APPENDIX C

POWER SPECTRA, PHASE SPECTRA, ESTIMATED FREQUENCIES, AND ESTIMATED DAMPING RATIOS: SECOND SET OF MEASUREMENTS

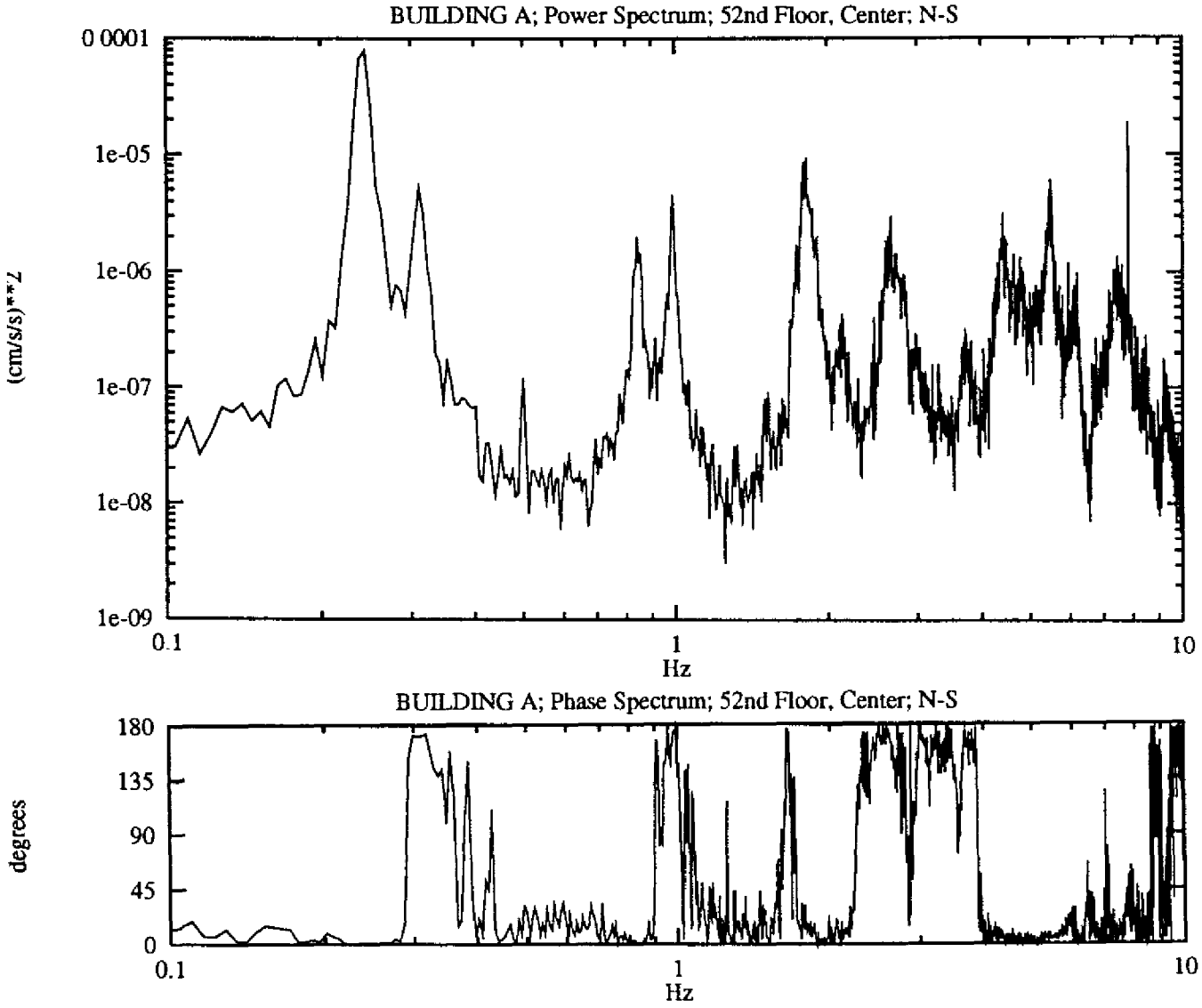
A second set of measurements from Buildings A and B were collected at later dates, when the buildings were at a different stage of construction. Although parameters estimated from these measurements are not presented in Section 6, the spectra are included in Appendix C for completeness and for reference in future reports. The measurement dates are as follows:

Building A:	August 15, 1991
Building B:	August 15, 1991

FIGURE C-1

BUILDING A - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	52nd Floor, Center
Reference	Sensor Location	52nd Floor, West
	Sensor Direction	North-South



Peaks In G52cwn

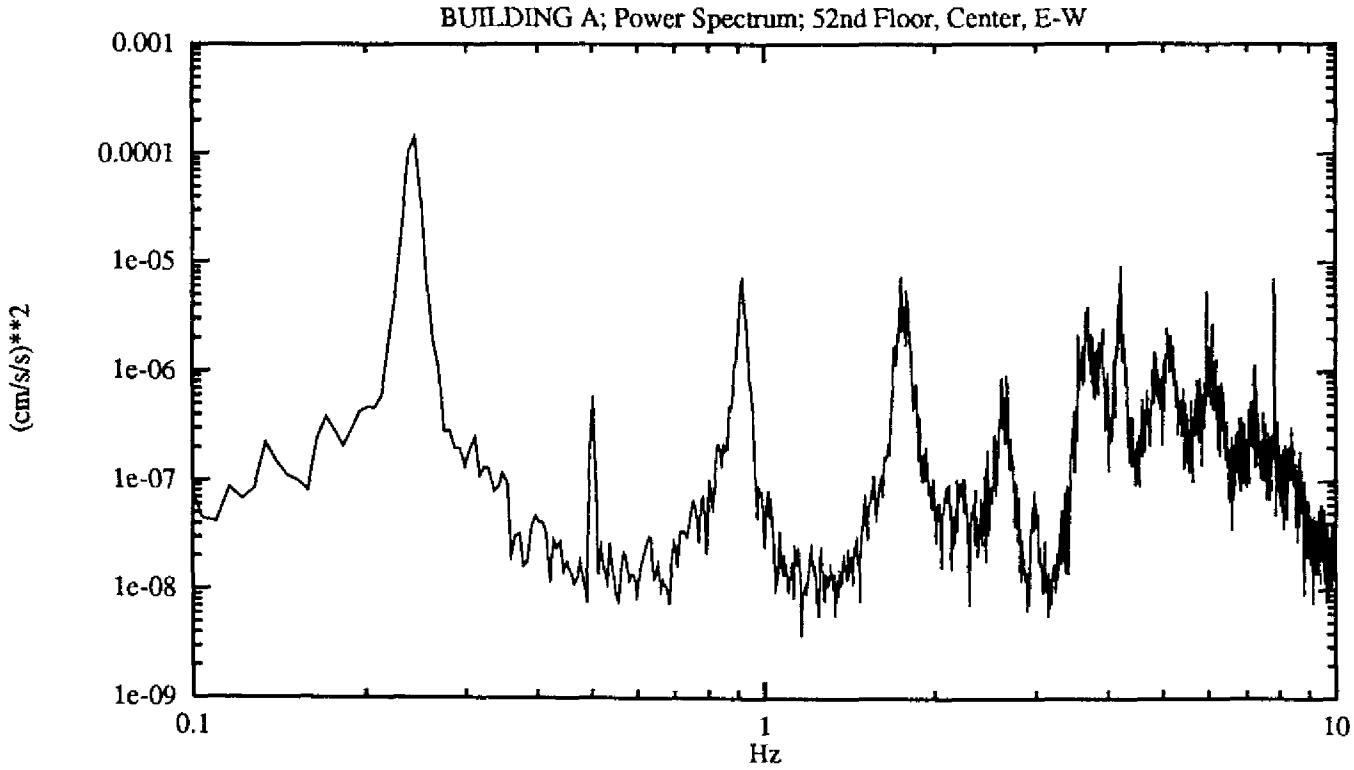
Hz-data	Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2441	0.2422	7.9021e-05	8.2292e-05	1	2.1
0.3113 T1	0.3117	5.2324e-06	5.2452e-06	171	2.0
0.5005	0.5000	1.1987e-07	1.2037e-07	27	0.7
0.8423	0.8426	2.0020e-06	2.0061e-06	0	0.9
0.9888	0.9894	4.5475e-06	4.5672e-06	174	0.7
1.8188	1.8185	9.4055e-06	9.3579e-06	10	0.4
2.1362	2.1379	4.4307e-07	4.5169e-07	13	0.4
2.6672 T5	2.6673	3.0255e-06	3.0100e-06	163	0.3

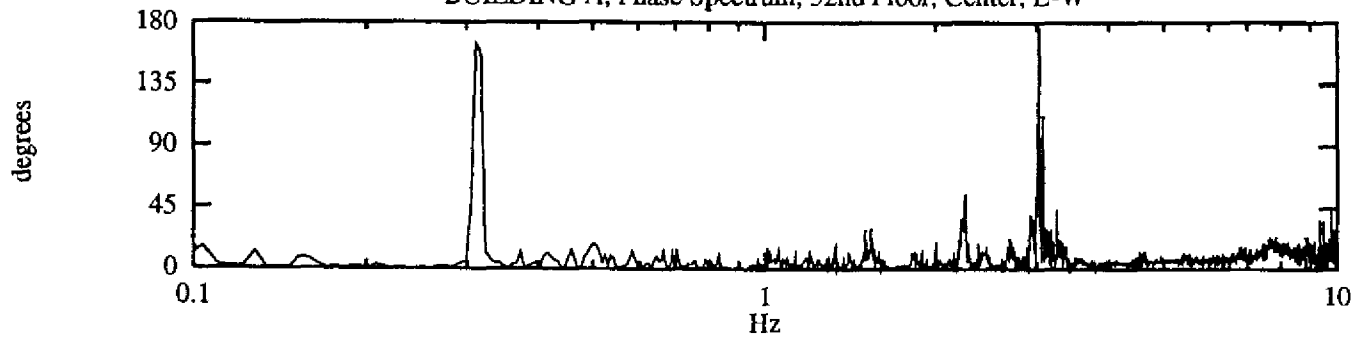
RMS-acc	RMS-vel	RMS-dsp
3.100e-02	9.962e-03	6.538e-03

FIGURE C-2

BUILDING A - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	52nd Floor, Center
Reference	Sensor Location	52nd Floor, West
	Sensor Direction	East-West





Peaks In G52cwe

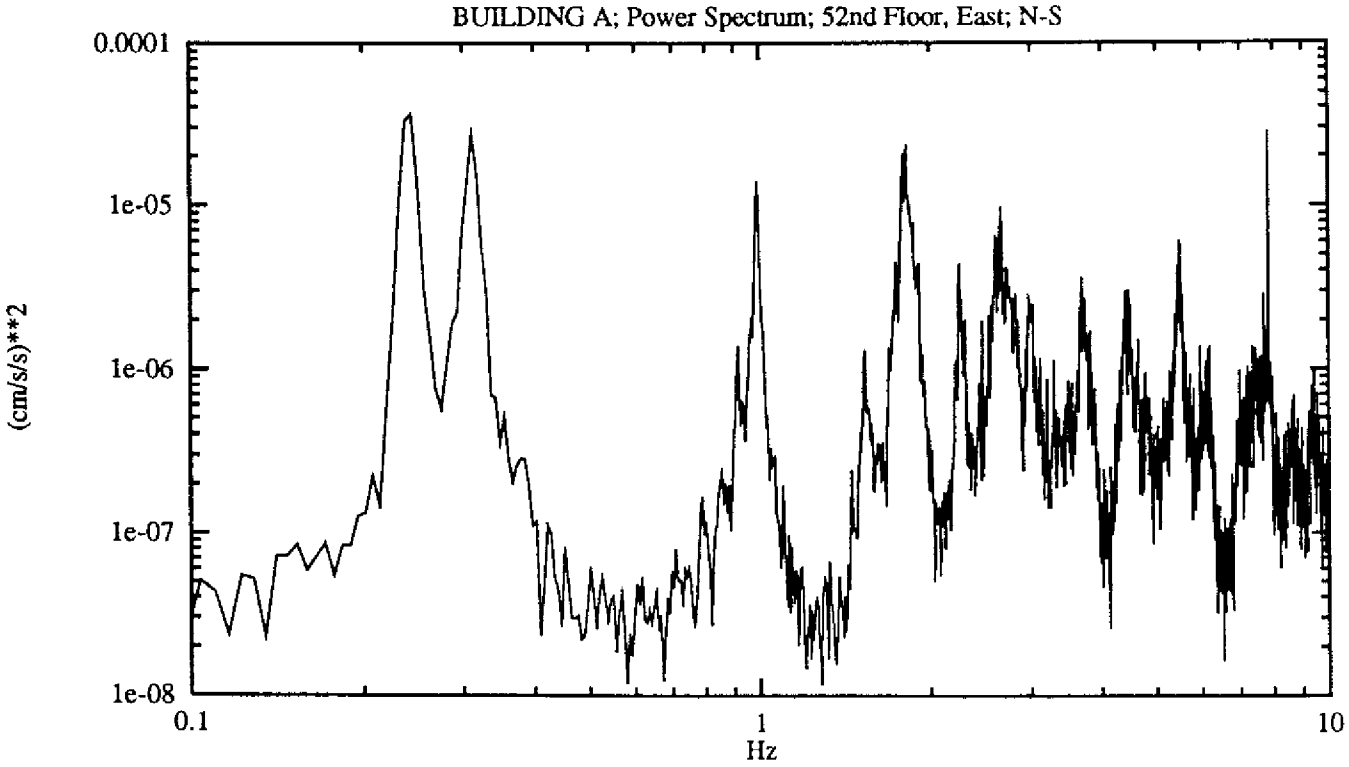
Hz-data	Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2441	0.2428	1.4131e-04	1.4474e-04	0	1.8
0.3113 T1	0.3100	2.5262e-07	2.5719e-07	165	1.9
0.5005	0.5002	5.9385e-07	5.9488e-07	18	0.6
0.9155	0.9136	6.8832e-06	7.0892e-06	1	0.9
1.7456	1.7429	7.0937e-06	7.5996e-06	2	0.3
2.6672 T5	2.6687	9.2164e-07	9.4622e-07	15	0.4
2.9846 T6	2.9818	7.7498e-08	8.1956e-08	1	0.3

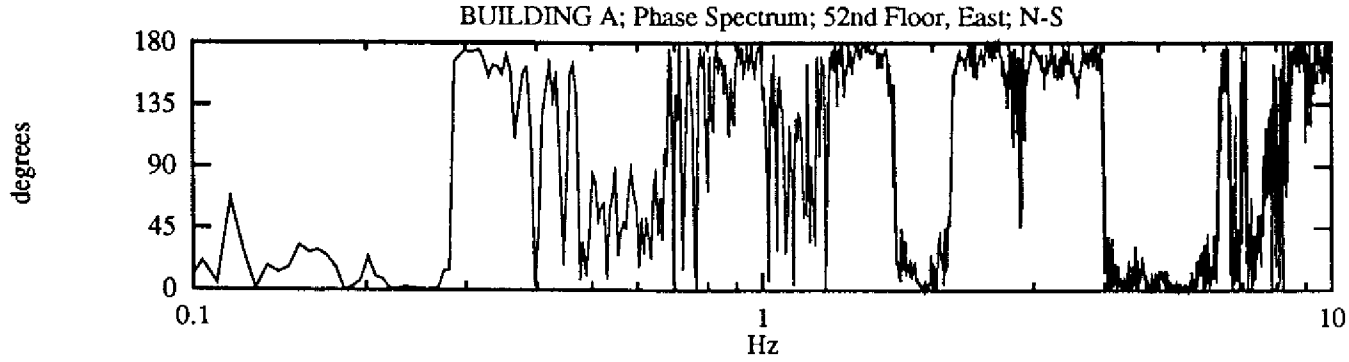
RMS-acc	RMS-vel	RMS-dsp
3.325e-02	1.225e-02	8.179e-03

FIGURE C-3

BUILDING A - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	52nd Floor, East
Reference	Sensor Location	52nd Floor, West
	Sensor Direction	North-South





Peaks In G52ewn

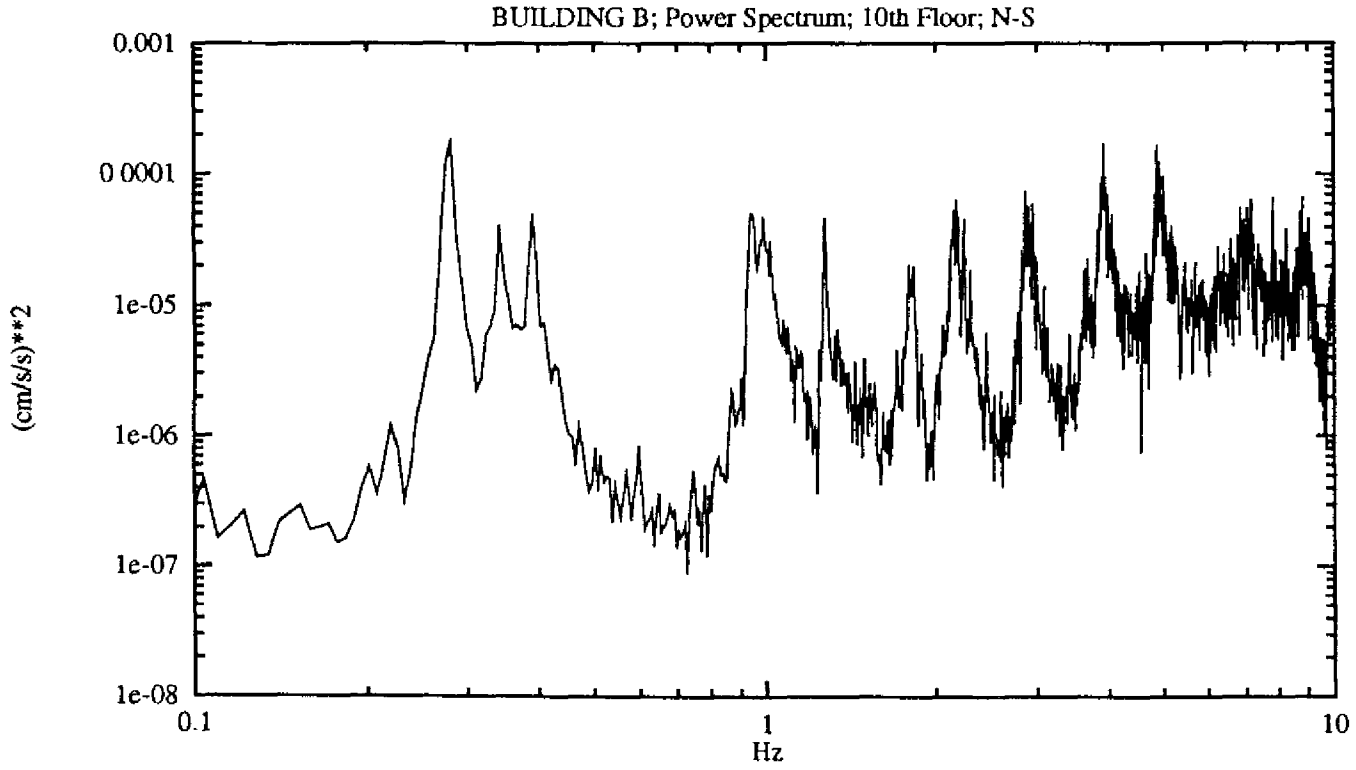
Hz-data	Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2441	0.2419	3.6481e-05	3.8264e-05	1	2.3
0.3113 T1	0.3116	2.7772e-05	2.7809e-05	174	2.0
0.9888 T2	0.9894	1.4237e-05	1.4290e-05	175	0.6
1.5320 T3	1.5304	1.2274e-06	1.2377e-06	174	0.8
1.8188	1.8184	2.3365e-05	2.3365e-05	13	0.4
2.2522 T4	2.2520	4.4082e-06	4.4256e-06	170	0.4
2.6672 T5	2.6672	9.7687e-06	9.7156e-06	168	0.3
2.9785 T6	2.9808	2.6913e-06	2.7344e-06	172	1.2

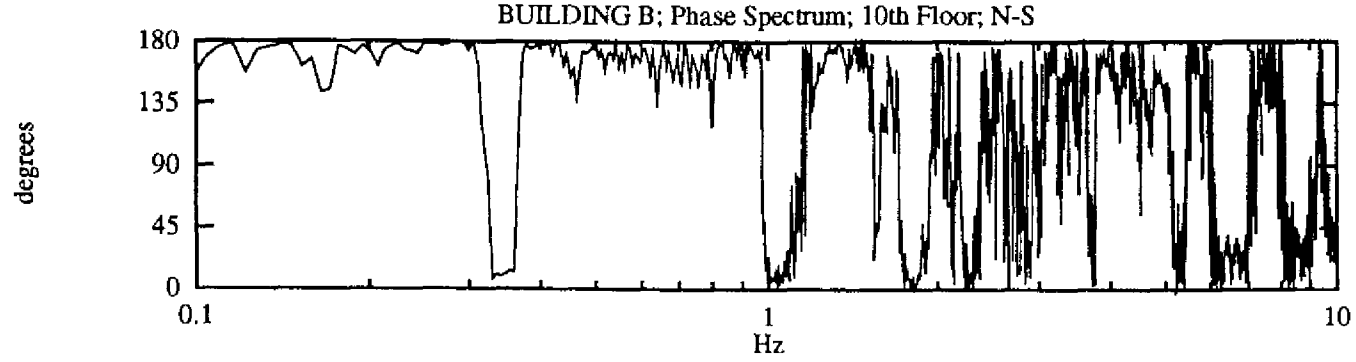
RMS-acc	RMS-vel	RMS-dsp
4.330e-02	8.567e-03	5.207e-03

FIGURE C-4

BUILDING B - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	10th Floor, Center
Reference	Sensor Location	40th Floor, Center
	Sensor Direction	North-South





Peaks In G1040n

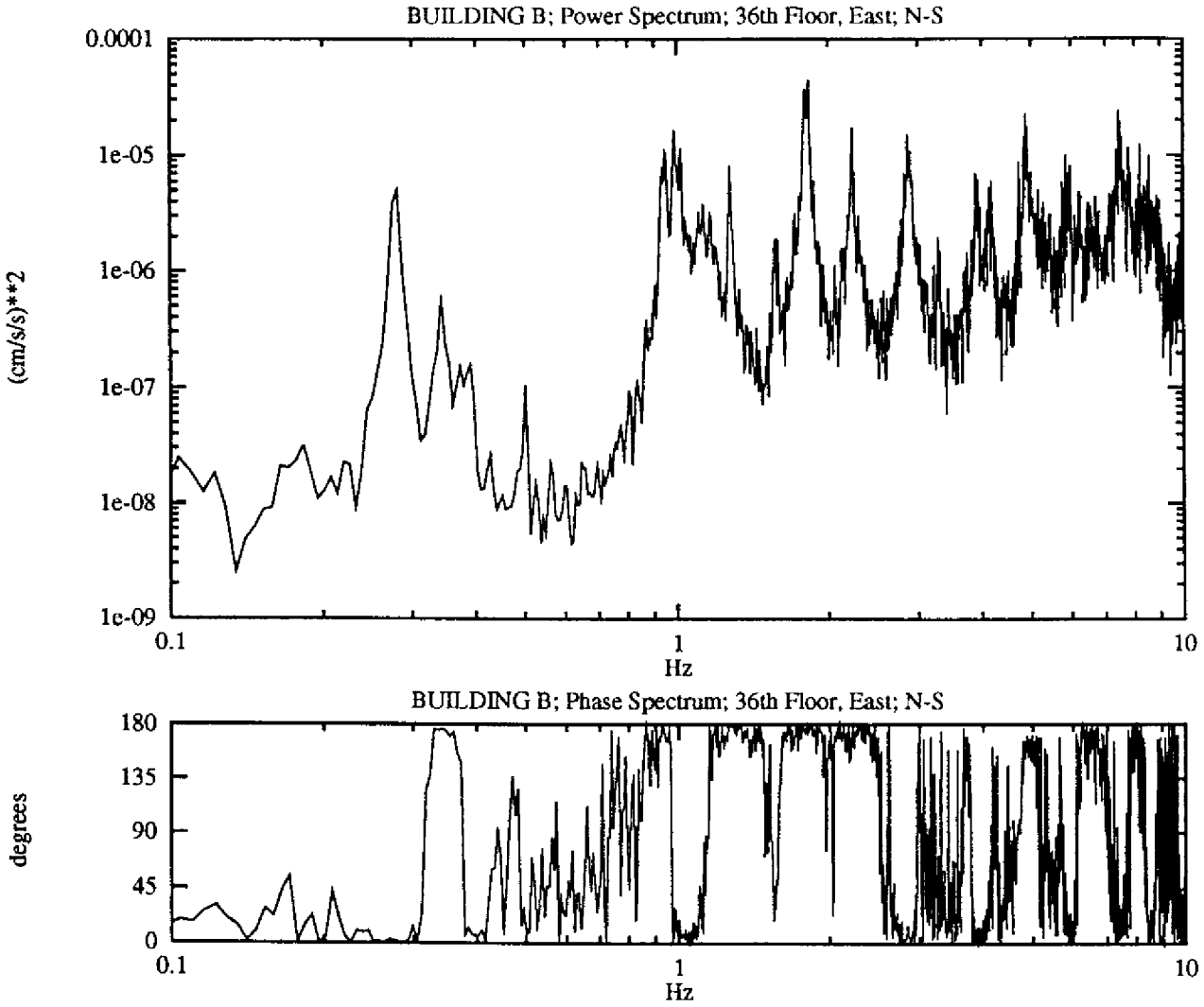
Hz-data		Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2808		0.2794	1.7395e-04	1.7871e-04	180	1.3
0.3418		0.3423	4.1376e-05	4.1530e-05	9	1.3
0.3906		0.3906	4.9758e-05	4.9755e-05	176	1.3
0.9399		0.9423	5.0123e-05	5.0642e-05	172	1.5
0.9888	T1	0.9894	4.7610e-05	4.7743e-05	18	1.3
1.2756		1.2762	4.6701e-05	4.6790e-05	172	0.6
1.7944		1.7944	2.0362e-05	2.0325e-05	2	0.4
2.1729		2.1728	6.3866e-05	6.3419e-05	169	0.4
2.8687	T4	2.8687	7.4298e-05	7.5340e-05	17	0.3
3.9307		3.9306	1.7064e-04	1.7548e-04	162	0.1
4.8828		4.8800	1.5445e-04	1.5831e-04	156	0.2

RMS-acc	RMS-vel	RMS-dsp
2.180e-01	1.549e-02	7.966e-03

FIGURE C-5

BUILDING B - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	36th Floor, East
Reference	Sensor Location	10th Floor, Center
	Sensor Direction	North-South



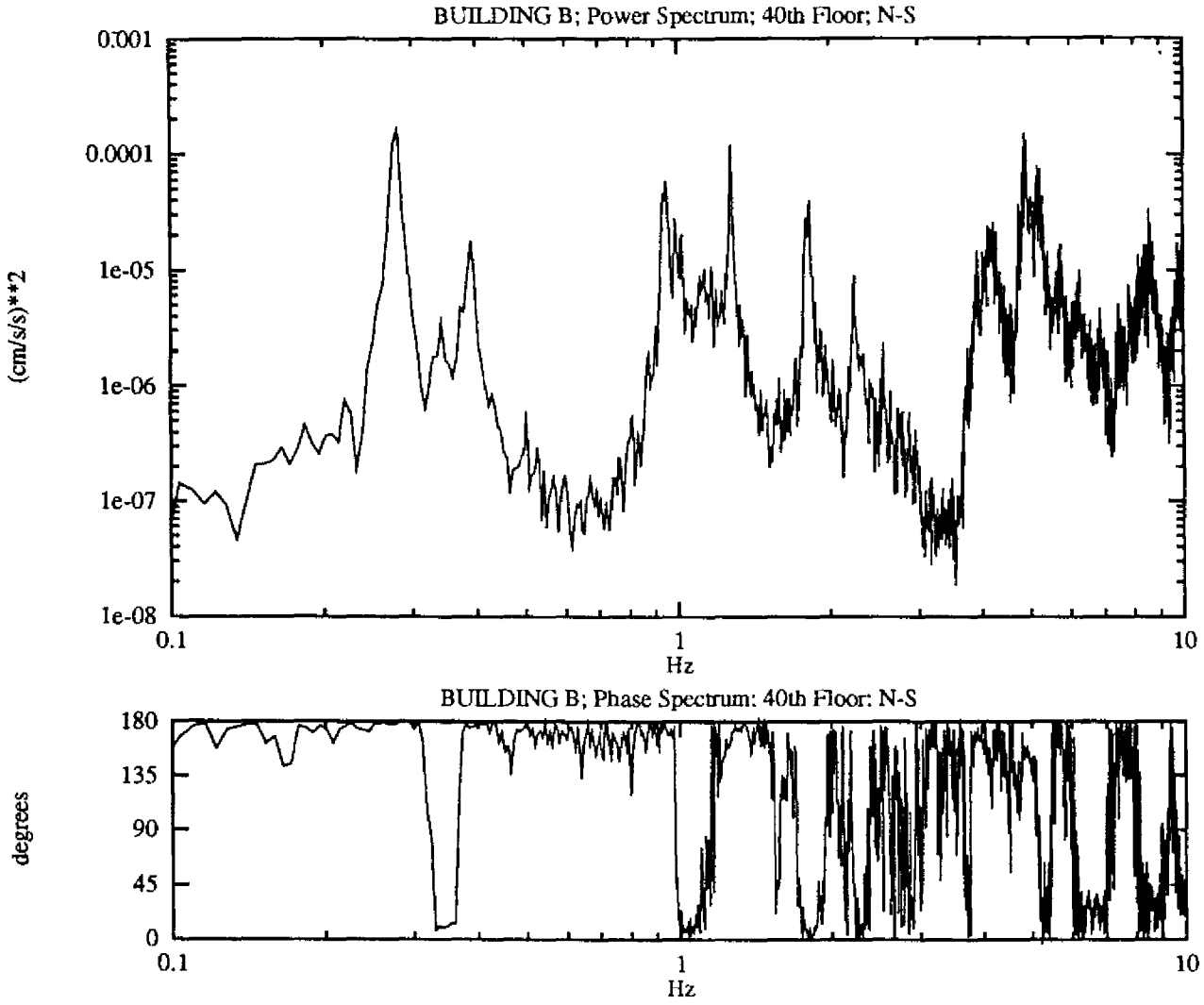
Peaks In G36e10n

Hz-data		Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2808		0.2791	5.1004e-06	5.2908e-06	1	1.2
0.3418		0.3419	6.1939e-07	6.1956e-07	176	1.3
0.3906		0.3890	1.6010e-07	1.6406e-07	7	1.7
0.5005		0.5005	1.0596e-07	1.0600e-07	7	0.7
0.9460		0.9483	1.0672e-05	1.1001e-05	166	0.8
0.9888	T1	0.9883	1.6407e-05	1.6436e-05	12	0.7
1.0193	T2	1.0174	1.1486e-05	1.1936e-05	7	0.5
1.1047		1.1049	3.3119e-06	3.3081e-06	39	0.7
1.1292		1.1287	3.8387e-06	3.8445e-06	40	0.7
1.1658	T3	1.1657	3.0323e-06	3.0333e-06	134	0.9
1.2756		1.2758	8.1588e-06	8.1360e-06	179	0.5
1.5747		1.5731	1.8998e-06	1.9595e-06	53	0.3
1.8311	T4	1.8286	4.4506e-05	4.7088e-05	180	0.3
2.2400		2.2385	1.7490e-05	1.8001e-05	160	0.2
2.8687		2.8687	1.5584e-05	1.5557e-05	3	0.4
3.9001		3.9018	6.9523e-06	7.0333e-06	4	0.2
4.1870	T5	4.1879	5.9891e-06	5.8413e-06	72	0.3
4.8828		4.8800	2.1977e-05	2.2888e-05	170	0.2

FIGURE C-6

BUILDING B - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	40th Floor, Center
Reference	Sensor Location	10th Floor, Center
	Sensor Direction	North-South



Peaks In G4010n

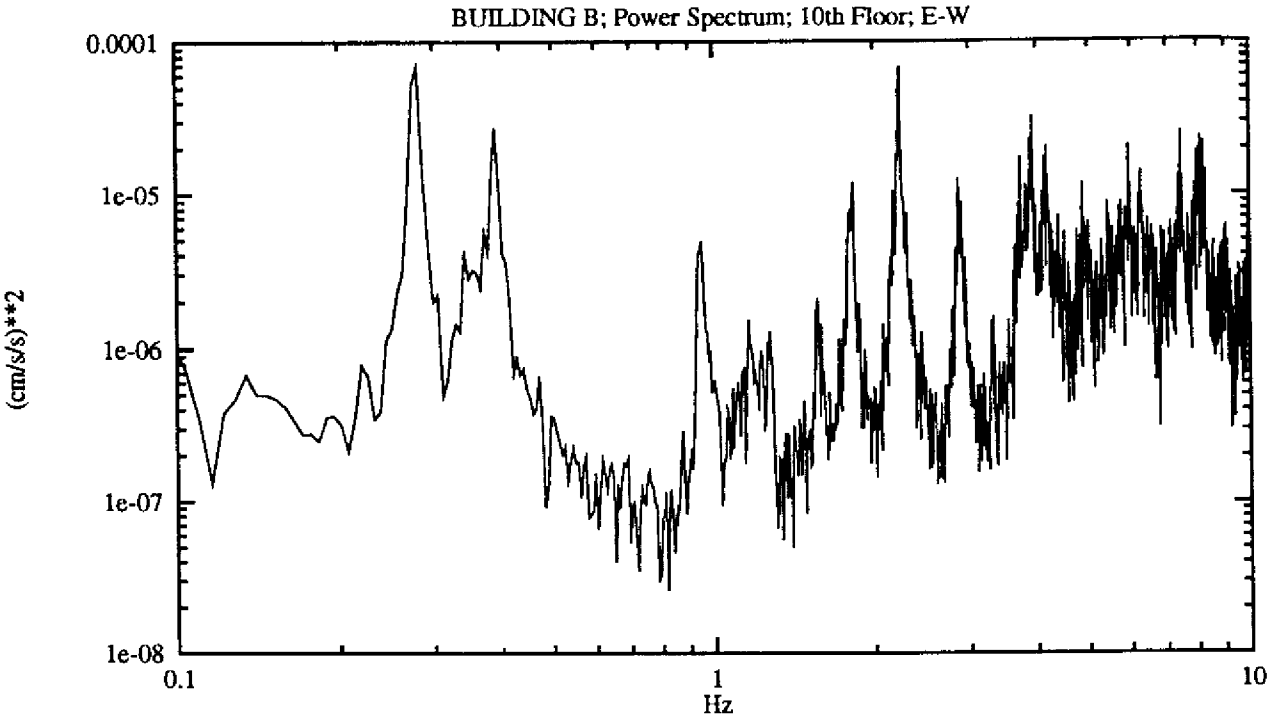
Hz-data		Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2808		0.2792	1.6510e-04	1.7114e-04	180	1.2
0.3418		0.3417	3.6197e-06	3.6210e-06	9	1.6
0.3906		0.3903	1.8138e-05	1.8161e-05	176	1.4
0.5005		0.4999	5.9891e-07	6.0233e-07	179	0.7
0.9460		0.9472	5.4627e-05	5.4985e-05	168	0.9
0.9888	T1	0.9878	2.7728e-05	2.7955e-05	18	0.7
1.0193	T2	1.0167	1.9907e-05	2.1294e-05	9	0.5
1.1292	T3	1.1281	1.0298e-05	1.0394e-05	42	0.7
1.1658		1.1653	1.0770e-05	1.0796e-05	39	0.8
1.2756		1.2757	1.1819e-04	1.1802e-04	172	0.6
1.8311	T4	1.8289	3.9541e-05	4.1246e-05	2	0.3
2.2400		2.2376	9.0358e-06	9.4771e-06	35	0.3
4.2297	T5	4.2291	2.5202e-05	2.5034e-05	173	0.2
4.8767		4.8788	1.4886e-04	1.5259e-04	156	0.3

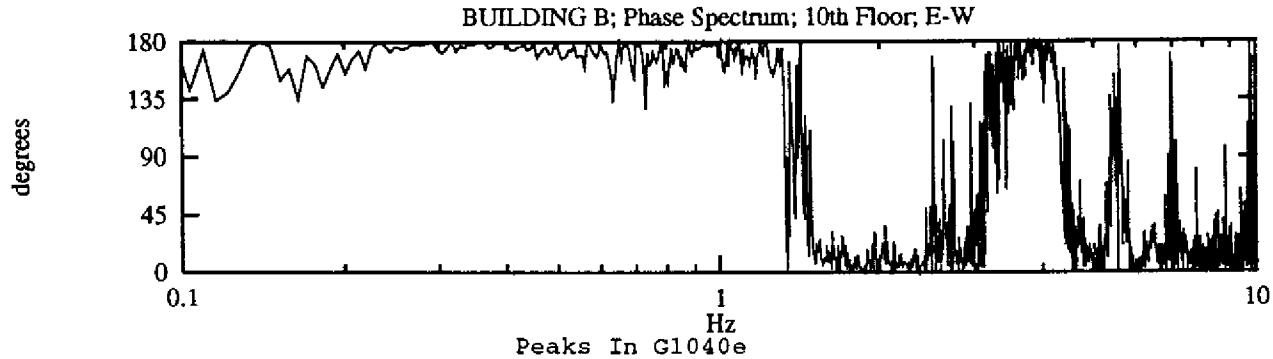
RMS-acc	RMS-vel	RMS-dsp
1.199e-01	1.331e-02	7.066e-03

FIGURE C-7

BUILDING B - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	10th Floor, Center
Reference	Sensor Location	40th Floor, Center
	Sensor Direction	East-West





Peaks In G1040e

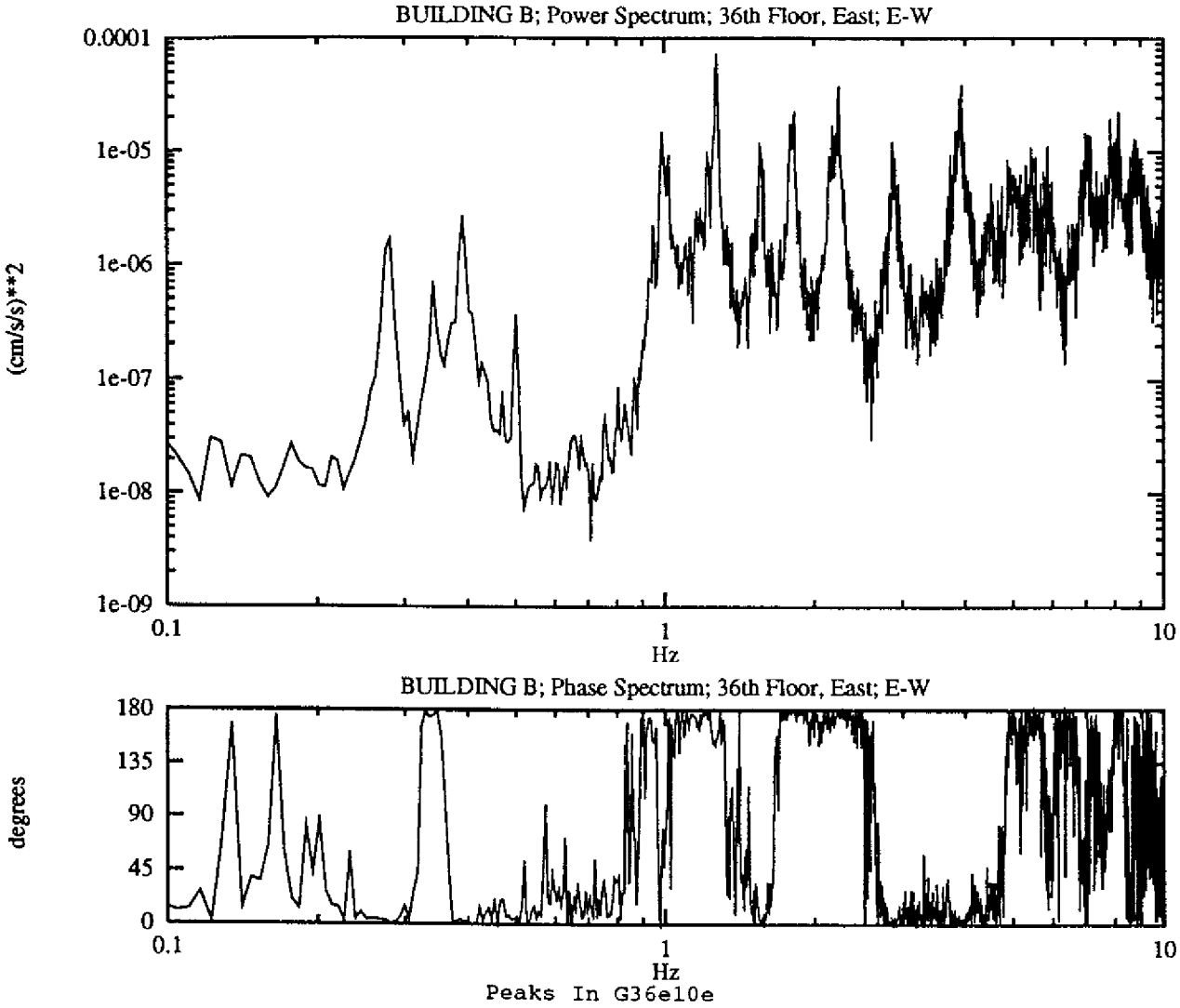
Hz-data	Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2808	0.2791	6.9478e-05	7.2129e-05	177	1.2
0.3418	0.3428	4.3055e-06	4.3698e-06	179	1.8
0.3906	0.3905	2.7249e-05	2.7262e-05	178	1.4
0.9460	0.9451	4.7272e-06	4.7404e-06	177	1.1
1.1597 T3	1.1600	1.5128e-06	1.5125e-06	169	0.7
1.2695	1.2700	1.2770e-06	1.2778e-06	157	1.1
1.5625	1.5600	1.9912e-06	2.0619e-06	8	0.7
1.8311 T4	1.8287	1.1918e-05	1.2577e-05	1	0.3
2.2400	2.2385	6.8093e-05	6.9857e-05	5	0.3
2.8687	2.8686	1.2595e-05	1.2636e-05	27	0.3
3.7415	3.7424	1.7382e-05	1.7405e-05	166	0.2
3.9429	3.9431	3.2101e-05	3.4332e-05	177	0.1
4.1870 T5	4.1878	2.0377e-05	2.0027e-05	168	0.2
4.8767	4.8775	1.1698e-05	1.1921e-05	11	0.2

RMS-acc	RMS-vel	RMS-dsp
1.049e-01	1.147e-02	8.392e-03

FIGURE C-8

BUILDING B - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	36th Floor, East
Reference	Sensor Location	10th Floor, Center
	Sensor Direction	East-West



Peaks In G36e10e

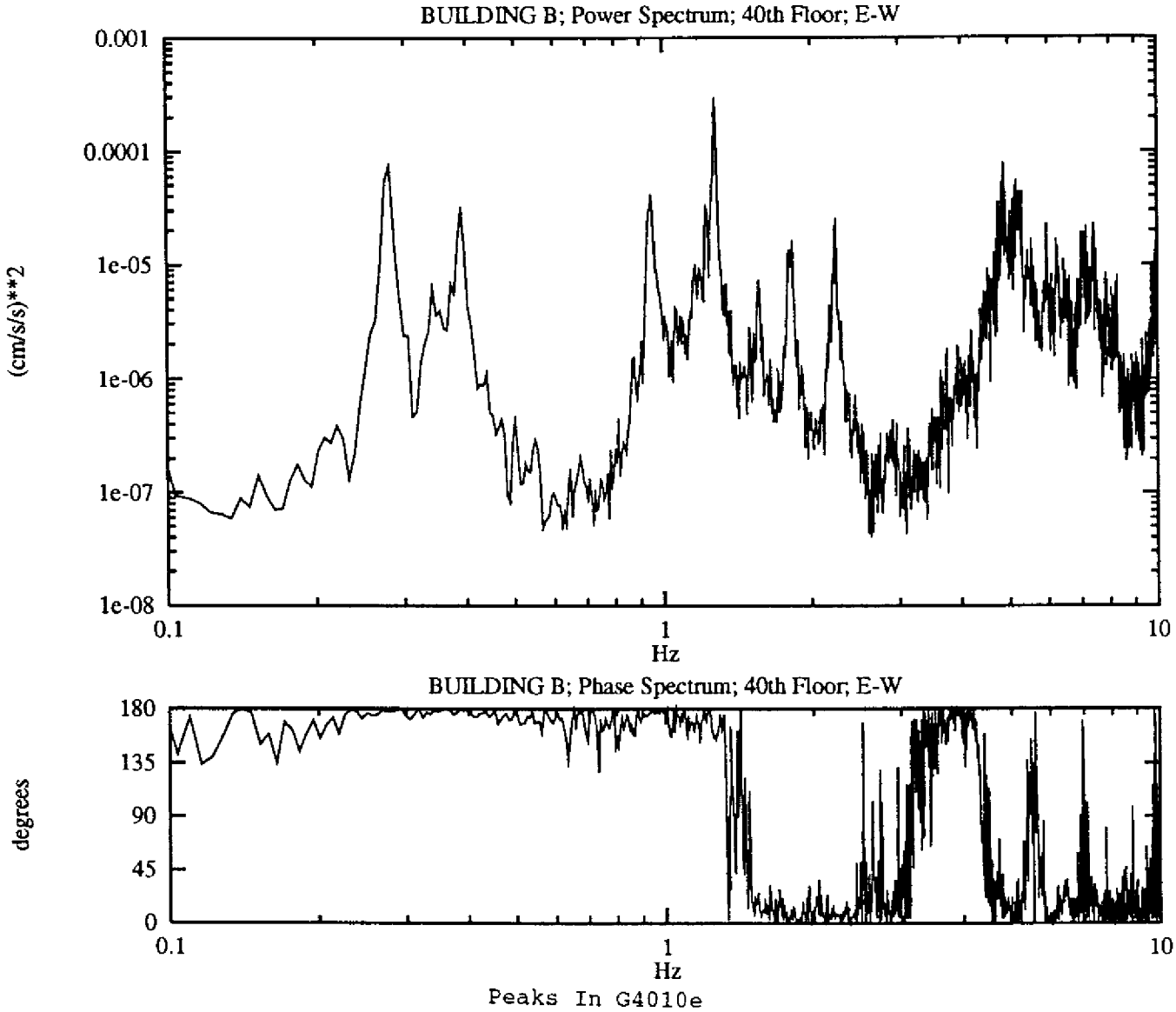
Hz-data	Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2808	0.2789	1.7068e-06	1.7844e-06	0	1.3
0.3418	0.3421	7.2849e-07	7.3016e-07	175	1.3
0.3906	0.3905	2.6772e-06	2.6776e-06	2	1.3
0.5005	0.5003	3.6844e-07	3.6880e-07	6	0.7
0.9460	0.9481	2.2606e-06	2.3637e-06	152	0.6
0.9888 T1	0.9884	1.4703e-05	1.4722e-05	80	0.7
1.0193 T2	1.0176	9.1861e-06	9.5144e-06	173	0.5
1.2756	1.2756	7.2635e-05	7.2598e-05	157	0.7
1.5625	1.5641	1.0616e-05	1.0788e-05	2	0.4
1.8311 T4	1.8283	2.1242e-05	2.2709e-05	178	0.3
2.2400	2.2389	3.7712e-05	3.8266e-05	174	0.4
2.8687	2.8684	1.2039e-05	1.2100e-05	4	0.3
3.9429	3.9432	3.8747e-05	3.9101e-05	1	0.1
4.5227	4.5228	5.2128e-06	5.0068e-06	33	0.1
4.8828	4.8825	8.4336e-06	8.1062e-06	152	0.2

RMS-acc	RMS-vel	RMS-dsp
1.037e-01	4.598e-03	1.828e-03

FIGURE C-9

BUILDING B - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	40th Floor, Center
Reference	Sensor Location	10th Floor, Center
	Sensor Direction	East-West



Peaks In G4010e

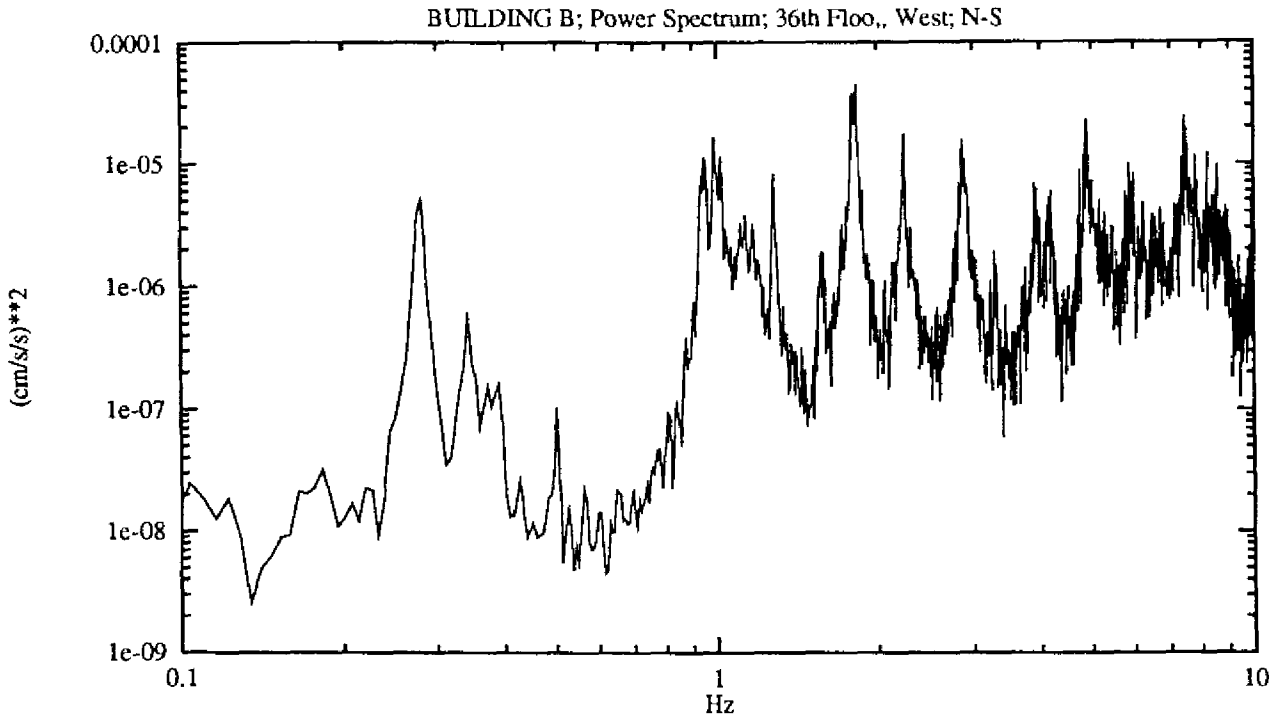
Hz-data	Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2808	0.2792	7.5738e-05	7.8365e-05	177	1.2
0.3418	0.3423	6.2898e-06	6.3088e-06	179	1.8
0.3906	0.3904	3.2550e-05	3.2596e-05	178	1.3
0.9460	0.9465	3.8941e-05	3.8981e-05	177	0.9
1.0559 T2	1.0584	4.1825e-06	4.3809e-06	175	0.8
1.0803	1.0800	3.5445e-06	3.5465e-06	171	0.7
1.1597	1.1589	1.0171e-05	1.0207e-05	169	0.8
1.1841	1.1841	9.0985e-06	9.1009e-06	169	1.6
1.2756	1.2755	2.9857e-04	2.9898e-04	160	0.6
1.5625	1.5636	7.3950e-06	7.4506e-06	8	0.8
1.8311 T4	1.8291	1.6281e-05	1.6868e-05	1	0.3
2.2400	2.2383	2.5475e-05	2.5988e-05	5	0.4
4.8767	4.8782	7.8683e-05	8.0109e-05	11	0.2

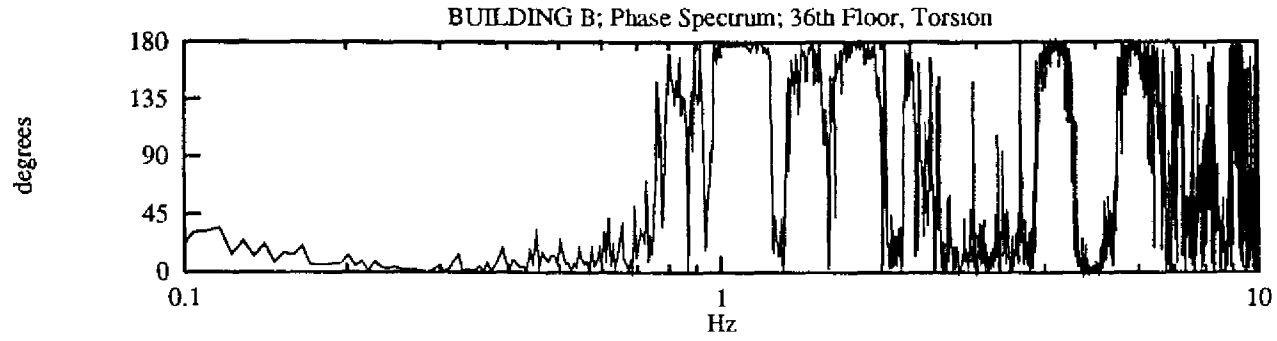
RMS-acc	RMS-vel	RMS-dsp
1.229e-01	1.126e-02	5.976e-03

FIGURE C-10

BUILDING B - Power and Phase Spectra of Measured Acceleration

Measurement	Date	August 15, 1991
Measurement	Sensor Location	36th Floor, West
Reference	Sensor Location	36th Floor, East
	Sensor Direction	North-South





Peaks In G36w36en

Hz-data	Hz-fit	Amp^2-data	Amp^2-fit	Phase	Damping %
0.2808	0.2791	5.1004e-06	5.2908e-06	0	1.2
0.3418	0.3419	6.1940e-07	6.1956e-07	1	1.3
0.3906	0.3890	1.6010e-07	1.6403e-07	19	1.7
0.5005	0.5005	1.0596e-07	1.0594e-07	23	0.7
0.9460	0.9483	1.0672e-05	1.1001e-05	61	0.8
0.9888 T1	0.9883	1.6407e-05	1.6436e-05	175	0.7
1.0193 T2	1.0174	1.1486e-05	1.1936e-05	177	0.5
1.1292 T3	1.1287	3.8387e-06	3.8445e-06	175	0.7
1.2756	1.2758	8.1588e-06	8.1360e-06	21	0.5
1.5747	1.5731	1.8998e-06	1.9595e-06	127	0.3
1.8311 T4	1.8286	4.4506e-05	4.6968e-05	175	0.3
2.2400	2.2385	1.7490e-05	1.8001e-05	137	0.2
2.8687	2.8687	1.5584e-05	1.5557e-05	13	0.4
3.9001	3.9018	6.9523e-06	7.0333e-06	115	0.2
4.1870 T5	4.1879	5.9891e-06	5.8413e-06	172	0.3
4.8828	4.8800	2.1977e-05	2.2888e-05	0	0.2

RMS-acc	RMS-vel	RMS-dsp
7.776e-02	4.319e-03	1.990e-03

APPENDIX D
SOURCE CODE LISTING FOR COMPUTER PROGRAM AMB

The computer program AMB make extensive use of the numerical analysis libraries from Numerical Recipes in C: The Art of Scientific Computing [Press et. al 1988]. These routines are copyright (C) 1987,1988,1992 Numerical Recipes Software and are reproduced by permission, from the book Numerical Recipes: The Art of Scientific Computing, published by Cambridge University Press. Comments in the code indicate page numbers from the book where the routines came from. Some routines from this book were taken verbatim, whereas others required some modification.

D.1 Input

Program AMB is interactive with the user, but requires no special graphics interface. The compiled version of the program has no command line arguments. Control information, such as sample rate and desired frequency resolution are supplied interactively.

D.1.1 Example of Terminal Session

```
reeltime 15% amb
 Reference data file name: duane.52e.n
 Response  data file name: duane.52c.n
 Number of data points: 60000
 Sample rate (Hz): 50
 Maximum frequency in the spectrum (Hz): 25.00
 Target  frequency interval (Hz): 0.01
 Actual  frequency interval (Hz): 0.012207
 Points per spectrum: 2048
 Number of  averages: 28
 Amplitude / Phase file name: G52cen
reeltime 16%
```

The input files must be in ASCII format. The entries in the input files must be separated by tabs, blank space, or carriage returns (new lines). The time axis of the data must be omitted. The data files require no special headers. The time histories from the measurement location sensor and the reference location sensor must be in separate files. (The UNIX command "more" displays a file page by page.)

D.1.2 Example of Input Files

```
reeltime 16% more duane.52e.n
-2.889475e-02
-1.002818e-02
1.206781e-02
```

... following data deleted ...

```
reeltime 17% more duane.52c.n
4.691148e-02
4.844121e-02
2.226596e-02
```

... following data deleted ...

D.2 Output

Program AMB creates a single output file of three-column, tab delimited, ASCII data. The first column contains the frequency axis data. The second column contains the power spectrum data as computed by equation (5.6). The third column contains the phase spectrum data as computed by equation (5.9).

D.2.1 Example of Output File

```
reeltime 18% more G52cen
1.22070313e-02   3.8438816e-06   1.3000222e+01
2.44140625e-02   3.1228686e-07   7.0814271e+00
3.66210938e-02   2.0304019e-07   1.2047212e+00
```

... following data deleted ...

```
/*************************************************************************/
/* Program amb.c  -  Ambient  Vibration  Spectral  Analysis              */
/* calculates the power spectra Gx, Gy, and Gxy using data sets x & y    */
/* Averaging and Windowing are Incorporated.                             */
/* W.H. Press et al. Numerical Recipes In C. Cambridge Press - Ch. 12    */
/*                                                                       */
/* to compile: cc -O -o amb amb.c fft.c nrutil.c                         */
/* (c)  H.P. Gavin, Dept. of Civil Eng., Princeton University, 2-91      */
/*************************************************************************/

#define  MXLN   85
#include <math.h>
#include <stdio.h>

main()
{
        char    x_filename[MXLN],
                y_filename[MXLN],
                G_filename[MXLN],
                c;

        FILE    *fpx,*fpy,      /* file pointers to x & y data files    */
                *fpG;           /* file pointer to the spectra file     */

        float   *gx,*gy,        /* power spectra of x & y data files    */
                *gxyr,*gxyi,    /* real & imaginary parts of cross spct */
                *Coh,           /* coherence spectrum                   */
                *Pha,           /*  phase    spectrum                   */
                sr,df,          /* sample rate and frequency interval   */
                w2,             /* circular frequency in rad/sec squared*/
                data,
                *vector();

        int     m = 2,          /* number of frequency values / segment */
                tg_pts,         /* target number of freq vals / segment */
                k,              /* number of segments to be averaged    */
                j,              /* counter                              */
                ovrlap = 1,     /* 1: overlap segments; 0: no overlap   */
                points = 0;     /* number of data points ( 10,000 )     */

        void    spectra(),      /* returns spectra and cross spectra    */
                coh(),          /* returns coherence and phase          */
                free_vector();

        printf (" Reference data file name: ");
        scanf ("%s",x_filename);
        if (( fpx=fopen(x_filename,"r"))==NULL) (
                printf ("  error: cannot open %s\n",x_filename);
                exit (0);
        )
        printf (" Response  data file name: ");
        scanf ("%s",y_filename);
```

```
if (( fpy=fopen(y_filename,"r"))==NULL) (
        printf ("  error: cannot open %s\n",y_filename);
        exit (0);
}

        /*      Size  The  Data         */

printf (" Number of data points: ");
scanf ("%d",&points);
if (points == 0) {
        while ( (c=fscanf(fpx,"%f",&data)) != EOF ) ++points;
        printf (" %d X data points\n",points);
        points = 0;
        while ( (c=fscanf(fpy,"%f",&data)) != EOF ) ++points;
        printf (" %d Y data points\n",points);
        rewind (fpx);
        rewind (fpy);
}

        /*      Choose  Frequency  Resolution   */

printf (" Sample rate (Hz): ");
scanf ("%f",&sr);
printf (" Maximum frequency in the spectrum (Hz): %.2f\n", sr/2);
printf (" Target  frequency interval (Hz): ");
scanf ("%f",&df);
tg_pts = sr/df;
while (m < tg_pts) m <<= 1; m >>= 2;
k = points/m - 1;
df = sr / (2*m);
printf (" Actual  frequency interval (Hz): %f\n", df);
printf (" Points per spectrum: %d\n", m);
printf (" Number of  averages: %d\n", k);
if (k < 1) {
        printf ("  error: %d is too small\n", k);
        exit(0);
}

gx = vector(1,m);
gy = vector(1,m);
gxyr = vector(1,m);
gxyi = vector(1,m);

spectra (fpx,fpy,gx,gy,gxyr,gxyi,m,k,ovrlap);

fclose (fpx);
fclose (fpy);

Coh = vector(1,m);
Pha = vector(1,m);

coh (gx,gy,gxyr,gxyi,Coh,Pha,m);
```

```
        /*      Print the Spectra       */

    printf (" Amplitude / Phase file name: ");
    scanf ("%s", G_filename);
    if (( fpG=fopen(G_filename,"w"))==NULL) {
        printf ("  error: cannot open %s\n",G_filename);
        exit (1);
    }
    for (j=2;j<=m;j++)
        fprintf (fpG,"%e\t%e\t%7.2f\n",(j-1)*df,gy[j],Pha[j]);
    fclose (fpG);

    free_vector(Coh,1,m);
    free_vector(Pha,1,m);
    free_vector(gx,1,m);
    free_vector(gy,1,m);
    free_vector(gxyr,1,m);
    free_vector(gxyi,1,m);



    /*              Function spectra()                          */
    /* Power Spectra Using The Fast Fourier Transform           */
    /* Adapted from Numerical Recipes in C - Ch 12              */
    /* normalization: SUM(gx[i]) i=1..m = variance(x)           */

static float sqrarg;
#define SQR(a) (sqrarg=(a), sqrarg*sqrarg)
#define tpi 6.28318530717959

#define WINDOW(j,c,b) (0.5-0.5*cos(tpi*((j)-1)*(c)))    /* Hanning      */
/* #define WINDOW(j,a,b) (1.0-fabs((((j)-1)-(a))*(b)))  /* Parzen       */
/* #define WINDOW(j,a,b) (1.0-SQR((((j)-1)-(a))*(b)))   /* Welch        */
/* #define WINDOW(j,a,b) 1.0                            /* Square       */

void spectra(fpx,fpy,gx,gy,gxyr,gxyi,m,k,ovrlap)
FILE *fpx,*fpy;
float gx[],gy[],gxyr[],gxyi[];
int m,k,ovrlap;
{
    int n,tn,tn3,tn4,kk,j,tj;               /* Useful Quantities    */
    float a,b,c,sumw=0.0,den=0.0;
    float *x1,*x2,*y1,*y2;                  /* Operating vectors    */
    float *Sx,*Sy;                          /* FFT'd vectors        */
    float *vector();
    void segments(),twofft(),free_vector();

    tn4 = (tn3 = (tn = n+(n=m+m))+3)+1;
    a = m - 0.5;
    b = 1.0/(m+0.5);
    c = 1.0/(n-1.0);
```

```
x1 = vector(1,m);
x2 = vector(1,n);
y1 = vector(1,m);
y2 = vector(1,n);
Sx = vector(1,tn);
Sy = vector(1,tn);
for (j=1; j<=n; j++) sumw += SQR(WINDOW(j,c,b));
for (j=1; j<=m; j++) gx[j]=gy[j]=gxyr[j]=gxyi[j]=0.0;
if (ovrlap)             /* Initialize the "save" half-buffer.    */
        for (j=1; j<=m; j++) {
                fscanf(fpx,"%f",&x1[j]);
                fscanf(fpy,"%f",&y1[j]);
        }
for (kk=1; kk<=k; kk++) {
        segments(fpx,x1,x2,a,b,c,m,ovrlap);
        segments(fpy,y1,y2,a,b,c,m,ovrlap);
                        /* Fourier transform the windowed data  */
        twofft(x2,y2,Sx,Sy,n,1);
                /* Sum the results into the previous segments   */
        gx[1] += (SQR(Sx[1]) + SQR(Sx[2]));
        gy[1] += (SQR(Sy[1]) + SQR(Sy[2]));
        for (j=2; j<=m; j++) {
                tj = j+j;
                gx[j] += (SQR(Sx[tj-1]) + SQR(Sx[tj])
                        + SQR(Sx[tn3-tj]) + SQR(Sx[tn4-tj]));
                gy[j] += (SQR(Sy[tj]) + SQR(Sy[tj-1])
                        + SQR(Sy[tn3-tj]) + SQR(Sy[tn4-tj]));
                gxyr[j] += (Sx[tj-1]*Sy[tj-1] + Sx[tj]*Sy[tj]
                                + Sx[tn3-tj]*Sy[tn3-tj]
                                + Sx[tn4-tj]*Sy[tn4-tj]);
                gxyi[j] += (Sx[tj]*Sy[tj-1] - Sx[tj-1]*Sy[tj]
                                - Sx[tn4-tj]*Sy[tn3-tj]
                                + Sx[tn3-tj]*Sy[tn4-tj]);
        }
        den += sumw;
}
den *= n;                       /* Correct normalization.       */
for (j=1; j<=m; j++) {
        gx[j] /= den;
        gy[j] /= den;
        gxyr[j] /= den;
        gxyi[j] /= den;
}
free_vector(x1,1,m);
free_vector(x2,1,n);
free_vector(y1,1,m);
free_vector(y2,1,n);
free_vector(Sx,1,tn);
free_vector(Sy,1,tn);
return;
```

```
        /*              Function segments()                          */
        /* Gets two complete segments into the work space          */
        /*              Numerical Recipes in C p446                  */

void segments(fp,z1,z2,a,b,c,m,ovrlap)
FILE    *fp;
float   *z1,*z2;
float   a,b,c;
int     m,ovrlap;
{
        int     j;
        if (ovrlap) {
                for (j=1; j<=m; j++) z2[j]=z1[j];
                for (j=1; j<=m; j++) fscanf(fp,"%f",&z1[j]);
                for (j=1; j<=m; j++) z2[m+j]=z1[j];
        } else {
                for (j=1; j<=m+m; j++)
                        fscanf(fp,"%f",&z2[j]);
        }
        for (j=1; j<=m+m; j++)          /* Apply the window to the data.*/
                z2[j] *= WINDOW(j,c,b);
        return;
}


        /*              Function coh()                               */
        /* Computes the coherence and the phase functions using */
        /* the power spectra gx and gy and the cross spectrum gxy*/

void coh(gx,gy,gxyr,gxyi,Coh,Pha,m)
float   *gx,*gy,*gxyr,*gxyi,
        *Coh,*Pha;
int     m;
{
        int     j;
        float   den1, den2;

        for (j=1; j<=m; j++) {
                den1 = SQR(gxyr[j]) + SQR(gxyi[j]);
                if (den2 = (gx[j]*gy[j]) != 0.0)
                        Coh[j] = den1/den2;
                else    Coh[j] = 0.0;
                if (gxyi[j] != 0.0)
                        Pha[j] = fabs(atan2(gxyi[j],gxyr[j])) * 57.29578;
                else    Pha[j] = Pha[j-1];
        }
        return;
```

```
/*                              FILE fft.c                                  */
/* Fast Fourier Transform routines from Numerical Recipes in C, by Press*/
/* et al. Cambridge University Press, 1988                                  */

#include <math.h>

        /*              Function four1()                            */
        /* Fast Fourier Transform - Numerical Recipes in C p411.*/

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

void four1(data,nn,isign)
float   data[];
int     nn,isign;
{
        int     n,mmax,m,j,istep,i;
        double  wtemp,wr,wpr,wpi,wi,theta;
        float   tempr,tempi;

        n=nn << 1;
        j=1;
        for (i=1;i<n;i+=2) {                            /* Bit Reversal */
                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 = 2*mmax;
                theta = 6.28318530717959 / (isign*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;           /* Danielson-Lanczo
                                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;
                                                /* Trigonometric Recurrence
```

```
                        wr = (wtemp=wr)*wpr - wi*wpi + wr;
                        wi = wi*wpr + wtemp*wpi + wi;
                }
                mmax = istep;
        }
        return;
}


        /*              function twofft()                       */
        /* The simultaneous FFT of two real valued functions    */
        /*      from  Numerical Recipes In C  p 415             */

void twofft(data1,data2,fft1,fft2,n)
float data1[],data2[],fft1[],fft2[];
int n;
{
        int nn3,nn2,jj,j;
        float rep,rem,aip,aim;
        void four1();

        nn3=1+(nn2=2+n+n);
        for (j=1,jj=2;j<=n;j++,jj+=2) {
                fft1[jj-1]=data1[j];
                fft1[jj]=data2[j];
        }
        four1(fft1,n,1);
        fft2[1]=fft1[2];
        fft1[2]=fft2[2]=0.0;
        for (j=3;j<=n+1;j+=2) {
                rep=0.5*(fft1[j]+fft1[nn2-j]);
                rem=0.5*(fft1[j]-fft1[nn2-j]);
                aip=0.5*(fft1[j+1]+fft1[nn3-j]);
                aim=0.5*(fft1[j+1]-fft1[nn3-j]);
                fft1[j]=rep;
                fft1[j+1]=aim;
                fft1[nn2-j]=rep;
                fft1[nn3-j] = -aim;
                fft2[j]=aip;
                fft2[j+1] = -rem;
                fft2[nn2-j]=aip;
                fft2[nn3-j]=rem;
        }
        return;
}


        /*              function realft()                       */
        /* The FFT of 2n real valued discrete function pointS   */
        /*      from  Numerical Recipes In C  p 417             */

void realft(data,n,isign)
```

```
float data[];
int n,isign;
{
        int i,i1,i2,i3,i4,n2p3;
        float c1=0.5,c2,h1r,h1i,h2r,h2i;
        double wr,wi,wpr,wpi,wtemp,theta;
        void four1();

        theta=3.141592653589793/(double) n;
        if (isign == 1) {
                c2 = -0.5;
                four1(data,n,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;
        n2p3=2*n+3;
        for (i=2;i<=n/2;i++) {
                i4=1+(i3=n2p3-(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);
        }
}
```

```
/*                          FILE nrutil.c                                   */
/* Memory allocation functions from Numerical Recipes in C, by Press,      */
/* Cambridge University Press, 1988                                        */

/* #include <malloc.h> */
#include <stdio.h>

typedef struct FCOMPLEX {float r,i;} fcomplex; /* */

void nrerror(error_text)        /* print error message to stderr           */
char error_text[];
{
        void exit();

        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}


float *vector(nl,nh)    /* allocate storage for a vector                   */
int nl,nh;
{
        float *v;

        v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

fcomplex *Cvector(nl,nh)
int nl,nh;              /* allocate storage for a complex vector           */
{
        fcomplex *v;

        v=(fcomplex *)malloc((unsigned) (nh-nl+1)*sizeof(fcomplex));
        if (!v) nrerror("allocation failure in Cvector()");
        return v-nl;
}

double *dvector(nl,nh)  /* allocate storage for a vector                   */
int nl,nh;
{
        double *v;

        v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
}

int *ivector(nl,nh)     /* allocate storage for a vector                   */
```

```
int nl,nh;
{
        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl;
}

float **matrix(nrl,nrh,ncl,nch) /* allocate storage for a matrix        */
int nrl,nrh,ncl,nch;
{
        int     i;
        float   **m;

        m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;
        for (i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float))
                if (!m[i]) nrerror("allocation failure 2 in matrix()");
                m[i] -= ncl;
        }
        return m;
}

fcomplex **Cmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;    /* allocate storage for a Complex matrix        */
{
        int     i;
        fcomplex **m;

        m=(fcomplex **)malloc((unsigned) (nrh-nrl+1)*sizeof(fcomplex*));
        if (!m) nrerror("allocation failure 1 in Cmatrix()");
        m -= nrl;
        for (i=nrl;i<=nrh;i++) {
                m[i]=(fcomplex *)malloc((unsigned)(nch-ncl+1)*sizeof(fcompl
                if (!m[i]) nrerror("allocation failure 2 in Cmatrix()");
                m[i] -= ncl;
        }
        return m;
}

double **dmatrix(nrl,nrh,ncl,nch)   /* allocate storage for a matrix    */
int nrl,nrh,ncl,nch;
{
        int     i;
        double  **m;

        m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
        if (!m) nrerror("allocation failure 1 in dmatrix()");
        m -= nrl;
```

```
        for (i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double
                if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
                m[i] -= ncl;
        }
        return m;
}

int **imatrix(nrl,nrh,ncl,nch)  /* allocate storage for a matrix        */
int nrl,nrh,ncl,nch;
{
        int     i;
        int     **m;

        m=(int **) malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
        if (!m) nrerror("allocation failure 1 in imatrix()");
        m -= nrl;
        for (i=nrl;i<=nrh;i++) {
                m[i]=(int *) malloc((unsigned) (nch-ncl+1)*sizeof(int));
                if (!m[i]) nrerror("allocation failure 2 in imatrix()");
                m[i] -= ncl;
        }
        return m;
}

float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
        int i,j;
        float **m;

        m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure in submatrix()");
        m -= newrl;

        for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

        return m;
}


void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
        free((char*) (v+nl));
}

void free_Cvector(v,nl,nh)
fcomplex *v;
int nl,nh;
```

```
{
        free((char*) (v+nl));
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
        free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v;
int nl,nh;
{
        free((char*) (v+nl));
}

void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
        int     i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_Cmatrix(m,nrl,nrh,ncl,nch)
fcomplex **m;
int nrl,nrh,ncl,nch;
{
        int     i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
        int     i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
```

```
        int     i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}

float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
        int i,j,nrow,ncol;

        float **m;

        nrow=nrh-nrl+1;
        ncol=nch-ncl+1;

        m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
        if (!m) nrerror("allocation failure in convert_matrix()");
        m -= nrl;
        for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
        return m;
}

void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}
```