Stanford Exploration Project, Report 73, November 18, 1997, pages 77–101

Stanford Exploration Project, Report 73, November 18, 1997, pages 77–101

Seismic data processing with Mathematica[1]

Ray Abma[1]

keywords: *graphics, Fourier transform, algorithm*

ABSTRACT

When treated as vectors or matrices, seismic traces may be manipulated with Mathematica's large set of mathematical algorithms and displayed in a variety of presentations. A program for converting seismic files into Mathematica format and examples of Mathematica processes are given. A variety of plots are shown to illustrate Mathematica's display capabilities, including wiggle-trace plots, variable-density plots, 3-D plots, and others.

INTRODUCTION

Mathematica is a computer algebra program that is generally used to manipulate mathematical expressions and display the results. For long, complex expressions, computer algebra programs have proven useful for simplifying and checking work done manually. While mathematical manipulations are not considered here, the mathematical tools available may simplify the tasks of programming and testing new algorithms as well as for displaying results.

Creating simple programs with Mathematica takes less work than similar tasks done with FORTRAN or C. Numbers with any precision may be used, complex numbers are treated naturally, and the syntax is generally simple within Mathematica. Where FORTRAN or C require programming structures that pass numerical values through an algorithm, Mathematica allows relationships to be defined and evaluated as needed. Variables may become complex numbers as needed and may be just symbolic values until they are supplied with numerical values.

Displaying seismic data is generally done with black and white rasters. While these displays may be used from Mathematica, many other displays are available for special purposes, including color displays. Most of these displays have automatic scaling and annotation options for easy use, but options to control scaling, annotation, color, and

[1]Mathematica is a registered trademark of Wolfram Research, Inc.

[1]**email:** not available

other features are available. In addition, new displays may be created with plotting primitives.

This paper shows how seismic files may be converted to Mathematica matrix files and then manipulated as vectors or matrices. A set of displays is presented to illustrate Mathematica's graphics power.

USING EXTERNAL FILES IN MATHEMATICA

Real seismic data is not always necessary for algorithm development; however the ability to input seismic data collected or generated elsewhere is important. Seismic files may be represented within Mathematica as matrices and then processed and displayed. Since the SEP format used at the Stanford Exploration Project is the most common data format here, a program was needed to convert SEP format to the external Mathematica matrix format. Mathematica matrix format does not support the FORTRAN E format, and care must be taken to retain the numerical range of the input. A program listing of the routine used here is given in the appendix.

Once an SEP file is converted to Mathematica matrix format, the data are input into a Mathematica session by entering:

```
a = << wz25.m;
```

This expression takes the external file 'wz25.m' and creates the matrix 'a'. The size and type of the matrix are automatically taken from the input and do not need to be prespecified. Notice the ';' following the command. This is needed to avoid having Mathematica print out the matrix after it is input. Seismic files are generally too large to make the standard matrix printout useful.

SAMPLE PROCESS DEVELOPMENT

As a comparison between developing a process using Mathematica and using the standard SEP software, hyperbola whitening is considered. Here, a hyperbola is generated with constant amplitude at each sample. The hyperbola is whitened by transforming the matrix containing it with a two-dimensional Fourier transform, dividing the transformed data by its absolute value, and inverse transforming back to the original domain. The new hyperbola will have a amplitude different from the original hyperbola. Using Mathematica, the process is implemented as follows:

```
v=6000.
t0=0.05
n1=100
n2=50
```

```
xinc = 25.0
tinc = 0.004
          (* create an empty matrix *)
data = Table[N[0.],{i,1,n1},{j,1,n2}];
          (* build a hyperbola within the matrix *)
For[i=1, i<n1+1,i ++, x=(i-n1/2) xinc; t=Sqrt[(x^2/(v^2) + t0^2)]; j= Floor[t/
tinc];data[[i,j]] = N[1.0] ]
          (* display the hyperbola *)
ListDensityPlot[data,Mesh->False]
          (* do a 2-D Fourier transform *)
fdata =Fourier[data];
ListDensityPlot[Re[fdata],Mesh->False]
          (* divide the transformed data by its absolute value *)
sdata = fdata/Abs[fdata];
newdata = InverseFourier[sdata];
          (* display the final result  *)
ListDensityPlot[Re[newdata],Mesh->False]
```

Notice that the matrix '*fdata*' does not have to be dimensioned or specified as an array of complex numbers. However, note that the final display of '*newdata*' plotted only the real part of the matrix, since the ListDensityPlot routine expects real numbers. ListDensityPlot produces a raster-like display using various gray-scale densities. The 'Mesh->False' option removes the default overlay mesh that generally obliterates seismic data. The Fourier routine applies a Fourier transform to an object supplied to it. If the object has two dimensions, such as our matrix, it will do a two-dimensional Fourier transform. To apply a Fourier transform to only one axis of a matrix, use Map[Fourier,a]. The expression 'fdata/Abs[fdata]' divides each element of the complex matrix fdata by the absolute value of the corresponding element. The actual matrix multiplication and matrix inversion are done with different syntax.

The three displays created by this run are shown in Figures 1 to 3.

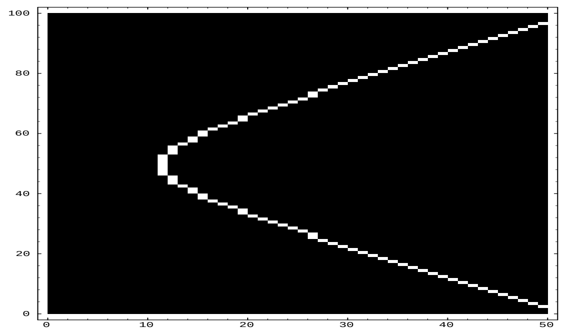

Figure 1: The original hyperbola with constant amplitude sample values. ray2-hyp [NR]

The comparable run using standard SEP software is as follows:

```
Spike n1=50 n2=100 k1=10 >spike.H
```

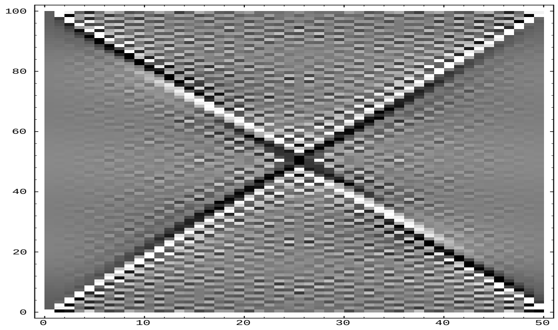

Figure 2: The real part of the Fourier transform of the original hyperbola. ray2-hypfk [NR]

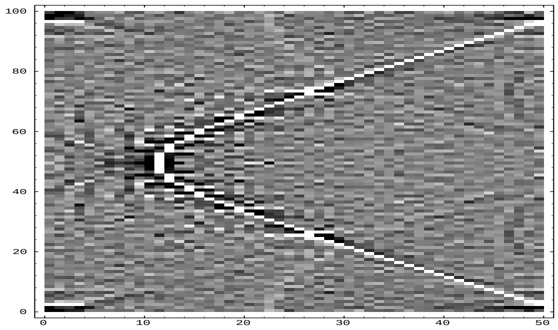

Figure 3: The hyperbola created from the inverse Fourier transform of the scaled data. ray2-newhyp [NR]

```
~/fort/hyper <spike.H >hyper.H
Bandpass <hyper.H >filter.H fhi=100
Rtoc <filter.H >cfilter.H
Ft3d sign1=1 sign2=1 <cfilter.H >cfft.H
Real <cfft.H >rfft.H
Cabs <cfft.H >absfft.H
Rtoc <absfft.H >cabsfft.H
Real <cfft.H >rjunk.H
Imag <cfft.H >ijunk.H
Add  rjunk.H absfft.H mode=d > rdjunk.H
Add  ijunk.H absfft.H mode=d > idjunk.H
Cmplx rdjunk.H idjunk.H >dfft.H
Rm *unk.H
Ft3d sign1=-1 sign2=-1 <dfft.H >cdinvfft.H
Real <cdinvfft.H >invfft.H
Taplot <invfft.H|Ta2vplot|Tube
Taplot < filter.H  | Ta2vplot | Tube
Taplot <rfft.H|Ta2vplot|Tube
```

The only nonstandard routine used here is the hyper routine that corresponds to the first ten lines of the Mathematica code. It is used to replace the spike synthetic with a hyperbola and is not shown here.

For small processes like this one, where the run time is insignificant compared to the coding time, Mathematica provides a quick method of getting results. For runs with significantly longer run times or for larger datasets, the standard SEP software is preferable since it is not limited to the memory size of the machine being used. The SEP software will run faster in computationally intense jobs, where Mathematica would be inpractical because of the overhead associated with the extra flexibility provided. Mathematica's interactive mode does provide more flexibility for observing the intermediate results than do the traditional FORTRAN or C codes.

A second sample shows a job building a Cagniard-De Hoop impulse response. The expression for the impulse response is shown here as given in Aki and Richards(1980).

$$v^{refl}(x,z,t) = \frac{A}{2\pi\rho_1\beta_1^2}\Re(\frac{\mu_1\eta_1 - \mu_2\eta_2}{\mu_1\eta_1 + \mu_2\eta_2})\frac{H(t - R_0/\beta_1)}{(t^2 - R_0^2/\beta_1^2)^{1/2}} \tag{1}$$

$$v^{refr}(x,z,t) = \frac{A}{2\pi\rho_1\beta_1^2}\Im(\frac{\mu_1\eta_1 - \mu_2\eta_2}{\mu_1\eta_1 + \mu_2\eta_2})\frac{H(t - t_h) - H(t - R_0/\beta_1)}{(R_0^2/\beta_1^2 - t^2)^{1/2}} \tag{2}$$

$$v^{direct}(x,z,t) = \frac{A}{2\pi\rho_1\beta_1^2}\frac{H(t - x/\beta_1)}{(t^2 - x^2/\beta_1^2)^{1/2}} \tag{3}$$

Coding and debugging these equations in FORTRAN or C would take some time, especially since several variables in the above equations are dependent on other variables, as seen below. The Mathematica input corresponding to these equations is as follows:

```
(* define a Heaviside function *)
H[xtemp_Real] :=
   ytemp = If[xtemp >= 0., 1., 0. ];Return[ytemp]

(* Set up the Cagniard-De Hoop equations *)

c = A/(2 Pi rho1 beta1^2)
c2 = (mu1 eta1 - mu2 eta2)/(mu1 eta1 + mu2 eta2)
refl = c Im[c2] (H[t-tn] - H[t-R0/beta1])/Sqrt[R0^2/beta1^2 - t^2]
refr = c Re[c2] (H[t-R0/beta1])/Sqrt[-R0^2/beta1^2 + t^2]
dirt = A/(2 Pi rho1 beta1^2) H[t-x/beta1]/( (t^2 - x^2/beta1^2)^(1/2) )

(*  fill in the variables *)
mu1 = beta1^2 rho1
mu2 = beta2^2 rho2
R0 = (x^2 + (z+z0)^2)^(1/2)
eta1 = (1/( beta1^2 ) - p^2)^(1/2)
eta2 = (1/( beta2^2 ) - p^2)^(1/2)
tn = x/beta2 + Abs[z+z0] (1/beta1^2 - 1/beta2^2)^(1/2)
p = ( x t + I Abs[z+z0] (t^2 - R0^2/beta1^2)^(1/2) )/( R0^2)

(* now define the actual values *)
beta1 = 3.5
rho1 = 2700
beta2 = 4.5
rho2 = 3300
x = 200
z0 = 25
z = 25
A = 1.

(* put the direct, reflected, and refracted arrivals together *)
all = dirt + refl + refr

(* Plot it *)
axx = Re[all]
plot = Plot[axx,{t,50,70},PlotRange->All,AxesLabel->{"t-sec.","amp."}]
```

A FORTRAN program is likely to be similar, except that the first few blocks of code would be reversed. With Mathematica, each expression may be printed as the variables are filled in, allowing the intermediate expressions to be examined to save debugging time. The standard Plot routine is used to display the output shown in figure 4.

DISPLAY SAMPLES

A wide variety of displays may be generated from within Mathematica. Several auxiliary programs are available to convert displays for common terminal types. These displays may also be output to a disk file in PostScript form for use in paper documents

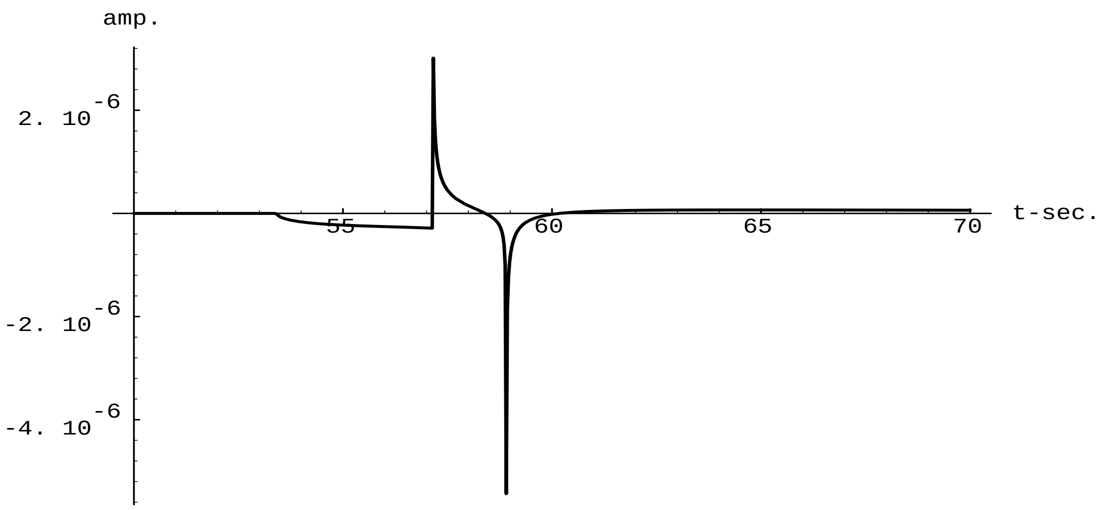

Figure 4: A Cagniard-De Hoop impulse response. $\boxed{\text{ray2-Cagniard}}$ [NR]

such as this one. The YPrint.m routine in the Appendix gives one method of doing this.

While several display options are available within Mathematica, displays may be custom-made by writing external packages from the display primitives provided. My package to display wiggle traces follows.

```
BeginPackage["Wiggle`"]

   Wiggle::usage = " plot a 2-D seismic section as a wiggle plot."
   Sigma::usage = " Get the sum of a vector."
   RMS::usage = " Get the Root mean squared of a vector."
   MRMS::usage = " Get the Root mean squared of a matrix"
   MDot::usage = " Get the dot product of a matrix with a matrix "

  Begin["`Private`"]
  Wiggle::badarg = " Bad input"

  Wiggle[a_List,s_] :=
   (
   Block[ {n,m,i,j,dim},
          dim = Dimensions[a];
          n = dim[[1]];
          m = dim[[2]];
          Show[
             Graphics[
              {
               Thickness[0.001],
               Table[Line[Table[{
                           Max[{ Min[{a[[j,i]],2 s}],-2 s}]
                                          +s j,m-i},{i,1,m}]],{j,1,n}]
              }
                    ]
              ] (* end of Show *)
         ] (* end of Block *)
   )

   Sigma[x_List] := ( Block[ {iii},
                      N[
                        Sum[ x[[iii]],{iii,1,Dimensions[x][[1]]}
                           ]
                       ]
                           ]
                    )
   RMS[x_List] := (
                      N[
                         Sqrt[ Sigma[x^2]
                           /Dimensions[x][[1]]
                             ]
                       ]
                  )

   MRMS[x_List] := (
                       N[ Sqrt[
```

```
                              Sigma[ Map[Sigma,N[x^2]] ]/
                              (Dimensions[x][[1]] Dimensions[x][[2]])
                             ]
                        ]
                  )
  MDot[x_List,y_List] := (
                      N[ Sum[
                             x[[i,j]] y[[i,j]],
                              { i,1,Dimensions[x][[1]] },
                              { j,1,Dimensions[x][[2]] }
                           ]
                       ]
                  )

End[]
EndPackage[]
```

Note that only the first part of this package is concerned with the actual plotting. The Sigma, RMS, MRMS, and MDot routines are used for data scaling.

To invoke Wiggle to display a matrix a, the routine is typed as follows:

```
Wiggle[a,MRMS[a]]
```

The Wiggle command takes the matrix to be plotted as the first parameter and requires a scale factor to be supplied as the second parameter. Here the routine to get the RMS of an entire matrix is used to supply the scale factor. An alternative to using MRMS[a] would be to use the maximum of the absolute values of all the values in a matrix.

```
Wiggle[a,Max[Abs[a]]]
```

If several displays are made for comparison, a numerical scale factor may be used.

```
Wiggle[a,3438.8]
```

A disadvantage of this plot is that it is slow and produces a useful result only when relatively few traces are plotted. Another disadvantage is that the axis-scales are not automatically provided. These scales might be added to the Wiggle package later. Another useful option would be a variable area addition to the wiggle traces.

A sample wiggle trace display is shown in Figure 5.

While the wiggle trace display is a common seismic display format, the built-in ListDensityPlot is faster and easier to use.

```
ListDensityPlot[a,Mesh->False]
```

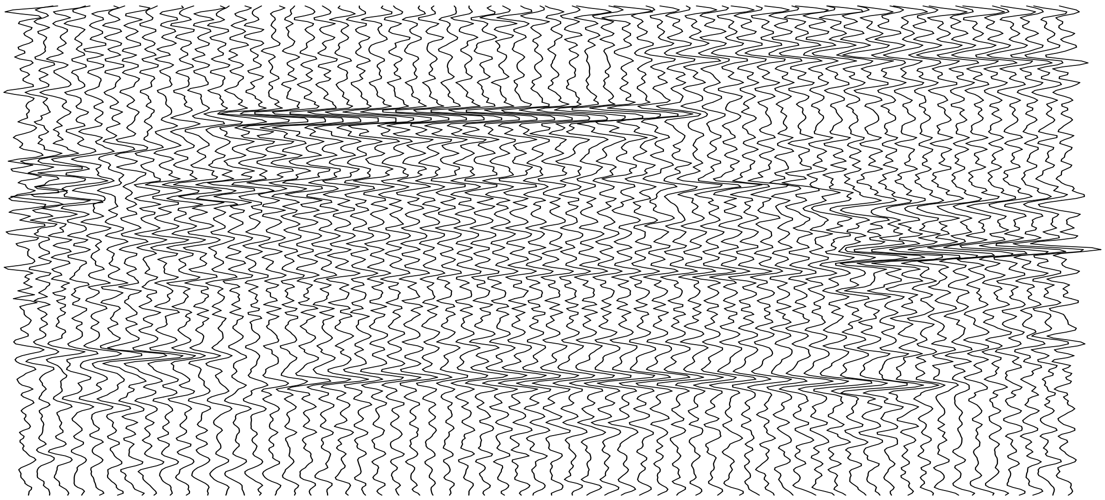

Figure 5: A wiggle trace plot. $\boxed{\text{ray2-wiggle}}$ [NR]

The 'Mesh->False' parameter suppresses the usual mesh that the ListDensityPlot covers the display. Seismic files generally contain so many samples that the mesh blocks out the data. Various scaling options are available, but the default generally does a good job. Samples of this type of plot are seen in Figures 1 to 3.

Occasionally, the standard plots may be supplemented by other displays. In this case, consider the case of a set of traces and their Hilbert transforms as seen in Figure 6 and 7.

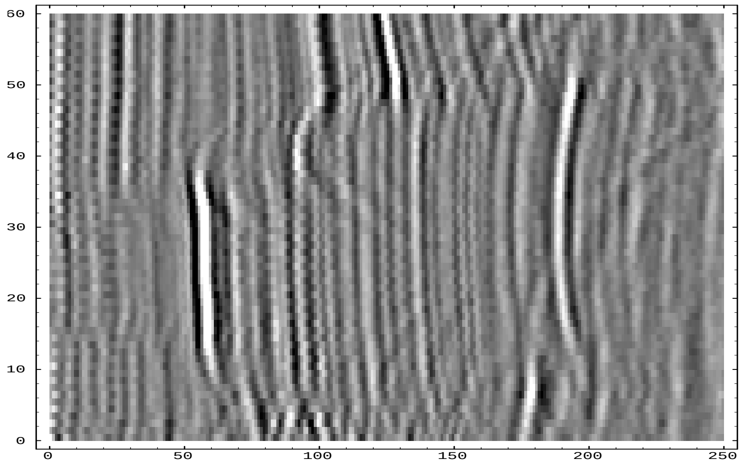

Figure 6: The original traces. ray2-origden [NR]

The envelope of the set of original traces may be generated using the square-root of the sums of the squares of the the corresponding samples in the original and transformed traces. The code to produce the envelope follows:

```
b = Map[Hilbert,a];
env=Sqrt[a^2+b^2];
```

The Hilbert transform routine in provided in the Appendix.

Figure 8 shows the ListDensityPlot display of the envelope trace. Figure 9 shows the data plotted with the ListPlot3D routine, subsampled in time. The 3-D plot gives a better idea of the variations of the envelope values.

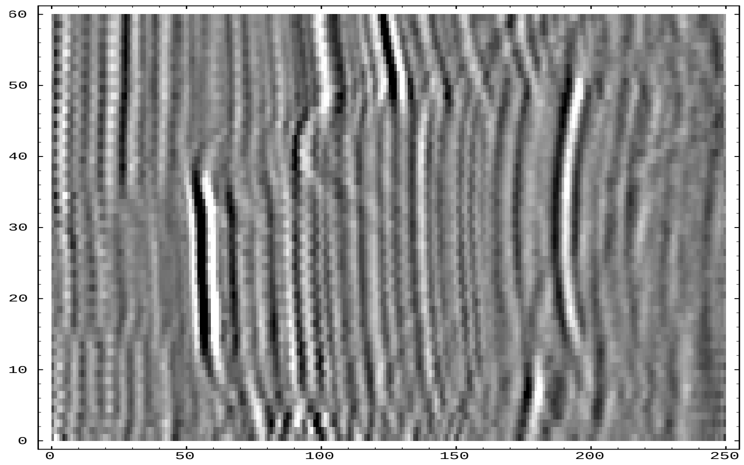

Figure 7: The Hilbert transform of the traces seen in Figure 6. ray2-hilbertden [NR]

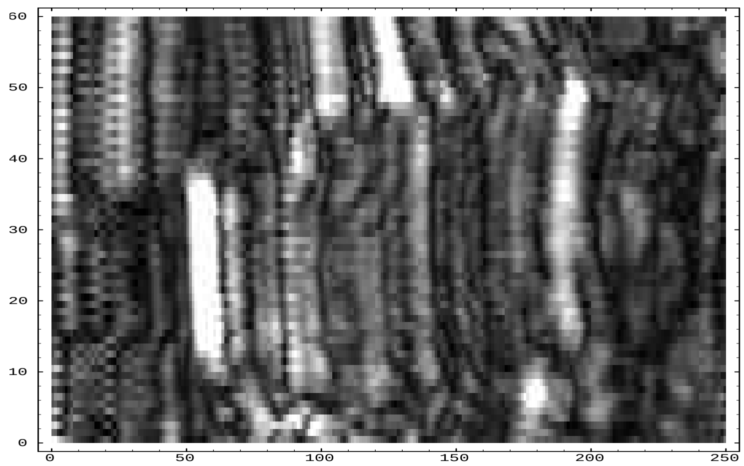

Figure 8: Density Plot of the envelope of the traces in Figure 6. ray2-envden [NR]

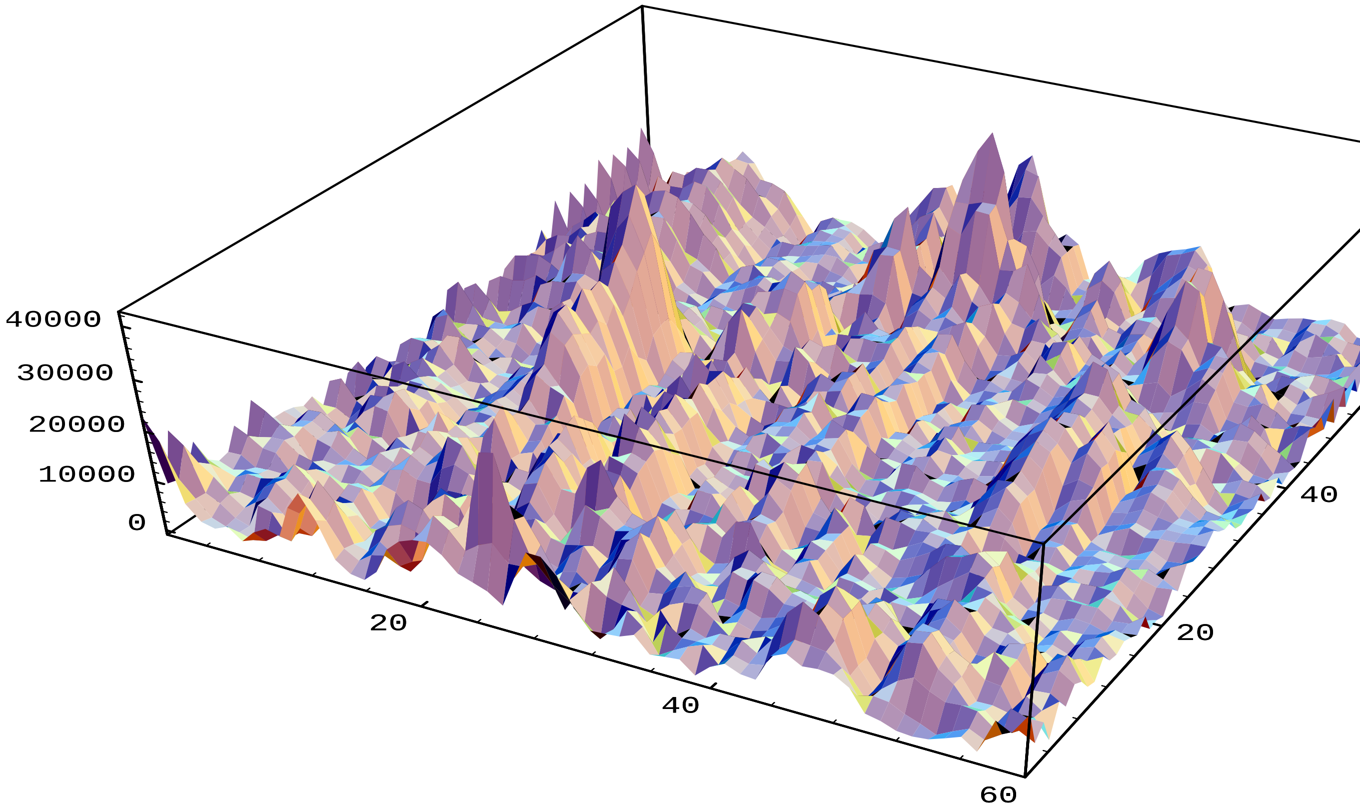

Figure 9: A three-D plot of the envelope data in Figure 8. ray2-threeDplot [NR]

A combination of Figures 8 and 9 may be created using a three-D plot with a shadow display. Figure 10 shows such a plot. Notice that the plot is rotated with respect to Figure 9, and that the grid on the shadow portion of the plot hides much of the detail there. The grid on the shadow is the result of a bug in the shadow-graphics package. The black and white display of these data is much less effective than a color display.

An interesting alternative to the separate trace displays in Figures 6 and 7 is the vector plot in Figure 11. This display allows the amplitude and a combination of the original and Hilbert-transformed traces to be displayed on one plot. These data have been subsampled in both time and space to allow a less cluttered plot.

Another similar set of displays involves showing the two-D Fourier transforms of seismic data. The Fourier transform of the data in Figure 6 is shown in Figures 12, 13, and 14, displaying the real part, the imaginary part, and the absolute value of the transformed data. Figure 15 shows a three-D display of the data in Figure 14.

A final example shows the singular value decomposition of a seismic file. In this case a simple line plot has been mixed with the density plots of the original file and the U and V matrices.

CONCLUSIONS

Mathematica will not replace software written in the FORTRAN or C languages, but it does have a place in algorithm development. Mathematica is more flexible than traditional languages and has many mathematical algorithms supplied in forms that are less complex and more flexible than the mathematical libraries usually available in FORTRAN or C. Mathematica's graphics are well developed and easy to use.

This flexibility and simplicity comes with a price. Mathematica runs a given job slower than a comparable job using FORTRAN or C. For small prototypes, this lack of speed may be acceptable, but for many seismic datasets, the run time soon becomes unacceptable. Another serious limitation is Mathematica's memory size limitations. With Mathematica, all the data must reside in memory, where FORTRAN or C allows moving data in and out of auxiliary storage. Where a FORTRAN array requires only the space needed for the floating point numbers, Mathematica stores the array in a more generalized format that takes several times more space. Finally, FORTRAN and C are generally familiar to software developers; Mathematica is a new language to be learned.

The issue of computer algebra have not been addressed in this paper but is an important enough topic to generate interest for anyone using more than the simplest mathematical manipulations. Useful work may be done with only a quick introduction to the system, while most of the possible complexity of the system may be ignored. I recommend reading the Part 1 of Wolfram (1991) and ignoring the largest part of this book. Much of the material in the book is only needed for reference and may be

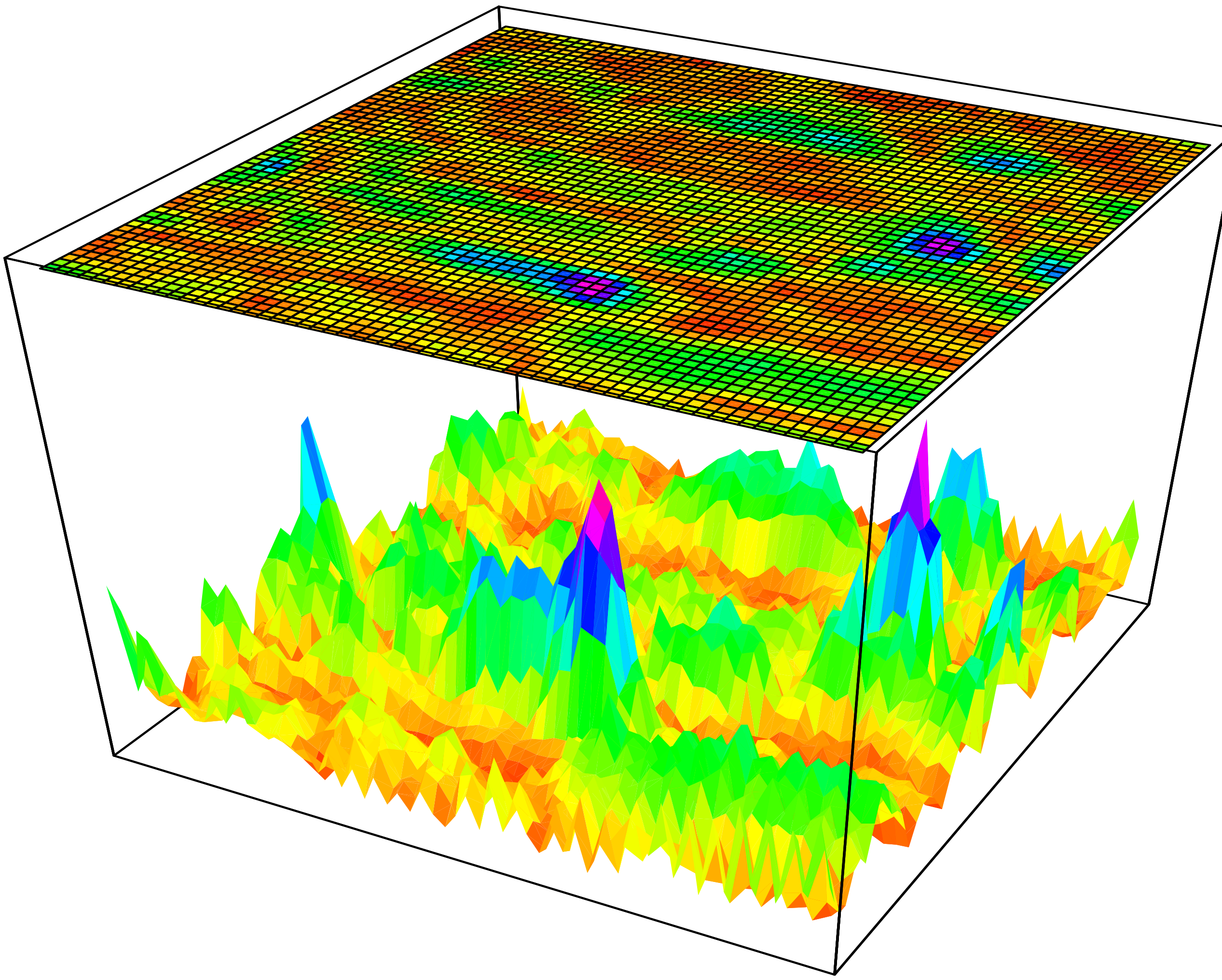

Figure 10: A three-D plot with a shadow display. This looks better in color. ray2-shadow [NR]

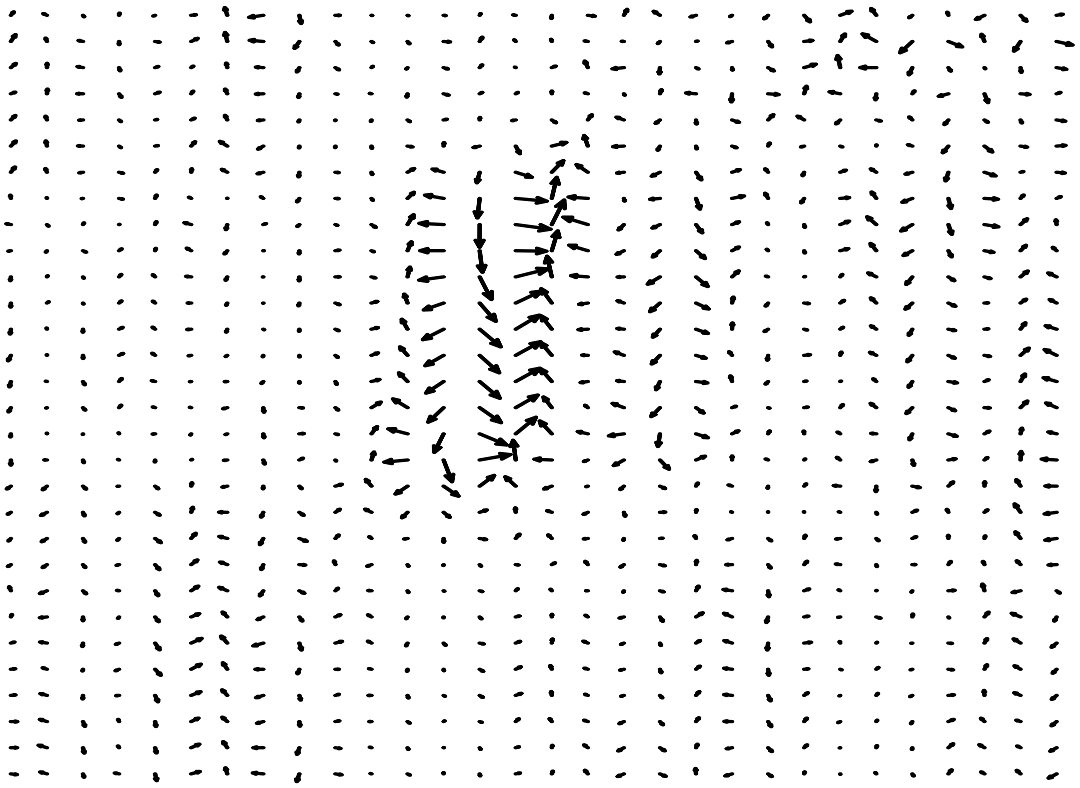

Figure 11: A three-D plot. $\boxed{\text{ray2-vector}}$ [NR]

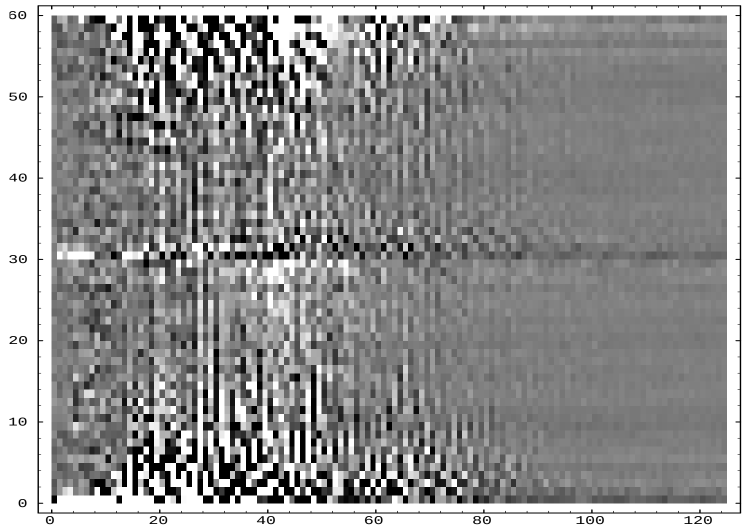

Figure 12: The real part of the 2-D Fourier-transformed data. ray2-rs [NR]

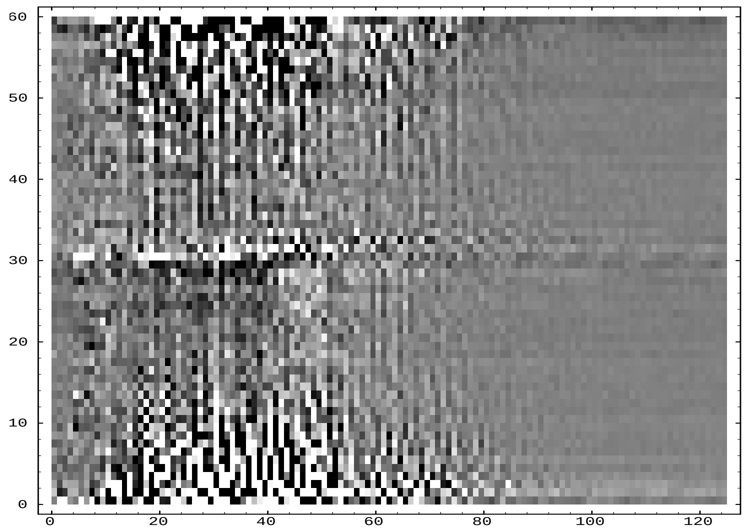

Figure 13: The imaginary part of the 2-D Fourier-transformed data. ray2-imag [NR]

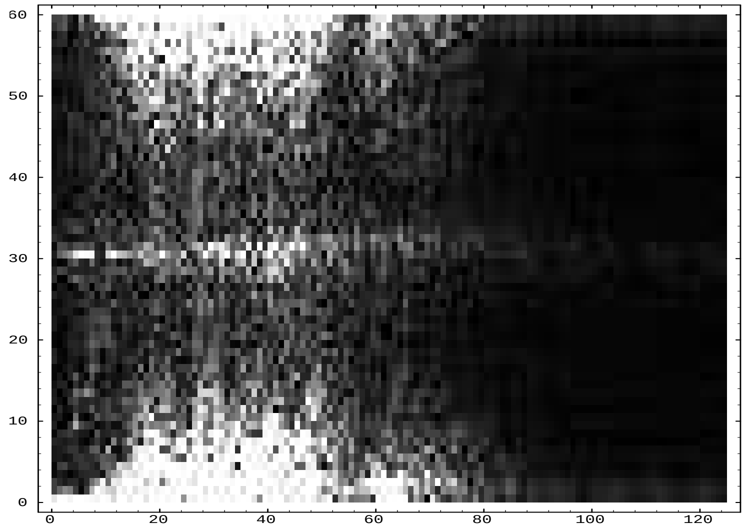

Figure 14: The absolute values of the 2-D Fourier-transformed data. ray2-abs [NR]

used only when the particular topic is required.

REFERENCES

Aki, K. and Richards, P., 1980, Quantitative Seismology: W. H. Freeman and Company.

Wolfram, S., 1991, Mathematica: A System for Doing Mathematics by Computer: Addison Wesley.

APPENDIX

Several Mathematica packages and run streams are listed here as samples for seismic processing and display.

The following routine generates the Hilbert transform of a trace.

```
BeginPackage["hilbert‘"]

Hilbert::usage =
             "Hilbert- create the Hilbert transform of a trace."
Begin["‘Private‘"]
```

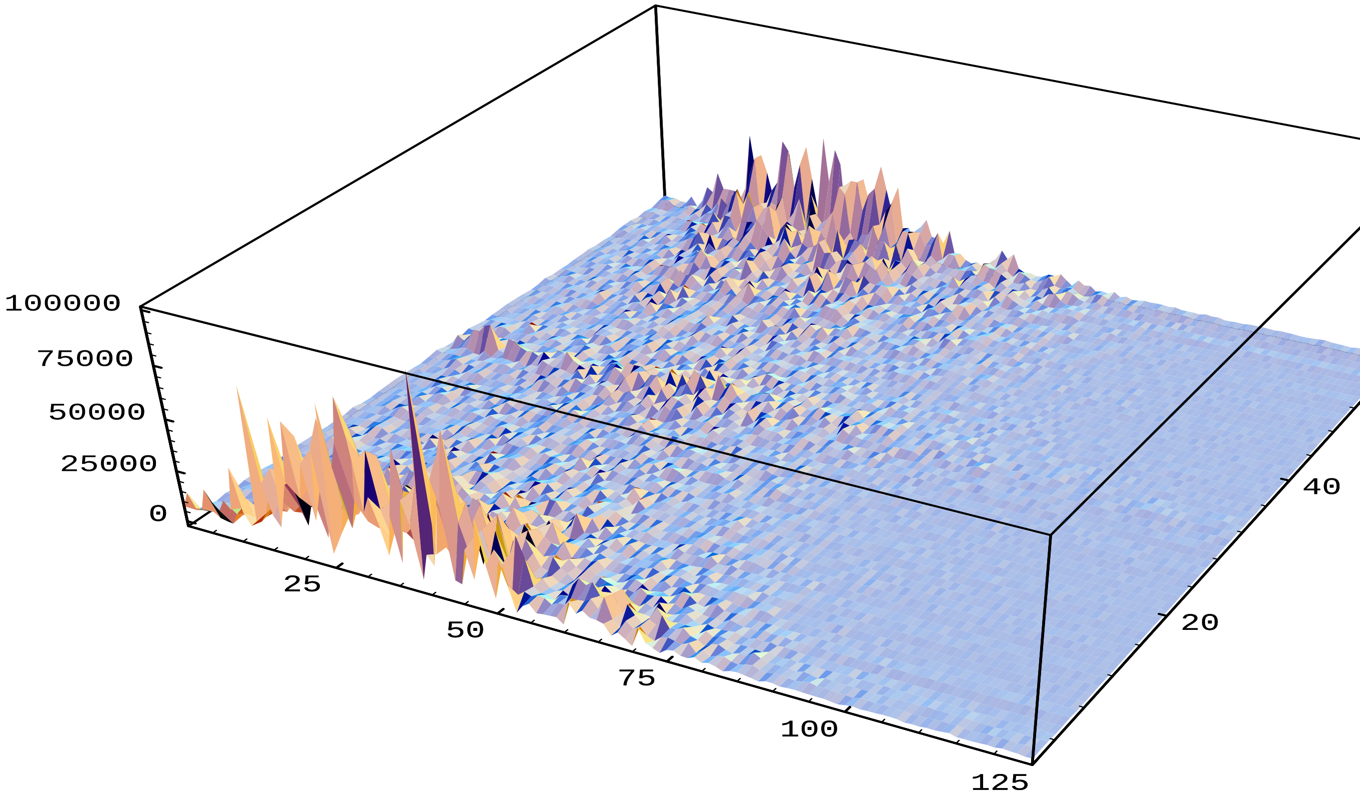

Figure 15: A three-D plot of the data shown in Figure 14. ray2-FT3D [NR]

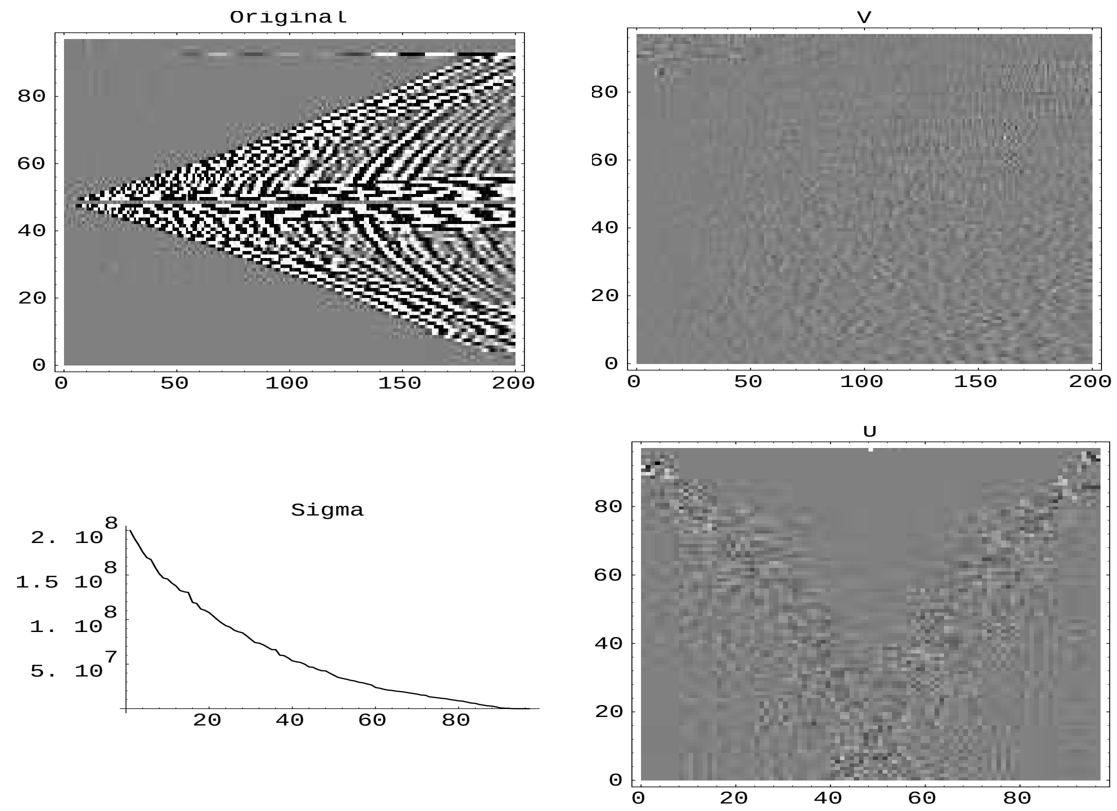

Figure 16: A seismic shot gather and its singular value decomposition. ray2-shotsvd [NR]

```
Hilbert::Badarg = " Bad input to Hilbert"

Hilbert[x_] := (
 Block[
        {i,half,junk123},
        half= Floor[ N[ Dimensions[x][[1]]/2 +1 ] ];
        junk123 =RotateRight[ Fourier[N[x]],half-1 ];
        Re[InverseFourier[
                         RotateLeft[
                          Table[
                                +Sign[ i-half ] I junk123[[i]],
                                {i,1,Dimensions[x][[1]]}]
                            ,half-1]
                         ]
          ]
      ]
                )
End[]
EndPackage[]
```

The run stream used to generate the Hilbert transforms in the paper follows. Note that the YPrint.m package produces Postscript files on disk for use in this paper. The GraphicsArray function is used to get around what appears to be a bug in the Sun system's display of these Postscript files. The DEC and IBM do not appear to be troubled by the output not processed with GraphicsArray.

```
ia = <<WSstack.m;
<<YPrint.m
<<~/amath/Hilbert.m
<<~/amath/Wiggle.m
<<Graphics`Graphics3D`
<<Graphics`PlotField`
a = ia;
ah = Map[Hilbert,a];
env = N[Sqrt[ah^2+a^2]];
p1 = GraphicsArray[{ListDensityPlot[a,Mesh->False]}]
YPrint[p1]
!mv out.ps origden.ps
p2 = GraphicsArray[{ListDensityPlot[ah,Mesh->False]}]
YPrint[p2]
!mv out.ps hilbertden.ps
p3 = GraphicsArray[{ListDensityPlot[env,Mesh->False]}]
YPrint[p3]
!mv out.ps envden.ps
```

The wiggle trace plot and the vector plot were generated with the following run.

```
<<YPrint.m
<<~/amath/Hilbert.m
<<~/amath/Wiggle.m
<<Graphics`PlotField`
```

```
ia = <<WSstack.m;
a = Table[ia[[i,j]],{i,1,60,2},{j,31,90,2}];
ah = Map[Hilbert,a];
env = N[Sqrt[ah^2+a^2]];

p5 = Wiggle[ia,2 MRMS[ia]]
YPrint[p5]
!mv out.ps wiggle.ps

t = Table[{a[[i,j]],ah[[i,j]]},{i,30},{j,30}];
p4 = ListPlotVectorField[t,ScaleFactor->1.0]
YPrint[p4]
!mv out.ps vector.ps
```

The Fourier transform of the data in Figure 6 was generated with the following run stream.

```
<<YPrint.m

a = <<WSstack.m;
bb = Fourier[a];
b=N[ Table[ bb[[i,j]],{i,1,Dimensions[bb][[1]]},{j,1,Dimensions[bb][[2]]/2} ] ];
p1 = GraphicsArray[{ListDensityPlot[Re[N[b]],Mesh->False]}]
YPrint[p1]
!mv out.ps rs.ps
p2 = GraphicsArray[{ListDensityPlot[Im[N[b]],Mesh->False]}]
YPrint[p2]
!mv out.ps imag.ps
p3 = GraphicsArray[{ListDensityPlot[Abs[N[b]],Mesh->False]}]
YPrint[p3]
!mv out.ps abs.ps
p4 = GraphicsArray[{ListPlot3D[Abs[N[b]],Mesh->False,PlotRange->All]}]
YPrint[p4]
!mv out.ps FT3D.ps
```

A small package that allows a plot to be output to disk in PostScript format is the YPrint.m routine listed below. It outputs the plot as a file out.ps that can be renamed from within Mathematica as seen in the previous example.

```
System`YPrint::usage =
             "YPrint[-graphics-] sends graphics to out.ps file."
Begin["System`Private`"]
YPrint[x_] := ( Display["!psfix > out.ps", x]; x )
End[]
```

Finally a FORTRAN program to convert SEP files to Mathematica format is given here. Perhaps a better method of converting the numbers would be to built the output in terms of powers of ten, to avoid the many format statements.

```
# Convert SEP format to Mathematica format
#
# Main.x head=/dev/null < input.H | cat outfile
#
#%
integer nt, nx
from history:    integer n1 : nt
from history:    integer n2 : nx
allocate:       integer gather(nt,nx)

subroutine doit( nt,nx, gather )
integer it,in,nt, nx
real gather(nt,nx),rnum

#        ----   call hclose()
100     format(a8)

        call sreed( 'in', gather, nt*nx*4)
        write(*,200) '{'
200     format(a)
        do in=1,nx
        {
          write(*,200) '{'
          do it=1,nt
          {
#          write(*,*) gather(it,in)
#          print *, gather(it,in)
           rnum = gather(it,in)
#           write(*,300) gather(it,in)
           if (abs(rnum) .lt. 1)
            write(*,298) rnum
           else if (abs(rnum) .lt. 1000)
           write(*,299) rnum
           else if (abs(rnum) .lt. 1000000)
            write(*,300)  rnum
           else if (abs(rnum) .lt. 1.E9)
                write(*,301)  rnum
           else if (abs(rnum) .lt. 1.E12 )
                write(*,302)  rnum
           else if (abs(rnum) .lt. 1.E16 )
                write(*,303)  rnum
           else if (abs(rnum) .lt. 1.E20 )
                write(*,304)  rnum
           else if (abs(rnum) .lt. 1.E25 )
                write(*,305)  rnum
           else if (abs(rnum) .lt. 1.E30 )
                write(*,306)  rnum
                else {
                       write(*,306)  sign(1.E30,rnum)
                     }
298        format(f14.10)
299        format(f14.7)
300        format(f14.4)
301        format(f17.4)
```

```
302        format(f20.4)
303        format(f24.0)
304        format(f28.0)
305        format(f32.0)
306        format(f34.0)

           if (it .ne. nt) write(*,200) ','
          }
          write(*,200) '}'
           if (in .ne. nx) write(*,200) ','
        }
        write(*,200) '}'

#       call srite( 'out', gather, nt*nx*4)
return; end
```