, pp. 545-551.

21. C.Rajagopalan, P.Kalyanasundaram and Baldev Raj,"An Expert System to Aid Ultrasonic Testing of Austenitic Welds", Proc. of NDE 92, Vol. II, pp. 155-168.

22. C.Rajagopalan, Baldev Raj, and P.Kalyanasundaram, "Acoustic Emission Monitoring for Nuclear Applications", presented at the first National Workshop on Acoustic Emission (NAWACE-90), June 28-29, 1990, Sriharikota, Andhra Pradesh.

23. Baldev Raj, P.Kalyanasundaram and C.Rajagopalan, "Emerging Trends in Engineering Materials and Components Testing Using Acoustic Methods", Proceedings of National Symposium on Acoustics, (NSA-91), New Delhi.

24. C.Rajagopalan, B.Venkatraman and Baldev Raj, "An Expert System to aid Radiography Testing", presented at the International Conference on Application of RadioIsotopes and Radiation in Industrial Development (ICARID-94), Feb. 7-9, 1994, Bombay.

25. C.V.Subramanian, M.Thavasimuthu, Baldev Raj, D.K.Bhattacharya, P.Kalyanasundaram and C.Rajagopalan, "Ultrasonic Testing of Carbon Fibre Composites", presented at the National Seminar on NDT and Inspection, (NDTI-89), Feb. 89, National Physical Laboratory, New Delhi.

26. C.Rajagopalan, M.T.Shyamsunder, P.Kalyanasundaram, Baldev Raj and G.Hariharan, "The Utility of Artificial Neural Networks in Nondestructive Testing and Evaluation", Published in the Proceedings of the National Conference on Neural Networks and Fuzzy Systems, 16 - 18, March, 1995, Anna University, Madras, pp. 225 - 235.

27. C.Rajagopalan, M.T.Shyamsunder and Baldev Raj, "Characterisation of Eddy Current Signals using a binary Hopfield Neural Network", Proceedings of the 8th Asia-Pacific Conference on NDT, December 11 - 14, 1995, Taipei, Taiwan, pp. 547 - 557.

28. M.T.Shyamsunder, C.Rajagopalan, Baldev Raj and Mayur Deshpande, "Characterisation of Eddy Current Signals Using a Boltzmann Machine", communicated for presentation in the National Seminar on NDE (NDE-95), New Delhi, November 1995.

29. M.T.Shyamsunder, C.Rajagopalan and Baldev Raj, "EDDYEX - A Knowledge Based System for Eddy Current Testing", ibid.

30. C.Rajagopalan, P.Kalyanasundaram and Baldev Raj, "Issues in Developing Knowledge Based Systems for Nondestructive Testing", presented at the International Conference in Trends in Industrial Measurements and Automation, Madras, 3 - 7, January, 1996.

31. Baldev Raj and C.Rajagopalan, "Artificial Intelligence to Maximise Contributions of Nondestructive Evaluation to Materials Science and Technology", Special Session Paper delivered during the 14th World Conference on Nondestructive Testing, New Delhi, India, Vol. 1, 47 - 57.

32. C.Rajagopalan, P.Kalyanasundaram and Baldev Raj, "Uncertainty Management in Knowledge based systems for Nondestructive Testing - An example from Ultrasonic Testing", ibid, Vol. 4, pp. 2127 - 2131.

33. C.Rajagopalan, P.Kalyanasundaram and Baldev Raj, "The Evolution of the Use of Artificial Intelligence in NDT&E", presented at the National Seminar on NDE (NDE-1994), Bombay, December 1994.

34. M.T.Shyamsunder, C.Rajagopalan, K.K.Ray, T.Jayakumar, P.Kalyanasundaram and Baldev Raj, "Characterisation of Eddy Current Signals using Different Types of Artificial Neural Networks", ibid, Vol. 3, pp. 1875 - 1879.

35. M.T.Shyamsunder, C.Rajagopalan, K.K.Ray and P.Kalyanasundaram, "Application of Neural Networks to Eddy Current Defect Characterisation in Materials", poster presentation at the 51st ATM-NMD, Jamshedpur, November 1997.

36. M.T.Shyamsunder, S.K.Dewangan, C.Rajagopalan, K.K.Ray and Baldev Raj, "Eddy Current Data Inversion using Neural Networks".

37. M.T.Shyamsunder, C.Rajagopalan, S.K.Dewangan and Baldev Raj, "Role of Artificial Neural Networks for Defect Detection and Characterisation by Eddy Current Testing", Proceedings of the National Seminar on Artificial Neural Networks and Cognitive Systems (ANCS-98), Cochin University, September 1998.

38. C.Rajagopalan, Baldev Raj and P.Kalyanasundaram, "Knowledge-based Decision Support Systems for NDT", presented during the National Seminar on NDE - 1999, Vadodara.

39. S.K.Dewangan, M.T.Shyamsunder, C.Rajagopalan, Baldev Raj, K.K.Ray, T.Jayakumar and P.Kalyanasundaram, "Application of Artificial Neural Network to Defect detection and characterization for Life Management", Proceedings of the International Symposium on Materials Ageing and Life Management, 3-6, October 2000, Kalpakkam, India, pp. 1329 - 1334.

40. C.Rajagopalan, Baldev Raj and P.Kalyanasundaram, "Synergism of NDE & IT: A Generic Knowledge-Base System for Effective and Reliable NDE", Proc. of the

10th Asia-Pacific Conf. on NDT, Brisbane, Australia, 2001. (http://www.ndt.net/article/apcndt01/papers/802/802.htm)

41. C.Rajagopalan, Baldev Raj and P.Kalyanasundaram, "Role of Information Technology in Nondestructive Testing for Total Quality Management", accepted for presentation at the 10th World Congress on Total Quality, 22nd - 24th January 2001, New Delhi.

42. C.Rajagopalan, P.Kalyanasundaram and Baldev Raj, "Signal Analysis Basics for Acoustic Emission Testing", 5th National Workshop on Acoustic Emission, Satish Dhwan Space Center, SHAR, Sriharikota, 18 - 19 October 2002, pp.157 - 170.

43. A.S.Ramesh, C.Rajagopalan, R.Chellapandian, Hassan Sheikh, P.Kalyanasundaram, Baldev Raj, "Evaluation of Stress Corrosion Cracks in an inaccesible Lattice Tube Weld using Ultrasonic Signal Analysis - A Case Study", presented at the NDE 2002 Symposium, December 5-7, 2002, Hotel Taj Connemara, Madras, pp. 25 -26.

44. C.Rajagopalan, B.Venkatraman, T.Jayakumar, P.Kalyanasundaram, Baldev Raj, "Multi-Layered Perceptron based Artificial Neural Network for the Prediction of Temperature Generated during Tensile Deformation", ibid., pp. 80 -81.

45. C.Rajagopalan, P.Kalyanasundaram, Baldev Raj, "Assets and Infrastructure Management System Software Framework", NDE 2003, Thiruvananthapuram, 11 - 13, December 2003, TP-78, pp.174 - 175.

46. P.Kalyanasundaram, C.Rajagopalan, Baldev Raj, "Eigenvalue Analysis for the Classification of 1-d Signals: An Exploratory Review", ibid., TP-77, p.173.

47. C.Rajagopalan, B.Venkataraman, T.Jayakumar, P.Kalyanasundaram and Baldev Raj, "A Novel Method for Automated Evaluation of Radiographic Weld Images", To appear in the Proceedings of the 16th World Conference on NDT, August 30 - September 3, 2004, Montreal, Canada.

48. B.Venkataraman, C.Rajagopalan, and Baldev Raj, "Predicting Strain Rate During IR Imaging of Tensile Deformation Using MLP-based ANN", To appear in the Proceedings of the 16th World Conference on Nondestructive Testing, August 30 - September 3, 2004, Montreal, Canada.

49. C.Rajagopalan, "A Generic Knowledge-Based Systems' Architecture for Materials Evaluation", Ph.D. Thesis, University of Madras, Chennai, December 2000.

50. C.Rajagopalan, http://deskpack.tripod.com/index.html

Glossary of Signal Processing Terms

A

Acquisition, Data – Data acquisition is a process of collecting and storing signals, for further processing.

Aliasing – It is a phenomenon by which an analog signal when sampled below the Nyquist rate of sampling, the information about the frequency components higher than Nyquist frequency are lost and also these higher frequency components take on the identify of lower frequency components.

Amplifier - A device, which increases the voltage or power level of a signal, introducing as minimum a distortion as possible to the signal.

Amplifier, Lock-in – It is a selective kind of voltmeter based on detection by cross correlation for measuring a D.C. or very slowly varying signal embedded in an independent noise.

Amplitude – The instantaneous value of a signal at any given time.

Amplitude, Peak – The maximum value of a signal within a specified time interval, or over a time record.

Analog-to-Digital Conversion – A process of sampling an analog signal at specified intervals of time and representing sampled values as sequence of numbers. Also it is a process of converting each sampled value into binary form, of finite resolution, determined by the number of bits per sample.

Analysis, Real-Time – An analysis method where, the signals are processed, analysed and evaluated for necessary action, as and when they area acquired.

A-Scan – A CRT (cathode ray tube) display in which the received signal amplitude is shown as a vertical excursion from the horizontal sweep time trace.

Attenuation – A phenomenon by which energy is reduced when a signal passes through a system/medium. It represents the loss in acoustic energy that occurs between any two points of travel. This loss may be caused by absorption, reflection, scattering or other material characteristics.

Attenuator – A device for causing or measuring attenuation.

Averaging, Frequency – A process of averaging frequency spectra of finite number of successive repetitive signals (time records) so that magnitudes of corresponding frequencies are added, vectorially.

Averaging, Time – A process of averaging successive repetitions of the signal records (time records) so that the periodic signals add coherently, while the random element is averaged to a small value, by virtue of its incoherence.

` **B**

Bandwidth – Bandwidth of a signal is the range of frequencies bounded by its upper and lower cutoff frequencies in the frequency spectrum. Also see 'Cutoff Frequency'.

Bias – The bias of an estimate is the difference between the estimate mean value and the true mean of the estimated values.

B-Scan – A data presentation method typically applied to pulse echo ultrasonic testing. It produces a 2-D view of a cross-sectional plane through the test object. The horizontal sweep is proportional to the distance along the test object and vertical sweep is proportional to the depth.

C

Cepstrum – It is a time function defined as wither the Fourier transform of the logarithm of its power spectral density, or modulus square of this transform.

Clipping – Signal bit quantization, in two levels, positive or negative so that only the algebraic sign of the signal is preserved.

Cluster – A group or a class of elements having one or many common properties.

Cluster Analysis – Study and extraction of information about and from the clusters, so as to classify different clusters based on the information obtained above.

Cluster Classification – Grouping of cluster elements based on their properties.

Clustering – A process, in which a set of data is organized into groups that have strong internal similarity.

Convolution – A process in which the output signal at time 't' is the weighted sum of past values of the input signal x(t). Mathematically, convolution of x(n) and h(n) can be represented as

$$y(n) = \sum_{k=0}^{N-1} x(n-k)h(k).$$

Convolution, Circular – Aliasing that can occur in the time domain when frequency domain signals are multiplied. Each period in the time domain overflows into adjacent periods.

Correlation, Auto – A process, which compares the function x(t) at time 't' with its value at time t-τ. See also, Correlation Function, Auto.

Correlation, Cross – The sum of all products of two samples over entire record length, of a signal such that the samples are separated by a search parameter 'm' when expressed as a function of 'm' mathematically,

$$R_{xy}(m) = \sum_{m=-(N-1)}^{+(N-1)} x(n)y(n+m)$$, where 'm' is an integer.

This can also be viewed as a process which characterizes the relationship of one signal x(t) at an instant 't' with another signal y(t) at an instant 't-τ'.

Correlogram, Auto / Cross – A graphical representation of the auto / cross correlation function with respect to the search parameter 'm'.

Correlation Function, Auto

The sum of all products of two samples over the entire record length of a signal such that the samples are separated by a search parameter 'm' when expressed as a function of 'm' mathematically.

$$R_x(m) = \sum_{m=-(N-1)}^{+(N-1)} x(n)\, x(n+m)$$, where 'm' is an integer.

Count, Ringdown – It is the number of threshold crossings of a signal, in specified direction, during an interval.

Covariance Function, Auto – It is an autocorrelation function in which the process variables are replaced by the deviation of the process variables with respect to their means. Mathematically, this can be expressed as:

$$c_x = \sum_{m=-(N-1)}^{+(N-1)} [x(n) - \mu_x]\, [x(n+m) - \mu_x]$$

Cross-talk – The unwanted signal leakage (acoustical or electrical) across an intended barrier, such as leakage between the transmitting and receiving elements of a dual transducer. Also called cross-noise or cross-coupling.

C-Scan – A data representation applied to pulse echo and transmission ultrasonic testing. It yields a 2-D plan view of the object.

D

Damping – It is a process by which the signal amplitude is gradually reduced to zero.

Data – Representation of information in discrete form.

Data Length – Number of data points per record.

Decibel – It is the ratio of two values of voltage or power expressed in logarithmic scale with a multiplication constant 20 or 10, respectively, expressed as :

$$\text{Voltage Gain} = 20 \log (v/v_r)$$
$$\text{Power Gain} = 10 \log (p/p_r)$$

Decimation – Reducing the sampling rate of a digitized signal, generally involves low-pass filtering followed by discarding samples.

Decimation-in-Time -- It is the technique used for computing Fast Fourier Transform, which results in two non-interleaved final sequence from the interleaved final sequences from the interleaved time sequence.

Decimation-in-frequency – It is a technique used for computing Fast Fourier Transform, which divides the time sequence into two non-interleaved sequences and interleaves the values of the final.

Deconvolution – A process of expressing the impulse response of a linear system in terms of its output and the parameters of the input.

Detection – A process attempted to extract an useful signal from the background noise, which is superimposed on it.

Digital-to-Analog Conversion – This is an inverse process of Analog-to-Digital Conversion, where a digitized input sequence is transformed into an output analog signal.

Digitization – A process of converting an analog signal into samples expressed as such or in discrete amplitude.

Dispersion – See standard deviation.

Distortion – A phenomenon by which the signal characteristics are disturbed.

Distribution, Binomial – It is the statistical law of the discrete random variable obtained by counting the number of occurrences (k) of an event, during finite number of independent trials (n). Mathematically, the probability of occurrence of 'k' events out of 'n' trials is given by

$$P(k,n) = {}^{n}C_{k}\ p^{k}(1-p)^{n-k}$$

Where 'p' is the probability of occurrence of an event.

Distrubution , Gaussian – A random variable is said to follow Gaussian distribution if it is a outcome of a physical process made up of many component process, none of which is dominant over the other. Mathematically, the corresponding probability density function can be given as

$$P(x) = [1/\sigma_x\sqrt{2\pi}\,]\ \exp^{[-(x-\mu_x)^2/2\sigma_x^{2}]}$$

Distribution, Normal – See Gaussian Distribution

Distribution, Poisson – It is a special case of the Binomial distribution where the probability of occurrence of an event is very small and the number of independent trials is large. Mathematically, the probability of occurrence of 'k' events out on 'n' trials is given by

$p(k,n) = \chi^{(K/K_1)}\exp[-x]$,where 'x' is the mean occurrence of an event.

Domain – A space in which signal parameters/characteristics can be defined/processed as a function of a certain variable of the domain.

Domain, Frequency – A space where signal parameters/characteristics can be defined processed as a function of frequency.

Duration, Pulse – It is the time duration for which the pulse is above the threshold value.

Dynamic Range – It is the difference in decibels between the overload level and the minimum signal level (usually fixed by noise level or low level distortion or interference or resolution level) in a system or sensor.

E

Ensemble – It is the experimental realization of a set of similar signals produced by the same stochastic process.

Ensemble Averaging – The process of arriving at the average value existing at time 't' or discrete variable 'n' summed over the ensemble and continued over the entire record length.
Mathematically,

$$A(n) = (1/M)\sum_{i=1}^{M} x_i(n)$$

where 'M' is the number of time records.

Error, Overflow – Error arising in the measuring process during acquisition, due to overflow of the signal's value, above a predefined level.

Error, Quantization – Error arising due to finite resolution of the sampling process.

Estimate – It is an outcome found by using certain rules or methods (called 'estimators'), or known variables.

F

Filter – A device that minimizes any undesired components in the input signal so as to deliver a signal with desired properties.

Filter Bank – A group of filters connected in series or parallel or both, wherein one or more can be chosen at will.

Filter, Analog – A system that accepts an analog signal as its inputs and outputs an analog signal with desired properties.

Filter, Anti-aliasing – An ideal low-pass filter with a bandwidth of the Nyguist frequency.

Filter, Auto-regressive Moving Average (ARMA) – A digital filter having the properties of both autoregressive filter and moving average filter. (See auto-regressive and Moving Average). The general equation is given by

$$y(n) = (1/a_o) \left[\sum_{k=0}^{N} b_k x(n-k) - \sum_{k=1}^{M} a_k y(n-k) \right]$$

Filter, Auto-regressive – A digital filter whose every sampled output depends on a finite number of past output values and the present input value.

$$y(n) = -(1/a_o) \left[\sum_{k=1}^{M} a_k y(n-k) \right] + \frac{b_o x(n)}{a_o}$$

Filter, Band-Pass (Ideal) – A filter whose output is unchanged for input frequencies above its lower cutoff frequency and below its upper cutoff frequency, removing other frequencies. The range of frequencies between the lower and upper cutoff frequencies is called the pass band.

Filter, Bessel – It is a class of low pass filters characterised by the property that the group delay is maximally flat at the origin of the S-plane.

Filter, Butterworth – It is a class of low pass filters characterized by the property that the magnitude characteristic is maximally flat at the origin of the S-plane.

Filter, Chebyshev - It is a class of low pass filters characterized by the property that over a prescribed band of frequencies the peak magnitude of the approximation error is minimized.

Filter, Digital – A system that accepts digital signal as its input and outputs a digital signal with desired properties.

Filter, Elliptic – It is a class of low pass filters characterized by the magnitude response that is equiripple in both the passband and stopband.

Filter, Finite Impulse Response (FIR) – An auto-regressive filter whose duration of impulse response is finite.

Filter, High-Pass (Ideal) – A filter whose output is unchanged for input frequencies above its cutoff frequency, removing the frequencies below the cutoff frequency.

Filter, Infinite Impulse Response (IIR) – An auto-regressive filter whose duration of impulse response is infinite. Also called a recursive filter.

Filter, Matched – This is a particular case of a linear filter designed to optimize the signal to noise ratio, when attempting to detect a signal, say x(t), of known shape and duration 'T' embedded in background noise.

Filter, Moving Average – A filter whose output depends only on a finite number of present and past values of the input.
Mathematically, this is expressed as:

$$y(n) = (1/a_o) \left[\sum_{k=0}^{N} b_k x(n-k) \right]$$

Filter, Non-recursive - See FIR filters.

Filter, Recursive – See IIR filters.

Filter, smoothing – A filter, which removes the unwanted random roughness in the signal caused by the noised.

Filter, Stop-Band – A filter whose output is unchanged for input frequencies below its lower cutoff frequency and above its upper cutoff frequency, removing other frequencies.

Fourier, Coefficients – Coefficients of the individual components of a Fourier series. For a continuous time signal of duration 'T', its k^{th} coefficient (harmonic) is given by

$$x(k) = (1/T) \int_{0}^{T} x(t) \exp(-j2\pi kt/T)dt$$

Fourier Series – A representation of any periodic time signal in terms of its basic cosine and sine components. Mathematically, the series can be expressed as:

$$x(t) = \sum_{k=0}^{\infty} x(k) \exp (j2\pi kt/T).$$

Fourier Magnitude Spectrum – A plot of the Fourier coefficients representing the magnitude of the harmonic components (Fourier Components) of a time signal as a function of its frequency content. Also see amplitude spectrum and phase spectrum.

Fourier Transform – A transformation, which converts time domain data into frequency domain data, resulting in amplitude and phase distribution with respect to frequency. Alternately, it is a function, which describes the amplitude and phase of each sinusoidal component.

Fourier Transform, Discrete – A Fourier transform of discrete time for main data into discrete frequency domain data. The expression is given as:

$$X(k) = \sum^{N-1} x(n) \exp(-j2\pi nk/N).$$

n=0

Fourier Transform, Fast – An algorithm used to compute discrete. Fourier transform using minimum memory space and time, for computation.

Fourier Transform, Inverse – A reverse process of Fourier transform, to convert frequency domain data into the original time domain data. Mathematically, the expression is given by:

$$x(n) = (1/N) \sum_{k=0}^{N-1} X(k) \exp(j2\pi nk/N)$$

Frequency – It is the rate at which a periodic signal repeats itself.

Frequency Response – A complex function, which defines the operation of a linear system as a function frequency.

Frequency, Cutoff (Lower and Upper) – The values of frequencies (Lower and Upper) at which the voltage gain of the filter is down by 3db from its maximum value in the pass band. Among these, the lower frequency is called the lower cutoff frequency and the upper one, the upper cutoff frequency.

Frequency, Folding – See, Nyquist frequency.

Function Coherence – Coherence function is one which is a measure of the output signal power portion, at frequency 'f' that is due to the input signal. Also, this function indicates, when its value is not unity, either the existence of additional noise, a nonlinear relationship between the input [x(t)] and output [y(t)] , or that [x(t)] does not depend upon excitation of [y(t)], solely. The expression for coherence function is given by,

$$\sqrt{xy}^{(k)} = \frac{|S_{xy}(k)|^2}{|S_x(k)|\,|S_y(k)|}$$

Function, Demodulated Autocorrelation – The logarithm of the square of the auto correlation function.

Function, Dirac impulse – A function whose integral with respect to time tends to unity as the duration of the function 'T', tends to zero.

Function, Even – Any function whose value is not altered both in sign and magnitude when the sign of the independent variable is reversed.

Function, Odd – Any function whose value is not altered in magnitude but changes sign when the sign of the independent variable is reversed.

Function, Orthogonal – A function that gives a finite value for the weighted average of the product of sine and sine or cosine and cosine components, if their frequencies and phase shifts are identical and given a null result otherwise.

Function, Probability Density – A function which indicates how likely the dependent variable (described by say, 'y') is found in the band, say 'y' at a location y, in the overall signal record.

Function, Probability Distribution – It is that function, that gives the area under the probability Density function [p(x)] curve from minus infinity to value of interest , say 'x' Mathematically expressed as,

$$p(x) = \int_{-\infty}^{x} p(x)\, dx$$

Function, Sine – It is the ratio of sine function to its argument, given by

$$\text{Sinc}(x) = [\sin(\pi x)]/(\pi x)$$

Function, Transfer – Transfer function of a system is the frequency response of that system, relating, the input and the output in the frequency domain.

Function, Windowing – It is a finite weighting sequence w(n), multiplied with the finite sequence x(n), so as to minimise the effect of Gibb's phenomenon.

FWHM – Full Width at Half-Maximum. A measure the width of a peak in a signal. Usually used with Gaussian functions. The width of the peak is measured at half of the peak amplitude.

G

Gain – It is the ratio of the output voltage level to the input voltage level, expressed in decibels.

Gate – It is a rectangular window of adjustable length and position.

Ghost – An indication arising from certain combinations of pulse repetition frequency and time base frequency.

Gibb's Phenomena – The limitation of a Fourier series to converge at discontinuities is Gibb's phenomena.

Group Delay – Group delay of filter is a measure of the average, delay of the filter as a function of frequency.

H

Hilbert Transform – Hilbert Transform of a function x(n), is the function's convolution with $1/\pi n$.

$$H(n) = \sum_{k=0} x(n-k)\ (1/\pi k)$$

I

Impulse – See Dirac Function.

Impulse Response – Total response of a system to Dirac impulse function, expressed as a weighting function, h(t).

Interleaving – In the case of repetitive signals, sampling of the waveform can be done over many periods by taking a certain sample point from each repetition and then combining them into one sample series.

M

Mean – Average of a set of numerical data.

Method, Maximum Entropy – It is a non-linear spectral estimation method in which among all the spectra that are consistent with the limited available data, the spectrum corresponding to a random signal of maximum entropy (informational entropy) is selected.

Method, Maximum Likelihood – It is a vector of parameter that are either non-random but with unknown statistics, the estimation must be based on the sole a prior knowledge of the conditional probability density p(x/a), of the observation vector, x, depending on the parameter vector, a, and on the noise statistics. This distribution is called likelihood function. The method of estimation of a vector a_{ML} maximizing p(x/a) is called the Maximum Likelihood Method.

Modulation – A process, in which a primary signal called the modulating signal, modifies an auxiliary signal called the carrier to create a secondary or modulated signal.

Modulation, Amplitude – A modulation in which the amplitude of a carrier varies as a function of the modulating signal. Mathematically represented as,

$$y(t) = [1+a_x(t)]\ \text{Cos}\omega_c t$$

Modulation, frequency – A modulation in which the phase of a carrier varies as a function of the modulating signal.

Modulation Phase – A modulation in which phase of a carrier varies as a function of the modulating signal.

Modulation, Pulse Code – A modulation where an analog signal is coded into a of pulse.

N

Noise – A varying signal having no desirable information.

Noise, Background – The extraneous signals caused by random signal sources within or exterior to the testing system.

Noise, Broadband – A noise whose power spectrum is wide.

Noise Flicker – It is a stochastic process whose power spectral density varies inversely with frequency.

Noise Gaussian – A noise whose amplitude values follow a Gaussian distribution, over long period of time.

Noise Random – A noise whose amplitude distribution follows no known distribution.

Noise, White – It is a stochastic process whose power spectral density is constant for any value of frequency.

Nyquist Criteria – In order to avoid aliasing, any signal should be sampled during the process of digitization, at a rate at least twice the Nyquist frequency.

Nyquist Frequency – Nyquist frequency is the highest of the individual frequency components present in any signal (also called folding frequency).

Nyquist Rate – It is the sampling rate, which is twice the Nyquist frequency.

O

Offset Value – Level by which every data point is increased or decreased.

P

Peak-to-Peak Value – The level difference between the maximum positive and the maximum negative amplitudes of a signal, within an interval.

Perseval Theorem – The total power contained in the fourier spectrum of a signal equals that in the signal itself.

$$\int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |x(f)|^2 df$$

Pattern Recognition – It is the categorisation of input data into identifiable classes, via the extraction of significant features of attributes of the data from a background of irrelevant detail.

Phase – A relative position of observation of a signal with respect to a reference point.

Phase Angle – The value of phase expressed in terms of degrees of radians or gradients.

Phase Difference – The temporal shift observed between two signals expressed in terms of a phase angle.

Power Spectral Density – See power Spectrum.

Power Spectrum – ***The power spectrum is the distribution of energy content of a signal record, say x(t), with respect to its frequency content. Alternately, the power spectrum is the Fourier transform of the auto correlation function. Or, power spectrum can be defined as the self-conjugate product of its Fourier Transform x(k). Mathematically expressed as,***

$$S_x(k) = x(k)x^*(k)$$

Power Spectrum, Auto - See Power Spectrum.

Power Spectrum, Cross – this is the Fourier transform of the cross-correlation function of two signals, say x(n) and y(n). Alternatively, cross power spectrum is the conjugate product of the Fourier Transforms x(k) and y(k) of the two signals x(n) and y(n) . Mathematically, the cross power is expressed as,

$$S_{xy}(k) = x(k)\, y^*(k)$$

Process Ergodic – It is a stochastic process exhibiting identical statistical averages and time averages of the same degree and order.

Process Random – Random process is one whose outcome can be predicted only in a probabilistic manner using statistical laws.

Process, Stationary – It is a stochastic process whose all statistical properties are time invariant.

Process, Stochastic – See Random Process.

R

Rate, Pulse Repetition – It is the rate at which the transmitting pulse is generated or applied to any system or probe.

Record Length – The number of data points used to represent a record. See Data Length.

Resolution – The interval between successive sampling instances or the successive levels of signal amplitude.

Rise Time (Instrument) – For an ideal square wave input, the instrument rise time is the time taken by any measuring instrument to reach from 10 to 90% of the maximum value of its output.

Rise Time (Pulse) – It is the time taken for the pulse to reach from 10 to 90% of its peak amplitude.

Rise Time (Signal) – It is the time duration between the first threshold crossing and the peak of the signal.

Root Mean Square (RMS) Value –The positive root of the mean square value of a signal record. It is a measure of the power content of a signal record. The root mean square value is given by the expression,

$$\sqrt{(1/N)\sum_{n=0}^{N-1} |x(n)|^2}$$

S

S-Plane – It is a complex plane where the frequency response of a system is evaluated.

Sample-and-Hold – A process in which the instantaneous value of a signal is sampled and is stored in analog form.

Sampling – Sampling is a process of examining a continuous function of time or any other independent variable, at equal intervals of the Independent variable.

Sampling Frequency – See Sampling Rate.

Sampling Rate – Number of samples examined at specified intervals per unit time.

Sequency – One half of the average number of zero crossings per unit time intervals.

Signal – An electrical quantity or any other physical quantity which provides information on the presence or change of a physical phenomenon.

Signal Analysis – A process of attempting to isolate the main components of interest of a signal of complex shape in order to understand its nature and origin better.

Signal Processing – It is a technical discipline which, based on the methods and signal and information theory, deals with the elaboration or interpretation of signals carrying information with the resources of electronics, computer engineering and applied physics.

Signal Record – See time History.

Signal Synthesis – It is the opposite operation of signal analysis consisting of creating a signal with desired properties by combining a set of elementary signals.

Signal, Analog – A signal which is characterised by continuous amplitude and time.

Signal, Analytic – A complex function, whose Fourier transform is the unilateral forms of the Fourier transform of its real part.

Signal, Bandlimited – Signal whose frequency component are limited to a certain range of frequencies.

Signal, Bounded – All physical signals whose amplitude cannot exceed certain limit (often enforced by electronic processing devices).

Signal, Casual – A signal is said to be causal if its amplitude is zero, for any time, 't' less than zero.

Signal, Deterministic – It is a signal, which is characterized by an evolution that is perfectly predictable by an appropriate mathematical model.

Signal, Digital – An ordered sequence of numbers generated by sampling a continuous time (analog) waveform at discrete time intervals. Alternatively, a Digital Signal is one, which is characterized by discrete amplitude and discrete time, represented by a sequence of numbers (digits).

Signal, Discrete – See Digital Signal.

Signal, Energy – Energy of a signal over a period of time is the integral value of the square of the signal amplitudes.

Signal, Ensemble – A set of time histories when each of them is referenced to an identical commencement of time.

Signal, Non-casual – A signal which has non-zero value for at least one instant of time 't' where 't' is less than zero.

Signal, Non -periodic – It is a signal which does not obey a regular cyclical repetition law, with a period.

Signal, Periodic – It is a signal, which obeys a regular, cyclical repetition law, with a fixed and finite period.

Signal, Quantized – A signal, which is characterized by discrete amplitude and continuous time.

Signal, Random – It is a signal, which has unpredictable behaviour and can generally be described only through statistical observations.

Signal, Sampled – A signal, which is characterized by continuous amplitude and discrete time.

Signal, Transient – A causal deterministic short-lived signal, which decays to zero value, after a finite length of time.

Signal-to-Noise Ratio – It is a measure of extent of signal contamination by noise.

Signum – It is a function of time, such that its value is equal to 1 for time less than zero, and +1 , otherwise.

Spectrum –Distribution of a certain characteristic parameter of interest of a signal, with respect to its frequency content.

Spectrum, Amplitude – Distribution of amplitude information of a signal with respect to its frequency content.

Spectrum, Continuous – A spectrum which contains a continuous range of frequency components, whose values are however finite.

Spectrum, Crosspower Autopower Difference – It is the difference between the crosspower (normalized to unit energy) of two signals (one called the reference signal and the other test signal) and the autopower (normalized to unit energy) of the test signal.

Spectrum, Line – A spectrum in which the number of frequency components and their values are finite and discrete and are located at precise positions on the frequency axis.

Spectrum, Phase – Distribution of phase information of a signal with respect to its frequency content.

Standard Deviation – The square root of the average of the squares of the instantaneous deviations about the mean value of the signal record. Mathematically,

$$\sigma = \sqrt{[1/(N-1)][\sum_{n=0}^{N-1}(x(n) - \mu_x)^2]}$$

System, Stable – A system where for a bounded input sequence the output sequence is also bounded. Also for such a system its impulse response has finite energy.

T

Threshold – It is a voltage reference level crossing, which a signal is detected for further process or operation.

Time, Dead – It is any interval during data acquisition when the instrument or system is unable to accept new data, for any reason.

Time History – A continuing but finite-length record of a process. Also, see Signal Record.

Trigger Level – It is that voltage level of a signal or an external input, to an instrument, which determines the instant at which data acquisition starts. Data acquisition may start at the instant, or a finite time before (pre-trigger) or a finite time after (delay) the voltage (of the signal or external input) crosses the above preset voltage level.

V

Variance – A measure of scatter of a set of data, about its mean value and is described as the mean square about the mean alternately, variance of random variable is the central moment of second degree. Variance is a mathematically expressed as:

$$(1/N)\sum_{n=0}^{N-1}[x(n) - \mu_x]^2$$

W

Window, Hamming – It is a class of finite weighting functions w(n) operated on a time record, expressed as

$$W(n) = a + (1...a)\text{Cos}(2\pi n/N)$$

for all n, greater than '0' and less than 'N-1' , and zero elsewhere, where 'a' assumes a value between zero and unity. Generally, a=0.54.

Window, Hamming – It is special case of Hamming window, where a=0.50.

Window, Rectangular – It is a windowing function where w(n) =1, for all 'm' where, 0 (N-1). 'N' is the data length.

Z

Z-plane – It is a complex plane in which the z-transform of a signal is studied.

Z-transform – Z-transform is the generalization of the Fourier transform such that, these two transforms are identical on the unit circle; (Unit circle is a circle of unit radius in the complex plane centred at (0,0) – Mathematically expressed as:

$$x(z) = \sum_{n=0}^{N-1} x(n)z^{-n}$$

Symbols Used in the Glossary

x(t), y(t)	Continuous time functions
x(n)	Input Discrete Signal
y(n)	Output Discrete Signal
h(n)	Impulse Response
$R_{xy}(n)$	Cross-correlation function
τ	Time Delay
P	Power

V	Voltage
	Standard Deviation
	Mean
N, M, n, k, m, I	Integers; Number of time records; Data length;
	Indices for summation
p(x)	Probability Density Function
P(x)	Probability of occurrence of events
p(k,n)	Probability fo occurrence of 'k' events in 'n'
	trials
A(n)	Ensemble Average
a_k , b_k , a_o , ab_o	Constants
X(k), Y(k), X(f)	Fourier coefficients
S_{xy}	Crosspower of two signal
S_x , S_y	Autopower of a signal
$xy_{(k)}$	Coherence Function
	Carrier Wave Frequency
P_r	Reference Power
V_r	Reference Voltage
T	Signal Duration
H()	Hilbert Transform
W(n)	Windowing Function
X(z)	Z-Transform of a Signal
C_x	Autocovariance Function
R_x	Autocorrelation Function
Z	Complex Variable