New Folder Name The Spectrum Analyzer

LIGO T900039-00-R

Interoffice Memorandum

CALIFORNIA INSTITUTE OF TECHNOLOGY

To:	All			**Date:**	October 25, 1990
From:	Yekta Gürsel	**Ext:**	2136	**Mail Code:**	130-33

Subject: **Lab Computer and The Spectrum Analyzer (PART II)**

This is a short memorandum describing the programs which are developed to enable the HP 3563A (or HP 3562A) Spectrum Analyzer to communicate with the laboratory computer. The data taken with the analyzer can be transferred to the laboratory computer in an easy-to-understand format. The transferred data can then be analyzed and plotted by anyone who has access to our LIGO network of computers.

What programs are available?

Currently, there are two programs installed on our computer. One of these programs is called "save_spect". This program transfers the data displayed on the spectrum analyzer into the laboratory computer and it stores the data in a user-readable file. The other program is called "extract". This program analyzes the file generated by the "save_spect" program and generates files to drive our graphics program called "sm". Using these two programs, one can get the data from the spectrum analyzer and have them plotted on the laser-printer within a few minutes.

How easy are they to use?

They are designed to be VERY easy to use. For example, to get a displayed spectrum from the spectrum analyzer, one simply runs the program "save_spect". The program ~~then~~ asks for a file name to put the data in. After this file name is given, the program then asks whether the user wants to put a comment line in the data file. After the input of the comment line, the data transfer automatically takes place and the program stops running. The total elapsed time is less than 30 seconds.

The print-out of the data is also very easy with the "extract" program. When one runs "extract", it first asks the name of the file which contains the spectrum analyzer data. After this file name is entered, the program asks for a file name to put the plot data. Then, it asks for a file name to store the plotting program commands. After this final entry, the program completes the analysis and stops running.

To get the plots printed on the laser-printer, one proceeds in the following way: Assume that the plotting program commands are stored in a file called “smfile.m”. To get the plots printed all one has to do is to give the command: “sm -m smfile.m”. The plot will be printed on the laser printer.

Where are these programs?

The “executable” versions of these programs are stored in the “/usr/local/bin” directory on the laboratory computer and they are called “save_spect” and “extract”. The source codes of the programs are stored in the “/usr/local/src/gpib” directory of the laboratory computer and they are called “save_spect.c” and “extract.f”. Note that “save_spect.c” is written in the language “C”, but “extract.f” is written in the language “FORTRAN”. The program “extract” can be compiled to run on our SUN computers, but “save_spect” can only run on the laboratory computer as only that machine has the necessary hardware to accomplish the data transfers. This is not a limitation since the data files can be transferred among all of our computers once they are stored on the laboratory computer.

How do we get these installed at MIT?

That is also extremely easy. Copy the files “extract.f”, “save_spect.c” and “Makefile” in the directory “/usr/local/src/gpib” directory of the laboratory computer to a directory on the MIT laboratory computer. Edit the “Makefile” to change “-DWEST_COAST” into “-DEAST_COAST”. If the GPIB address of the spectrum analyzer is different from “20”, also change “-DEQUIPM_ADDRESS=20” into “-DEQUIPM_ADDRESS=XX” where XX is the GPIB address of the spectrum analyzer at MIT. Then, execute the command “make all”. The programs “save_spect” and “extract” will be automatically generated.

Are there any nitty-gritty details we have to know??

There are only two of them. When you type a file name in response to the file name request from the “extract” program to store the “sm” commands, you should type a file name with a length less than or equal to 12 characters. The program will then generate a file with a name equal to the name you typed in with “.m” appended at the end. You should then feed this new file name to the “sm” program. For example, if th efile name you typed in is “smfile”, you execute the “sm” program by typing “sm -m smfile.m”. Also, make sure that the “.sm” file in your directory does not have a line like “device x11”. If it has, delete it.

The other important thing to remember is that the “save_spect” program saves only the DISPLAYED data on the active trace of the spectrum analyzer. This means that if the function you invoke generates both real and imaginary parts of the spectrum (as in a transfer function), you have to display these parts separately and save them. For example, you can display the magnitude and save it, then you can display the phase and save it. Alternatively, you can display the real part and save it, and then you can display the imaginary part and save it as well. Since the power spectra are real, they have to be saved only once.

Should we use these programs now?

Yes, by all means. You have nothing to lose by using them. You can still store your important spectra using the old ways as well until your confidence builds up. Please send your comments to me in the form of electronic mail. I will improve the programs based on these comments. But, in order for your comments to be really useful, you should use these programs. That way, they will get thoroughly tested. I do not have the time to sit and go through every possible state of the spectrum analyzer as this will take weeks.

Are there any examples of output?

Yes. Examples of the output, as well as the full source code listing of the programs are given at the end of this document. The listing labeled as "testfile" shows the typical file generated by the "save_spect" program. Most of the trace data are removed to keep the file short. The listing labeled as "datafile" is the plot data generated from "testfile" by the "extract" program. Again, not all the trace data are shown. The listing labeled as "smfile.m" shows the "sm" command generated by the "extract" program. It is shown in its entirety. The next plot is generated by the command "sm -m smfile.m". The other plots are similar test plots. The listing labeled as "Makefile" is the "Makefile" listing used in the installation. This is followed by the source code listings of "save_spect.c" and "extract.f"

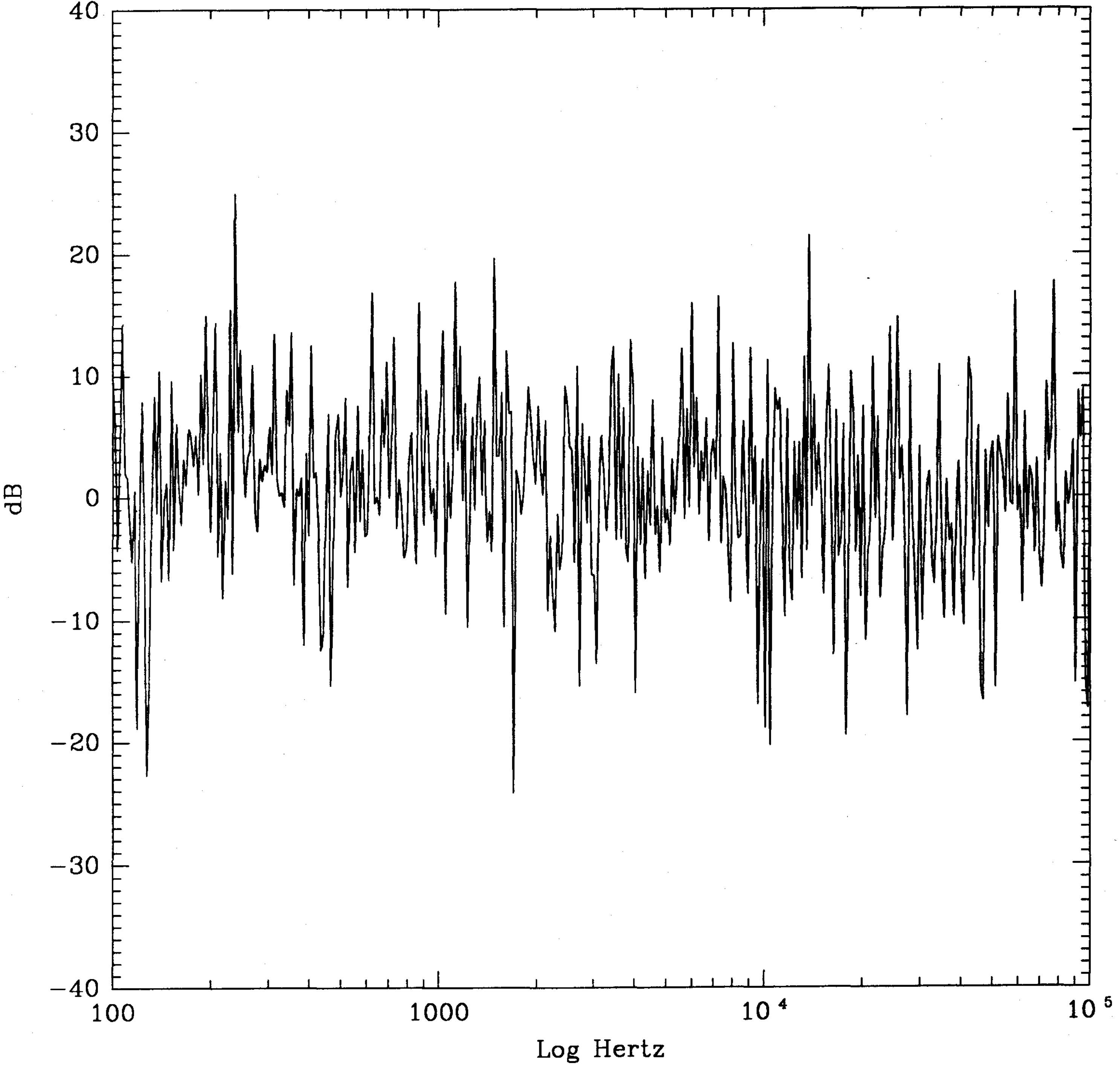
```
**DEVICE: HP 3563A**
**DATE AND TIME: Wed Oct 24 22:17:23 1990 PDT**
**COMMENTS: This is a test file.
**COORDINATE HEADER**
Y coordinates: dB
Number of display elements: 801
First element: 0
Total number of elements: 801
Display sampling: Not sampled (Number of displayed elements = Total number of elements)
Scaling: X and Y are fixed scale.
Data pointer: 6
In data: 6
Log or Linear x-axis? ( 1 or 0): 1
Sampled display data? (1=yes, 0=no): 0
Plot or graph mode? (1 or 0): 0
Phase wrap? (1=yes, 0=no) : 0
X scale factor: 1.000000
Grid minimum Y scale: -6.643854
Grid maximum Y scale: 6.643854
Y amount per division: 1.660964
Minimum value of data: -4.015350
Maximum value of data: 4.141708
Y cumulative minimum: 0.000000
Y cumulative maximum: 16.794922
Y scale factor: 6.020599
Stop value: 99999.694455
Left grid limit: 100.000024
Right grid limit: 99999.694455
Left data limit: 100.000024
Right data limit: 99999.694455
**END OF COORDINATE HEADER**
**DATA HEADER**
Displayed function: Frequency response
Number of elements: 801
Number of displayed elements: 801
Number of averages: 10
Channel selection: Channel 1 & 2
Overload status: No channels
Overlap percentage: 0
Domain: Frequency
Volts (peak or RMS): Volts (indicates peak only)
Amplitude units: No amplitude units
X-axis units: Hertz
Auto math label: AUTO MATH
Trace label:
EU label on active trace:
EU label on other trace:
Float or Integer? (1 or 0): 1
Complex or real? (1 or 0): 1
Live or recalled? (1 or 0): 1
Result of math operation? (1=yes, 0=no): 0
Real or complex input? (1 or 0): 0
Logarithmic or linear data? (1 or 0): 1
Auto Math? (1=yes, 0=no): 0
Real Time status? (1=yes, 0=no): 0
Measurement mode: Swept sine
Window type: Uniform
Channel 1 demodulator type: AM
Channel 2 demodulator type: AM
Channel 1 demodulator active? (1=yes, 0=no): 0
Channel 2 demodulator active? (1=yes, 0=no): 0
Average status: Averaged
(Sampling freq)/2 (real): 0.000000
(Sampling freq)/2 (imaginary): 800.000000
```

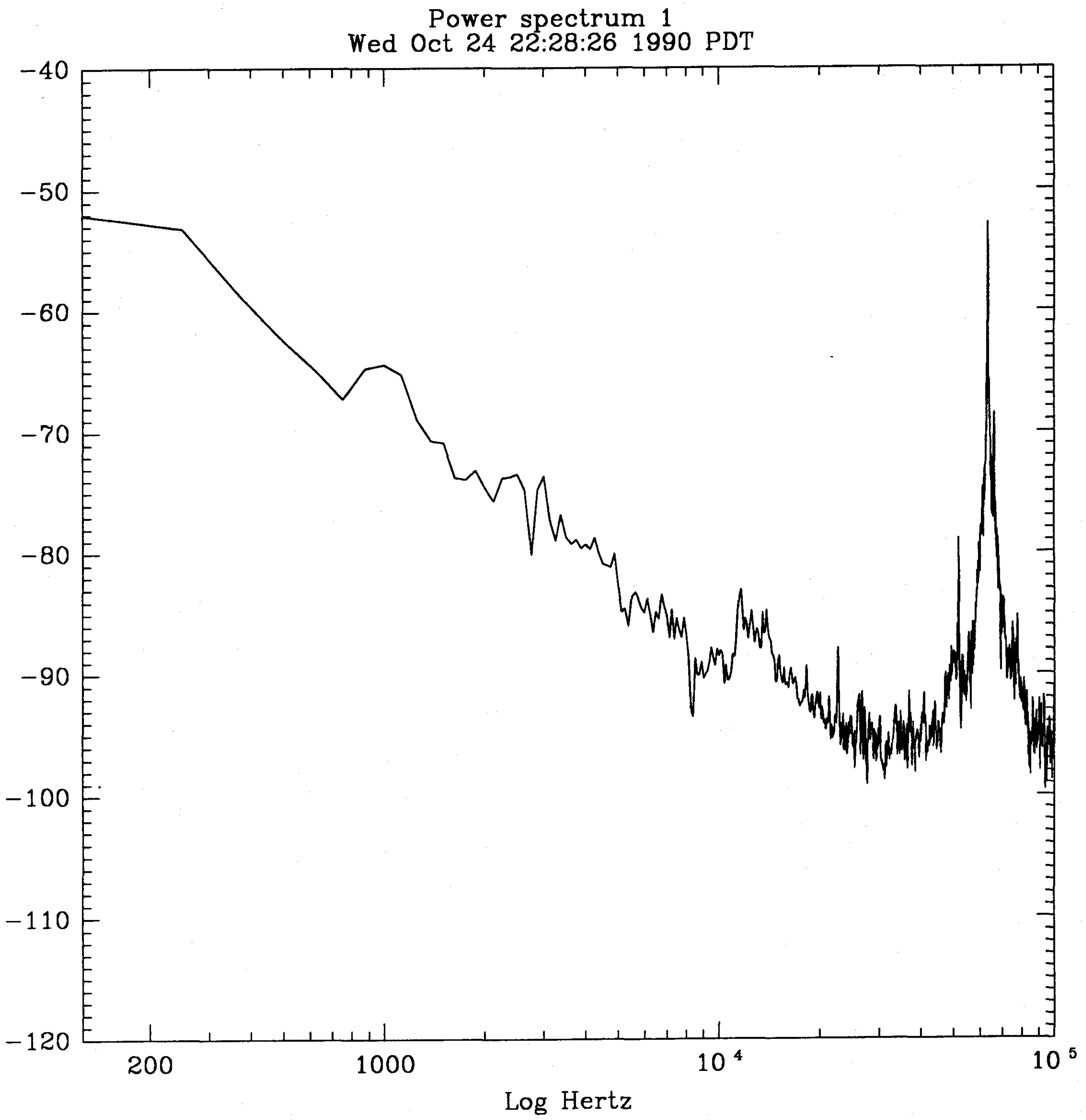
```
Delta X-axis: 0.003750
Maximum range (for scaling): 1.000000
Start time value: 0.000000
Exponential window constant 1: 0.000000
Exponential window constant 2: 0.000000
EU value (channel 1): 1.000000
EU value (channel 2): 1.000000
Trigger delay (channel 1): 0.000000
Trigger delay (channel 2): 0.000000
Start frequency value: 100.000024
Start data value: 100.000024
**END OF DATA HEADER**
**START OF SCALED DATA**
 2.000000   9.879292
 2.003750   7.784447
 2.007500   5.690429
 2.011250   0.691575
 2.015000   -4.307371
 2.018750   -0.076433
 .
 . A lot of data removed
 .
 4.969999   4.853888
 4.973749   6.835277
 4.977499   8.816849
 4.981249   -2.896477
 4.984999   -14.609710
 4.988749   -15.957215
 4.992499   -17.304905
 4.996249   -4.255558
 4.999999   8.793882
**END OF DATA**
```

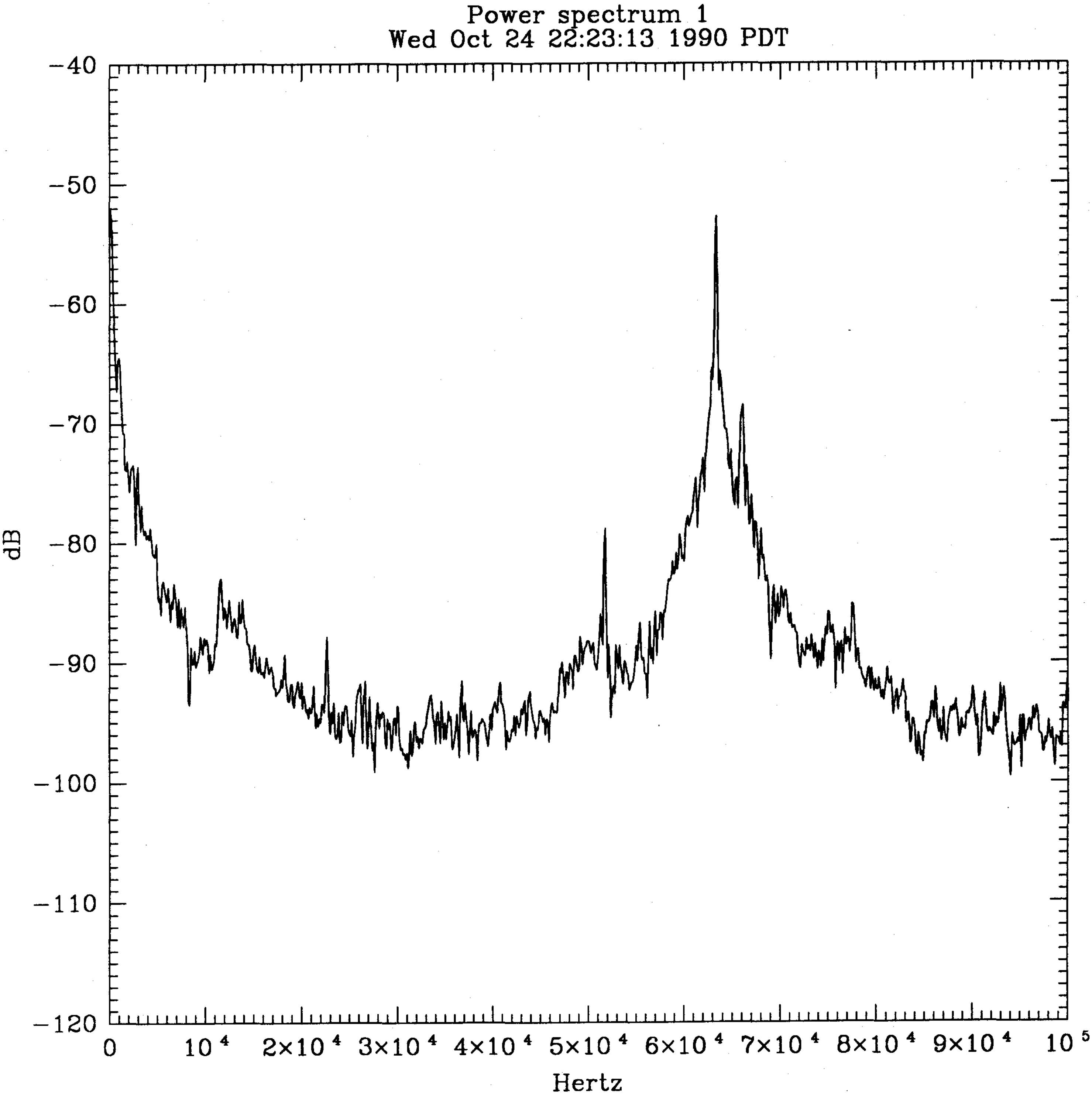
```
2.000000   9.879292
2.003750   7.784447
2.007500   5.690429
2.011250   0.691575
2.015000   -4.307371
2.018750   -0.076433
.
. A lot of data removed
.
4.969999   4.853888
4.973749   6.835277
4.977499   8.816849
4.981249   -2.896477
4.984999   -14.609710
4.988749   -15.957215
4.992499   -17.304905
4.996249   -4.255558
4.999999   8.793882
```

```
smfile
  device postscript
  expand 1.0001
  limits     2.0    5.0  -40.0   40.0
  ticksize  -.10E+01 0.10E+02 0.10E+01 0.10E+02
  box
  xlabel Log Hertz
  ylabel dB
  relocate     3.5   43.2
  putlabel 8 Frequency response
  relocate     3.5   40.5
  putlabel 8 Wed Oct 24 22:17:23 1990 PDT
  data datafile
  read x 1
  read y 2
  connect x y
  hardcopy
  quit
```

Frequency response
Wed Oct 24 22:17:23 1990 PDT




Power spectrum 1
Wed Oct 24 22:28:26 1990 PDT
dB
−40
−50
−60
−70
−80
−90
−100
−110
−120
200
1000
10^4
10^5
Log Hertz


Power spectrum 1
Wed Oct 24 22:23:13 1990 PDT
−40
−50
−60
−70
−80
−90
−100
−110
−120
dB
0
10^4
2×10^4
3×10^4
4×10^4
5×10^4
6×10^4
7×10^4
8×10^4
9×10^4
10^5
Hertz

```
all : save_spect extract

extract : extract.f
        f77 -u -o extract extract.f
        rm -f extract.o

save_spect : save_spect.c
        cc -DWEST_COAST -DEQUIPM_ADDRESS=20 save_spect.c -o save_spect -lmr -lm -fpp
```

```
#include <mr.h>
#include <math.h>
#include <stdio.h>
#include <time.h>

#define EQUIPM_NAME "HP 3563A"
#define ONLY_CNTRLR_LISTENS 0
#define GPIB_PATH "/dev/dacp0/gpib0"

#ifdef WEST_COAST
#define STANDARD_TIME "PST"
#define DAYLIGHT_TIME "PDT"
#endif

#ifdef EAST_COAST
#define STANDARD_TIME "EST"
#define DAYLIGHT_TIME "EDT"
#endif

int number_of_elem;
int complex_data;
int logarithmic_data;
int log_x_axis;

float delta_x_axis;
float x_scale_factor;
double left_data_limit;
double right_data_limit;
float y_scale_factor;

FILE *data_file;

main() {
        int pathno= -1;
        int charcount, maxcount=80, nlisteners=1;
        int *listeners, i, len;
        char devpath[80];
        char *dummy;
        char msgbuffer[80], file_name[80], comment[256];
        unsigned char *inp_data, temp_inp;
        unsigned char *i_p_ptr = inp_data;
        float delta_data;

        int it();
        long int lng_it();
        float ril();
        double lng_ril();
        char *strg();
        void prt_data_header(), prt_coord_header();

        printf ("\n\n");
        printf ("This program saves the displayed data on the %s\n",EQUIPM_NAME);
        printf ("Spectrum Analyzer.  The HPIB address of the analyzer should\n");
        printf ("be set to %d and the HPIB mode should be set to \"ADDRESS ONLY\".\n",
                 EQUIPM_ADDRESS);
        printf ("\n");
        printf ("         Version 1.0          October 22, 1990\n");
        printf ("Written by Yekta Gursel, based on a program by Noam Bernstein.\n\n");

        printf ("Please enter the file name for the data (14 characters max.):\n");
        dummy = gets(file_name);

        printf ("Please enter a comment string (256 characters max.):\n");
        dummy = gets(comment);

        listeners = (int *) malloc (nlisteners * sizeof (int));
        strcpy (devpath, GPIB_PATH);
        strcpy (msgbuffer,"DCBN");
        maxcount = strlen (msgbuffer);
        listeners [0] = EQUIPM_ADDRESS;

        mribopen (&pathno, devpath);
        mribenbremall (pathno);
        mribsettimout (pathno,32000);

        mribsend (pathno, msgbuffer, maxcount, nlisteners, listeners);

        inp_data = (unsigned char *) malloc (4 * sizeof (unsigned char));
        mribreceive (pathno, inp_data, &charcount, 4,
                     EQUIPM_ADDRESS, ONLY_CNTRLR_LISTENS, listeners);
        len = it (&inp_data[2]);

        inp_data = (unsigned char *) malloc (156 * sizeof (unsigned char));
        mribreceive (pathno, inp_data, &charcount, 156,
                     EQUIPM_ADDRESS, ONLY_CNTRLR_LISTENS, listeners);

        data_file = fopen (file_name,"w");
        prt_coord_header (inp_data, comment);

        inp_data = (unsigned char *) malloc (168 * sizeof (unsigned char));

        mribreceive (pathno, inp_data, &charcount, 168,
                     EQUIPM_ADDRESS, ONLY_CNTRLR_LISTENS, listeners);

        prt_data_header (inp_data);

        inp_data = (unsigned char *) malloc ((len-(168+156)) * sizeof (unsigned char));

        mribreceive (pathno, inp_data, &charcount, (len-(168+156)),
                     EQUIPM_ADDRESS, ONLY_CNTRLR_LISTENS, listeners);

        mribclose (pathno);

        fprintf(data_file,"**START OF SCALED DATA**\n");

        delta_data = (float) ((right_data_limit - left_data_limit) /
                             ((double) (number_of_elem - 1)));

        if ((log_x_axis == 1) && (logarithmic_data != 1)) {

          delta_data = (float) ((right_data_limit - left_data_limit) /
                              ((double) (number_of_elem - 1)));

          for (i=0; i<charcount; i+=4)  {

            fprintf (data_file, " %f   %f\n",
                     log10((left_data_limit+(float) (i/4)*delta_data)*x_scale_factor),
                     y_scale_factor*ril (&inp_data[i]));
          }

        }
        else if ((log_x_axis == 1) && (logarithmic_data == 1)) {

          delta_data = (float) ((log10(right_data_limit) - log10(left_data_limit)) /
                              ((double) (number_of_elem - 1)));

          for (i=0; i<charcount; i+=4)  {

            fprintf (data_file, " %f   %f\n",
                     (log10(left_data_limit)+(float) (i/4)*delta_data),
                     y_scale_factor*ril (&inp_data[i]));
          }

        }
        else {

          for (i=0; i<charcount; i+=4)  {

            fprintf (data_file, " %f   %f\n",
                     (left_data_limit+(float)(i/4)*delta_data)*x_scale_factor,
                     y_scale_factor*ril (&inp_data[i]));
          }

        }

        fprintf(data_file,"**END OF DATA**\n");

        fclose (data_file);

}

int it (x)
        unsigned char x[2];
        {
        int out_val;
        out_val = (x[1]+256*x[0]);
        return out_val;
}

long int lng_it (x)
        unsigned char x[2];
        {
        long out_val;
        out_val = (x[1]+256*x[0]+65536*x[3]+16777216*x[2]);
        return out_val;
}

double twoton(n)
int n;

{
  int m, i, value;

  if ( n == 0 )
    return ((double) 1.0);

  m = n;

  if ( m < 0)
    m = -m;

  value = 1;

  for (i=1;i<=m;i++)
    value=value*2;

  if (n < 0) {

    return(((double) 1.0) / ((double) value));

  }
  else {

    return((double) value);

  }

}

float ril (x)
        char x[4];
        {

        int i, exponent;
        float out_val;
        double mantissa;

        mantissa = ((double) x[0]) / ((double) 128.0) ;

        for (i=1; i<=2; i++) {

          exponent = 8*(i+1)-1;
          mantissa += ((double) ((unsigned char) x[i]))/twoton(exponent);

        }


        if (mantissa == 0.0) {

          out_val = 0.0;

        }
        else {

          exponent = (int) x[3];
          out_val = (float) (mantissa * twoton(exponent));

        }

        return out_val;
}

double lng_ril(x)
        char x[8];
```

```
{
        int i, exponent;
        double  mantissa, dexponent, two, out_val;

        two = (double) 2.0;

        mantissa = ((double) x[0]) / ((double) 128.0) ;

        for (i=1; i<=2; i++) {

          exponent = 8*(i+1)-1;
          mantissa += ((double) ((unsigned char) x[i])) / twoton(exponent);

        }

        for (i=3; i<=6; i++) {

          dexponent = (double) (8*(i+1)-1);
          mantissa += ((double) ((unsigned char) x[i])) / pow(two, dexponent);

        }

        if (mantissa == 0.0) {

          out_val = 0.0;

        }
        else {

          exponent=(int) x[7];
          out_val = mantissa * twoton(exponent);

        }

        return out_val;
}


char *strg (y, length)
        char *y;
        int length;
        {
        char *out_val, ch [2];
        int i;

        ch [1] = '\0';
        out_val = (char *) malloc (length * sizeof (char));
        out_val [0] = '\0';
        for (i=1 ; i<length; i++) {
                ch [0] = *(y+i) & 0177;
                strcat (out_val, ch);
        }
        return out_val;
}

void prt_data_header (x)
        unsigned char x[];
        {
        int inp_int;
        char inp_int_string[80];

#ifdef DEBUG
        printf ("\n");
#endif

        fprintf (data_file, "**DATA HEADER**\n");

        inp_int = it (&x[0]);

        switch (inp_int) {
                case 0 : strcpy (inp_int_string, "No data");
                        break;
                case 1 : strcpy (inp_int_string, "Frequency response");
                        break;
                case 2 : strcpy (inp_int_string, "Power spectrum 1");
                        break;
                case 3 : strcpy (inp_int_string, "Power spectrum 2");
                        break;
```

```
case 4 : strcpy (inp_int_string, "Coherence");
        break;
case 5 : strcpy (inp_int_string, "Cross Spectrum");
        break;
case 6 : strcpy (inp_int_string, "Input time 1");
        break;
case 7 : strcpy (inp_int_string, "Input time 2");
        break;
case 8 : strcpy (inp_int_string, "Input linear spectrum 1");
        break;
case 9 : strcpy (inp_int_string, "Input linear spectrum 2");
        break;
case 10 : strcpy (inp_int_string, "Impulse response");
        break;
case 11 : strcpy (inp_int_string, "Cross correlation");
        break;
case 12 : strcpy (inp_int_string, "Auto correlation 1");
        break;
case 13 : strcpy (inp_int_string, "Auto correlation 2");
        break;
case 14 : strcpy (inp_int_string, "Histogram 1");
        break;
case 15 : strcpy (inp_int_string, "Histogram 2");
        break;
case 16 : strcpy (inp_int_string, "Cumulative density function 1");
        break;
case 17 : strcpy (inp_int_string, "Cumulative density function 2");
        break;
case 18 : strcpy (inp_int_string, "Probability density function 1");
        break;
case 19 : strcpy (inp_int_string, "Probability density function 2");
        break;
case 20 : strcpy (inp_int_string, "Average linear spectrum 1");
        break;
case 21 : strcpy (inp_int_string, "Average linear spectrum 2");
        break;
case 22 : strcpy (inp_int_string, "Average time record 1");
        break;
case 23 : strcpy (inp_int_string, "Average time record 2");
        break;
case 24 : strcpy (inp_int_string, "Synthesis pole-zero");
        break;
case 25 : strcpy (inp_int_string, "Synthesis pole-residue");
        break;
case 26 : strcpy (inp_int_string, "Synthesis polynomial");
        break;
case 27 : strcpy (inp_int_string, "Synthesis constant");
        break;
case 28 : strcpy (inp_int_string, "Windowed time record 1");
        break;
case 29 : strcpy (inp_int_string, "Windowed time record 2");
        break;
case 30 : strcpy (inp_int_string, "Windowed linear spectrum 1");
        break;
case 31 : strcpy (inp_int_string, "Windowed linear spectrum 2");
        break;
case 32 : strcpy (inp_int_string, "Filtered time record 1");
        break;
case 33 : strcpy (inp_int_string, "Filtered time record 2");
        break;
case 34 : strcpy (inp_int_string, "Filtered linear spectrum 1");
        break;
case 35 : strcpy (inp_int_string, "Filtered linear spectrum 2");
        break;
case 36 : strcpy (inp_int_string, "Time capture buffer");
        break;
case 37 : strcpy (inp_int_string, "Captured linear spectrum");
        break;
case 38 : strcpy (inp_int_string, "Captured time record");
        break;
case 39 : strcpy (inp_int_string, "Throughput time record 1");
        break;
case 40 : strcpy (inp_int_string, "Throughput time record 2");
        break;
case 41 : strcpy (inp_int_string, "Curve fit");
        break;
case 42 : strcpy (inp_int_string, "Weighting function");
        break;
```

```
                case 44 : strcpy (inp_int_string, "Orbits");
                        break;
                case 45 : strcpy (inp_int_string, "Demodulation polar");
                        break;
                case 46 : strcpy (inp_int_string, "Preview demod record 1");
                        break;
                case 47 : strcpy (inp_int_string, "Preview demod record 2");
                        break;
                case 48 : strcpy (inp_int_string, "Preview demod linear spectrum 1");
                        break;
                case 49 : strcpy (inp_int_string, "Preview demod linear spectrum 2");
                        break;
                default : ;
        }

        fprintf (data_file, "Displayed function: %s\n", inp_int_string);

        number_of_elem = it(&x[2]);

        fprintf (data_file, "Number of elements: %d\n", number_of_elem);
        fprintf (data_file, "Number of displayed elements: %d\n", it(&x[4]));
        fprintf (data_file, "Number of averages: %d\n", it(&x[6]));

        inp_int = it (&x[8]);
        switch (inp_int) {
                case 0 : strcpy (inp_int_string, "Channel 1");
                        break;
                case 1 : strcpy (inp_int_string, "Channel 2");
                        break;
                case 2 : strcpy (inp_int_string, "Channel 1 & 2");
                        break;
                case 3 : strcpy (inp_int_string, "No channels");
                        break;
                default : ;
        }

        fprintf (data_file, "Channel selection: %s\n", inp_int_string);


        inp_int = it (&x[10]);

        switch (inp_int) {
                case 0 : strcpy (inp_int_string, "Channel 1");
                        break;
                case 1 : strcpy (inp_int_string, "Channel 2");
                        break;
                case 2 : strcpy (inp_int_string, "Channels 1 & 2");
                        break;
                case 3 : strcpy (inp_int_string, "No channels");
                        break;
                default : ;
        }

        fprintf (data_file, "Overload status: %s\n", inp_int_string);
        fprintf (data_file, "Overlap percentage: %d\n", it(&x[12]) );


        inp_int = it (&x[14]);

        switch (inp_int) {
                case 0 : strcpy (inp_int_string, "Time");
                        break;
                case 1 : strcpy (inp_int_string, "Frequency");
                        break;
                case 2 : strcpy (inp_int_string, "Amplitude (Voltage)");
                        break;
                default : ;
        }

        fprintf (data_file, "Domain: %s\n", inp_int_string);

        inp_int = it (&x[16]);

        switch (inp_int) {
                case 0 : strcpy (inp_int_string, "Peak");
                        break;
                case 1 : strcpy (inp_int_string, "RMS");
                        break;
                case 2 : strcpy (inp_int_string, "Volts (indicates peak only)");
                        break;
                default: ;
        }

        fprintf (data_file, "Volts (peak or RMS): %s\n", inp_int_string);

        inp_int = it(&x[18]);

        switch (inp_int) {
                case 0 : strcpy (inp_int_string, "Volts");
                        break;
                case 1 : strcpy (inp_int_string, "Volts squared");
                        break;
                case 2 : strcpy (inp_int_string, "PSD ((V^2)/Hz)");
                        break;
                case 3 : strcpy (inp_int_string, "ESD ((V^2)s/Hz)");
                        break;
                case 4 : strcpy (inp_int_string, "Sqrt(PSD) (V/Sqrt(Hz))");
                        break;
                case 5 : strcpy (inp_int_string, "No amplitude units");
                        break;
                case 6 : strcpy (inp_int_string, "Unit volts");
                        break;
                case 7 : strcpy (inp_int_string, "Unit volts^2");
                        break;
                default : ;
        }

        fprintf (data_file, "Amplitude units: %s\n", inp_int_string);

        inp_int = it(&x[20]);

        switch (inp_int) {
                case 0 : strcpy (inp_int_string, "No units");
                        break;
                case 1 : strcpy (inp_int_string, "Hertz");
                        break;
                case 2 : strcpy (inp_int_string, "RPM");
                        break;
                case 3 : strcpy (inp_int_string, "Orders");
                        break;
                case 4 : strcpy (inp_int_string, "Seconds");
                        break;
                case 5 : strcpy (inp_int_string, "Revs");
                        break;
                case 6 : strcpy (inp_int_string, "Degrees");
                        break;
                case 7 : strcpy (inp_int_string, "dB");
                        break;
                case 8 : strcpy (inp_int_string, "dBV");
                        break;
                case 9 : strcpy (inp_int_string, "Volts");
                        break;
                case 10 : strcpy (inp_int_string, "V/Sqrt(Hz) (Sqrt(PSD))");
                        break;
                case 11 : strcpy (inp_int_string, "Hertz/second");
                        break;
                case 12 : strcpy (inp_int_string, "Volts/EU");
                        break;
                case 13 : strcpy (inp_int_string, "Vrms");
                        break;
                case 14 : strcpy (inp_int_string, "V^2/Hz (PSD)");
                        break;
                case 15 : strcpy (inp_int_string, "Percent");
                        break;
                case 16 : strcpy (inp_int_string, "Points");
                        break;
                case 17 : strcpy (inp_int_string, "Records");
                        break;
                case 18 : strcpy (inp_int_string, "Ohms");
                        break;
                case 19 : strcpy (inp_int_string, "Hertz/Octave");
                        break;
                case 20 : strcpy (inp_int_string, "Pulse/Rev");
                        break;
                case 21 : strcpy (inp_int_string, "Decades");
                        break;
```

```
void prt_coord_header (x,comment)
        unsigned char x[];
        char comment[];
        {
        int inp_int;
        char inp_int_string[80];
        char *printed_time;
        long *clock;
        long seconds;
        struct tm *tmday;

        fprintf (data_file, "**DEVICE: %s**\n",EQUIPM_NAME);

        clock = (long *) malloc(sizeof(long));

        seconds=time(clock);
        tmday=localtime(clock);
        printed_time=asctime(tmday);
        *(printed_time + 24)= '\0';

        if ((*tmday).tm_isdst == 0) {

          fprintf (data_file, "**DATE AND TIME: %s %s**\n",printed_time,STANDARD_TIME);

        }
        else {

          fprintf (data_file, "**DATE AND TIME: %s %s**\n",printed_time,DAYLIGHT_TIME);

        }

        fprintf (data_file, "**COMMENTS: %s\n",comment);

        fprintf (data_file, "**COORDINATE HEADER**\n");

        inp_int = it (&x[0]);

        switch (inp_int) {
                case 1 : strcpy (inp_int_string, "Real");
                        break;
                case 2 : strcpy (inp_int_string, "Imaginary");
                        break;
                case 3 : strcpy (inp_int_string, "Linear magnitude");
                        break;
                case 4 : strcpy (inp_int_string, "Log magnitude");
                        break;
                case 5 : strcpy (inp_int_string, "dB");
                        break;
                case 6 : strcpy (inp_int_string, "Nyquist");
                        break;
                case 8 : strcpy (inp_int_string, "Phase");
                        break;
                case 9 : strcpy (inp_int_string, "Nichols");
                        break;
                case 10: strcpy (inp_int_string, "dBm");
                        break;
                default : ;
        }

        fprintf (data_file, "Y coordinates: %s\n", inp_int_string );
        fprintf (data_file, "Number of display elements: %d\n",it(&x[2]) );
        fprintf (data_file, "First element: %d\n", it(&x[4]));
        fprintf (data_file, "Total number of elements: %d\n", it(&x[6]));

        inp_int = it(&x[8]);

        switch (inp_int) {
                case 0 :
                        strcpy (inp_int_string,
                                "Not sampled (Number of displayed elements = Total number of elements)");
                        break;
                case 1 :
                        strcpy (inp_int_string,
                                "Half sampled (Number of displayed elements = Total number of elements/2)");
                        break;
                case 2 :
                        strcpy (inp_int_string,
                                "Sampled (Number of displayed elements < Total number of elements)");
                        break;
                default : ;
        }

        fprintf (data_file, "Display sampling: %s\n", inp_int_string );

        inp_int = it (&x[10]);

        switch (inp_int) {
                case 0 : strcpy (inp_int_string, "X and Y are automatically scaled.");
                        break;
                case 1 : strcpy (inp_int_string, "X is fixed scale, Y is automatically scaled.");
                        break;
                case 2 : strcpy (inp_int_string, "X is automatically scaled, Y is fixed scale.");
                        break;
                case 3 : strcpy (inp_int_string, "X and Y are fixed scale.");
                        break;
                default : ;
        }

        fprintf (data_file, "Scaling: %s\n", inp_int_string );
        fprintf (data_file, "Data pointer: %ld\n", lng_it(&x[12]) );
        fprintf (data_file, "In data: %ld\n",lng_it(&x[16]));

        log_x_axis = it(&x[20]);

        fprintf (data_file, "Log or Linear x-axis? ( 1 or 0): %d\n",log_x_axis);
        fprintf (data_file, "Sampled display data? (1=yes, 0=no): %d\n", it(&x[22]));
        fprintf (data_file, "Plot or graph mode? (1 or 0): %d\n", it(&x[24]));
        fprintf (data_file, "Phase wrap? (1=yes, 0=no) : %d\n", it(&x[26]));

        x_scale_factor = ril(&x[64]);

        fprintf (data_file, "X scale factor: %f\n", x_scale_factor);
        fprintf (data_file, "Grid minimum Y scale: %f\n", ril(&x[68]));
        fprintf (data_file, "Grid maximum Y scale: %f\n", ril(&x[72]));
        fprintf (data_file, "Y amount per division: %f\n", ril(&x[76]));
        fprintf (data_file, "Minimum value of data: %f\n", ril(&x[80]));
        fprintf (data_file, "Maximum value of data: %f\n", ril(&x[84]));
        fprintf (data_file, "Y cumulative minimum: %f\n", ril(&x[88]));
        fprintf (data_file, "Y cumulative maximum: %f\n", ril(&x[92]));

        y_scale_factor = ril(&x[96]);

        fprintf (data_file, "Y scale factor: %f\n", y_scale_factor);
        fprintf (data_file, "Stop value: %f\n", lng_ril(&x[116]) );
        fprintf (data_file, "Left grid limit: %f\n", lng_ril(&x[124]));
        fprintf (data_file, "Right grid limit: %f\n", lng_ril(&x[132]));

        left_data_limit = lng_ril(&x[140]);

        fprintf (data_file, "Left data limit: %f\n", left_data_limit);

        right_data_limit = lng_ril(&x[148]);

        fprintf (data_file, "Right data limit: %f\n", right_data_limit);
        fprintf (data_file, "**END OF COORDINATE HEADER**\n");
}
```

```
      PROGRAM EXTRACT

      INTEGER STRLENGTH, LOG_X_AXIS

      CHARACTER*100 INPUT_STR

      CHARACTER*14  INPUT_FILE, OUTPUT_FILE, SM_FILE

      CHARACTER*80 DATE_STR, Y_LABEL, X_LABEL, X_TMP, DISPLAY_LABEL

      REAL X_MIN, X_MAX, Y_MIN, Y_MAX, X_SCALE, Y_SCALE,
     $     Y_PERDIV, X_PERDIV, X_GRL, Y_GRL1, Y_GRL2

      LOGICAL IT_IS_THERE

      PRINT*, " "
      PRINT*, "Extract Version 1.0"
      PRINT*, "Written by:  Yekta Gursel    October 23, 1990"
      PRINT*, " "

      PRINT*, "Enter the input file name:"

      READ*, INPUT_FILE

      INQUIRE(FILE=INPUT_FILE,EXIST=IT_IS_THERE)

      IF (.NOT. IT_IS_THERE) THEN

         PRINT*, "The input file ",
     $           INPUT_FILE(1:STRLENGTH(INPUT_FILE)),
     $           " does not exist."

         STOP

      ENDIF

      PRINT*, "Enter the output data file name:"

      READ*, OUTPUT_FILE

      INQUIRE(FILE=OUTPUT_FILE,EXIST=IT_IS_THERE)

      IF (IT_IS_THERE) THEN

         PRINT*, "The output data file ",
     $           OUTPUT_FILE(1:STRLENGTH(OUTPUT_FILE)),
     $           " exists already."

         STOP

      ENDIF

      PRINT*, "Enter the SM command file name:"

      READ*, SM_FILE

      X_TMP = SM_FILE(1:STRLENGTH(SM_FILE)) // ".m"
```

```
      SM_FILE = X_TMP(1:STRLENGTH(X_TMP))

      INQUIRE(FILE=SM_FILE,EXIST=IT_IS_THERE)

      IF (IT_IS_THERE) THEN

         PRINT*, "The SM command file ",
     $           SM_FILE(1:STRLENGTH(SM_FILE)),
     $           " exists already."

         STOP

      ENDIF

      OPEN(10,FILE=INPUT_FILE,STATUS='OLD')

      OPEN(20,FILE=OUTPUT_FILE,STATUS='NEW')

      OPEN(30,FILE=SM_FILE,STATUS='NEW')

 10   READ(10,'(A100)') INPUT_STR

      IF (INPUT_STR(1:16) .EQ. "**DATE AND TIME:") THEN

         DATE_STR=INPUT_STR(18:45)

         GOTO 10

      ELSEIF (INPUT_STR(1:14) .EQ. "Y coordinates:") THEN

         Y_LABEL=INPUT_STR(16:STRLENGTH(INPUT_STR))

         GOTO 10

      ELSEIF (INPUT_STR(1:32)
     $        .EQ. "Log or Linear x-axis? ( 1 or 0):") THEN

         READ(INPUT_STR(33:STRLENGTH(INPUT_STR)), *) LOG_X_AXIS

         GOTO 10

      ELSEIF (INPUT_STR(1:15) .EQ. "X scale factor:") THEN

         READ(INPUT_STR(16:STRLENGTH(INPUT_STR)), *) X_SCALE

         GOTO 10

      ELSEIF (INPUT_STR(1:21) .EQ. "Grid minimum Y scale:") THEN

         READ(INPUT_STR(22:STRLENGTH(INPUT_STR)), *) Y_MIN

         GOTO 10

      ELSEIF (INPUT_STR(1:21) .EQ. "Grid maximum Y scale:") THEN

         READ(INPUT_STR(22:STRLENGTH(INPUT_STR)), *) Y_MAX

         GOTO 10
      ELSEIF (INPUT_STR(1:22) .EQ. "Y amount per division:") THEN

         READ(INPUT_STR(23:STRLENGTH(INPUT_STR)), *) Y_PERDIV

         GOTO 10

      ELSEIF (INPUT_STR(1:15) .EQ. "Y scale factor:") THEN

         READ(INPUT_STR(16:STRLENGTH(INPUT_STR)), *) Y_SCALE

         GOTO 10

      ELSEIF (INPUT_STR(1:16) .EQ. "Left grid limit:") THEN

         READ(INPUT_STR(17:STRLENGTH(INPUT_STR)), *) X_MIN

         GOTO 10

      ELSEIF (INPUT_STR(1:17) .EQ. "Right grid limit:") THEN

         READ(INPUT_STR(18:STRLENGTH(INPUT_STR)), *) X_MAX

         GOTO 10

      ELSEIF (INPUT_STR(1:19) .EQ. "Displayed function:") THEN

         DISPLAY_LABEL = INPUT_STR(21:STRLENGTH(INPUT_STR))

         GOTO 10

      ELSEIF (INPUT_STR(1:13) .EQ. "X-axis units:") THEN

         X_TMP = INPUT_STR(15:STRLENGTH(INPUT_STR))

         GOTO 10

      ELSEIF (INPUT_STR .NE. '**START OF SCALED DATA**') THEN

         GOTO 10

      ENDIF

20    READ(10,'(A100)') INPUT_STR

      IF (INPUT_STR .NE. '**END OF DATA**') THEN

         WRITE(20,*) INPUT_STR(1:STRLENGTH(INPUT_STR))

         GOTO 20

      ENDIF

      CLOSE(10)
      CLOSE(20)

      X_MIN = X_SCALE*X_MIN
      X_MAX = X_SCALE*X_MAX
```

```
      X_LABEL = X_TMP

      IF (LOG_X_AXIS .EQ. 1) THEN

         X_MIN = LOG10(X_MIN)
         X_MAX = LOG10(X_MAX)

         X_LABEL = "Log " // X_TMP(1:STRLENGTH(X_TMP))

      ENDIF

      Y_MIN = Y_SCALE*Y_MIN
      Y_MAX = Y_SCALE*Y_MAX

      X_PERDIV = (X_MAX - X_MIN)/10.0
      Y_PERDIV = Y_SCALE*Y_PERDIV

      X_GRL = (X_MIN + X_MAX)/2.0

      Y_GRL1 = Y_MAX+(Y_MAX - Y_MIN)/150.0
      Y_GRL2 = Y_MAX+(Y_MAX - Y_MIN)/25.0

      WRITE(30,*) SM_FILE(1:STRLENGTH(SM_FILE)-2)
      WRITE(30,*) "  device postscript"
      WRITE(30,*) "  expand 1.0001"
      WRITE(30,'(A10,4(1X,E8.2))') "  limits ", X_MIN, X_MAX, Y_MIN,
     $                                          Y_MAX

      IF (LOG_X_AXIS .NE. 1) THEN

         WRITE(30,'(A12,4(1X,E8.2))') "  ticksize ", X_PERDIV/10.0,
     $                                               X_PERDIV,
     $                                               Y_PERDIV/10.0,
     $                                               Y_PERDIV

      ELSE

         WRITE(30,'(A12,4(1X,E8.2))') "  ticksize ", -1.0,
     $                                               10.0,
     $                                               Y_PERDIV/10.0,
     $                                               Y_PERDIV

      ENDIF

      WRITE(30,*) "  box"

      INPUT_STR = "  xlabel " // X_LABEL
      WRITE(30,*) INPUT_STR(1:STRLENGTH(INPUT_STR))

      INPUT_STR = "  ylabel " // Y_LABEL
      WRITE(30,*) INPUT_STR(1:STRLENGTH(INPUT_STR))

      WRITE(30,'(A12,2(1X,E8.2))') "  relocate ", X_GRL, Y_GRL2

      INPUT_STR = "  putlabel 8 " // DISPLAY_LABEL
      WRITE(30,*) INPUT_STR(1:STRLENGTH(INPUT_STR))
      WRITE(30,'(A12,2(1X,E8.2.1))') "  relocate ", X_GRL, Y_GRL1

      INPUT_STR = "  putlabel 8 " // DATE_STR
      WRITE(30,*) INPUT_STR(1:STRLENGTH(INPUT_STR))

      WRITE(30,*) "  data ", OUTPUT_FILE
      WRITE(30,*) "  read x 1"
      WRITE(30,*) "  read y 2"
      WRITE(30,*) "  connect x y"
      WRITE(30,*) "  hardcopy"
      WRITE(30,*) "  quit"

      CLOSE(30)

      STOP

      END


      FUNCTION STRLENGTH(ASTRING)

      CHARACTER*(*) ASTRING

      INTEGER STRLENGTH,N

      INTEGER I

      N=LEN(ASTRING)

      DO 10 I=N,1,-1

         IF(ASTRING(I:I) .NE. ' ') THEN
            GOTO 20
         ENDIF

 10   CONTINUE

      I=0

      RETURN

 20   CONTINUE

      STRLENGTH=I

      RETURN
      END
```