A.1. Input Files

The program TIM-DLY.EXE is a program that was compiled using Watcom C/C^{++} v.10 to work in a DOS environment. It reads binary “raw” data files, applies a user-selected time delay, computes the power spectrum, and writes the result to an ASCII file. The time delay technique is described in §2.3.1. Since a given data set may consist of multiple data files, the program uses program line arguments so that large amounts of data can be processed in a “batch” format. The program line arguments are:

```
TIM-DLY.EXE <input file name> <log file name>
```
(required) (optional)

An *input file* is required for program execution in order to specify user selected options. The log file is optional. If a *log file name* is provided, a copy of all screen I/O is written to the that file. An example input file is,

```
    Output Root Filename = E:\DATA\SPECTRUM
             Record Size = 8192
                   Shift = 8192
Number of Raw Data Files = 5
    E:\RAW-DATA\DATA1.RAW  27.01 20.31
    E:\RAW-DATA\DATA2.RAW  27.01 20.31
    E:\RAW-DATA\DATA3.RAW  27.01 20.31
    E:\RAW-DATA\DATA4.RAW  27.01 20.31
    E:\RAW-DATA\DATA5.RAW  27.01 20.31
```

In the above example, The *Output root filename* is used to construct the output file names. One output file is created for each data channel. The number of the data channel is appended as an extension to the text string given as the *Output root filename*. For example, using the above input file and raw data files that each contain two channels of data, two output files of power spectra would be created– `E:\DATA\SPECTRUM.P1` and `E:\DATA\SPECTRUM.P2`. The *Record Size* is the number of contiguous samples to use when computing the spectral power density and the *Shift* is the number of samples to use as the equivalent time delay. The actual time delay is calculated by multiplying the *Shift* by 1/(sample frequency of data acquisition). The *Number of Raw Data Files* is followed by a list of each “raw” data file name. The static sensitivity (at 0 Hz) of each data channel in the “raw” data file is specified (in Pa/V) following each “raw” data file name. The frequency response of the transducer must be applied to the spectral power density, which is the output of TIM-DLY.EXE.

A.2. Output files

The primary output of TIM-DLY.EXE are several power spectra for each data channel (time series) in the “raw” data files. For example, using the above input file and raw data files that each contain two channels of data, two output files of power spectra would be created– `E:\DATA\SPECTRUM.P1` and `E:\DATA\SPECTRUM.P2`. An example output file is,

```
     Freq            p1             p2            p1-p2          p1+p2
 6.10352e+000  9.92157e-001  9.92038e-001  1.80845e+000  2.15994e+000
 1.42415e+001  3.14360e-001  3.11584e-001  6.13420e-001  6.38468e-001
 2.23796e+001  5.88380e-002  5.85823e-002  1.34346e-001  1.00495e-001
 3.05176e+001  6.28321e-002  6.19110e-002  1.88205e-001  6.12808e-002
 3.86556e+001  4.14870e-002  4.11076e-002  1.10290e-001  5.48988e-002
 4.67936e+001  1.23961e-002  1.33300e-002  3.15965e-002  1.98557e-002
```

The first column (`Freq`) is the center frequency, in Hertz, of each spectral estimate (spectral level). The second column (`p1`) contains the spectral power density of the input time series *without* any time-delay applied. The third column (`p2`) contains the spectral power density of the time-delayed time series. For a perfectly stationary and ergodic time series, the values in `p1` and `p2` are equal. The fourth column (`p1-p2`) contains the spectral power density of the time-delayed and subtracted time series. The fifth column (`p1+p2`) contains the spectral power density of the time-delayed and added time series. The spectral power density due to the turbulent contributions can be separated from the spectral power density due to the coherent contributions (acoustic, vibration) using the method described in §2.3.1.

The other files created by this program are used to check the calculation. The first and last spectral value for each power density spectrum calculated by this program should be very near zero, since each time series is made to have zero mean by the program before the power density spectrum is calculated. This is done in order to prevent spectral energy from the mean value from *leaking* into (contaminating) the adjacent, low frequency spectral estimates (side-lobe leakage is discussed by Bendat and Piersol, 1986, pp. 393-400). The first and last spectral value are not written to the output files that contain the rest of the power density spectrum, since these two estimates should be zero. Therefore, the first and last spectral estimate from each record is written to a separate file. Following the convention adopted for the power density spectrum output, the first and last spectral estimates for each data channel are written to a separate file. For example, using the above input file and raw data files that each contain two channels of data, two additional output files would be created– `E:\DATA\SPECTRUM.M1` and `E:\DATA\SPECTRUM.M2`. An example output file is,

```
Record    FirstEst       LastEst
    0   6.24075e-009   6.37123e-009
    1  -3.44704e-009  -3.55237e-009
    2  -1.11723e-009  -1.26382e-009
    3  -3.44646e-009  -3.36124e-009
    4   3.72054e-010   3.72128e-009
    5   8.98667e-009   8.88884e-009
    6  -6.51994e-009  -6.62983e-009
    7   2.18546e-009   2.15682e-009
```

A.3. Program Listings

The source code for TIM-DLY.EXE is split up into 8 files:

`TIM-DLY.C`	Main functions of the program
`DATHEAD.H`	Contains the definition of the "raw" data file header structure
`FILE-OPS.H` `FILE-OPS.C`	These two files contain utility functions used for file I/O
`LOG-OPS.H` `LOG-OPS.C`	These two files contain redefinitions of the functions `printf` and `scanf` in order to copy screen I/O to a log file, if one is open.
`REALFFT.H` `REALFFT.C`	These two files contain the REALFFT algorithm given on p. 513 of Press *et al*. (1994) modified to use double precision.

TIM-DLY.C

```
/****************************************************************************
*   THIS PROGRAM CALCULATES THE POWER SPECTRUM OF A SIGNAL AFTER APPLYING   *
*       A USER SELECTED TIME DELAY                                          *
****************************************************************************/
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdarg.h>
```

```
#include <math.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
#include <malloc.h>
#include <errno.h>
#include <graph.h>

#include "dathead.h"
#include "realfft.h"
#include "log-ops.h"
#include "file-ops.h"

#define NUM_SPEC    4

//
//      GLOBAL VARIABLES
//
DataHeader      h ;
FileInfo        LogF, IniF, DataF, OutF ;
char            OutRoot[80] ;
unsigned long   RecSize, Shift, BlockSize, TotalRecs, Bins, SampPerBlock,
                NumRecs, RecsDone ;
unsigned short  NumFiles, SampPerBank, ****Data ;
double          delF, ***avg, sens[16], **scratch, ***ms, ***power,
                ***FirstEst, ***LastEst ;

//
//      FUNCTION DECLARATIONS
//
          void ProcessArgs (int ArgCount, char **Arg) ;
unsigned short ReadConfig (void) ;
unsigned short GetNextDataFname (void) ;
          void WrapUp (void) ;
          void PrintConfig (void) ;
          void PrintGlobals (void) ;
unsigned short GetMem(void) ;
          void FreeMem (void) ;
unsigned short OpenDataFile (DataHeader *Header) ;
unsigned short ReadData (void) ;
          void PrintDataHeader (void) ;
          void CalcPowerSpectra (void) ;
unsigned short SaveSpec (void) ;

//
//      MAIN ROUTINE
//
void main (int ArgCount, char **Arg)
{ unsigned short    sp, ch, file, error ;
  unsigned long     bn ;

  _clearscreen(_GCLEARSCREEN) ;
  printf ("\n\n    STARTING TIME DELAY PROGRAM\n\n") ;
  if (ArgCount < 2)
   { printf ("  Usage : tim-dly <Ini Filename> <Log Filename>\n") ;
     printf ("                    (required)     (optional)\n\n") ;
   }
  atexit (WrapUp) ;
  ProcessArgs(ArgCount, Arg) ;
  if (!IniF.loaded) return ;
  LogFptr = LogF.ptr ;

//
//  THE INI (CONFIGURATION) FILE IS READ AND PRINTED TO THE SCREEN
//
  printf (" Reading Configuration ... ") ;
  error = ReadConfig() ;
  if (error)
   { printf ("ERROR %hu\n\n", error) ;
     return ;
   }
  printf ("O.K.\n") ;
  PrintConfig() ;
//
//  THE FIRST DATA FILE IS OPENNED AND THE HEADER IS READ AND DISPLAYED
//
  fscanf (IniF.ptr, "%s", DataF.name) ;

  error = OpenDataFile (&h) ;
  if (error) return ;

  for (ch=0; ch<h.Channels; ch++)  fscanf (IniF.ptr, "%lg", &sens[ch]) ;

  PrintDataHeader() ;
//
//  THESE GLOBAL PARAMTERS ARE CALCULATED BASED ON THE FIRST FILE HEADER
//      IT IS ASSUMED THAT THE OTHERS WILL BE THE SAME
//
  BlockSize = RecSize + Shift ;
  SampPerBank = h.Samples / h.Channels ;
  SampPerBlock = (unsigned long)SampPerBank * (unsigned long)h.Banks ;
  NumRecs = SampPerBank / BlockSize ;
  Bins = RecSize/2 ;
  TotalRecs = NumFiles * h.Blocks * NumRecs ;
  delF = 1e6 / ((double)RecSize * (double)h.SPeriod) ;
  PrintGlobals() ;
//
//  MEMORY IS ALLOCATED
//
  printf (" Allocating memory ... ") ;
  error = GetMem() ;
  if (error)
   { printf ("ERROR %hu\n\n", error) ;
     return ;
   }
  printf ("O.K.\n") ;

  RecsDone = 0L ;

  for (sp=0; sp<NUM_SPEC; sp++)
   for (ch=0; ch<h.Channels; ch++)
    for (bn=0; bn<Bins; bn++)
     power[sp][ch][bn] = 0.0 ;

//
//  THIS LOOPS THROUGH THE DATA FILES LISTED IN THE INI FILE
//
  for (file=0; file<NumFiles; file++)
```

```
   {//
    //  DATA IS READ IN FROM FILE
    //
     printf (" Reading Data ...  ") ;
     error = ReadData () ;
     if (error)
      { printf ("ERROR %hu\n\n", error) ;
        return ;
      }
     printf ("O.K.\n") ;
    //
    //  SPECTRA ARE CALCULATED
    //
     printf (" Calculating ...    ") ;
     CalcPowerSpectra () ;
     printf ("O.K.\n") ;
     CloseFile(&DataF) ;
    //
    //  THE NEXT DATA FILE IS OPENNED AND THE HEADER SKIPPED
    //
     if (file < (NumFiles-1))
      { GetNextDataFname() ;
        error = OpenDataFile(0L) ;
        if (error)
         { printf (" ***ERROR SKIPPING HEADER\n\n") ;
           return ;
   }  }  }
  CloseFile(&IniF) ;

  printf (" Saving spectra ...\n") ; fflush(stdout) ;
  error = SaveSpec() ;
  if (error)
   { printf (" *** ERROR %i\n\n", error) ;
     return ;
   }

  printf ("\n             MEAN SQUARES\n");
  printf ("            ***********\n");
  for (ch=0; ch<h.Channels; ch++)
   { printf ("  Channel %02hu :  p1 = %-lg Pa%c\n", ch+1,ms[0][ch][0],253) ;
     printf ("                p2 = %-lg Pa%c\n", ms[1][ch][0], 253) ;
     printf ("             p1-p2 = %-lg Pa%c\n", ms[2][ch][0], 253) ;
     printf ("             p1+p2 = %-lg Pa%c\n", ms[3][ch][0], 253) ;
   }
  return ;
}


/***************************************************************************
*   THIS FUNCTION PROCESSES PROGRAM ARGUMENTS                              *
***************************************************************************/
void ProcessArgs (int ArgCount, char **Arg)
{
  if (ArgCount > 2)
   { strcpy (LogF.name, Arg[2]) ;
     OpenFile (&LogF, "wt") ;
   }

  if (ArgCount > 1)
   { strcpy (IniF.name, Arg[1]) ;
     OpenFile (&IniF, "rt") ;
}  }


/*****************************************************************************
*   THIS FUNCTION READS CONFIGURATION INFO FROM A CONFIGURATION FILE        *
*****************************************************************************/
unsigned short ReadConfig (void)
{
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%s", OutRoot) ;
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%lu", &RecSize) ;
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%lu", &Shift) ;
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%hu", &NumFiles) ;

  return 0 ;
}


/*****************************************************************************
*   THIS FUNCTION READS THE NEXT DATA FILENAME IN THE INI FILE AND THE      *
*       ASSOCIATED SENSITIVITIES                                             *
*****************************************************************************/
unsigned short GetNextDataFname (void)
{ unsigned short ch ;

  fscanf (IniF.ptr, "%s", DataF.name) ;
  for (ch=0; ch<h.Channels; ch++)  fscanf (IniF.ptr, "%lg", &sens[ch]) ;

  return 0 ;
}


/*****************************************************************************
*   THIS FUNCTION FREES MEMORY AND CLOSES FILES.  IT IS CALLED WHEN PROGRAM *
*       EXECUTION TERMINATES                                                 *
*****************************************************************************/
void WrapUp (void)
{ FreeMem() ;
  flushall() ;
  fcloseall() ;
}


/*****************************************************************************
*   THIS FUNCTION PRINTS THE CONFIGURATION INFORMATION                      *
*****************************************************************************/
void PrintConfig (void)
{
  printf ("\n") ;
  printf ("    Output Filename Root : %s\n", OutRoot) ;
  printf ("           Record Length = %lu\n", RecSize) ;
  printf ("                   Shift = %lu\n", Shift) ;
  printf ("         Number of Files = %hu\n", NumFiles) ;
  printf ("\n") ; fflush(stdout) ;
}

/*****************************************************************************
*   THIS FUNCTION PRINTS SOME GLOBAL VARIABLES                              *
*****************************************************************************/
```

```
void PrintGlobals (void)
{
  printf ("\n") ;
  printf ("              BlockSize = %lu\n", BlockSize) ;
  printf ("           Samples/Bank = %hu\n", SampPerBank) ;
  printf ("          Samples/Block = %lu\n", SampPerBlock) ;
  printf ("                NumRecs = %lu\n", NumRecs) ;
  printf ("                   Bins = %lu\n", Bins) ;
  printf ("              TotalRecs = %lu\n", TotalRecs) ;
  printf ("                   delF = %g\n", delF) ;
  printf ("\n") ; fflush(stdout) ;
}




/*****************************************************************************
*   THIS FUNCTION ALLOCATES MEMORY FROM THE HEAP                             *
*****************************************************************************/
unsigned short GetMem(void)
{ unsigned short  sp, ch, bl, ba ;

  scratch = (double **) calloc (NUM_SPEC, sizeof(double *)) ;
  avg     = (double ***)calloc (NUM_SPEC, sizeof(double **)) ;
  ms      = (double ***)calloc (NUM_SPEC, sizeof(double **)) ;
  FirstEst= (double ***)calloc (NUM_SPEC, sizeof(double **)) ;
  LastEst = (double ***)calloc (NUM_SPEC, sizeof(double **)) ;
  power   = (double ***)calloc (NUM_SPEC, sizeof(double **)) ;
  if (!scratch || !ms || !power)  return 1 ;

  for (sp=0; sp<NUM_SPEC; sp++)
   { scratch[sp] = (double *) calloc (RecSize, sizeof(double)) ;
     avg[sp]     = (double **)calloc (h.Channels, sizeof(double *)) ;
     ms[sp]      = (double **)calloc (h.Channels, sizeof(double *)) ;
     FirstEst[sp]= (double **)calloc (h.Channels, sizeof(double *)) ;
     LastEst[sp] = (double **)calloc (h.Channels, sizeof(double *)) ;
     power[sp]   = (double **)calloc (h.Channels, sizeof(double *)) ;
     if (!scratch[sp] || !ms[sp] || !power[sp])  return 2 ;

     for (ch=0; ch<h.Channels; ch++)
      { avg[sp][ch]   = (double *)calloc (TotalRecs, sizeof(double)) ;
        ms[sp][ch]    = (double *)calloc (TotalRecs, sizeof(double)) ;
        FirstEst[sp][ch] = (double *)calloc (TotalRecs, sizeof(double)) ;
        LastEst[sp][ch]  = (double *)calloc (TotalRecs, sizeof(double)) ;
        power[sp][ch] = (double *)calloc (Bins, sizeof(double)) ;
        if (!ms[sp][ch] || !power[sp][ch])  return 3 ;
   }  }

  Data = (unsigned short ****) calloc(h.Channels, sizeof(unsigned short ***));
  if (!Data)  return 4 ;

  for (ch=0; ch<h.Channels; ch++)
   { Data[ch]= (unsigned short ***)calloc(h.Blocks,sizeof(unsigned short **));
     if (!Data[ch])  return 5 ;

     for (bl=0; bl<h.Blocks; bl++)
      { Data[ch][bl] = (unsigned short **)calloc(h.Banks,
                                         sizeof(unsigned short *));
        if (!Data[ch][bl]) return 6 ;

        Data[ch][bl][0] = (unsigned short *)calloc(SampPerBlock,
                                                        sizeof(unsigned short)) ;
        if (!Data[ch][bl][0]) return 7 ;

        for (ba=1; ba<h.Banks; ba++)
          Data[ch][bl][ba] = Data[ch][bl][ba-1] + (unsigned long)SampPerBank ;
   }  }

  return 0 ;
}


/*****************************************************************************
*   THIS FUNCTION RETURNS MEMORY TO THE HEAP                                *
*****************************************************************************/
void FreeMem (void)
{ unsigned short    sp, ch, bl ;

  for (ch=0; ch<h.Channels; ch++)
   { for (bl=0; bl<h.Blocks; bl++)
      { free (Data[ch][bl][0]) ;
        free (Data[ch][bl]) ;
      }
     free (Data[ch]) ;
   }
  free (Data) ;

  for (sp=0; sp<NUM_SPEC; sp++)
   { for (ch=0; ch<h.Channels; ch++)
      { free (avg[sp][ch]) ;
        free (ms[sp][ch]) ;
        free (FirstEst[sp][ch]) ;
        free (LastEst[sp][ch]) ;
        free (power[sp][ch]) ;
      }
     free (scratch[sp]) ;
     free (avg[sp]) ;
     free (ms[sp]) ;
     free (FirstEst[sp]) ;
     free (LastEst[sp]) ;
     free (power[sp]) ;
   }
  free (scratch) ;
  free (avg) ;
  free (ms) ;
  free (FirstEst) ;
  free (LastEst) ;
  free (power) ;
}


/*****************************************************************************
*   THIS FUNCTION OPENS A DATA FILE AND EITHER READS THE HEADER INFO (IF A  *
*       NON-ZERO POINTER IS PASSED) MOVES THE FILE POINTER PAST THE HEADER  *
*       INFO                                                                *
*****************************************************************************/
unsigned short OpenDataFile (DataHeader *Header)
{ size_t    numr ;
  int       error ;

  error = OpenFile (&DataF, "rb") ;
  if (error) return 1 ;

  if (Header)
   { numr = fread (Header, sizeof(DataHeader), 1, DataF.ptr) ;
     if (numr != 1) return 2 ;
   }
  else
   { error = fseek (DataF.ptr, (long)sizeof(DataHeader), SEEK_SET) ;
     if (error) return 3 ;
   }

  return 0 ;
}



/*****************************************************************************
*   THIS FUNCTION READS RAW BINARY DATA FROM THE DATA FILE                   *
*****************************************************************************/
unsigned short ReadData (void)
{ unsigned short bl, ba, ch, DataPt ;
  unsigned long  numr, sa, n ;

  n=0L ;
  for (bl=0; bl<h.Blocks; bl++)
   for (ba=0; ba<h.Banks; ba++)
    {  switch (n%4)
        { case 0 : fprintf (stdout, "\b-");  break ;
          case 1 : fprintf (stdout, "\b\\"); break ;
          case 2 : fprintf (stdout, "\b|");  break ;
          case 3 : fprintf (stdout, "\b/");  break ;
        }
       fflush(stdout) ;
       n++ ;

       for (sa=0; sa<SampPerBank; sa++)
        for (ch=0; ch<h.Channels; ch++)
         { numr = fread (&DataPt, 2, 1, DataF.ptr) ;
           if (numr != 1) return errno ;
           Data[ch][bl][ba][sa] = DataPt ;
    }    }

  fprintf (stdout, "\b ");  fflush(stdout);
  return 0 ;
}



/*****************************************************************************
*   THIS FUNCTION PRINTS THE DATA FILE HEADER TO THE SCREEN                  *
*****************************************************************************/
void PrintDataHeader (void)
{ printf ("\n") ;
  printf ("                 DATA FILE HEADER\n") ;
  printf ("      Number of Channels = %-hu\n", h.Channels) ;
  printf ("           Sample Period = %-hu %cs\n", h.SPeriod, 230) ;
  printf ("        Number of Blocks = %-hu\n", h.Blocks) ;
  printf ("         Number of Banks = %-hu\n", h.Banks) ;
  printf ("        Samples per Bank = %-hu\n", h.Samples) ;
  printf ("              A/D offset = %-hu\n", h.ADoffset) ;
  printf ("               A/D slope = %-g\n\n", h.ADslope) ;
}
```

```
/*****************************************************************************
*   THIS FUNCTION CALCULATES THE POWER SPECTRA                              *
*****************************************************************************/
void CalcPowerSpectra (void)
{ unsigned short    ch, bl ;
  unsigned long     rec, sp, sa, bn, start ;
  double            raw[4] ;

  for (bl=0; bl<h.Blocks; bl++)
   for (rec=0; rec<NumRecs; rec++)
    {
    //  THIS JUST PRINTS A STATUS INDICATOR TO THE SCREEN
    //
      fprintf (stdout, "\b\b\b\b\b\b%03lu   ", RecsDone) ;
      fflush (stdout) ;

      for (ch=0; ch<h.Channels; ch++)
       { for (sp=0; sp<NUM_SPEC; sp++)     //
          { avg[sp][ch][RecsDone] = 0.0 ;  //  ZERO STATISTICS ARRAYS
            ms[sp][ch][RecsDone]  = 0.0 ;  //
          }
    //
    //  THIS JUST PRINTS A STATUS INDICATOR TO THE SCREEN
    //
         fprintf (stdout, "\b\b%02hu", ch) ;
         fflush (stdout) ;

    //
    //  THE SCRATCH ARRAYS ARE FILLED AND THE AVERAGE IS CALCULATED
    //
         bn = 0 ;
         start = rec * BlockSize ;
         for (sa=start; sa<(start+RecSize); sa++)
          { raw[0] = (double)Data[ch][bl][0][sa] ;            //  p1
            raw[1] = (double)Data[ch][bl][0][sa+Shift] ;      //  p2
            raw[2] = raw[0] - raw[1] ;                        //  p1 - p2
            raw[3] = raw[0] + raw[1] ;                        //  p1 + p2

            for (sp=0; sp<NUM_SPEC; sp++)
             { scratch[sp][bn] = raw[sp] * (double)h.ADslope * sens[ch] ;
               avg[sp][ch][RecsDone] += scratch[sp][bn] ;
             }
            bn++ ;
          }
    //
    //  THE MEAN SQUARE, FFT, AND POWER SPECTRUM ARE CALCULATED
    //
         for (sp=0; sp<NUM_SPEC; sp++)
          { avg[sp][ch][RecsDone] /= (double)RecSize ;
            for (bn=0; bn<RecSize; bn++)
             { scratch[sp][bn] -= avg[sp][ch][RecsDone] ;
               ms[sp][ch][RecsDone] += scratch[sp][bn]*scratch[sp][bn] ;
             }
            ms[sp][ch][RecsDone] /= (double)RecSize ;

            drealft (scratch[sp]-1L, RecSize, 1) ;

            FirstEst[sp][ch][RecsDone] = scratch[sp][0] ;
            LastEst[sp][ch][RecsDone] = scratch[sp][1] ;
            for (bn=0; bn<Bins; bn++)
              power[sp][ch][bn] += scratch[sp][2*bn] * scratch[sp][2*bn] +
                                   scratch[sp][2*bn+1]*scratch[sp][2*bn+1] ;
       }  }

      RecsDone++ ;
    }
  fprintf (stdout, "\b\b\b\b\b\b") ;
  printf ("%03lu %02hu ", RecsDone, ch) ;  fflush (stdout);
}


/***************************************************************************
*   THIS FUNCTION NORMALIZES THE COMPUTED POWER SPECTRA TO BE A SPECTRAL    *
*       POWER DENSITY.  IT THEN WRITES THIS TO FILE.                        *
***************************************************************************/
unsigned short SaveSpec (void)
{ unsigned short    sp, ch, error ;
  unsigned long     bn, rec ;
  char              OutExt[4] ;
  double            Denom, TrueAvg, Adjustment, Freq ;

  Denom = (double)RecSize*(double)RecSize*(double)RecsDone*delF / 2.0 ;

  for (ch=0; ch<h.Channels; ch++)
   {for (sp=0; sp<NUM_SPEC; sp++)
      {
      //    POWER SPECTRA ARE NORMALIZED SUCH THAT THEY INTEGRATE
      //        TO THE MEAN SQUARE
      //
        for (bn=0; bn<Bins; bn++)  power[sp][ch][bn] /= Denom ;
      //
      //    THE AVERAGE OF THE INDIVIDUAL BLOCK AVERAGES IS CALCULATED
      //
        TrueAvg = avg[sp][ch][0] ;
        for (rec=1; rec<RecsDone; rec++)  TrueAvg += avg[sp][ch][rec] ;
        TrueAvg /= (double)RecsDone ;
      //
      //    THE MEAN SQUARE IS CALCULATED FROM THE BLOCK MEAN SQUARES
      //        ACCOUNTING FOR THE BLOCK AVERAGES BEING DIFFERENT
      //
        Adjustment = avg[sp][ch][0] - TrueAvg ;
        ms[sp][ch][0] += Adjustment*Adjustment ;
        for (rec=1; rec<RecsDone; rec++)
         { Adjustment = avg[sp][ch][rec] - TrueAvg ;
           ms[sp][ch][0] += ms[sp][ch][rec] + Adjustment*Adjustment ;
         }
        ms[sp][ch][0] /= (double)RecsDone ;
      }
   //
   //   THE OUTPUT FILENAME IS CONSTRUCTED AND OPENNED
   //
     sprintf (OutExt, "p%-u", ch+1) ;
     _makepath (OutF.name, NULL, NULL, OutRoot, OutExt) ;
     error = OpenFile (&OutF, "wt") ;
     if (error) return 1 ;
   //
   //   COLUMN HEADINGS ARE WRITTEN TO THE OUTPUT FILE
   //
     fprintf (OutF.ptr, "      Freq     ") ;
```

```
      fprintf (OutF.ptr, "      p1      ") ;
      fprintf (OutF.ptr, "      p2      ") ;
      fprintf (OutF.ptr, "     p1-p2    ") ;
      fprintf (OutF.ptr, "     p1+p2\n") ;
    //
    //   ADJACENT BINS ARE AVERAGED AND WRIITEN TO FILE. THROUGH TRIAL AND
    //       ERROR IT WAS FOUND THAT THIS SIGNIFICANTLY SMOOTHED THE SPECTRUM
    //
      Freq = delF ;
      for (bn=1; bn<Bins; bn++)
       { fprintf (OutF.ptr, " % 13.5le", Freq) ;
         for (sp=0; sp<NUM_SPEC; sp++)
           fprintf (OutF.ptr, " % 13.5le", power[sp][ch][bn]) ;

         fprintf (OutF.ptr, "\n") ;
         Freq += delF ;
       }

      CloseFile (&OutF) ;

      sprintf (OutExt, "m%-u", ch+1) ;
      _makepath (OutF.name, NULL, NULL, OutRoot, OutExt) ;
      error = OpenFile (&OutF, "wt") ;
      if (error) return 1 ;
     //
    //   COLUMN HEADINGS ARE WRITTEN TO THE OUTPUT FILE
    //
      fprintf (OutF.ptr, "  Record ") ;
      fprintf (OutF.ptr, "   FirstEst   ") ;
      fprintf (OutF.ptr, "   LastEst\n") ;

      for (rec=0; rec<RecsDone; rec++)
       { fprintf (OutF.ptr, " % 6lu ", rec) ;
         fprintf (OutF.ptr, " % 13.5le", FirstEst[3][ch][rec]) ;
         fprintf (OutF.ptr, " % 13.5le\n", LastEst[3][ch][rec]) ;
       }
      CloseFile (&OutF) ;
    }
   return 0 ;
 }
```

DATHEAD.H

```
#ifndef _DATHEAD_H_
#define _DATHEAD_H_
/****************************************************************************
*   THIS FILE DEFINES THE DATA FILE HEADER STRUCTURE                        *
****************************************************************************/
//
//  Data File Header Structure
//
typedef struct
 {  unsigned short  Channels,
                    SPeriod,
                    Blocks,
                    Banks,
                    Samples,
                    ADoffset ;
    float           ADslope ;
 }
DataHeader ;

#endif
```

FILE-OPS.H

```
#ifndef _FILE_OPS_H_
#define _FILE_OPS_H_
/****************************************************************************
*   THIS FILE CONTAINS DEFINITIONS USED FOR FILE I/O                        *
****************************************************************************/

#include <stdio.h>
#include <errno.h>

typedef struct
 { FILE             *ptr ;
   char             name[80] ;
   unsigned short   loaded ;
 }
FileInfo ;

extern unsigned short OpenFile (FileInfo *f, char *mode) ;
extern void CloseFile (FileInfo *f) ;
extern void GotoChar (FileInfo *f, char c) ;
extern void SkipLine (FileInfo *f) ;

#endif
```

FILE-OPS.C

```
/***************************************************************************
*   THIS FILE CONTAINS FUNCTION DECLARATIONS USED FOR FILE I/O            *
***************************************************************************/

#include "file-ops.h"

/***************************************************************************
*   THIS FUNCTION OPENS A FILE                                            *
***************************************************************************/
unsigned short OpenFile (FileInfo *f, char *mode)
{
  printf ("  Openning %s ... ", f->name) ;

  if ((f->ptr = fopen (f->name, mode)) == NULL)
   { printf ("ERROR #%i", errno) ;
     f->loaded = 0 ;
     return 1 ;
   }

 printf ("O.K.\n") ;
 f->loaded = 1 ;

 return 0 ;
}

/***************************************************************************
*   THIS FUNCTION CLOSES A FILE                                           *
***************************************************************************/
void CloseFile (FileInfo *f)
{
  fclose (f->ptr) ;
  f->loaded = 0 ;
}

/***************************************************************************
*   THIS FUNCTION MOVES THE GIVEN ASCII FILE POINTER TO THE GIVEN CHARACTER *
***************************************************************************/
void GotoChar (FileInfo *f, char c)
{ char format[] = "%*[^ ]%*[ ]" ;


  format[4] = format [9] = c ;
  fscanf (f->ptr, format) ;
}

/***************************************************************************
*   THIS FUNCTION ADVANCES THE FILE POINTER PAST THE NEXT LINE OF AN ASCII  *
*       FILE                                                              *
***************************************************************************/
void SkipLine (FileInfo *f)
{ char buffer[256] ;
  fgets (buffer, 256, f->ptr) ;
}
```

LOG-OPS.H

```
#ifndef _LOG_OPS_H_
#define _LOG_OPS_H_
/***************************************************************************
*   THIS FILE CONTAINS FUNCTION DECLARATIONS FOR LOGGING SCREEN I/O         *
***************************************************************************/

#include <stdio.h>
#include <stdarg.h>

extern FILE *LogFptr ;

extern int scanf (const char *format, ...) ;
extern int printf (const char *format, ...) ;

#endif
```

LOG-OPS.C

```
/***************************************************************************
*   THIS FILE CONTAINS FUNCTION DEFINITIONS FOR LOGGING SCREEN I/O          *
***************************************************************************/

#include "log-ops.h"

FILE *LogFptr = 0L ;

/***************************************************************************
*   THIS FUNCTION REMAPS THE SCREEN INPUT THAT IS TRANSFERRED USING SCANF   *
*       TO ALSO PRINT A COPY TO THE LOGFILE IF ONE IS OPEN                  *
***************************************************************************/
int scanf (const char *format, ...)
{ va_list   args ;
  char      buffer[80] ;
  int       result ;

  va_start (args, format) ;

  gets (buffer) ;
  result = vsscanf (buffer, format, args) ;
  va_end (args) ;

  if (LogFptr)
   { va_start (args, format) ;
     fprintf (LogFptr, "%s", buffer) ;
     fprintf (LogFptr, "\n") ;
     va_end (args) ;
     fflush (LogFptr) ;
   }

  return result ;
}
```

```
/****************************************************************************
*   THIS FUNCTION REMAPS THE SCREEN OUTPUT THAT IS TRANSFERRED USING PRINTF *
*       TO ALSO PRINT TO A LOG FILE IF ONE IS OPEN                          *
****************************************************************************/
int printf (const char *format, ...)
{ va_list   args ;
  int       result ;

  va_start (args, format) ;

  result = vprintf (format, args) ;
  va_end (args) ;

  if (LogFptr)
   { va_start (args, format) ;
     vfprintf (LogFptr, format, args) ;
     va_end (args) ;
     fflush (LogFptr) ;
   }

  return result ;
}
```

REALFFT.H

```
#ifndef _REALFFT_H_
#define _REALFFT_H_
/****************************************************************************
*   THIS IS THE HEADER FILE FOR FFT FUNCTIONS                               *
****************************************************************************/
#include <math.h>

extern void drealft (double data[], unsigned long nn, int isign) ;

#endif
```

REALFFT.C

```
#include "realfft.h"

#define SWAP(a,b) tempr=(a); (a)=(b); (b)=tempr

/*****************************************************************************
*   This function calculates the fast Fourier Transform of a time series.    *
*     Taken from Numerical Recipes in C.                                     *
*****************************************************************************/

void four1(double data[], unsigned long nn, int isign)
{
   unsigned long n,mmax,m,j,istep,i;
   long double wtemp,wr,wpr,wpi,wi,theta;
   double tempr,tempi;
   n=nn << 1;
   j=1;
   for (i=1;i<n;i+=2) {
      if (j > i) {
            SWAP(data[j],data[i]);
            SWAP(data[j+1],data[i+1]);
      }
      m=n >> 1;
      while (m >= 2 && j > m) {
            j -= m;
            m >>= 1;
      }
      j += m;
   }
   mmax=2;
   while (n > mmax) {
      istep=mmax << 1;
      theta=isign*(6.28318530717959/mmax);
      wtemp=sin(0.5*theta);
      wpr = -2.0*wtemp*wtemp;
      wpi=sin(theta);
      wr=1.0;
      wi=0.0;
      for (m=1;m<mmax;m+=2) {
            for (i=m;i<=n;i+=istep) {
                  j=i+mmax;
                  tempr=wr*data[j]-wi*data[j+1];
                  tempi=wr*data[j+1]+wi*data[j];
                  data[j]=data[i]-tempr;
                  data[j+1]=data[i+1]-tempi;
                  data[i] += tempr;
                  data[i+1] += tempi;
            }
            wr=(wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
      }
      mmax=istep;
   }
}
#undef SWAP


/******************************************************************************
*   This function uses FOUR1 to calculate the Fourier Transform of a set of  *
*     n real-valued data points.  Replaces this data by the POSITIVE         *
*     frequency half of its complex Fourier Transform.  The real-valued      *
*     first and last components of the complex transform are returned as     *
*     elements data[1] and data[2], respectively.                            *
******************************************************************************/

void drealft(double data[], unsigned long n, int isign)
{
   void four1(double data[], unsigned long nn, int isign);
   unsigned long i,i1,i2,i3,i4,np3;
   double c1=0.5,c2,h1r,h1i,h2r,h2i;
   long double wr,wi,wpr,wpi,wtemp,theta;

   theta=3.141592653589793/(long double) (n>>1);
   if (isign == 1) {
      c2 = -0.5;
      four1(data,n>>1,1);
   } else {
      c2=0.5;
      theta = -theta;
   }
   wtemp=sin(0.5*theta);
   wpr = -2.0*wtemp*wtemp;
   wpi=sin(theta);
   wr=1.0+wpr;
   wi=wpi;
   np3=n+3;
   for (i=2;i<=(n>>2);i++) {
      i4=1+(i3=np3-(i2=1+(i1=i+i-1)));
      h1r=c1*(data[i1]+data[i3]);
      h1i=c1*(data[i2]-data[i4]);
      h2r = -c2*(data[i2]+data[i4]);
      h2i=c2*(data[i1]-data[i3]);
      data[i1]=h1r+wr*h2r-wi*h2i;
      data[i2]=h1i+wr*h2i+wi*h2r;
      data[i3]=h1r-wr*h2r+wi*h2i;
      data[i4] = -h1i+wr*h2i+wi*h2r;
      wr=(wtemp=wr)*wpr-wi*wpi+wr;
      wi=wi*wpr+wtemp*wpi+wi;
   }
   if (isign == 1) {
      data[1] = (h1r=data[1])+data[2];
      data[2] = h1r-data[2];
   } else {
      data[1]=c1*((h1r=data[1])+data[2]);
      data[2]=c1*(h1r-data[2]);
      four1(data,n>>1,-1);
   }
}
```