B.1. Input Files

The program BIN-AV.EXE is a program that was compiled using Watcom C/C++ v.10 to work in a DOS environment. It reads an ASCII file that nominally contains a spectral power density. BIN-AV.EXE creates another ASCII file with the spectral estimates (data points) spaced differently. The program was written in order to decrease the data file size and to prevent the spectral estimates from being bunched together at high frequencies when plotted on a logarithmically-spaced frequency axis. It does this by making multiple passes through the data file. Each pass starts at a user-specified frequency. Adjacent spectral estimates at frequencies that are higher than the user-specified frequency are averaged. Then, the center frequency of the new, averaged spectral estimate is adjusted accordingly in order to satisfy Parseval's theorem and preserve the mean square integral. 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:

```
BIN-AV.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,

```
 Input data file = E:\DATA\P-SPEC.P1
Output data file = E:\DATA\P-SPEC.AV
        StartRow = 2
         NumRows = 8191
         NumCols = 5
         Key Col = 0
   NumBoundaries = 10
 Boundaries (Hz) =  50
                   100
                   300
                   500
                   700
                  1000
                  2000
                  4000
                  5000
                  7000
```

In the above example, the *Input data file* and *Output data file* are self-explanatory. The *StartRow* is the row number of the input ASCII data file that the program starts reading data from. In the above example, *StartRow* = 2. So, BIN-AV will transfer the first line of the input data file directly to the output data file, then start reading data values beginning at row 2 of the input data file. This feature is intended to handle column names or other header-type information. *NumRows* and *NumCols* specify the organization of the data. *NumRows* is the number of rows **of data**. For example, an input data file may have 8192 lines, however, if the first line contains column labels, then there are only 8191 rows of data. Therefore, *NumRows* would be 8191. *KeyCol* is the column number that contains the frequency of each spectral estimate. For the *KeyCol* **only**, the numbering starts with zero– the first column is column 0, the second column is column 1, etc. *KeyCol* specifies what column will be searched by BIN-AV in order to find the boundary values which are specified by *NumBoundaries* and *Boundaries (Hz)*. The *Boundaries (Hz)* values specify the start frequencies for each successive pass of BIN-AV through the input data file to average adjacent bins. Using the above input file, BIN-AV would first find the first spectral estimate that is at a frequency that is not less than 50. Then, adjacent spectral estimates at higher frequencies would be averaged. Next, BIN-AV would find the first spectral estimate that is at a frequency that is not less than 100, and adjacent spectral estimates at higher frequencies would be averaged. This process would be carried out for each *boundary* frequency.

B.2. Output files

The output data file is the only output file of this program. The format of the output data file is identical to the format of the input data file.

B.3. Program Listings

The source code for BIN-AV.EXE is split up into 5 files:

`BIN-AV.C`	Main functions of the program
`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.

BIN-AV.C

```
/*************************************************************************
*   THIS PROGRAM PERFORMS A BIN AVERAGE ON AN INPUT ASCII SPECTRUM FILE.    *
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <graph.h>
#include <time.h>

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

#define MAX_LINE_SIZE   256

FileInfo        IniF, LogF, InSpecF, OutSpecF ;
unsigned long   StartRow, SkipLines, NumRows ;
unsigned short  NumCols, KeyCol, NumBoundaries ;
double          **InSpec, *Boundary ;
char            **SkipRow ;

void ProcessArgs (int ArgCount, char **Arg) ;
unsigned short ReadConfig (void) ;
void           PrintConfig (void) ;
unsigned short GetMem(void) ;
void           FreeMem (void) ;
unsigned short ReadInSpec (void) ;
unsigned long  MakeEven (unsigned long Number) ;
unsigned short DoAveraging (void) ;
unsigned short WriteOutSpec (void) ;

void main (int ArgCount, char **Arg)
{ unsigned short    error ;
  time_t            TheTime ;

  _clearscreen(_GCLEARSCREEN) ;
  printf ("\n\n    STARTING BIN AVERAGE PROGRAM\n\n") ;
  if (ArgCount < 2)
   { printf ("  Usage : bin-av <Ini Filename> <Log Filename>\n") ;
     printf ("                    (required)     (optional)\n\n") ;
   }
  atexit (FreeMem) ;

  ProcessArgs(ArgCount, Arg) ;
  if (!IniF.loaded) return ;
  LogFptr = LogF.ptr ;
```

```
  time (&TheTime) ;
  printf ("\n\n    STARTING BIN AVERAGE PROGRAM : %s\n\n", ctime(&TheTime)) ;

  printf (" Reading Configuration ... ") ;
  error = ReadConfig() ;
  if (error)
   { printf ("ERROR %hu\n", error) ;
     return ;
   }
  printf ("O.K.\n") ;
  PrintConfig() ;

  printf (" Allocating memory ... ") ;
  error = GetMem() ;
  if (error)
   { printf ("ERROR %hu\n", error) ;
     return ;
   }
  printf ("O.K.\n") ;

  printf (" Reading spectra ...\n") ;
  error = ReadInSpec () ;
  if (error)
   { printf (" READING ERROR %hu\n", error) ;
     return ;
   }
  printf ("\n") ;

  printf (" Calculating ... ") ;
  error = DoAveraging() ;
  if (error)
   { printf ("ERROR %hu\n", error) ;
     return ;
   }
  printf ("O.K.\n") ;

  printf (" Saving bin-averaged spectra ...\n") ;
  error = WriteOutSpec () ;
  if (error)
   { printf (" WRITING ERROR %hu\n", error) ;
     return ;
   }

  time (&TheTime) ;
  printf ("\n   SUCCESSFUL COMPLETION : %s", ctime(&TheTime)) ;
  printf ("                            New NumRows = %lu\n\n", NumRows) ;
}



/*************************************************************************
*   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)
{ unsigned short b ;

  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%s", InSpecF.name) ;
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%s", OutSpecF.name) ;
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%lu", &StartRow) ;
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%lu", &NumRows) ;
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%hu", &NumCols) ;
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%hu", &KeyCol) ;
  GotoChar (&IniF, '=') ;  fscanf (IniF.ptr, "%hu", &NumBoundaries) ;

  SkipLines = StartRow - 1 ;

  Boundary = (double *)calloc(NumBoundaries, sizeof(double)) ;
  if (!Boundary)  return 1 ;

  GotoChar (&IniF, '=') ;
  for (b=0; b<NumBoundaries; b++)
    fscanf (IniF.ptr, "%lg", &(Boundary[b])) ;

  return 0 ;
}


/*************************************************************************
*   THIS FUNCTION PRINTS THE CONFIGURATION INFORMATION                   *
*************************************************************************/
void PrintConfig (void)
{ unsigned short b ;

  printf ("\n") ;
  printf ("\t\t       Input Filename : %s\n", InSpecF.name) ;
  printf ("\t\t      Output Filename : %s\n", OutSpecF.name) ;
  printf ("\t\t            Start Row = %lu\n", StartRow) ;
  printf ("\t\t       Number of Rows = %lu\n", NumRows) ;
  printf ("\t\t    Number of Columns = %hu\n", NumCols) ;
  printf ("\t\t           Key Column = %hu\n", KeyCol) ;
  printf ("\t\t Number of Boundaries = %hu\n", NumBoundaries) ;
  for (b=0; b<NumBoundaries; b++)
    printf ("\t\t          Boundary %02hu : %lg\n", b+1, Boundary[b]) ;
  printf ("\n") ; fflush(stdout) ;
}


/*************************************************************************
*   THIS FUNCTION ALLOCATES MEMORY FROM THE HEAP                         *
*************************************************************************/
unsigned short GetMem(void)
{ unsigned long     TotalSize = NumRows * NumCols ;
  unsigned short    row, col ;
```

```
  InSpec = (double **) calloc (NumCols, sizeof (double *)) ;
  if (!InSpec)  return 1 ;

  InSpec[0] = (double *) calloc (TotalSize, sizeof (double)) ;
  if (!InSpec[0])  return 2 ;

  for (col=1; col<NumCols; col++)
    InSpec[col] = InSpec[col-1] + NumRows ;

  if (SkipLines)
   { SkipRow = (char **) calloc (SkipLines, sizeof(char *)) ;
     if (!SkipRow)  return 3 ;

     SkipRow[0] = (char *) calloc (SkipLines*(MAX_LINE_SIZE+1), sizeof(char));
     if (!SkipRow[0])  return 4 ;

     for (row=1; row<SkipLines; row++)
       SkipRow[row] = SkipRow[row-1] + (MAX_LINE_SIZE+1L) ;
   }

  return 0 ;
}


/*************************************************************************
*   THIS FUNCTION RETURNS MEMORY TO THE HEAP                             *
*************************************************************************/
void FreeMem (void)
{ free (Boundary) ;
  free (InSpec[0]) ;
  free (InSpec) ;
  free (SkipRow[0]) ;
  free (SkipRow) ;
}


/*************************************************************************
*   THIS FUNCTION READS THE INPUT SPECTRA                                *
*************************************************************************/
unsigned short ReadInSpec (void)
{ unsigned long     line, row, numr ;
  unsigned short    error, col ;

  error = OpenFile (&InSpecF, "rt") ;
  if (error) return 1 ;

  for (line = 0; line < SkipLines; line++)
   if (fgets(SkipRow[line], MAX_LINE_SIZE, InSpecF.ptr) == NULL) return 2 ;

  for (row = 0; row < NumRows; row++)
   for (col = 0; col < NumCols; col++)
    { numr = fscanf (InSpecF.ptr, "%lg", &(InSpec[col][row])) ;
      if (numr != 1)  return 3 ;
    }

  CloseFile(&InSpecF) ;
  return 0 ;
}
```

```
/*************************************************************************
*   THIS FUNCTION IS PASSED A NUMBER.  IT TESTS IF THE NUMBER IS EVEN. IF   *
*       THE NUMBER IS NOT EVEN THE RETURNED VALUE IS THE NUMBER MINUS ONE   *
*       IN ORDER TO MAKE IT EVEN                                            *
*************************************************************************/
unsigned long MakeEven (unsigned long Number)
{ unsigned long Left, Right ;

  Left  = Number >> 1 ;
  Right = Left << 1 ;

  if (Right == Number) return Number ;

  return Number-- ;
}



/*************************************************************************
*   THIS FUNCTION DOES THE BIN AVERAGING                                  *
*************************************************************************/
unsigned short DoAveraging (void)
{
  unsigned short    avg, c ;
  unsigned long     r, FirstRow, NewNumRows, NewIndex ;

  NewNumRows = MakeEven(NumRows) ;

  for (avg=0; avg<NumBoundaries; avg++)
   { FirstRow = 0L ;
     while (InSpec[KeyCol][FirstRow] < Boundary[avg])
      { FirstRow++ ;
        if (FirstRow > NewNumRows)
         { printf ("%lu %lg ", NewNumRows, Boundary[avg]);
           return 1 ;
      }  }

     NewIndex = MakeEven(FirstRow) ;

     for (r=NewIndex; r < NewNumRows; r+=2L)
      { for (c=0; c<NumCols; c++)
         InSpec[c][NewIndex] = 0.5 * (InSpec[c][r] + InSpec[c][r+1L]) ;
        NewIndex++ ;
      }
     NewIndex-- ;
     NewNumRows = NewIndex ;
   }

  NewIndex = MakeEven(NumRows) ;
  if (NewIndex != NumRows)
   { for (c=0; c<NumCols; c++)
       InSpec[c][NewNumRows] = InSpec[c][NewIndex] ;
     NewNumRows++ ;
   }

  NumRows = NewNumRows ;
  return 0 ;
}
```

```
/*************************************************************************
*   THIS FUNCTION WRITES THE OUTPUT SPECTRA                              *
*************************************************************************/
unsigned short WriteOutSpec (void)
{ unsigned long     row ;
  int               numw ;
  unsigned short    error, col ;

  error = OpenFile (&OutSpecF, "wt") ;
  if (error) return 1 ;

  for (row = 0; row < SkipLines; row++)
   { numw = fprintf (OutSpecF.ptr, "%s", SkipRow[row]) ;
     if (numw < 0)  return 2 ;
   }

  for (row = 0; row < NumRows; row++)
   { for (col = 0; col < NumCols; col++)
      { numw = fprintf (OutSpecF.ptr, " % 13.5le", InSpec[col][row]) ;
        if (numw < 0)  return 3 ;
      }
     fprintf (OutSpecF.ptr, "\n") ;
   }

  CloseFile(&OutSpecF) ;
  return 0 ;
}
```

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) ;
   }

  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) ;
   }

  return result ;
}
```