National Radio Astronomy Observatory
Tucson, Arizona
July 18, 1995

MEMORANDUM

To: 12–m Memo Series
From: J. Wren
Subject: Fourier Analysis of 12 m Structural Tilt Measurements

1 Introduction

We are currently looking for new methods to analyze the data from tilt meters on the 12–m structure. The overall goal of this analysis is to improve the pointing model. To accomplish this goal, the tilt data must be visualized in new ways.

One method of analyzing the tilt meter data is fourier analysis. Fourier analysis allows tilt meter data to be broken up into it's component sine terms. These terms reflect the periodic shifting of the telescope mount due to changes in flexure and alignment. Once these shifts are defined, the pointing model can then be adjusted to compensate for the major periodic terms. To this end, I have written a routine in *PV Wave* that displays the fourier components of the tilt meter data.

2 PV Wave

PV Wave is a mathematics package for data reduction and analysis. It is geared especially for the display of data using 2D and 3D graphics and image processing. One advantage of *PV Wave* is that it works well communicating with programs written in FORTRAN and C. The user can transfer data between *PV Wave* and external programs and between *PV Wave* and the operating system. This is advantageous to us because we may eventually want to link *PV Wave* to software already written for pointing analysis, such as gpoint written by Phil Jewell.

PV Wave uses the Cooley-Tukey Fast Fourier Transform algorithm. The input is an array of data points of length N defining the function to be transformed. In our case this would be the tilt measurements. The output is an complex array of the same dimension containing the fourier components of the function. The first term in the fft array is the dc term. The second contains the amplitude and phase of the cosine term with a wavelength equal to the array size. The third contains the amplitude and phase of a cosine term with a wavelength equal to one half the array size, etc.

The procedure I have written is called `jimpoint.pro`. It looks at a tilt data file and:

1. Displays a magnitude vs. frequency plot.

2. Displays a phase vs. frequency plot.

3. Displays the sum of the first ten cosine components, and the first three cosine components plotted over the tilt data.

4. Displays the residuals of the difference of the tilt data and the fourier sum and the rms of the difference.

5. Writes the first $N/2$ cosine components (amplitude, frequency, and phase) to a file in the current directory. The second half of the fft array contains the same information as in the first, and is therefore discarded.

Users may also request `jimpoint.pro` to print out a ps file of the displayed data to their home directory. The usage is as follows.

```
WAVE> jimpoint,'path',[comp]
```

Here, *path* is the path from the current directory to the file containing the tilt data. The tilt data files should be at least columns long. The first column should be the azimuth, the second should be elevation, the third should be the tilt data for axis 1, and the fourth should be the data for axis 2. Users may also include 'comp' as an optional parameter. When included, 'comp' will be returned as an array containing the first $N/2$ cosine components.

There is some concearn that a discrete fourier transform (dft) might provide better information than a fft due to the fact that the fft makes several assumptions about the input array to speed up calculation. I have noticed no particular drawbacks in using the package fft and the increase in speed when transforming large datasets is substantial. I have created two dft procedures called `jft.pro` for forward transforms and `jift.pro` for inverse transforms. The usage is as follows.

```
WAVE> jft,x,y,k,fk
or
WAVE> jift,k,fk,x,y
where:
x=independant variable
y=dependant variable
k=independant variable in fourier space
fk=the fourier transform of y
```

To use `jft.pro` or `jift.pro`, all the arguments must be the same dimension. In `jft.pro` only `fk` is changed. In `jift.pro` only `y` is changed. It would be a simple matter to change `jimpoint.pro` so that it uses a dft instead of a fft.

PV Wave provides for least squares fitting of arbitrary functions. Included with *PV Wave* is a procedure called `curvefit.pro` which does a least squares fit on a function defined by another procedure called `funct.pro`. Initially these procedures were set up to fit for a gaussian-polynomial combination. I have modified them to fit for a cosine curve and return it's parameters. The new procedures are called `jcosfit.pro` and `jcosfunct.pro`. The

usage is as follows:

```
WAVE> yfit=jcosfit(x,y,wt,parms,[sigma])
```

where:

x=the independant variable (a vector of N elements)

y=the dependant variable (to be fit by jcosfit)

wt= a N element vector of the weighting factors

Use $wt = 1$ for no weighting

Use $wt = 1/y_i$ for statistical weighting

Use $wt = 1/(stddevofy_i)$ for instrumental weighting

parms=the parameters of cosine function, $parms = (a_0, a_1, a_2)$

$y = a_0 cos(x - a_1) + a_3$ where x and a_1 are in degrees

3 Figure Captions

- Figure 1: The tilt meter data for El=15deg.
- Figures 2–3: The first three cosine components of tilt data and their sum.
- Figures 4–10: Using `jimpoint.pro` on several test cosine functions.
- Figures 11–13: Using `jimpoint.pro` on test cosine functions with random variations introduced.
- Figures 14–21: Using `jimpoint.pro` on tilt meter data.
- Figures 22–24: Comparing the results of `cosfit.pro` with the results of the fft on a test curve and on actual tilt data.

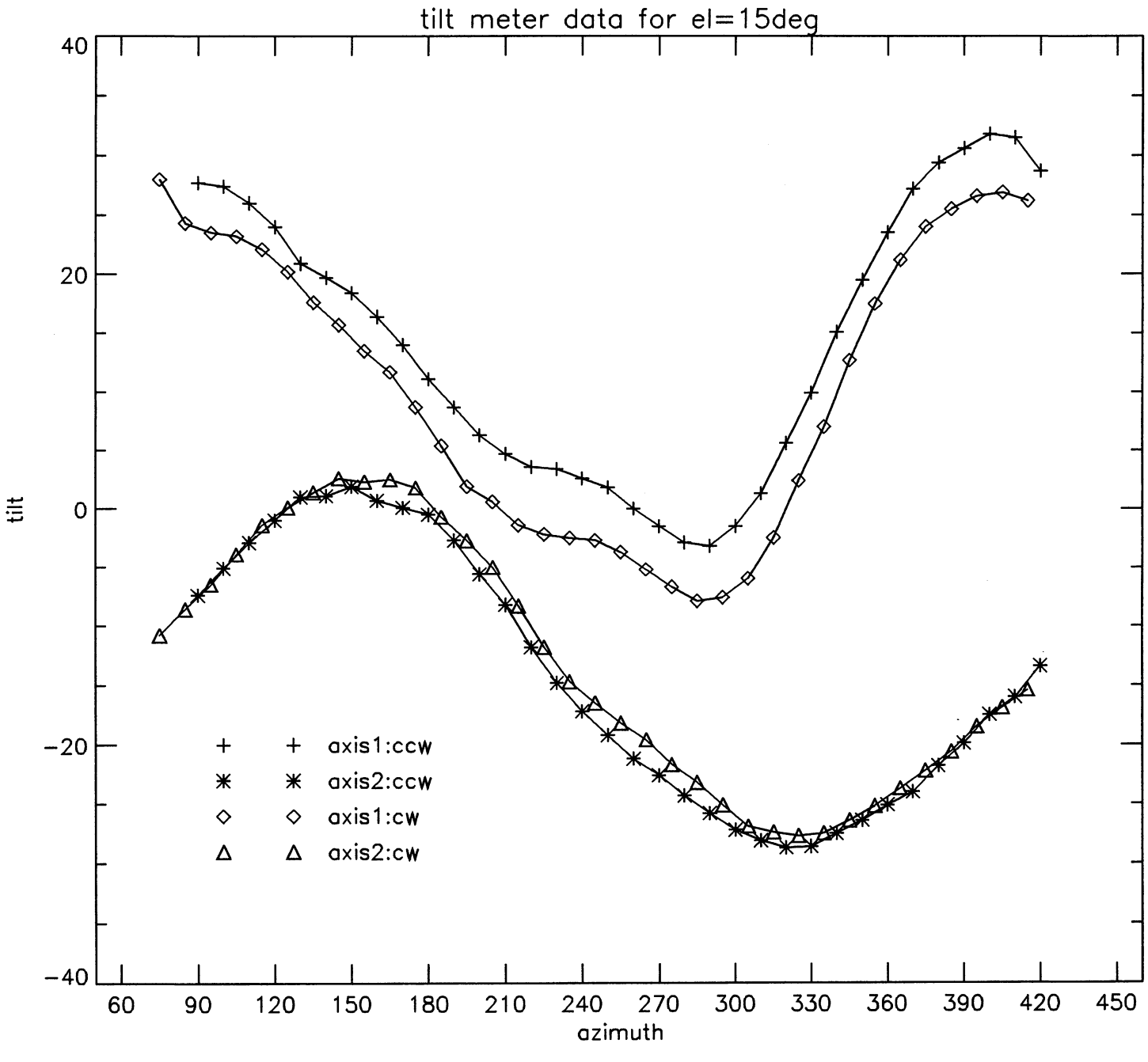

tilt meter data for el=15deg
40
20
0
−20
−40
tilt
60
90
120
150
180
210
240
270
300
330
360
390
420
450
azimuth
axis1:ccw
axis2:ccw
axis1:cw
axis2:cw

Figure 1.

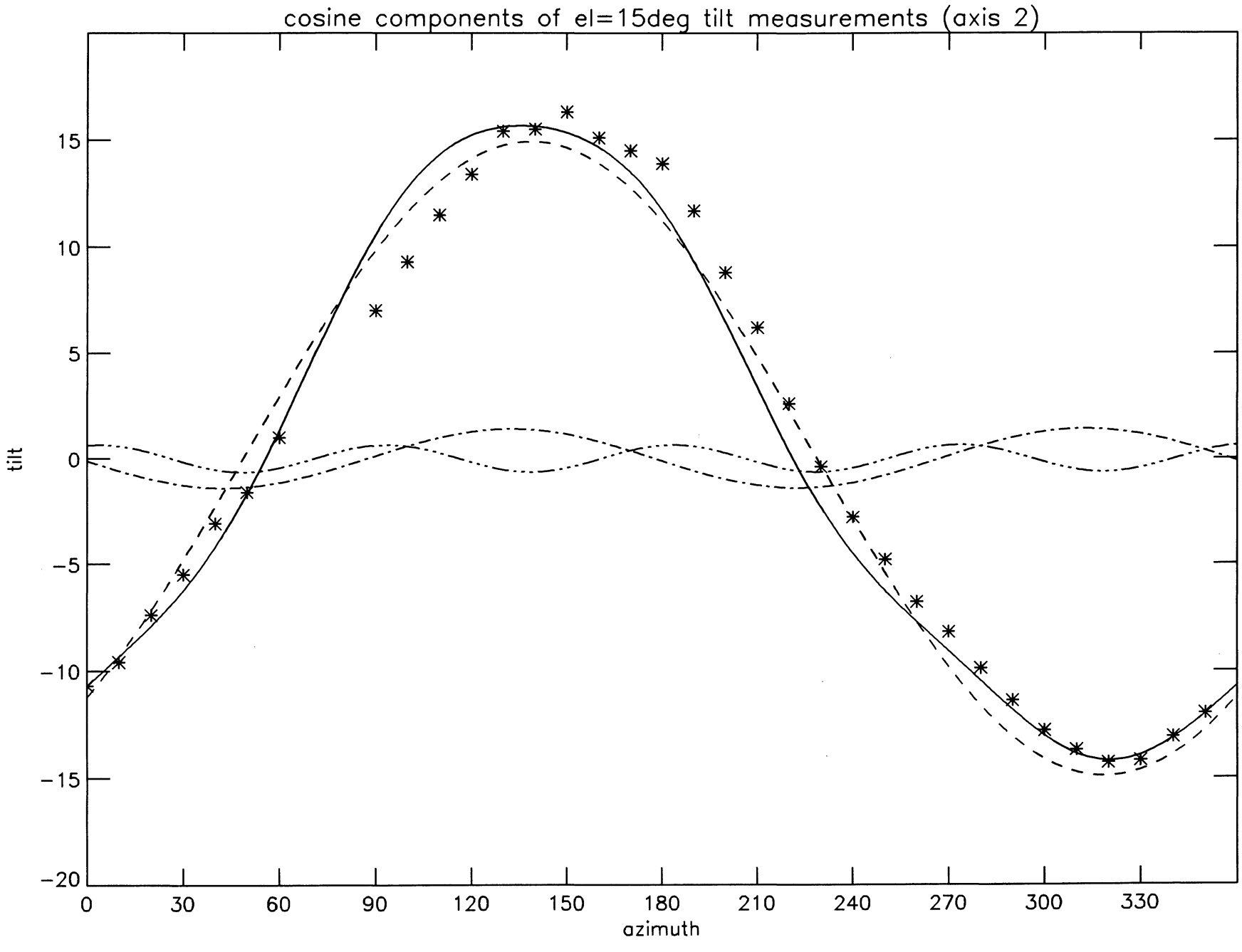


Figure 2.

tiltcosines2.ps

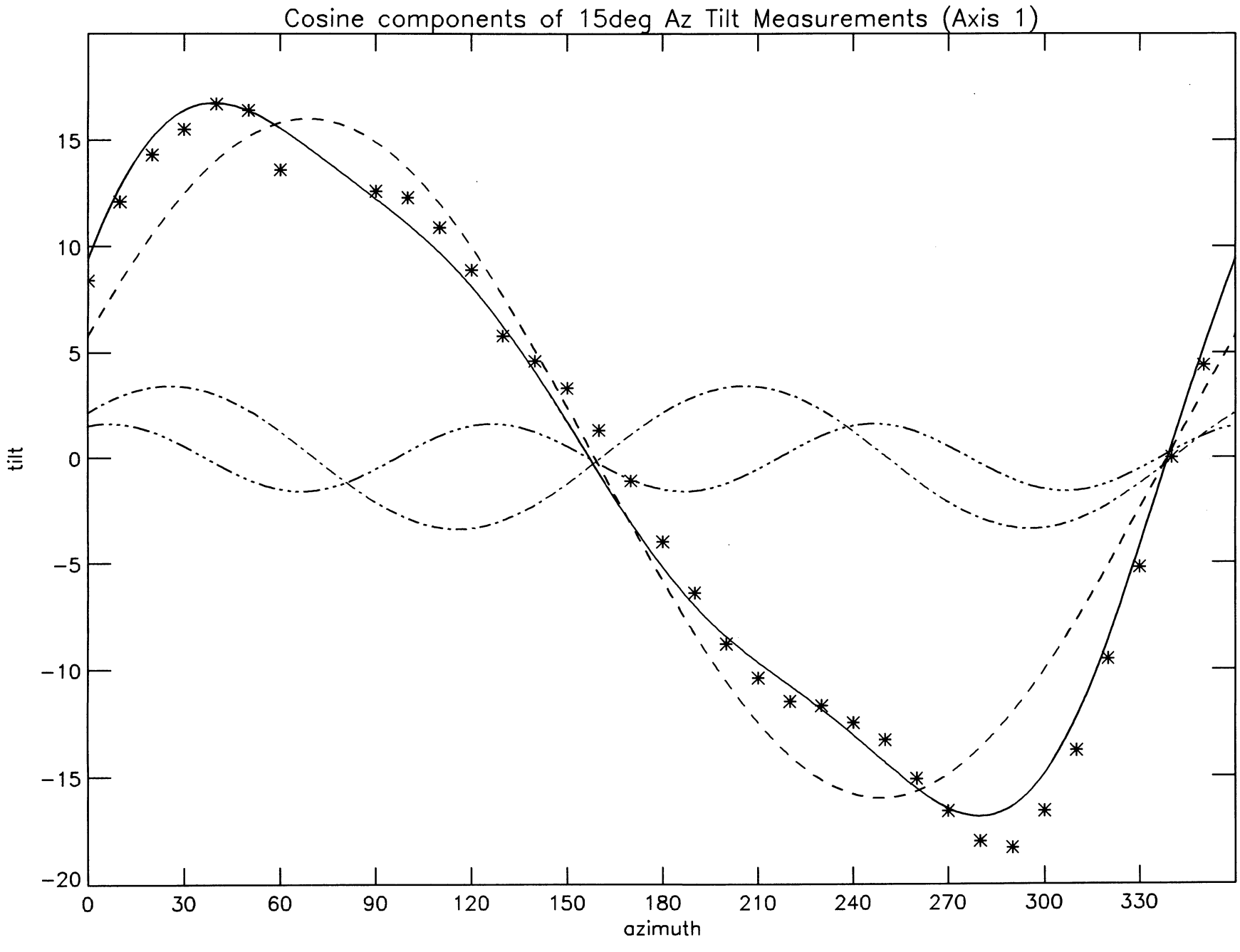
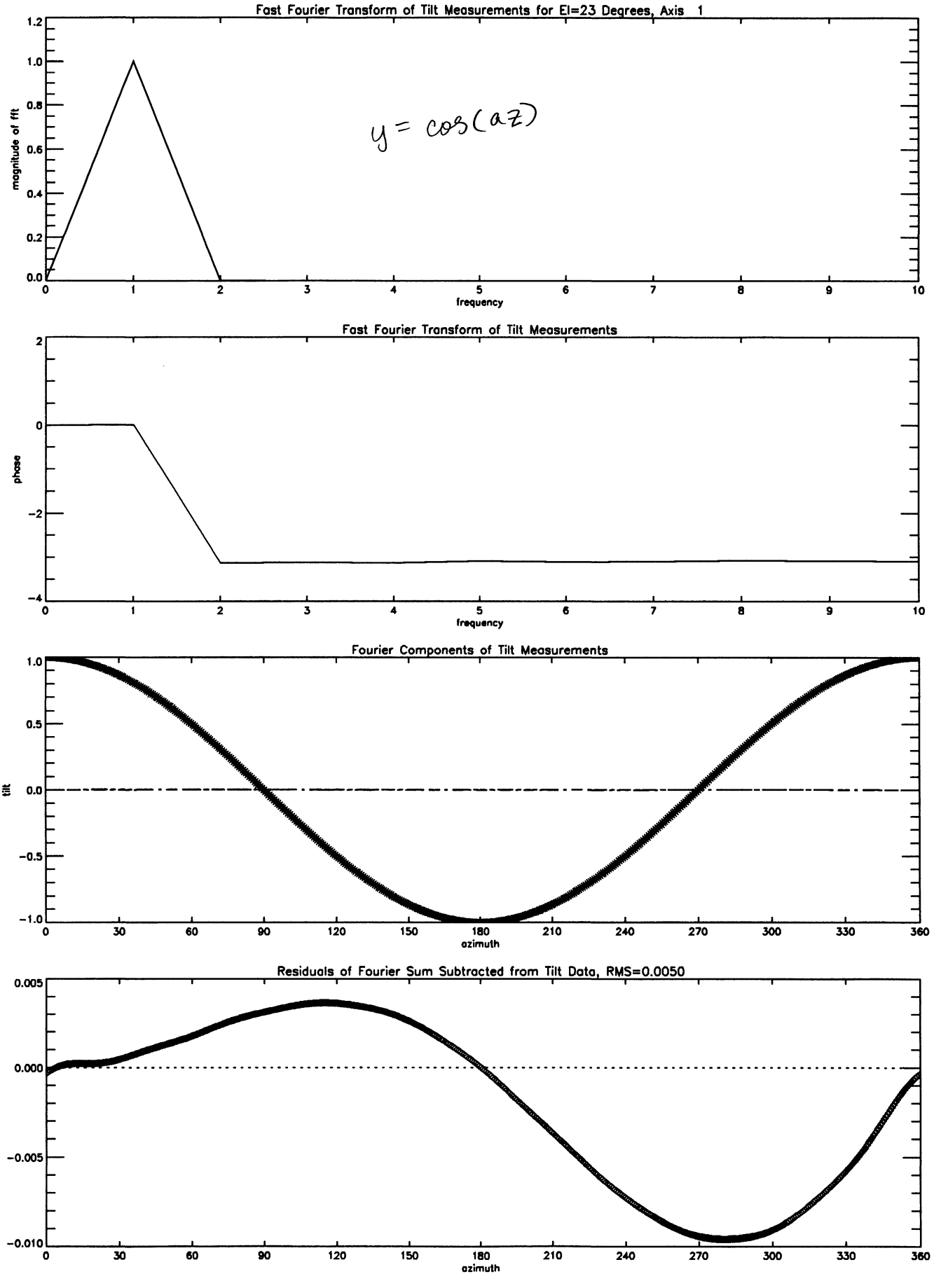


Figure 3.

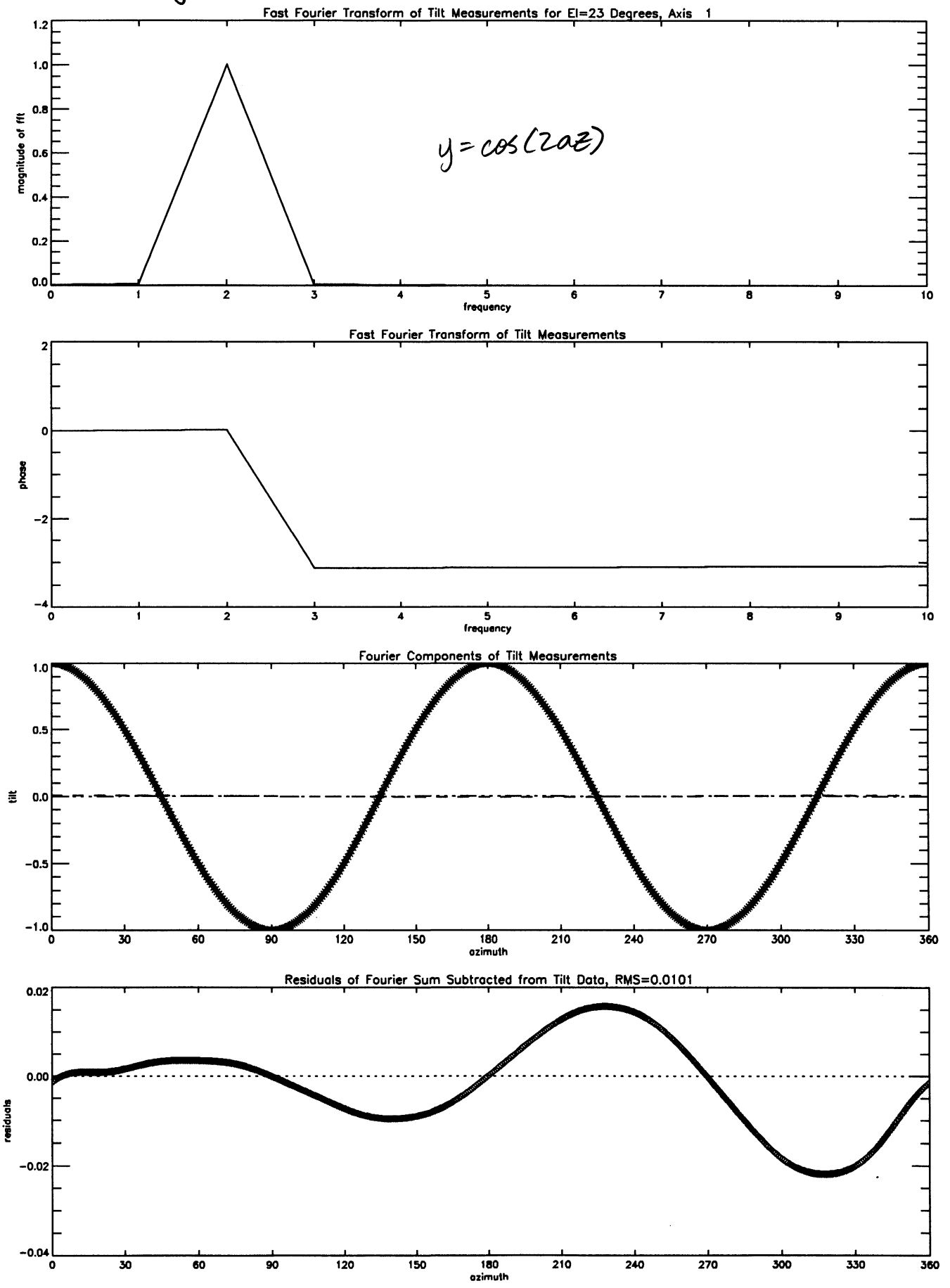
tiltcosines1.ps

Figure 4.



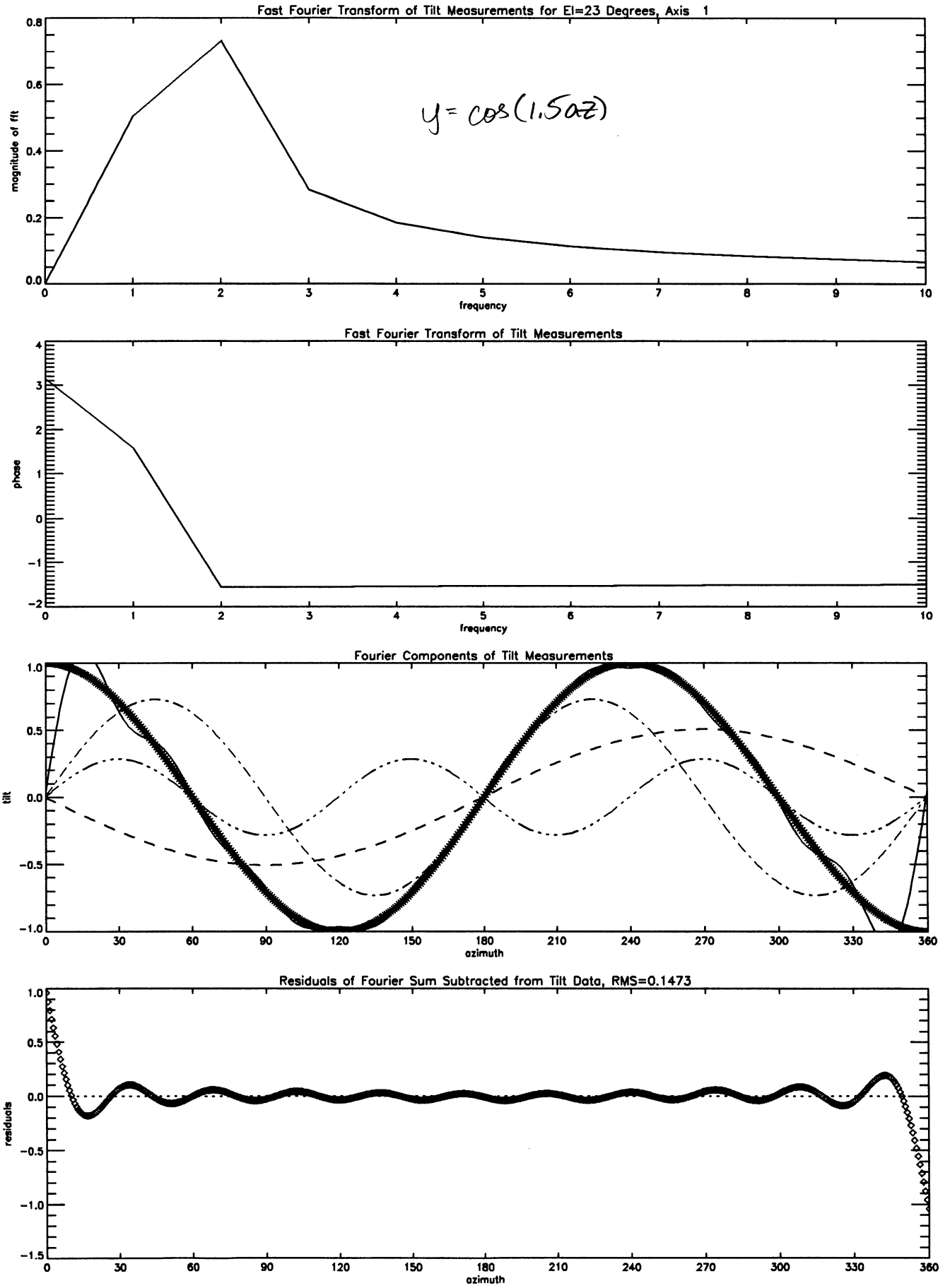
jimpoint-tl.ps

Figure 5.



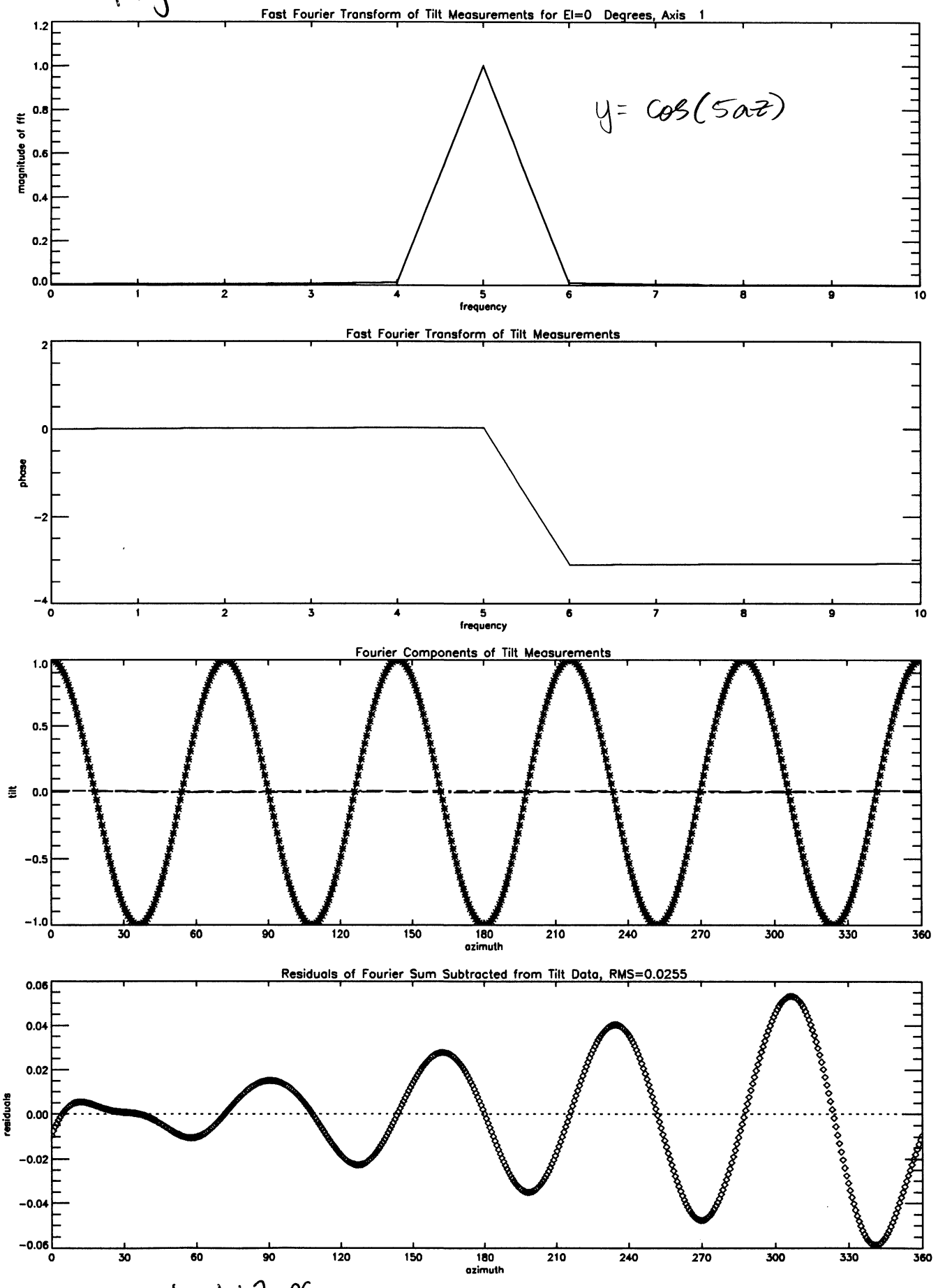
jimpoint-+1.5.ps

Figure 6.



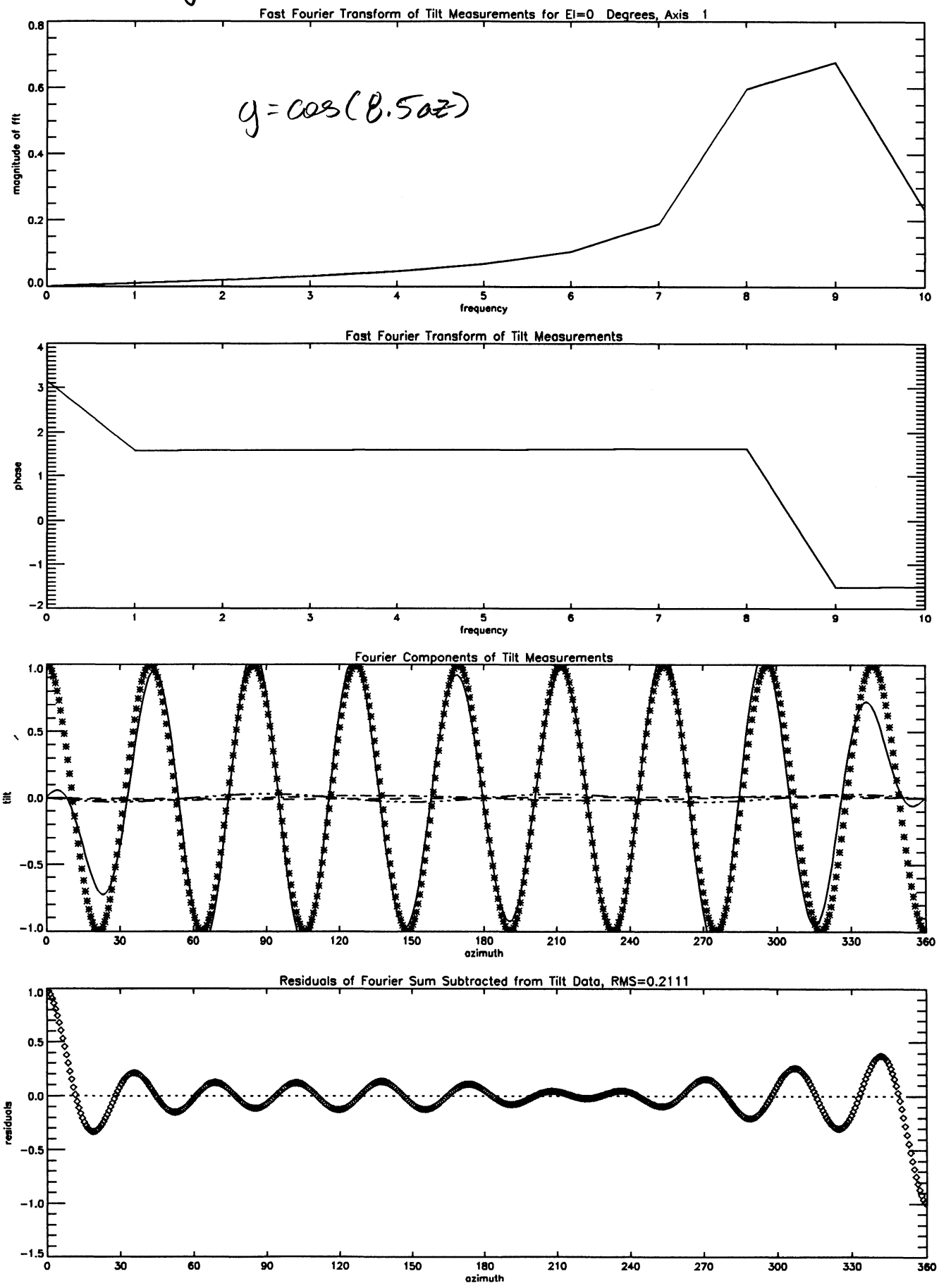
jimpoint-t1.6.ps

Figure 7.



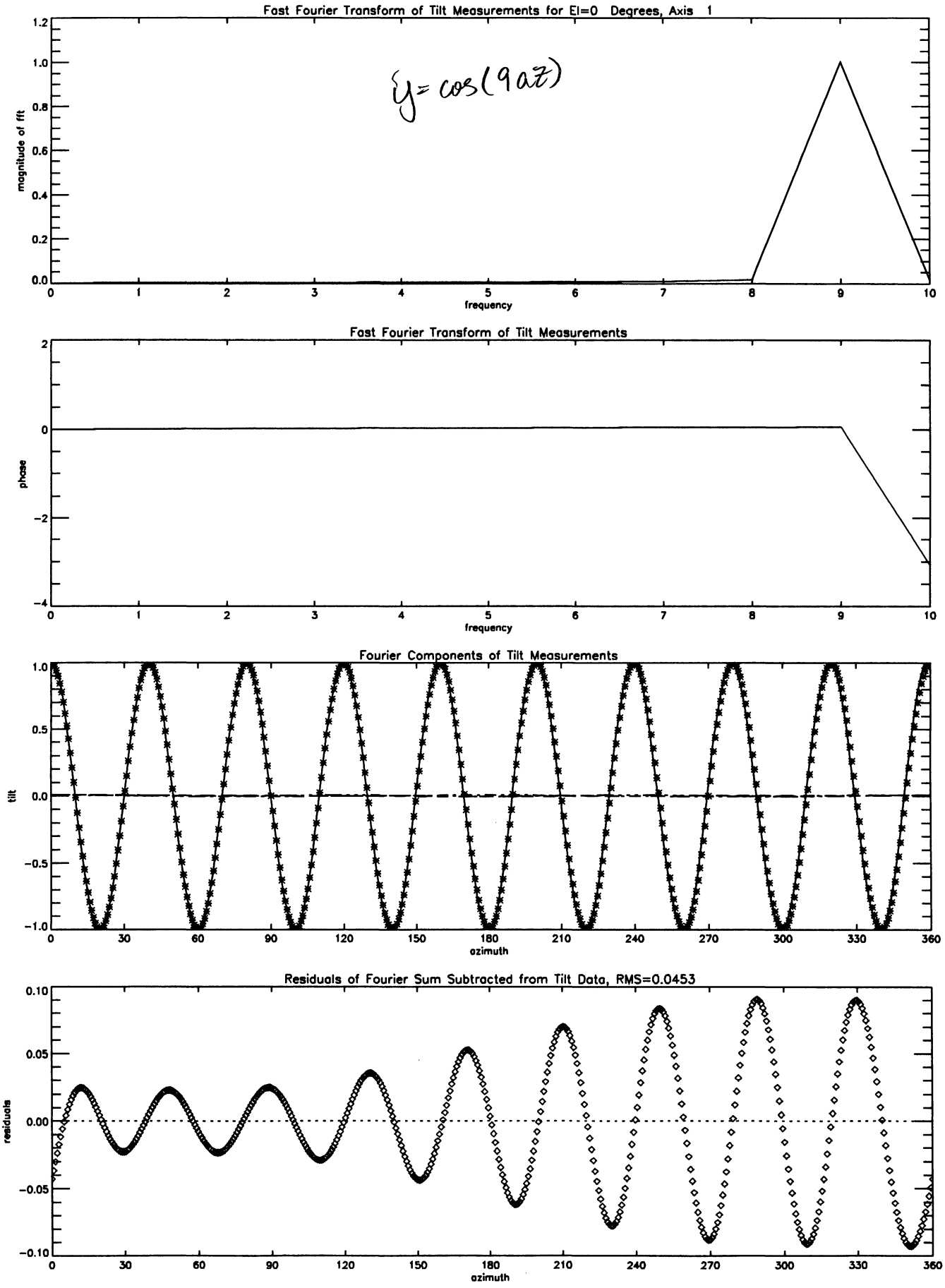
jimpoint-f1.7.ps

Figure 8.



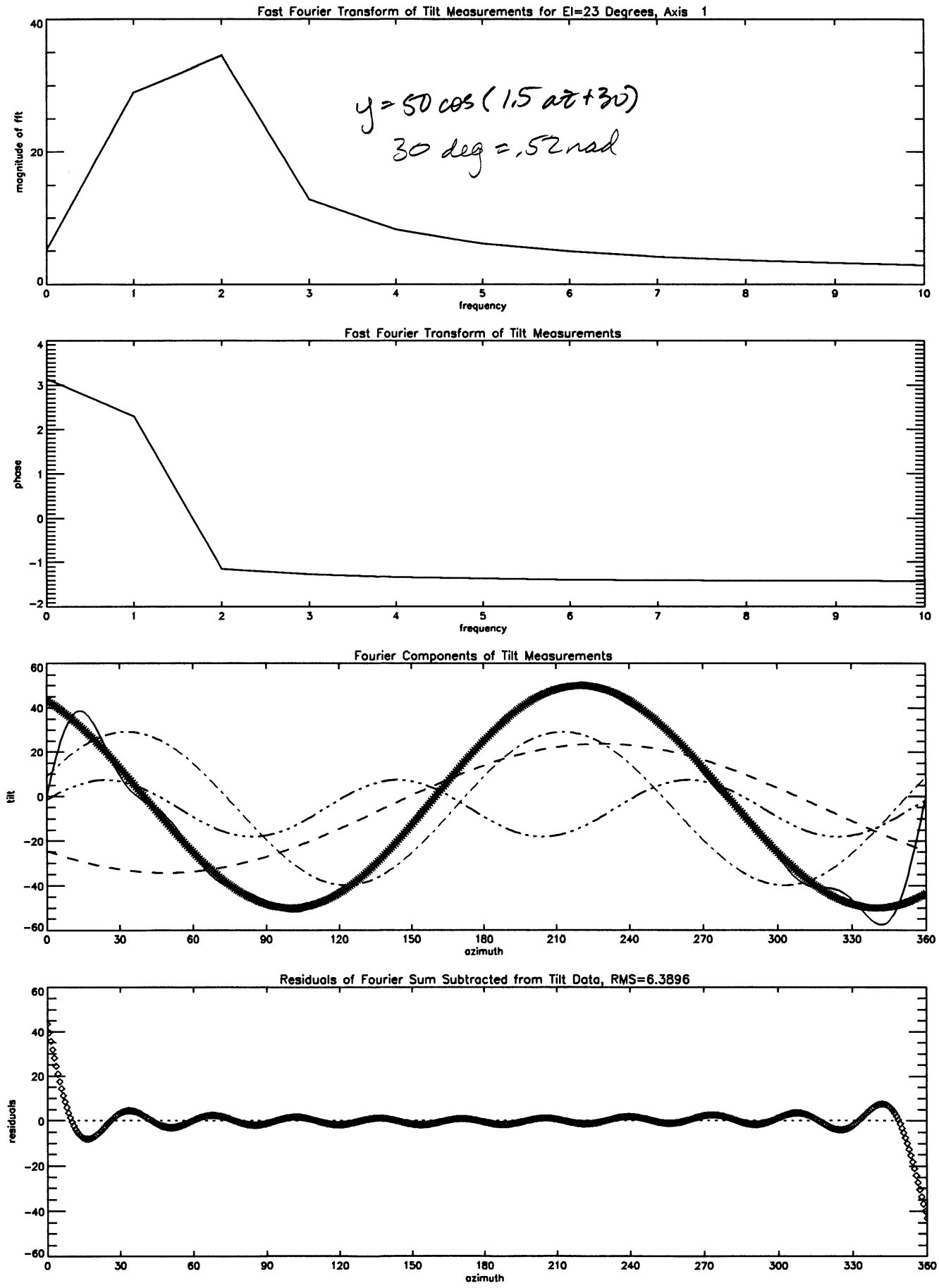
jimpoint-t1.9.ps

Figure 9.



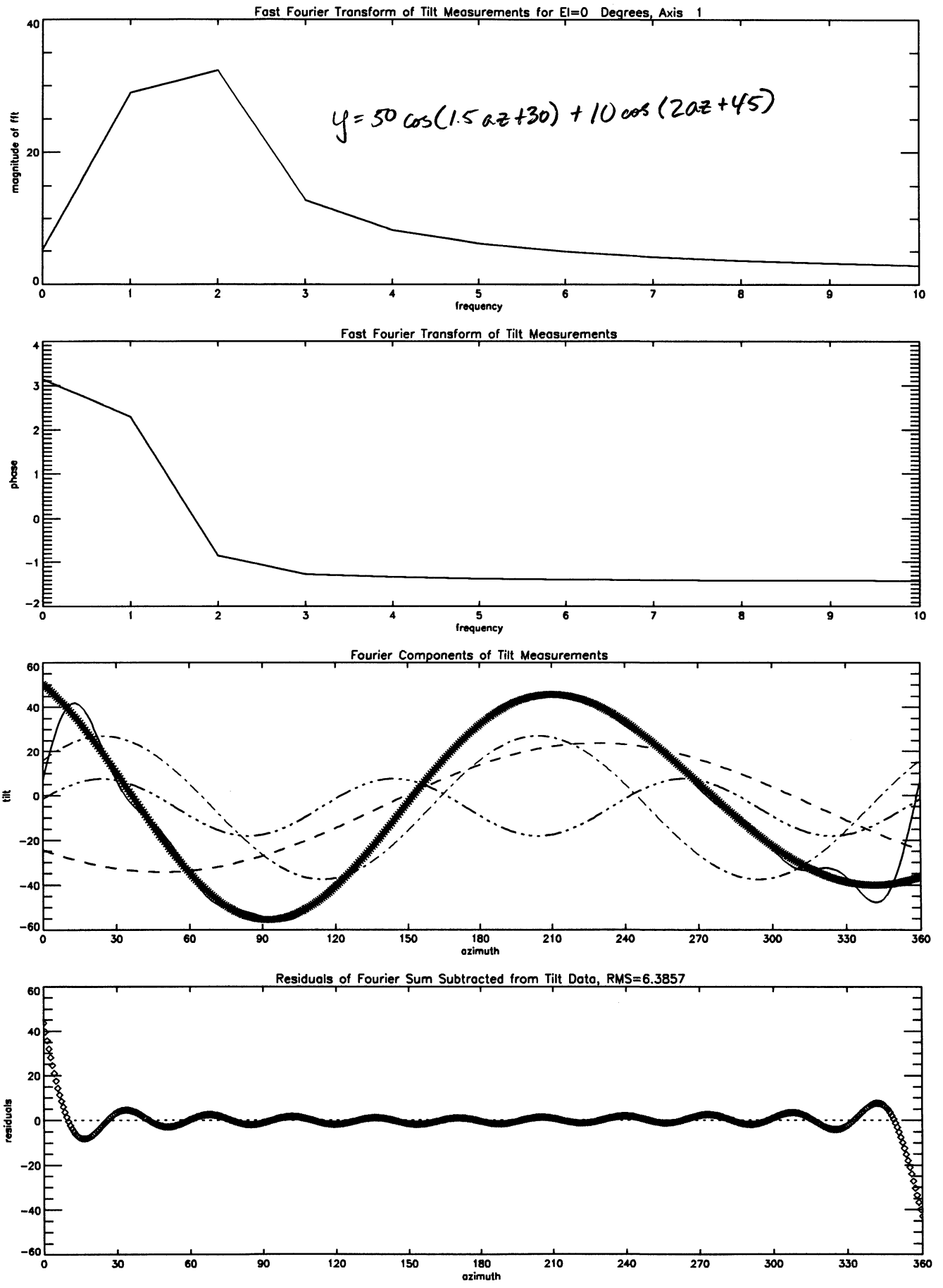
jimpoint -t1.8.ps

Figure 10.



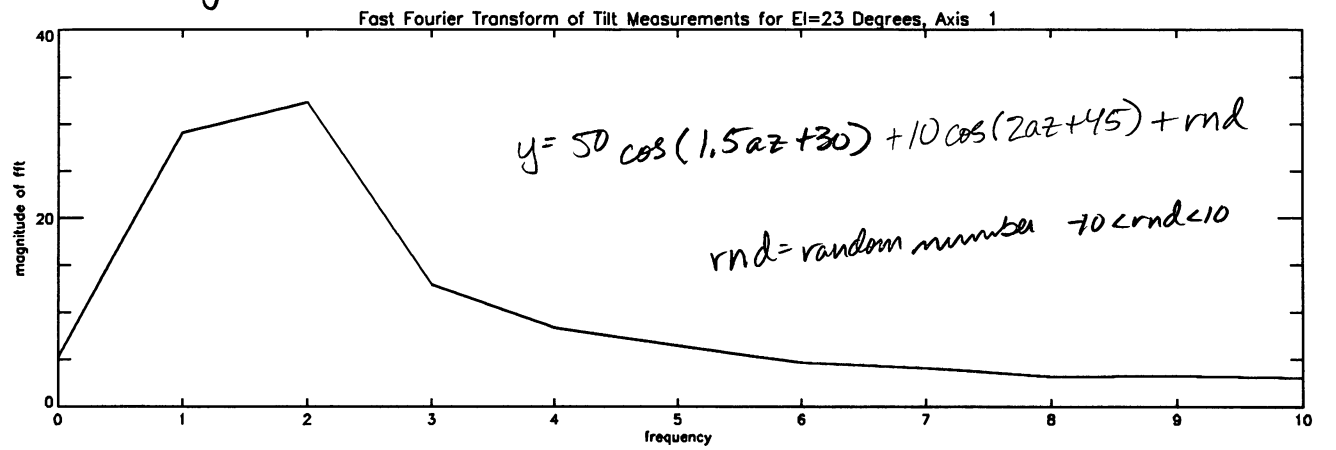
jimpoint-13.ps

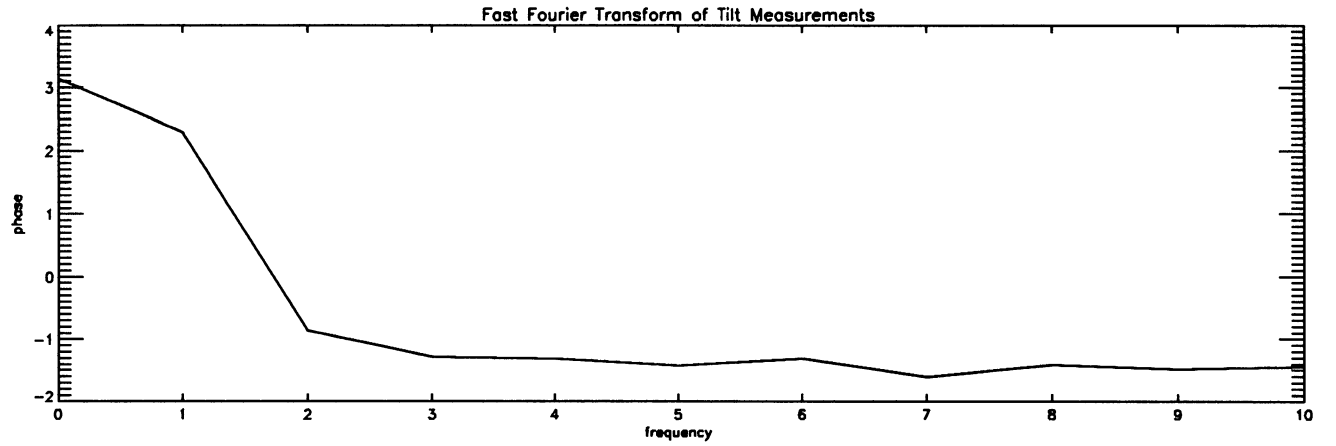
Figure 11.

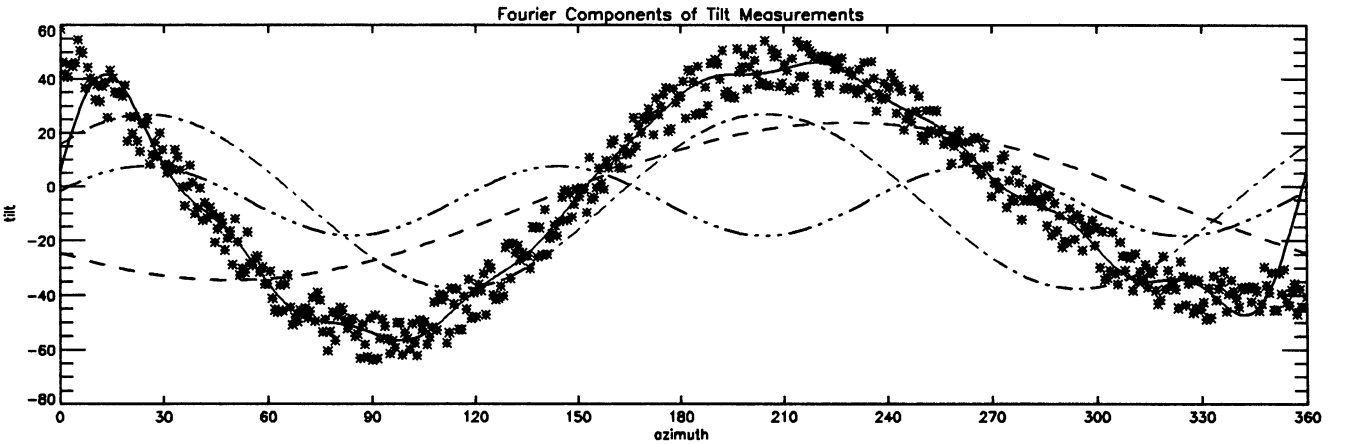


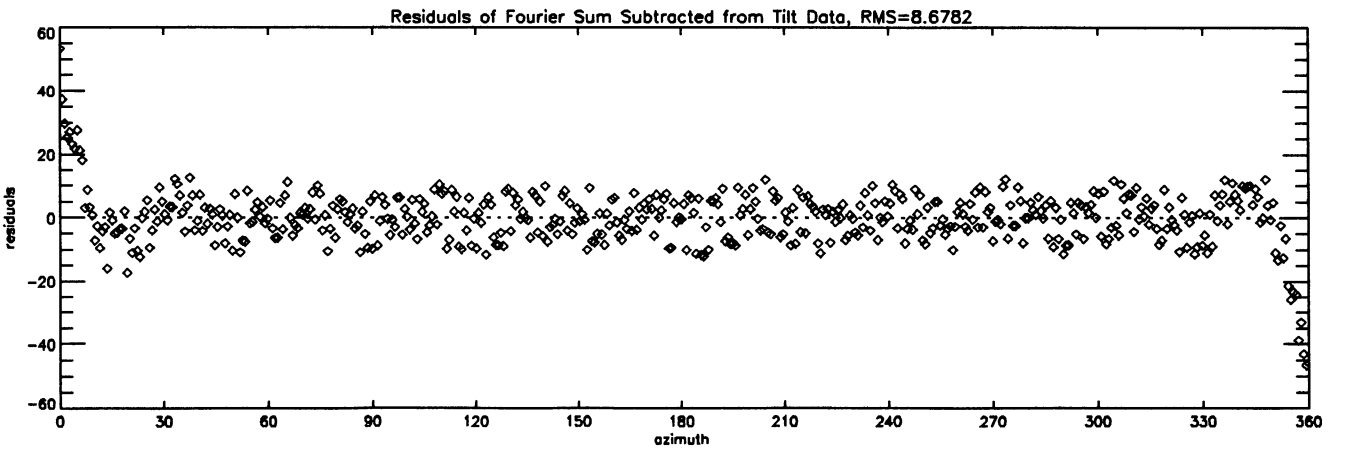
jimpoint -tz.ps

Figure 12.



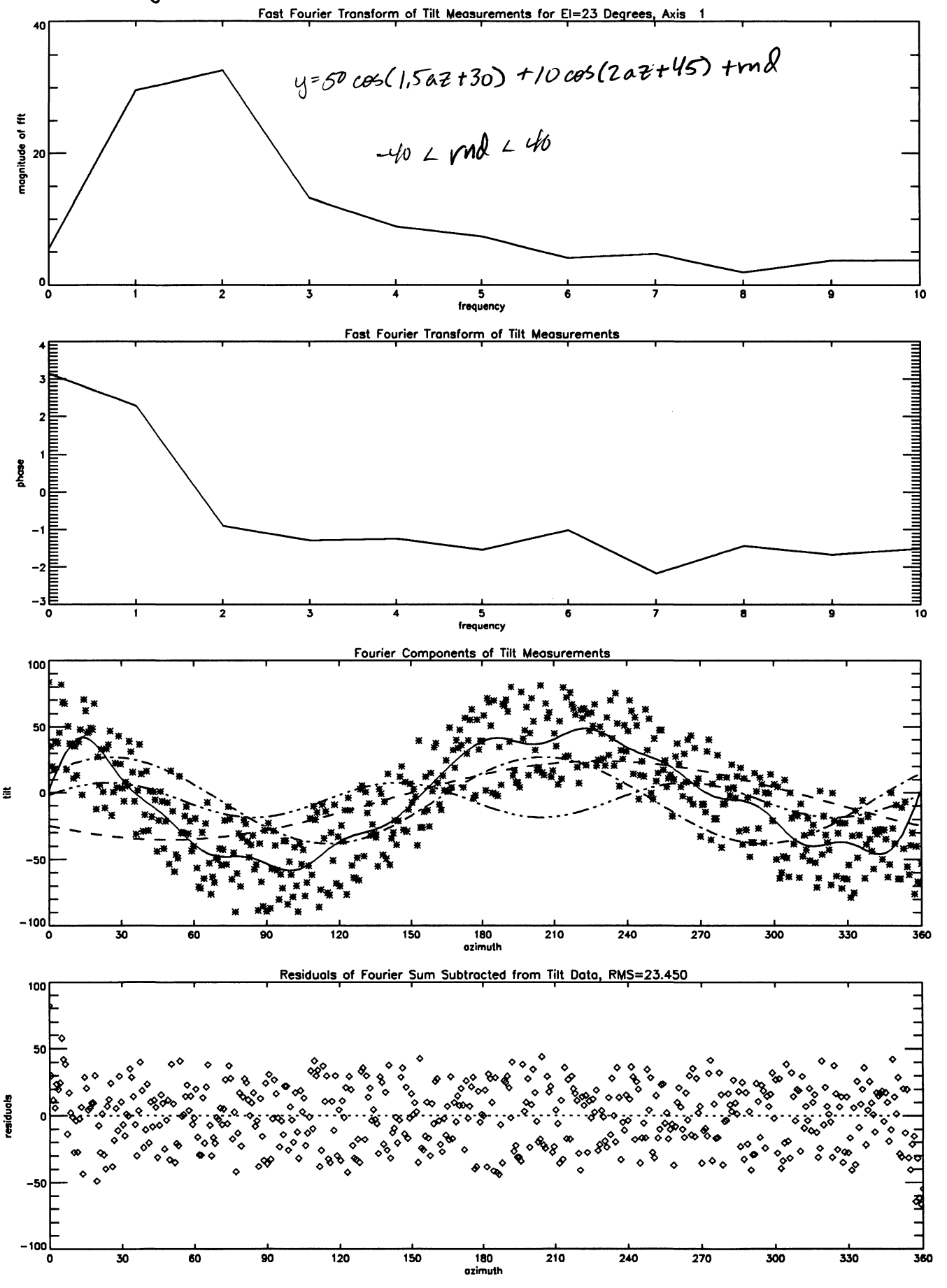






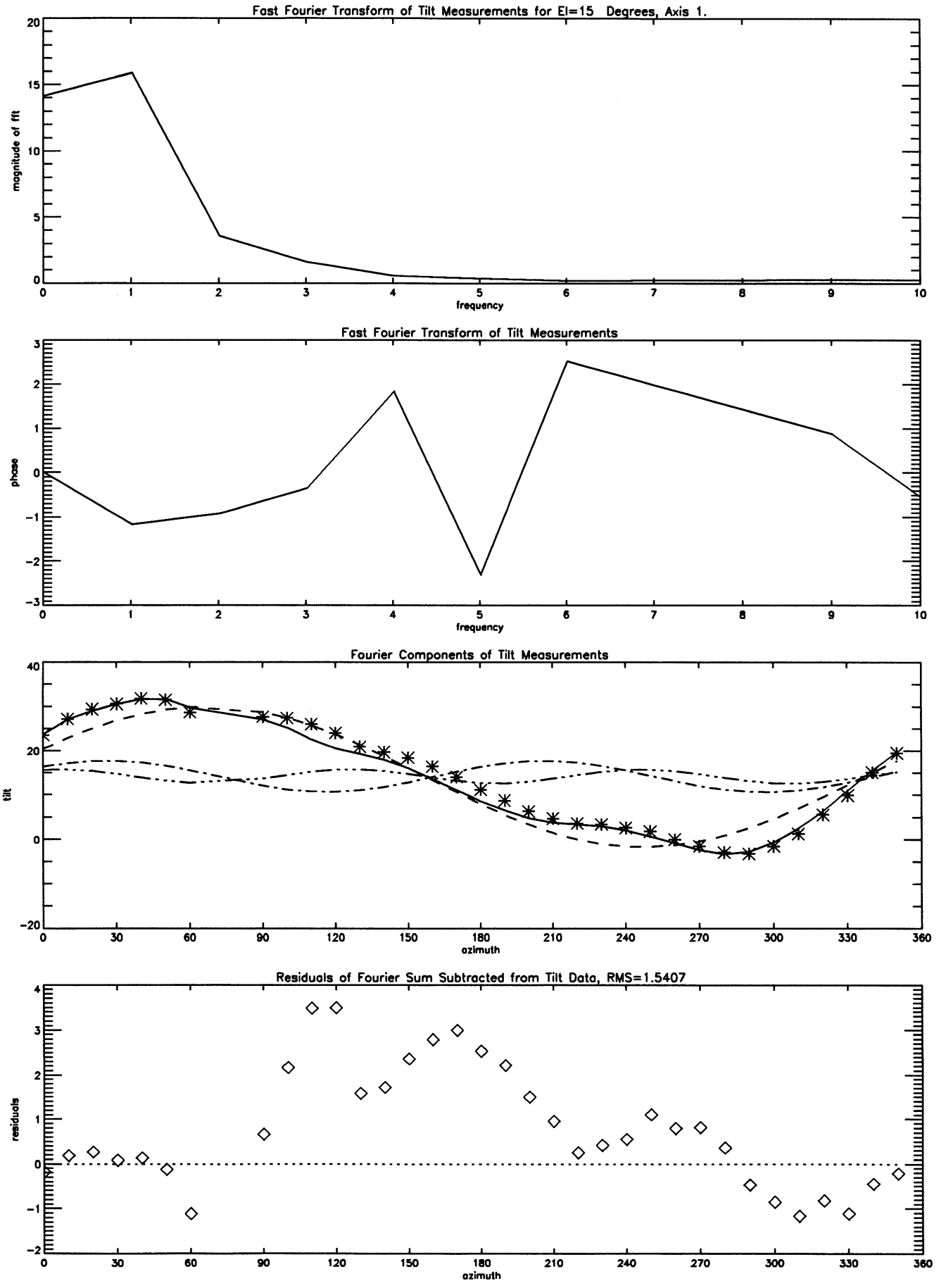
jimpoint-tcl.ps

Figure 13.



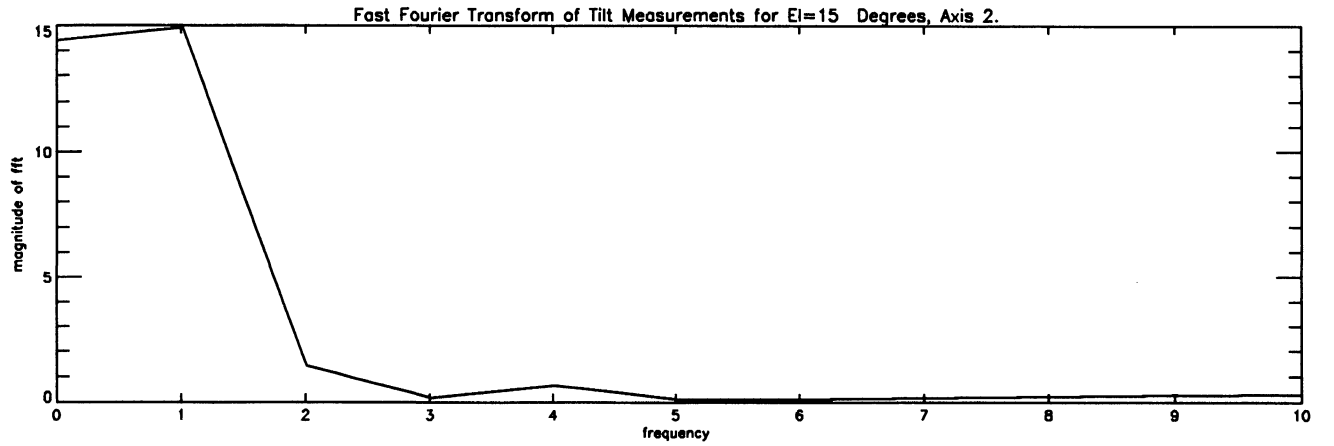
jimpoint-15.ps

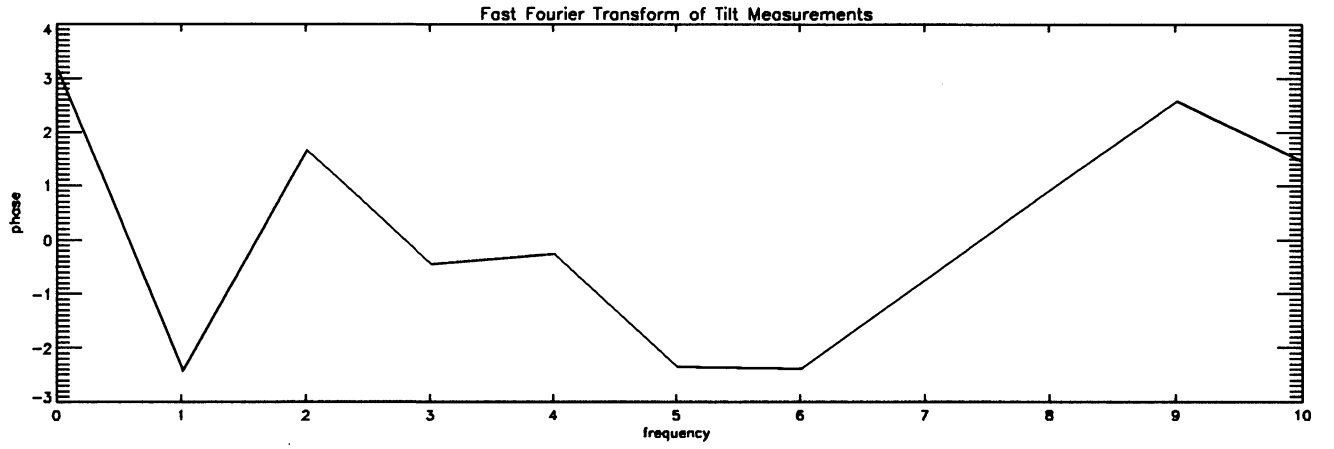
Figure 14.

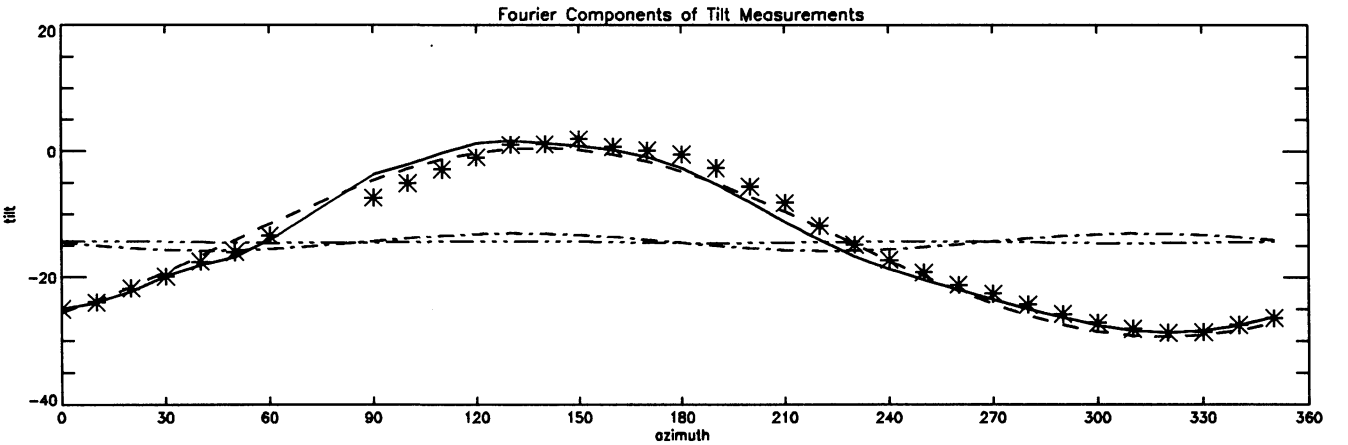


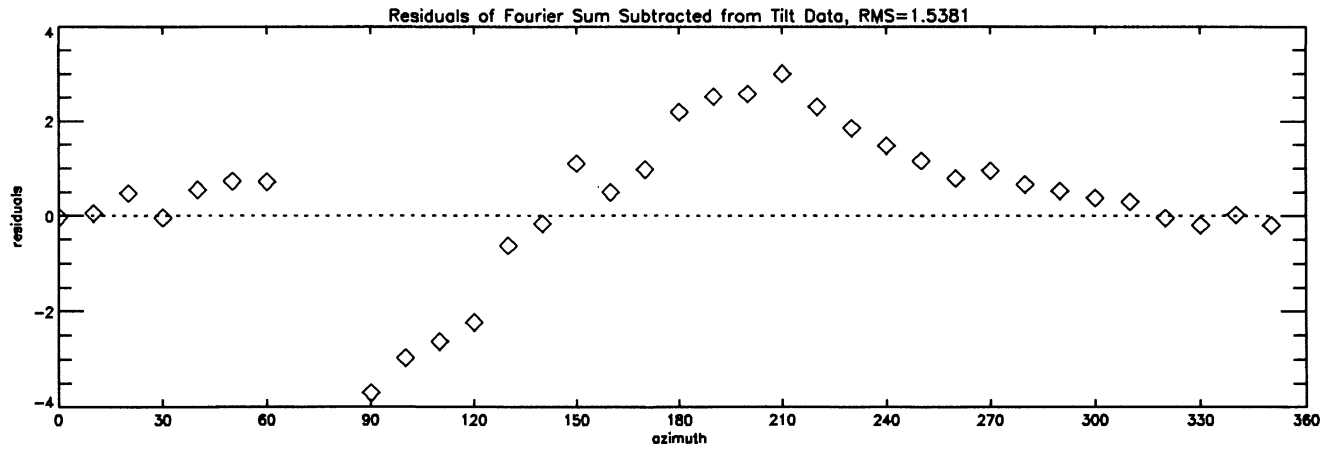
jimpoint1.ps

Figure 15.



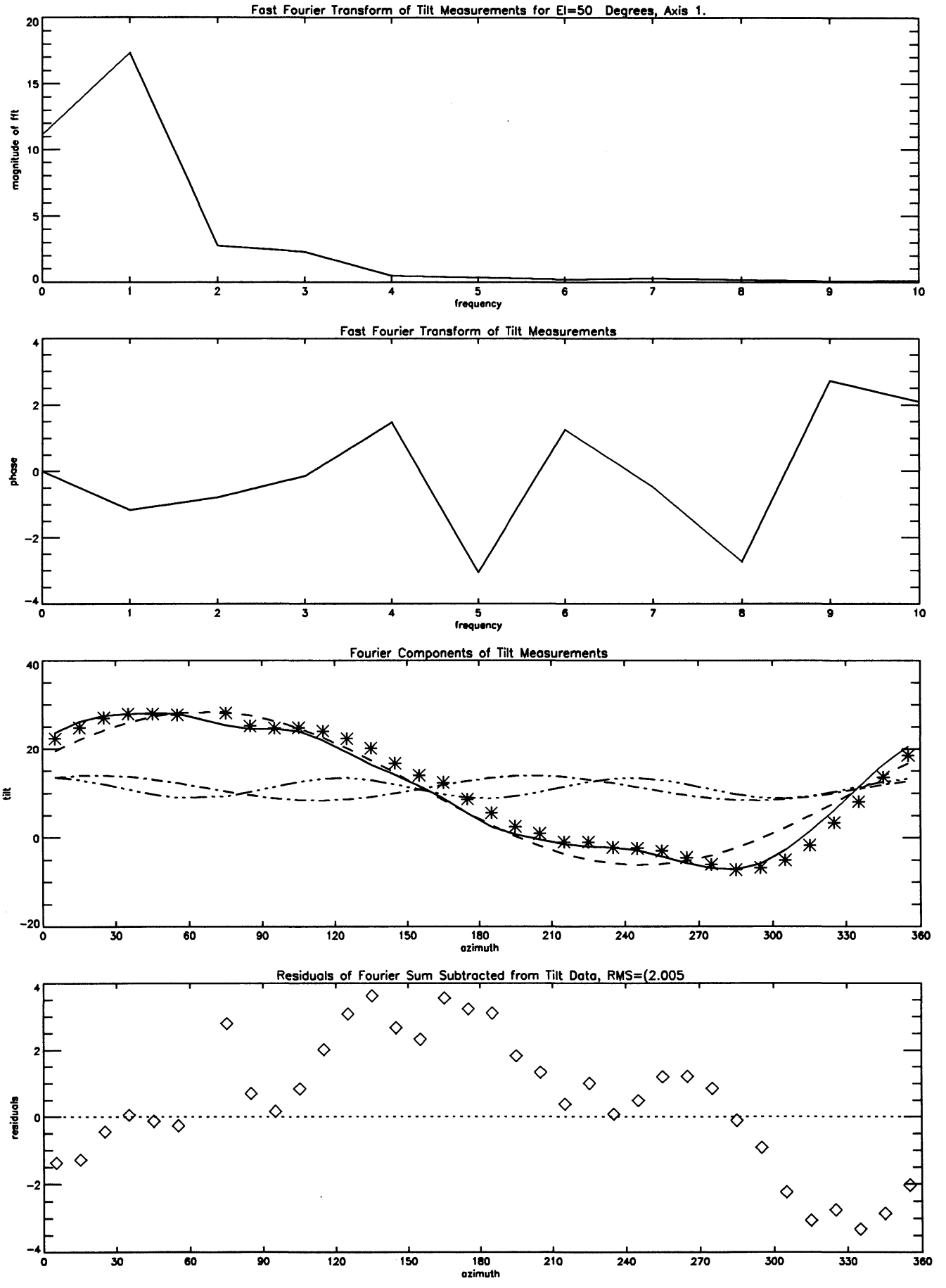






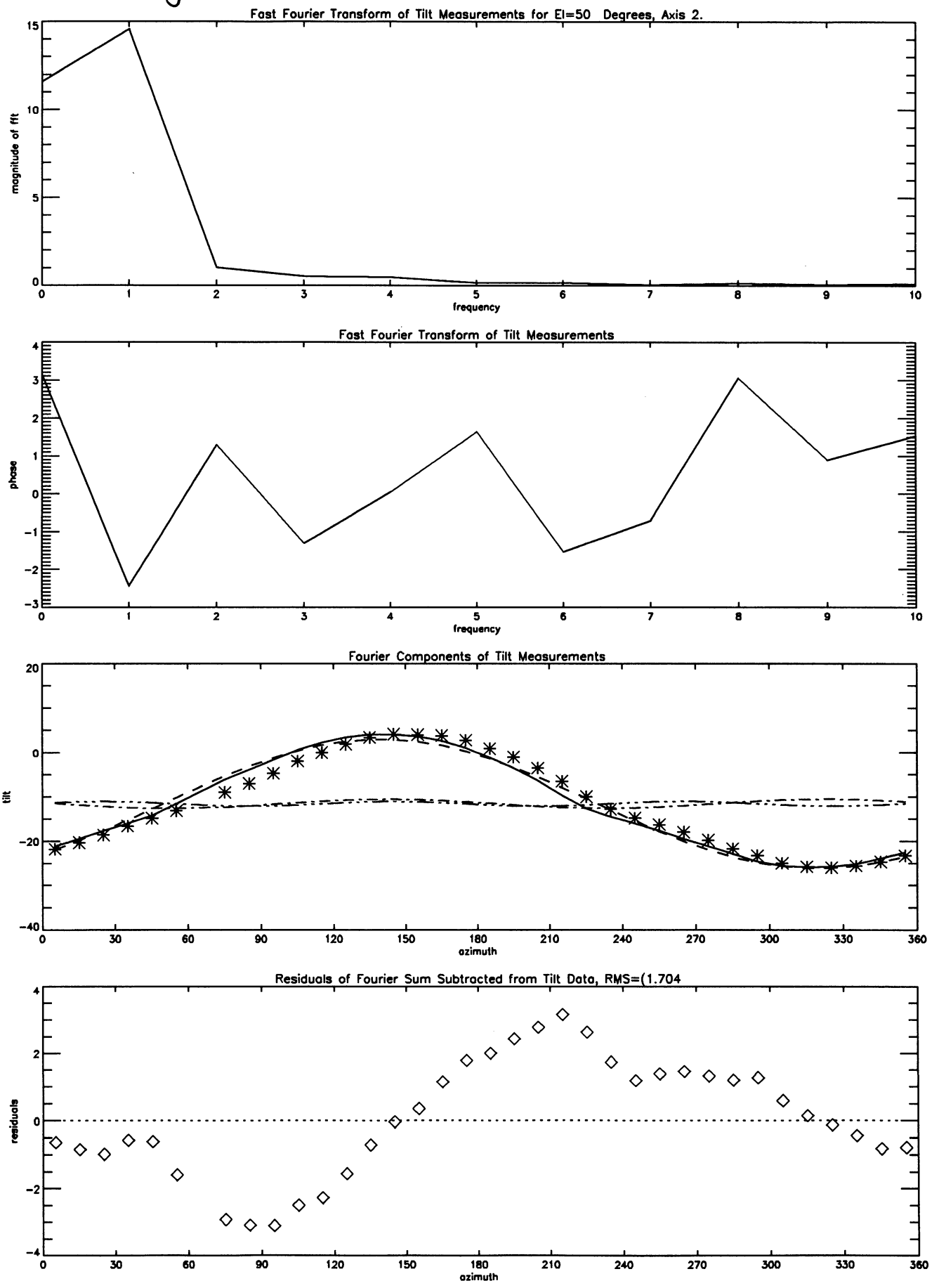
jimpoint2.ps

Figure 16.



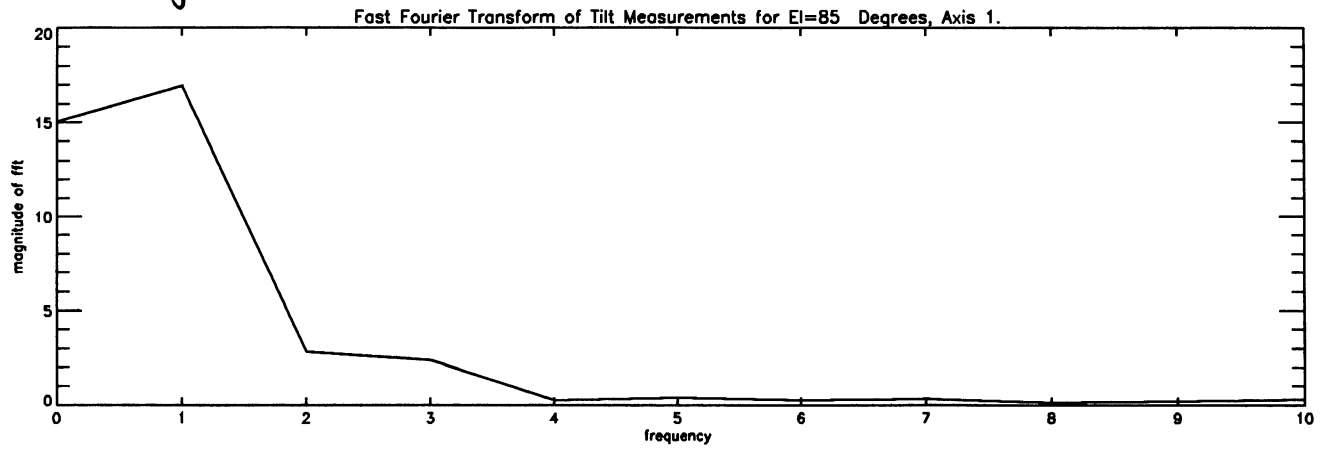
jimpoint3.ps

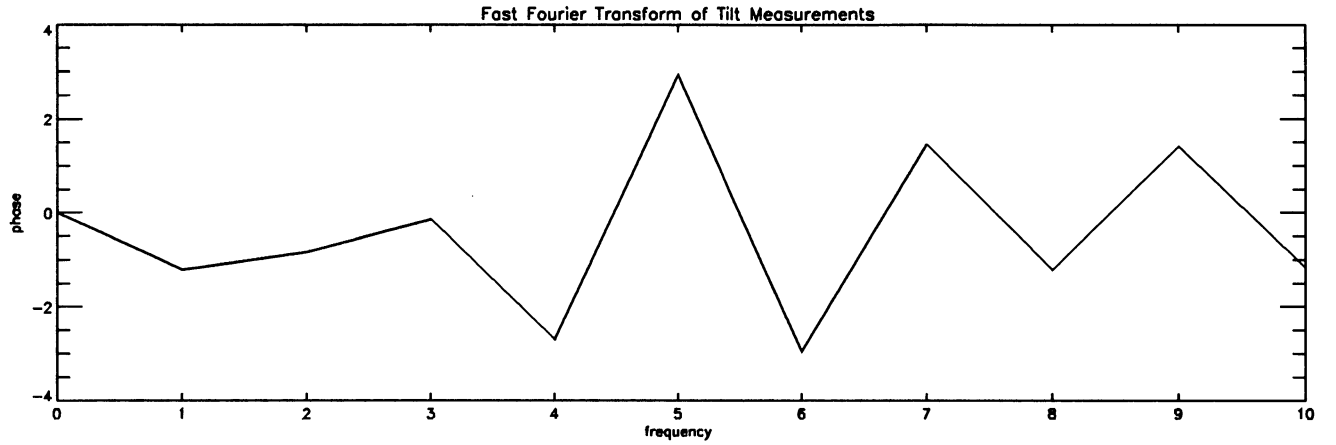
Figure 17.

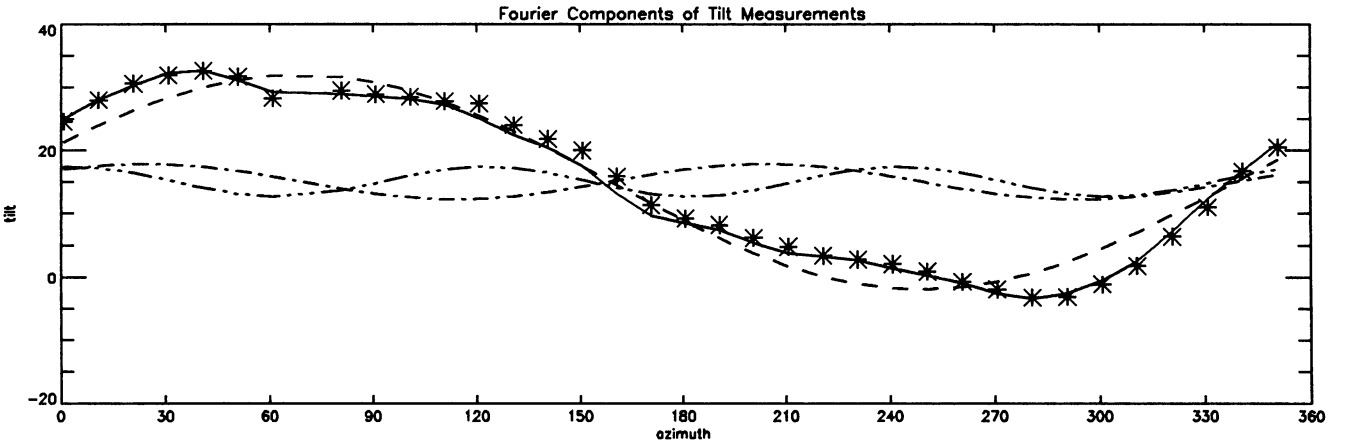


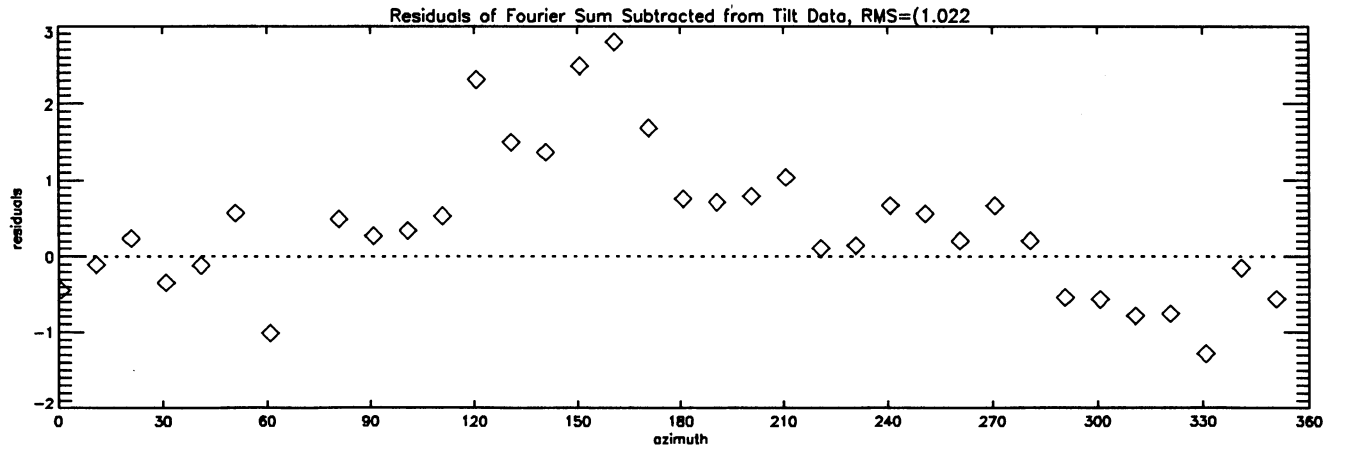
jimpoint4.ps

Figure 18.



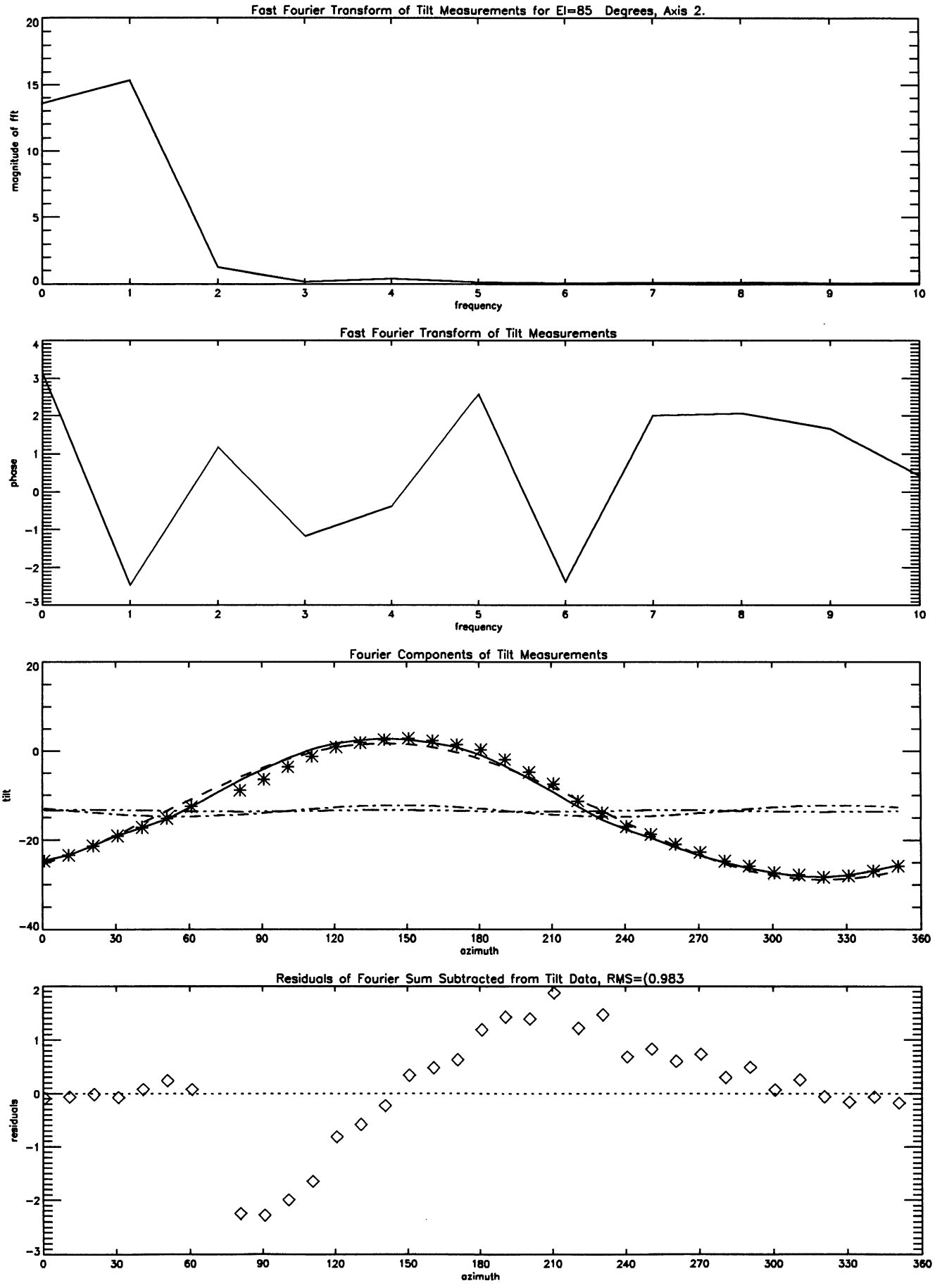






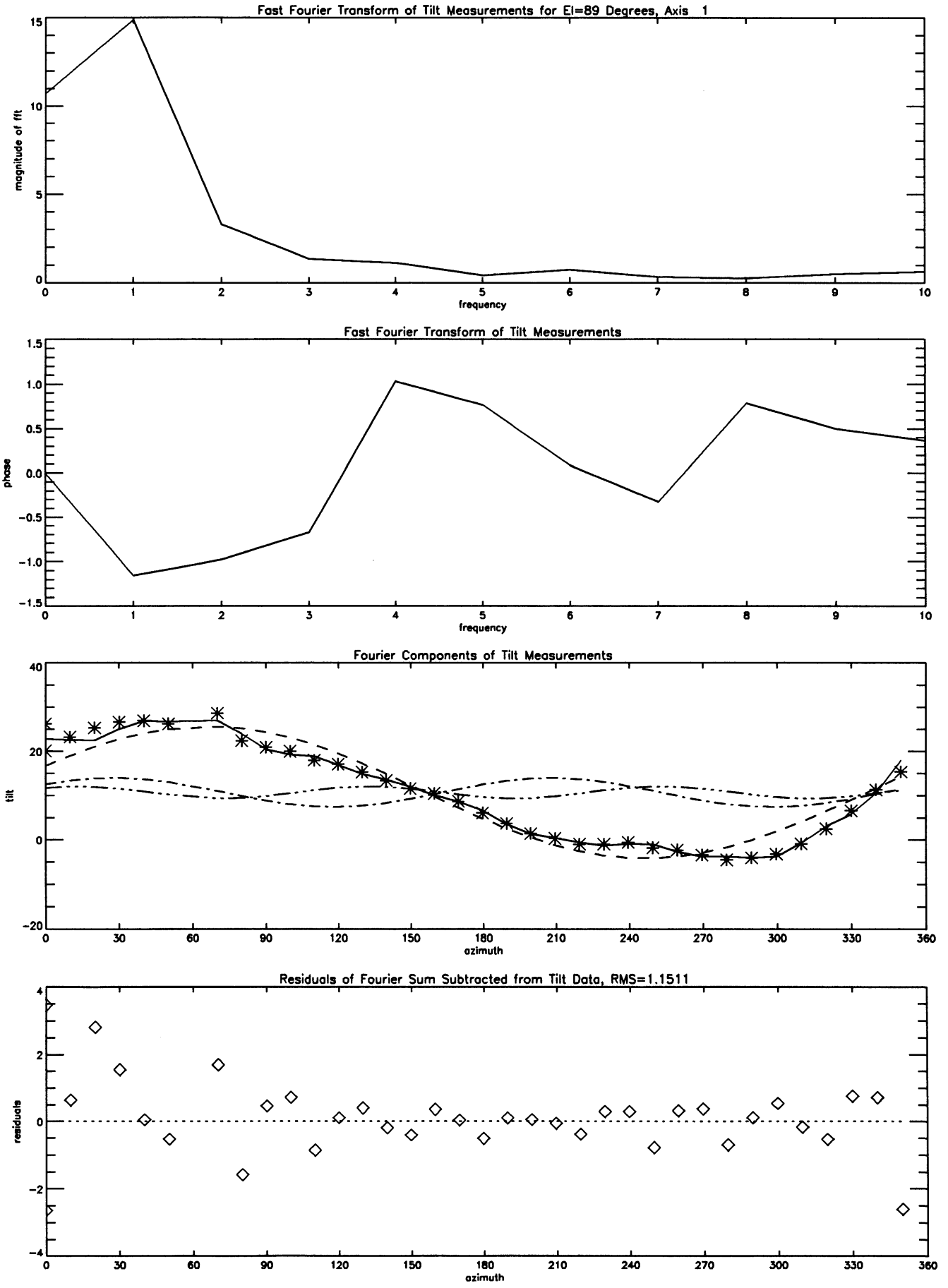
jimpoint5.ps

Figure 19.



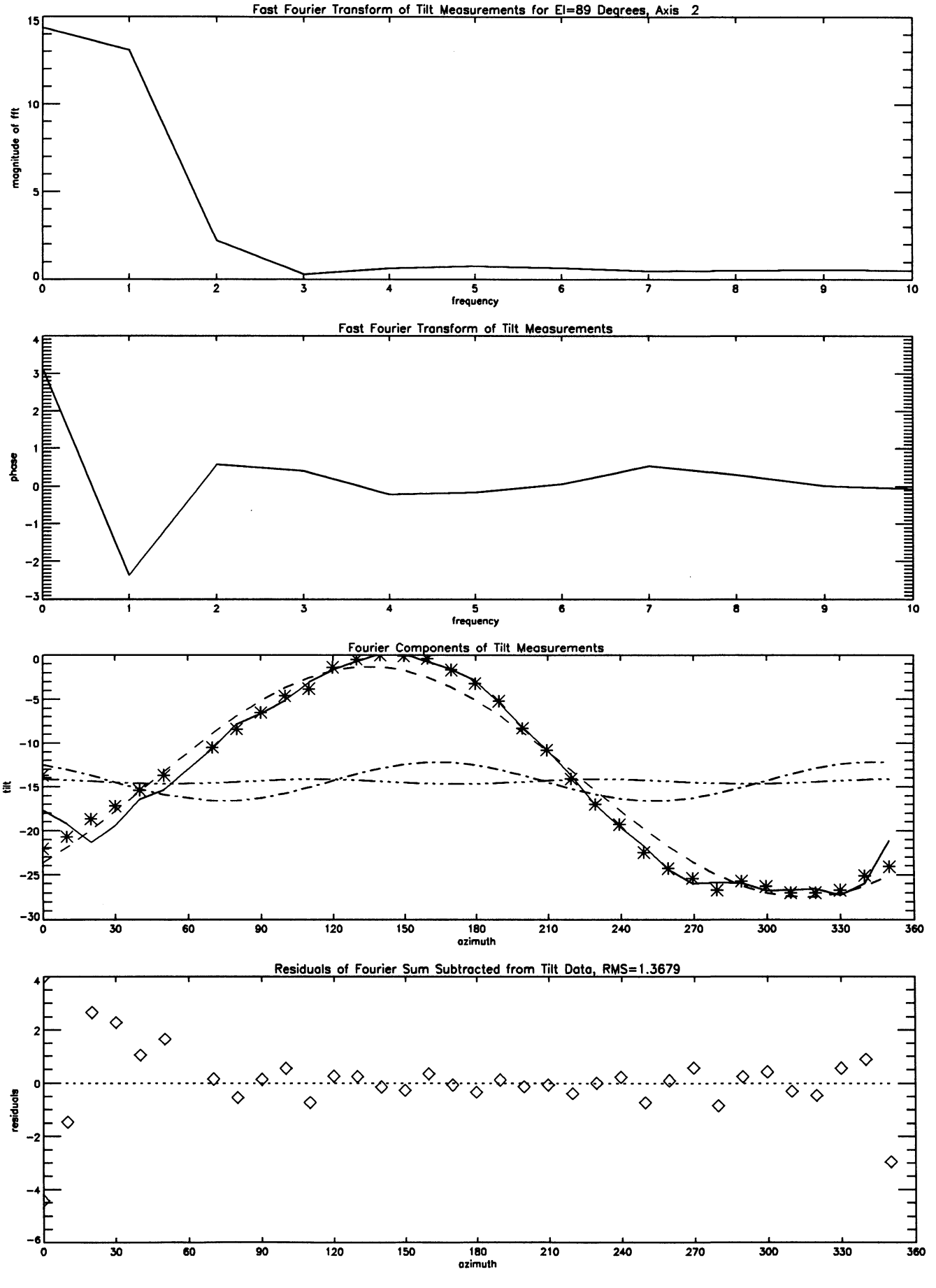
jimpoint6.ps

Figure 20.



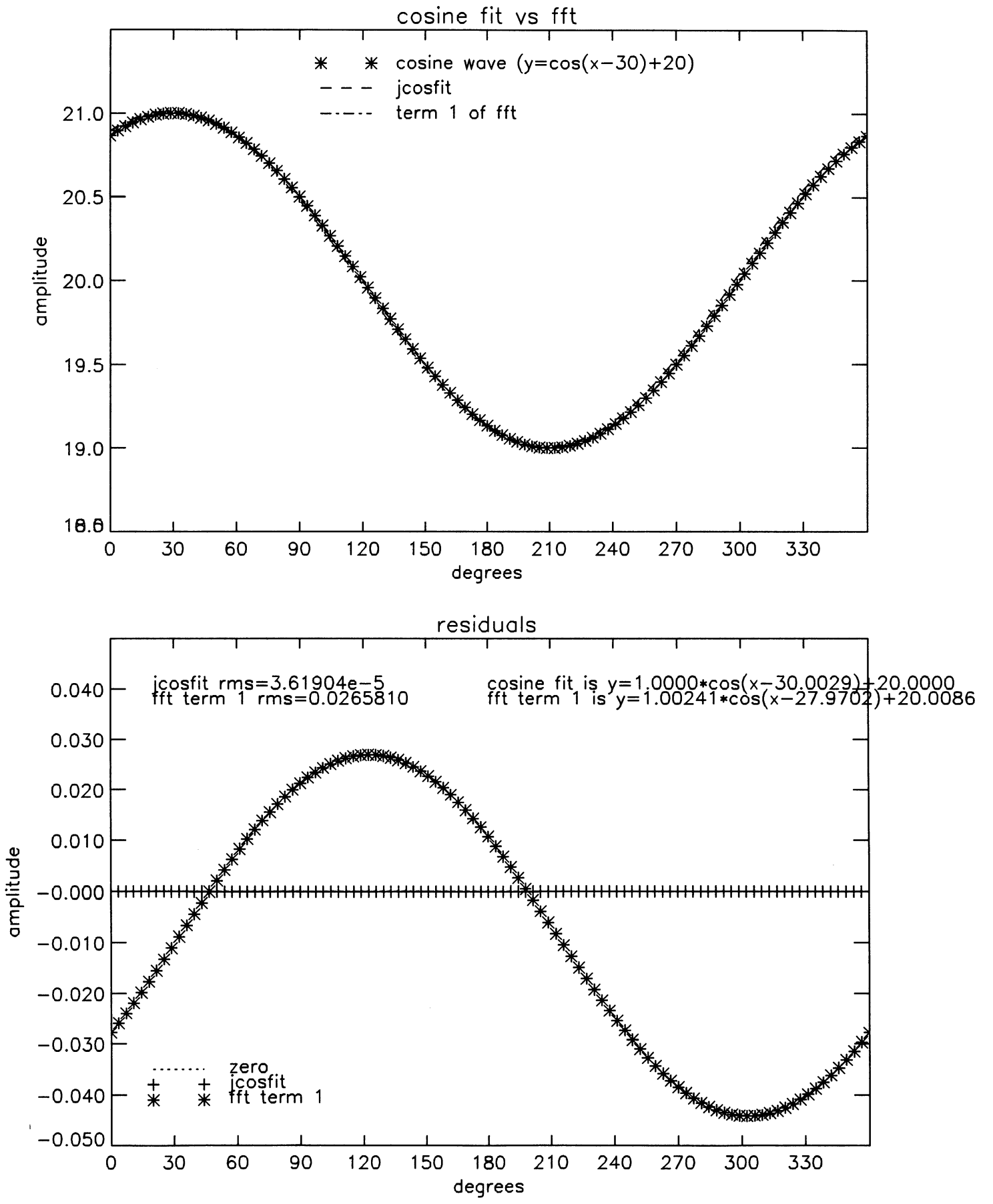
jimpoint7.ps

Figure 21.



jimpoint8.ps

Figure 22.



fit vs fft3.ps

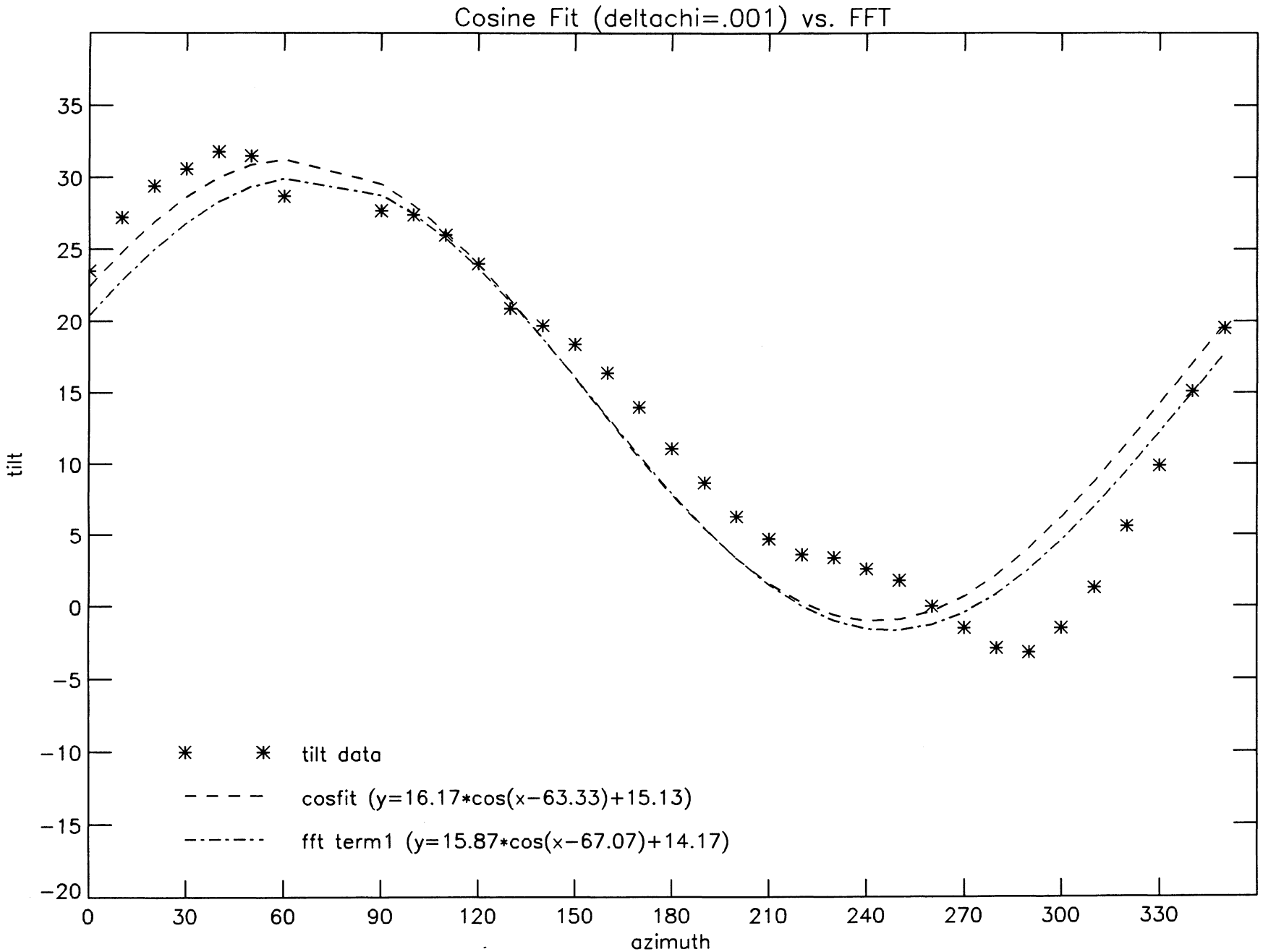


Figure 23.

fitvsfft1.ps

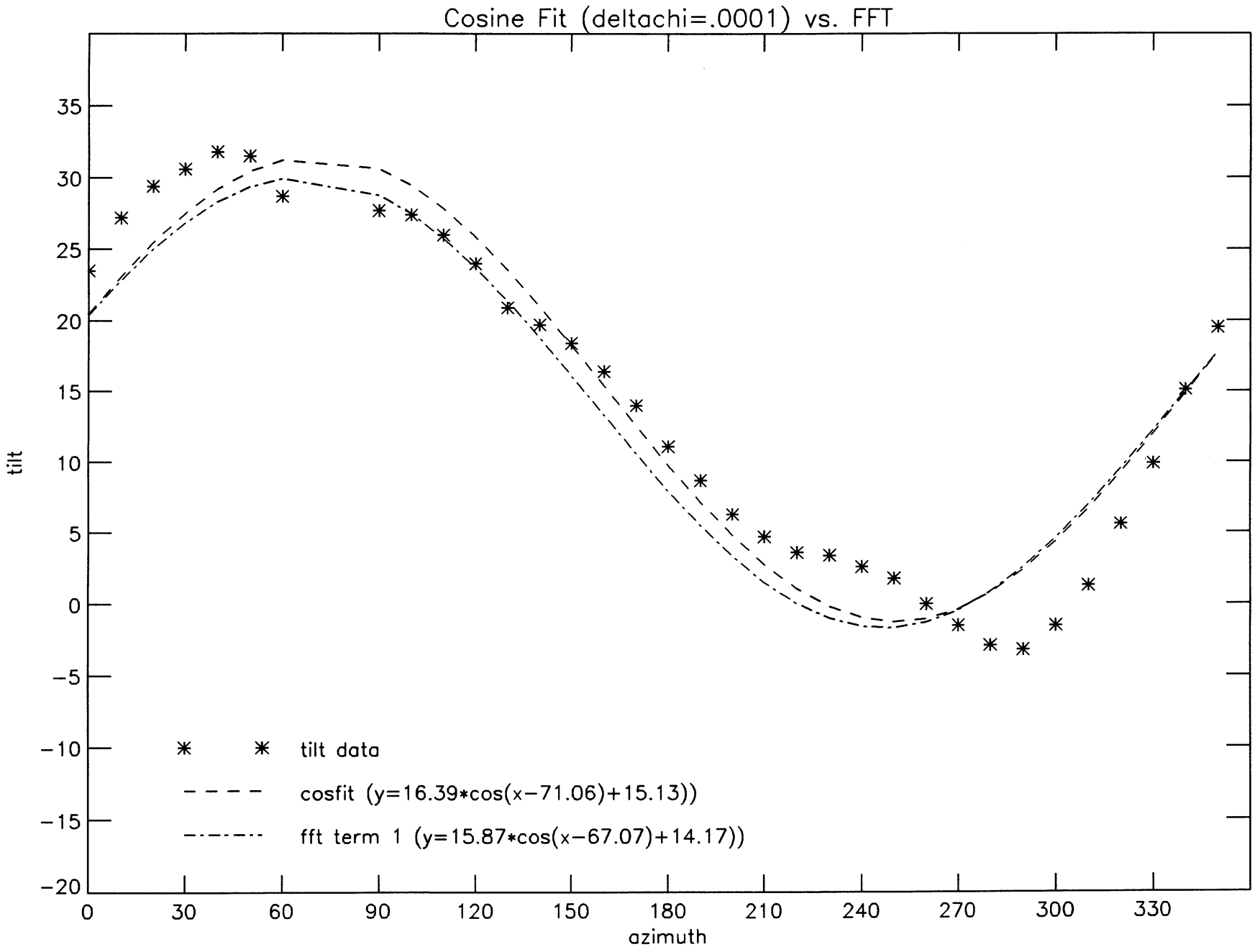


Figure 24

fitvsfft2.ps

Draft

Draft

National Radio Astronomy Observatory
Tucson, Arizona
August 15, 1995

MEMORANDUM

To: 12–m Memo Series
From: J. Wren and P. Jewell
Subject: Analysis of 12–m Pointing Measurements

1 Introduction

Using the *PV Wave* mathematics package, we have developed some tools for analyzing the data from optical and radio pointing runs. Specifically, we want to look for terms that the pointing model might be missing. To do this we have created procedures in *PV Wave* that display the data in ways that may provide new insight into periodic trends in pointing offsets.

The two main methods we have developed for pointing analysis are:

1. Create a contour or surface map that displays the pointing data.
2. Create a tool that performs a fourier analysis and displays the results.

2 Pointing the 12–m

There are several problems in the pointing of the 12–m. The pointing model may be missing some high order terms that may be significant to the pointing accuracy of the telescope. In order to compensate for these, we need to extract these terms from the residuals of the pointing model fit.

Pointing data can be taken in three ways.

1. Pointing data is kept from pointing tests that are made during regular observing sessions.
2. A radio pointing run can be executed manually so that a number of sources are observed across the entire field of the telescope.
3. A optical pointing run can be executed using an attached optical telescope.

These methods allow the pointing model to be updated periodically. The program that does this for the 12–m is called `gpoint` which was written by Phil Jewell. This program performs a non-linear least squares fit to the pointing data and adjusts the coefficients of the pointing equations accordingly. We are concerned with identifying any remaining terms that `gpoint` may be missing.

3 Working with the Data

There are some special problems to overcome when working with a pointing data set. For one thing, it is incomplete when arranged in azimuth and elevation. In order to do a proper 3–D display of the data, we needed to arrange the data into azimuth and elevation bins. This can be done with the *PV Wave* procedure called `jpoint_bin.pro`. This procedure takes a data array of one dimension and arranges it into a two dimensional array as a function of azimuth and elevation.

If the bins are smaller, then there will be gaps in the pointing data. We have written a function called `jsurface_fill.pro` which then fills in the gaps in the data using a *PV Wave* function called `spline.pro`. We have also written procedures that can do a linear interpolation if desired. The bin size can also be enlarged so that all the bins are filled with data points. Because this lowers the resolution of the data trends, a slightly lower bin size may be desirable. The usage for these two functions follows:

```
WAVE> result1=jpoint_bin(az,el,array)
WAVE> result2=jsurface_fill(result1)
```

When the data is properly arranged then it is possible to create some three dimensional displays which show data trends.

4 3–D Displays

Now that the data is binned, it is possible to create some 3–D displays. One simple way to display the data is to create a surface plot using the binned data array. To do this in *PV Wave* simply enter:

```
WAVE> surface,result2
```

This plot is not very helpful, however. A contour plot produces similar results. A more helpful display is a contour map in polar coordinates. To generate this display, we have written a procedure in *PV Wave* that takes a set of binned data and converts it to a polar coordinate system. This procedure is called `jconvert_polar.pro`. This procedure produces a polar contour plot that can be shaded or labeled based on user preference. The results can then be printed on a regular or color postscript printer. The usage for `jconvert_polar.pro` is as follows:

```
WAVE> jconvert_polar,result2
```

These display techniques may be used to augment `gpoint`. Currently the only available method for visualization of residuals is a 2–D vector plot on an az–el grid. It may be possible to later modify `gpoint` to call *PV Wave* and use some the plotting tools mentioned above. One of *PV Wave*'s strengths is that it works well with FORTRAN and C programs. *PV Wave* can call an external FORTRAN or C program or the programs themselves can be made to call *PV Wave* and use it as a display tool. In this way, we may be able to update `gpoint` to use these display tools we have created in *PV Wave*.

5 Fourier Analysis

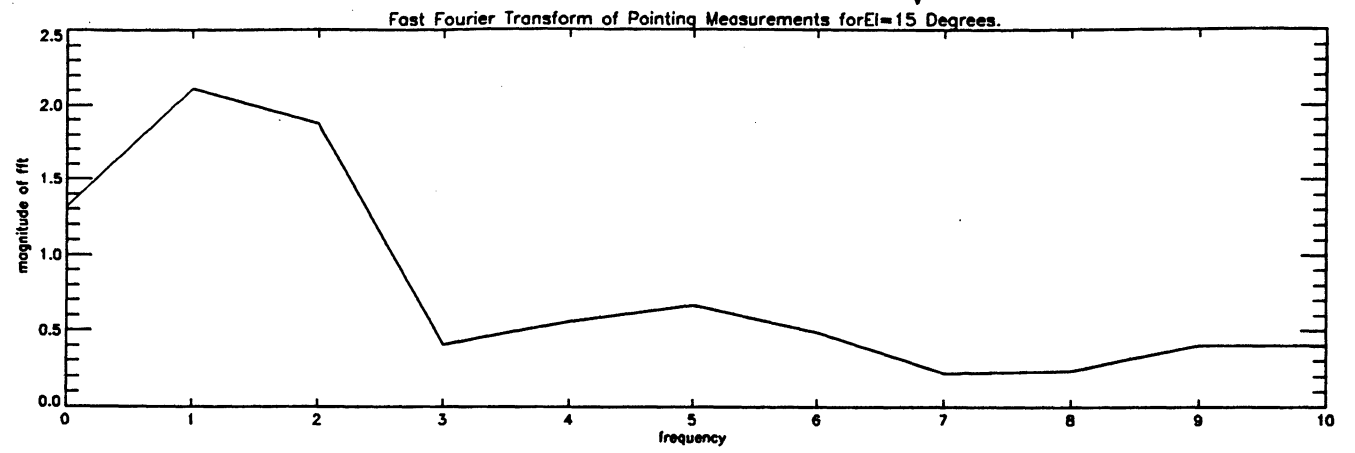
Currently gpoint offers no Fourier analysis of the pointing residuals. To help solve this problem, we have written a procedure called jimpoint.pro. This procedure takes data from an optical or radio pointing run, bins it, interpolates it, and then displays the fourier components and residuals. The resulting display has four windows. The first window shows the magnitude of the fft as a function of frequency. The second window shows the phase of the fft as a function of frequency. The third window shows the first three terms and their sum plotted over the data points. The fourth window shows the residuals of the fourier sum subtracted from the actual data. The usage is as follows.

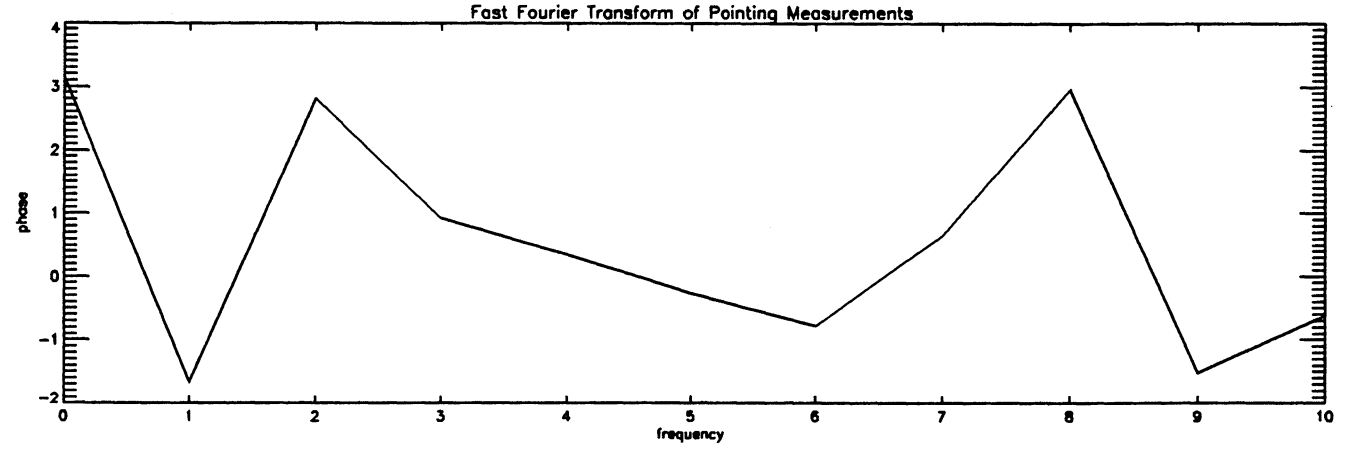
```
WAVE> jimpoint,'path'
```

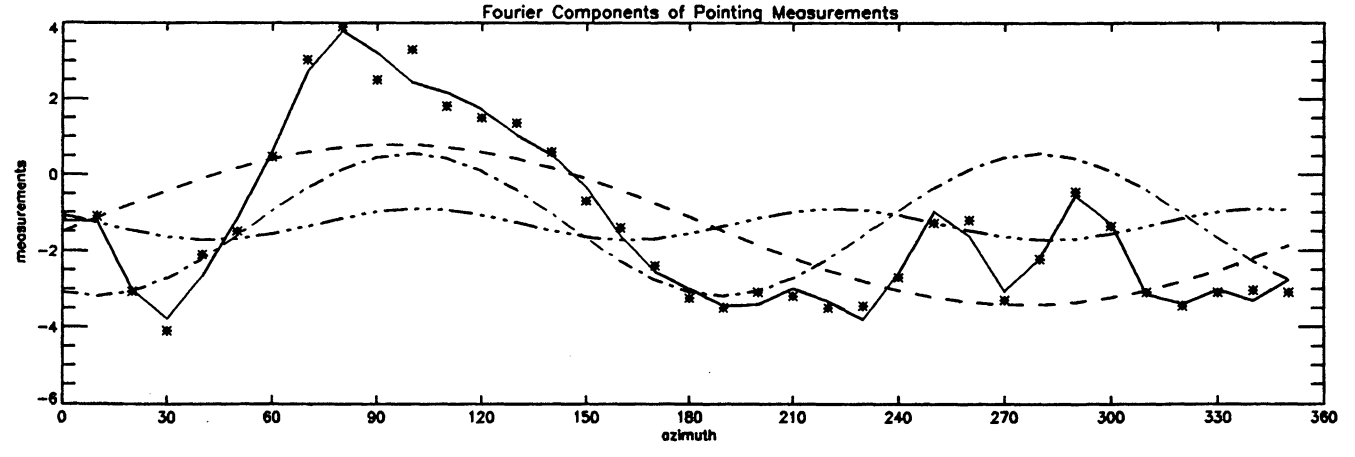
The procedure then reads the file indicated by *path* and asks which data columns the user wishes to analyze. The first two columns should be the azimuth and elevation data respectively. The third column should contain the pointing data to be analyzed.

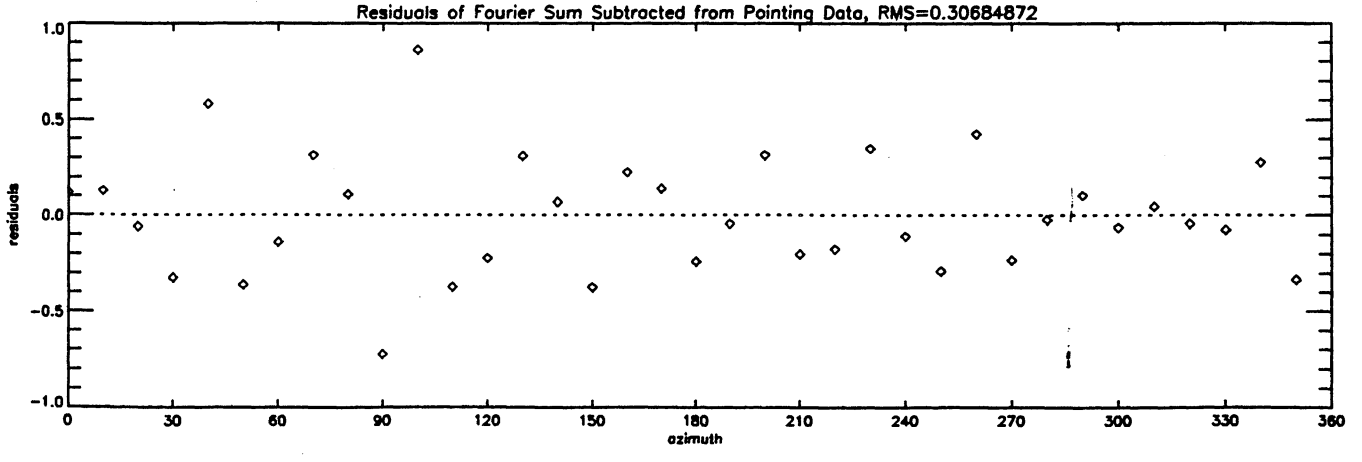
We may also be able to integrate these procedures into gpoint.

res-daz for ptout-091594d.lis Figure 1.



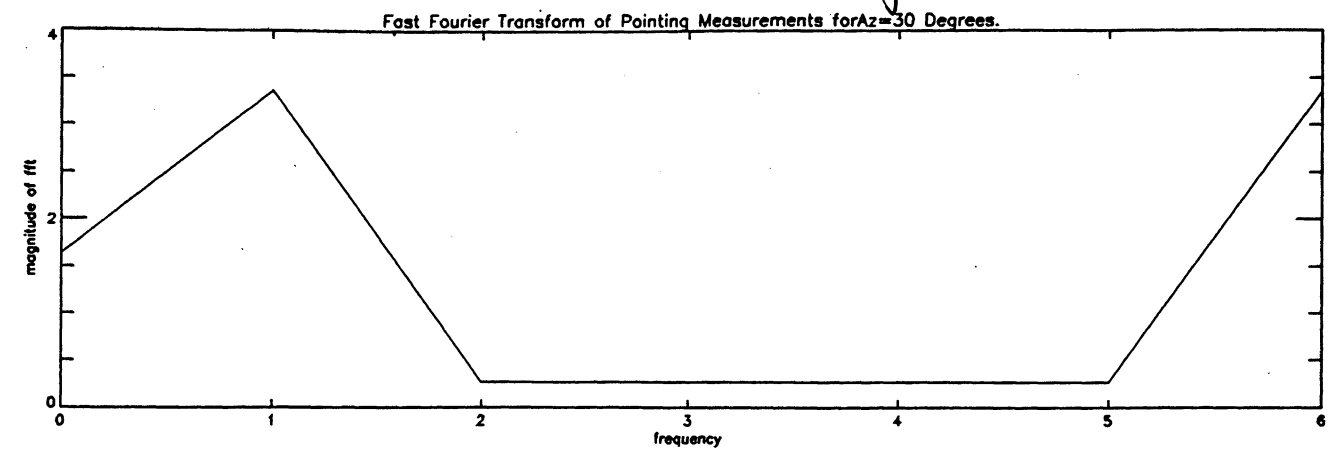


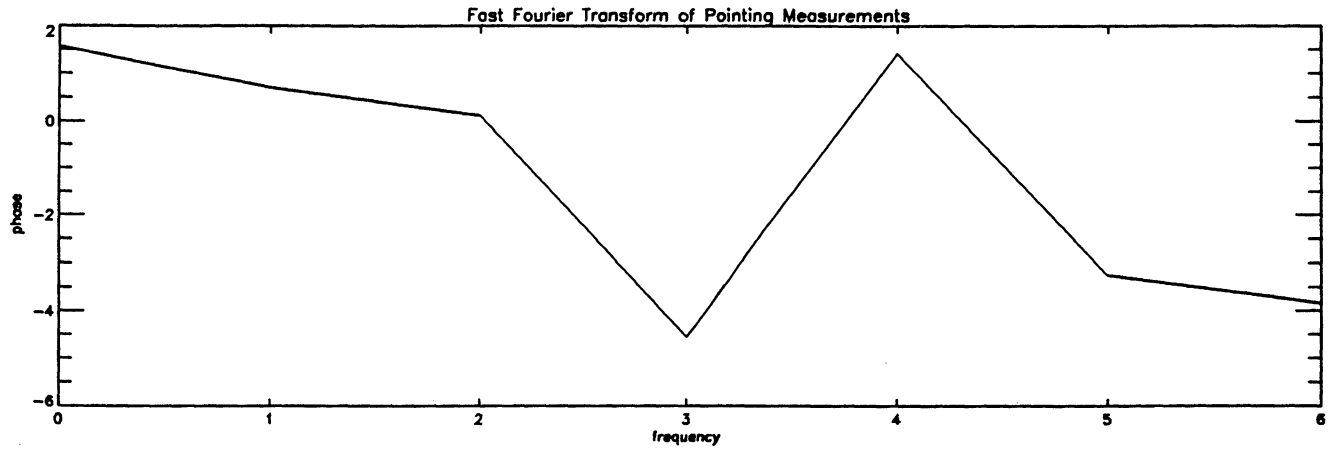


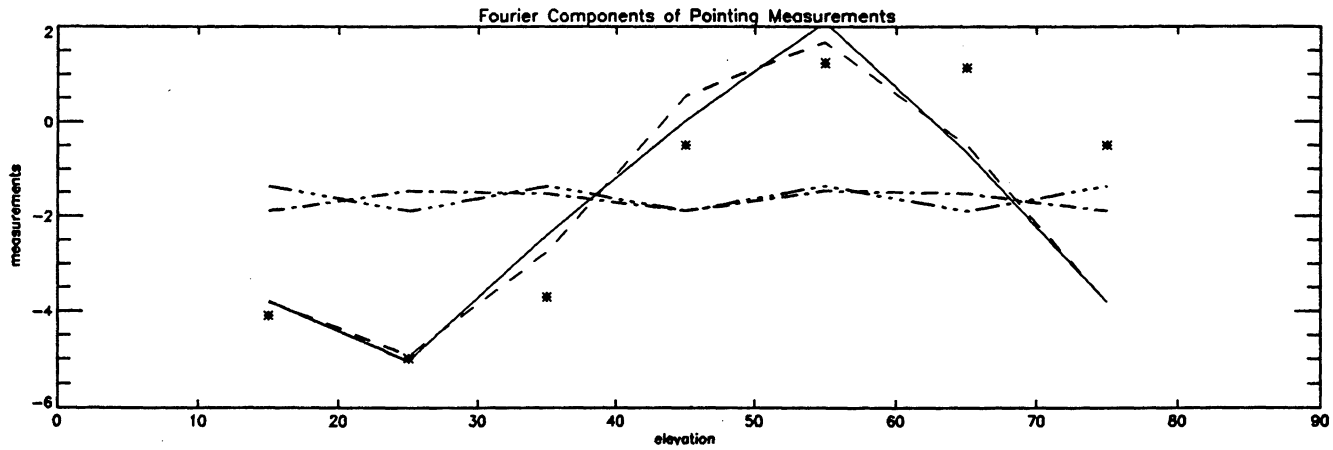


ptout_091594d.115

Figure 2.







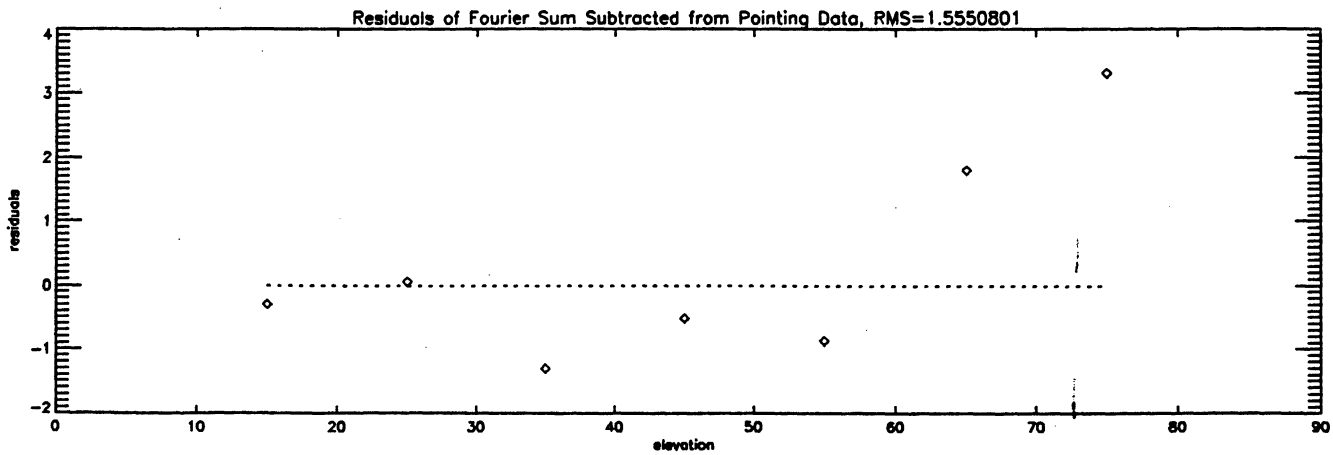
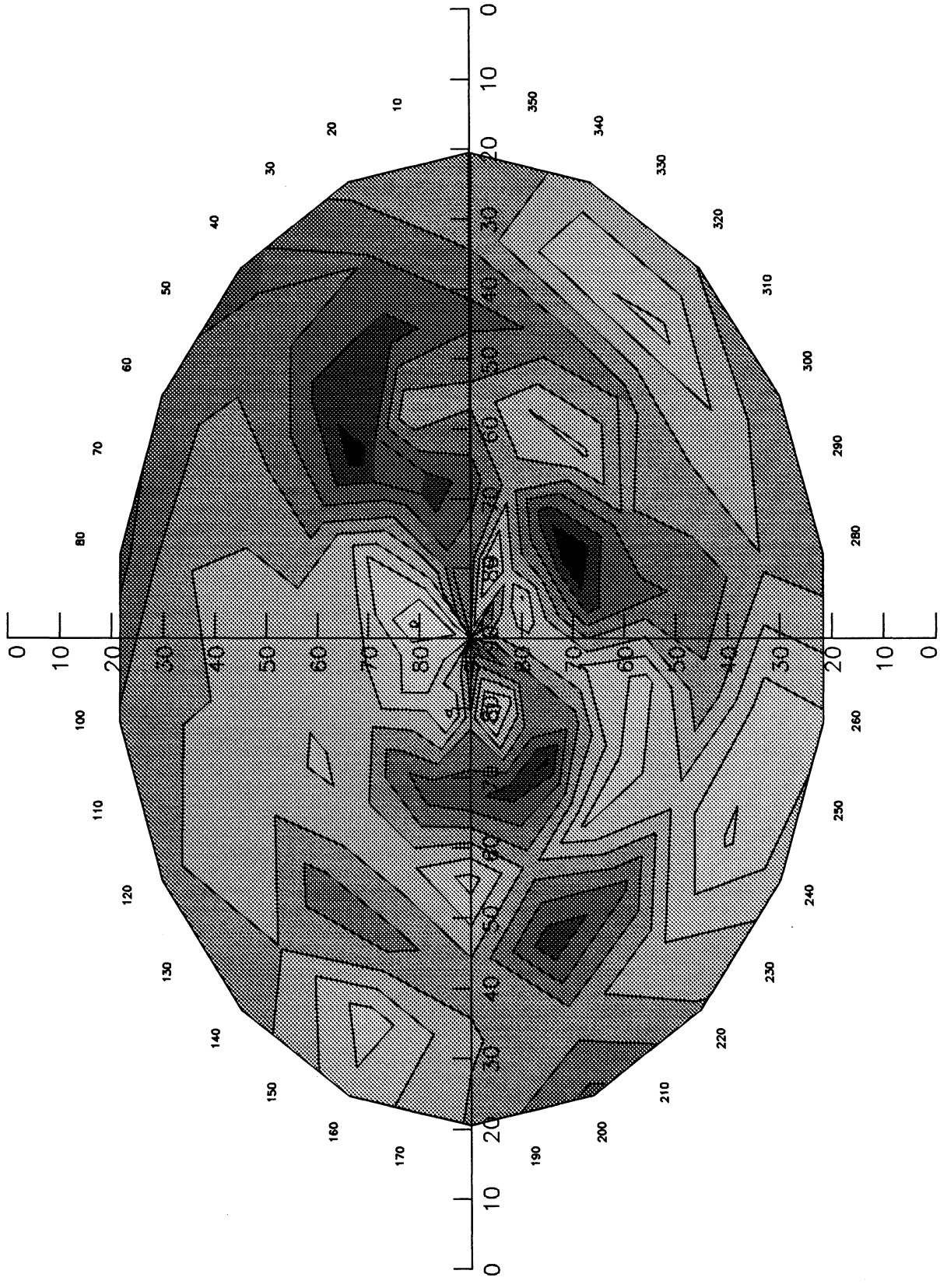


Figure 3_


0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300
310
320
330
340
350