DETECTION OF REFLECTOR SURFACE FROM NEAR FIELD PHASE MEASUREMENTS

Final Report
Grant No. NCC-3-146

February 27, 1991

Final Report
Grant No. NCC-3-146

DETECTION OF REFLECTOR SURFACE FROM NEAR FIELD PHASE MEASUREMENTS

Submitted by:
Dr. Nathan Ida
Department of Electrical Engineering
The University of Akron
Akron, OH. 44325-3904

Submitted to
NASA Lewis Research Center

DETECTION OF REFLECTOR SURFACE FROM NEAR FIELD PHASE MEASUREMENTS

INTRODUCTION

The deviation of a reflector antenna surface from a perfect parabolic shape causes degradation of the performance of the antenna. The shape of the antenna is therefore desired for several applications. If the shape of the antenna can be determined quickly during its manufacture, localized deviations from a perfect surface might be eliminated. If an antenna should become damaged, the location of the damage may allow easier repair. This is particularly important since the damage is not easy to see and is difficult to measure directly.

The problem of determining the shape of the reflector surface in a reflector antenna using near field phase measurements in not a new one. A recent issue of the IEEE transactions on Antennas and Propagation (June, 1988) contained numerous descriptions of the use of these measurements, including works by Y Rhamat-Samii, et al, W. Chujo, et al., and J. J. Lee, et al. These accounts use one of two methods: holographic reconstruction or inverse Fourier transform.

Holographic reconstruction, used by Rahmat-Samii, makes use of measurement of the far field (amplitude and phase) of the reflector and then applies the Fourier transform relationship between the far field and the current distribution on the reflector surface.

Inverse Fourier transformation uses the phase measurements to determine the far field pattern using the method of Kerns. After the far field pattern is established, an inverse Fourier transform is used to determine the phases in a plane between the reflector surface and the plane in which the near field measurements were taken.

These calculations are time consuming since they involve a relatively large number of operations. For the holographic reconstruction technique, the calculations are of the order of $n^2\log(2n)$ floating point operations per phase measurement. The inverse Fourier transform method requires $n^2\log(2n)$ calculations to obtain the far field pattern, followed by $n^2\log(2n)$ operations to obtain the near filed phases again.

A much faster method can be used to determine the position of the reflector. This method makes use of simple geometric optics to determine the path length of the ray from the feed to the reflector and from the reflector to the measurement point. This method takes only 57 floating point operations per phase measurement and gives the specular reflection point directly, rather than the phase at a plane near the reflector, as the inverse Fourier transform method does.

For small physical objects and low frequencies, diffraction effects have a major effect on the error, and the algorithm provides incorrect results. It is believed (but not proven) that the effect is less noticeable for large distortions such as antenna warping, and more noticeable for small, localized distortions such as bumps and depressions such as might be caused by impact damage.

Determination of the applicable distortion feature sizes is outside the scope of this work.

THE REFLECTOR SURFACE ESTIMATION ALGORITHM

Necessary assumptions.

The Reflector Surface Estimation (RSE) algorithm developed here, requires that there be no caustic points between the reflector surface and the measurement plane. If this assumption is met, each point on the phase measurement plane corresponds to either zero or one specular reflection point on the antenna surface.

Geometry of the problem.

The geometry used in the discussions throughout this document are shown in Figure 1. In accordance with normal conventions, the antenna radiates in the z-direction. A feed horn is located at the apparent focus (x_f,y_f,z_f). A ray emitted from the feed intersects the reflector surface at the point (x,y,z). The ray is reflected from (x,y,z) and intersects the near field plane at a point (x_a,y_a,z_z).

Required data.

The RSE algorithm requires transform phase measurements in the near field of a reflector antenna to the point on the antenna which caused the specular reflection. Required inputs to the basic algorithm are:

Phase measurements	Absolute phase measurements or relative phase measurements which can be converted to absolute.
Frequency	The frequency at which the phase measurements were taken.
Antenna feed location	The location of the antenna feed in the coordinate system in which the results are desired.
Reference length	One physical measurement which must be made to provide relative phase length.
Phase measurement	The distance from the origin of the coordinate system to the plane in which the phase measurements are made.

Theory.

The electrical distance from the feed of the antenna to the phase measurement plane can be found from:

$$d = d_{ref} - \frac{\phi_{ref} - \phi_{ij}}{k} \qquad (1)$$

The distance consists of two components: the distance from the feed to the reflector and the distance from the reflector to the measurement point.

$$d_{ij} = \sqrt{(x-x_f)^2 + (y-y_f)^2 + (z-z_f)^2} + \sqrt{(x-x_a)^2 + (y-y_a)^2 + (z-z_a)^2} \qquad (2)$$

It is desired to know the location of that reflector point. Since we know the phase at many points in the near field, we can calculate the angle of arrival of the ray. The partial derivatives are:

$$\frac{\partial \phi_{ij}}{\partial x} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{x_{i+1,j} - x_{i-1,j}}, \qquad \frac{\partial \phi_{ij}}{\partial y} = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{y_{i,j+1} - y_{i,j-1}} \qquad (3)$$

$$m_x = \frac{1}{k}\frac{\partial \phi_{ij}}{\partial x}, \qquad m_y = \frac{1}{k}\frac{\partial \phi_{ij}}{\partial y}, \qquad m_z = \frac{1}{\sqrt{1 - m_x^2 - m_y^2}} \qquad (4)$$

From these derivatives, we can define the path of the incoming ray:

$$x = \frac{m_x}{m_z}(z-z_a) + x_a, \qquad y = \frac{m_y}{m_z}(z-z_a) + y_a \qquad (5)$$

We define constants representing the slopes of the lines:

$$C_1 = \frac{m_x}{m_z}, \qquad C_2 = \frac{m_y}{m_z} \qquad (6)$$

Substituting (5) and (6) into (2)

$$d_{ij} = \sqrt{(C_1(z-z_a) + x_a - x_f)^2 + (C_2(z-z_a) + y_a - y_f)^2 + (z-z_f)^2} + \sqrt{(x-x_a)^2 + (y-y_a)^2 + (z-z_a)^2} \qquad (7)$$

or:

$$d_{ij} = \sqrt{(-C_1 z_a + x_a - x_f + C_1 z)^2 + (-C_2 z_a + y_a - y_f + C_2 z)^2 + (z - z_f)^2} + \sqrt{(x-x_a)^2 + (y-y_a)^2 + (z-z_a)^2} \quad (8)$$

Two new constants are now defined,

$$d_1 = -C_1 z_a + x_a - x_f \qquad d_2 = -C_2 z_a + y_a - y_f \quad (9)$$

Substituting into equation (8)

$$d_{ij} = \sqrt{(d_1 + C_1 z)^2 + (d_2 + C_2 z)^2 + (z - z_f)^2} + \sqrt{(x-x_a)^2 + (y-y_a)^2 + (z-z_a)^2} \quad (10)$$

Expanding the first term and segregating powers of z,

$$d_{ij} = \sqrt{(d_1^2 + 2d_1 C_1 z + C_1^2 z^2) + (d_2^2 + 2d_2 C_2 z + C_2^2 z^2) + (z^2 - 2zz_f + z_f^2)} + \sqrt{(x-x_a)^2 + (y-y_a)^2 + (z-z_a)^2} \quad (11)$$

or:

$$d_{ij} = \sqrt{(d_1^2 + d_2^2 + z_f^2) + (2d_1 C_1 + 2d_2 C_2 - 2z_f)z + (C_1^2 + C_2^2 + 1)z^2} + \sqrt{(x-x_a)^2 + (y-y_a)^2 + (z-z_a)^2} \quad (12)$$

Three additional constants are defined:

$$f_1 = d_1^2 + d_2^2 + z_f^2 \quad (13)$$

$$f_2 = 2d_1C_1 + 2d_2C_2 - 2z_f \tag{14}$$

$$f_3 = C_1^2 + C_2^2 + 1 \tag{15}$$

Substituting (13) through (15) into (8)

$$d_{ij} = \sqrt{f_1 + f_2z + f_3z^2} + \sqrt{(x-x_a)^2 + (y-y_a)^2 + (z-z_a)^2} \tag{16}$$

Substituting (6) into equation (16) yields

$$d_{ij} = \sqrt{f_1 + f_2z + f_3z^2} + \sqrt{C_1^2(z-z_a)^2 + C_2^2(z-z_a)^2 + (z-z_a)^2} \tag{17}$$

$$d_{ij} = \sqrt{f_1 + f_2z + f_3z^2} + \sqrt{f_3(z-z_a)^2} \tag{18}$$

$$\sqrt{f_1 + f_2z + f_3z^2} = d_{ij} - \sqrt{f_3(z-z_a)^2} \tag{19}$$

The second term on the right hand side of equation (19) can be either $z-z_a$ or z_a-z. One root represents the desired solution and the other root represents a point along the ray but in the positive z direction from the near field plane.

Squaring both sides and selecting the proper root,

$$f_1 + f_2z + f_3z^2 = d_{ij}^2 - 2\sqrt{f_3}(z-z_a)d_{ij} + f_3(z-z_a)^2 \tag{20}$$

Separating the powers of z,

$$f_1 + f_2z + f_3z^2 = d_{ij}^2 - 2\sqrt{f_3}z_ad_{ij} + f_3z_a^2 + (2\sqrt{f_3}d_{ij} - 2f_3z_a)z + f_3z^2 \tag{21}$$

The z^2 terms cancel, so

$$f_1 + f_2 z = d_{ij}^2 - 2\sqrt{f_3}\, z_a d_{ij} + f_3 z_a^2 + (2\sqrt{f_3}\, d_{ij} - 2 f_3 z_a) z \tag{22}$$

Solving for z,

$$z = \frac{f_1 - d_{ij}^2 + 2\sqrt{f_3}\, z_a d_{ij} - f_3 z_a^2}{2\sqrt{f_3}\, d_{ij} - 2 f_3 z_a - f_2} \tag{23}$$

The x and y points may be found from equation (5).

To obtain these results, the floating point operations in table 1 must be performed.

EFFECT OF NEAR FIELD GRID SIZE ON ACCURACY

An attempt was made to determine the effect of the near field grid size on the accuracy. The accuracy should worsen with larger grid sizes because the partial derivatives are determined from the phases of the nearest neighbors, and, in the presence of distortion, the calculated partial derivatives differs from the true local partial derivative for large grid sizes.

The analysis uses as a reflector model a parabola with cosine distortion in one of the axes. The surface is described by the equation

$$z = \frac{x^2 + y^2}{2f} + \delta\cos\left(2\pi\frac{y_{max} - y}{y_{max} - y_{min}}\right)$$

with δ ranging from 0 to 0.007 meters.

The average error as a function of number of elements in the model antenna is plotted in figure 2 for several levels of distortion. As can be seen from the figure, the algorithm error varies nearly linearly with input distortion and is not greatly influenced by the element size. The invocations of RSE, and that the RSE algorithm failed for the large values of distortion for large grid sizes (evidenced by the curves which terminate early on the left of the plot). The RSE algorithm determined that some pairs of input phases were increasing or decreasing, wrote a message to the screen, and terminated the calculations for these cases.

For successful invocation of RSE, however, the accuracy of the result is not heavily influenced by grid size.

This is not to say that the number of grid points is not an important parameter. If the number of grid points is small, the position of the reflector surface will be known at only a few points.

EFFECT OF PHASE MEASUREMENT ERROR ON ACCURACY

An important performance measurement for any algorithm that uses real measurements is the effect of errors in the measurements on the accuracy of the results. In order to determine the output of the program to input noise, Gaussian noise of various amplitudes was added to the input phase measurements. The results are shown in figure 3.

The nonlinearity in the average output error as a function of input noise is apparently because the major effect on the error at low input noise levels is due to truncation errors in the algorithm. At higher levels, the error due to noise is the dominant part.

APPENDIX I

AUXILIARY PROGRAMS

This appendix contains description and listings of auxiliary programs used in the analysis. These programs include:

vary: A program which uses all of the subroutines and functions below to produce error data based on variations in grid spacing, phase accuracy, and frequency.

rnfgp: A subroutine which generates near field phase data on regular grid points based on user-supplied reflector distortion.

fixphi: A subroutine which accepts the near field phase data supplied from rnfpg or from actual phase measurements and eliminates discontinuities which normally occur either at π and at $-\pi$ or at 0 and 2π. The input range is either $(-\pi,\pi)$ or $(0.,\pi)$ and the output range is unlimited.

rseerr: A subroutine which includes the RSE algorithm and uses a user-supplied reflector distortion function (also supplied in rnfpg) to determine the error of the RSE algorithm.

reffun: A function which is supplied to rnfpg and rseerr which returns the z position of a simulated reflector surface given the x and y coordinates. The partial derivatives with respect to x, y, and z are also given.

Subroutine RNFPG.

RNFGP theory.

Program rnfpg is an iterative procedure used to determine a point on a reflector, x_r, y_r, z_r, which will reflect incident rays from a known feed point to a known point in a near field plane. Only two of the variables are required; the third can be determined because it is known that the point lies on the reflector surface. The projection of the geometry in the y=0 plane is shown in figure 4.

The procedure begins with the selection of a starting value for the solution. The assumption is made that the x and y coordinates on the reflector are close to the x and y coordinates in the near field plane. About this point, four rays are used to probe the location of the exact solution. These rays originate form the feed location, intersect the reflector at four points arranged about the assumed solution (see figure 5).

Each of the rays is bounced off the reflector, following the laws of geometric optics. The intersection of the resulting ray and the near field plane is then calculated. These projections and the target grid point are shown in figure 5.

From these points, a new value of x_r, y_r is selected by linear interpolation:

$$x_r' = x_r - \delta + 2 * \delta * (x_a - x_{a3}) / (x_{a1} - x_{a3})$$

$$y_r' = y_r - \delta + 2 * \delta * (y_a - y_{a4}) / (y_{a2} - y_{a4})$$

Up to this point, we have not discussed the selection of δ. Obviously, to converge to a solution, δ must decrease with each iteration. The speed of convergence to a solution is directly related to the rate at which δ decreases. If δ is decreased too quickly, however, x_r,y_r may fall outside of the bundle of rays. This is usually not fatal, if the function is well behaved, but if it happens too often, divergence may occur.

The parameter which will determine how quickly δ can be reduced is related to the linearity of the mapping from the reflector position to the near field position. If the mapping is totally linear (e.g., no distortion), only one iteration is necessary. The more non-linearity, the more iterations will be needed.

One rough indication of linearity can be obtained from the points already calculated. If the transformation were totally linear, the distance in the x axis from x_{a1} to x_{a2} would be the same as the distance in the x direction x_{a4} to x_{a3}. Using the difference between the two distances divided by the total distance from x_{a1} to x_{a3} as the measure of non-linearity, we have:

$$skew_x = |(x_{a1} + x_{a3} - x_{a2} - x_{a4}) / (x_{a1} - x_{a3})|$$

A similar measure can be made in the y direction.

$$skew_y = |(y_{a1} + y_{a3} - y_{a2} - y_{a4}) / (y_{a1} - y_{a3})|$$

The program uses the non-linearity as the basis for the decrease in the spread of the packet of rays. The program starts with a δ of 10% of the largest dimension of the antenna. After the first iteration, δ is calculated from

$$\delta_{n+1} = \delta_n * (skew')$$

Where

$$skew' = \max(0.1,\ \min(0.9, \max(skew_x, skew_y))$$

The maximum value of $skew_x$ or $skew_y$ is used, so long as that value is greater than 0.1 and less than 0.9.

After the new value for δ is determined, a new bundle of rays is launched. A test is made to determine if the bundle of rays does enclose the solution. This can be determined by examining the intersection of the rays with the near field plane. x_{a3} should be less than x_a and x_{a1} should be greater than x_a, with similar requirements in the y axis. If one of these conditions is not met, an informative message is sent to the console and the value of δ is automatically multiplied by 2. The iteration then continues.

After each iteration, a test is made to determine if the error in the near field plane has converged to within the maximum error used in the program's calling argument. If it has, the value is printed out to a file and the program continues with the next point in the near field plane.

Subroutine fixphi.

Subroutine fixphi takes as its input the results of rnfpg of near field phase measurements from an antenna facility and transforms the relative phase measurements ($-\pi < ø < \pi$) or ($0 < ø < 2\pi$) to measurements which can be used to determine phase length. For example, if the following line were input into the program:

1.0	1.5	2.5	0.5	1.5

the program would convert the line to:

1.0	1.5	2.5	3.64159	4.64159

The program works by first examining the data to see if it meets one of the conditions: ($-\pi < ø < \pi$) or ($0 < ø < 2\pi$). If it meets neither condition, an error message is displayed on the console and the program terminates. If either condition is met, the program continues.

The program continues by rewriting the input file and reading input while processing and printing the output. The first input value is special in that its value is always preserved. After the first value, each measurement is examined to determine if it appears that the data has gone through a transition from $-\pi$ to π or from 2π to 0.

RNFPG performance.

There were two figures of merit of the routine which were traded against each other to obtain maximum performance: computational speed and accuracy. Because of the iterative nature of the algorithm, additional accuracy can always be obtained by allowing more time for the computations, up to the precision limits of the machine. Double precision numbers were used as the default for the algorithm to limit the effect of machine precision on the output.

Required accuracy is an argument in the invocation of RNFPG, and the algorithm will execute until that accuracy is obtained.

In order to determine the effect of accuracy on expected execution time. number of iterations was plotted as a function of required accuracy. The results are shown in figure 6.

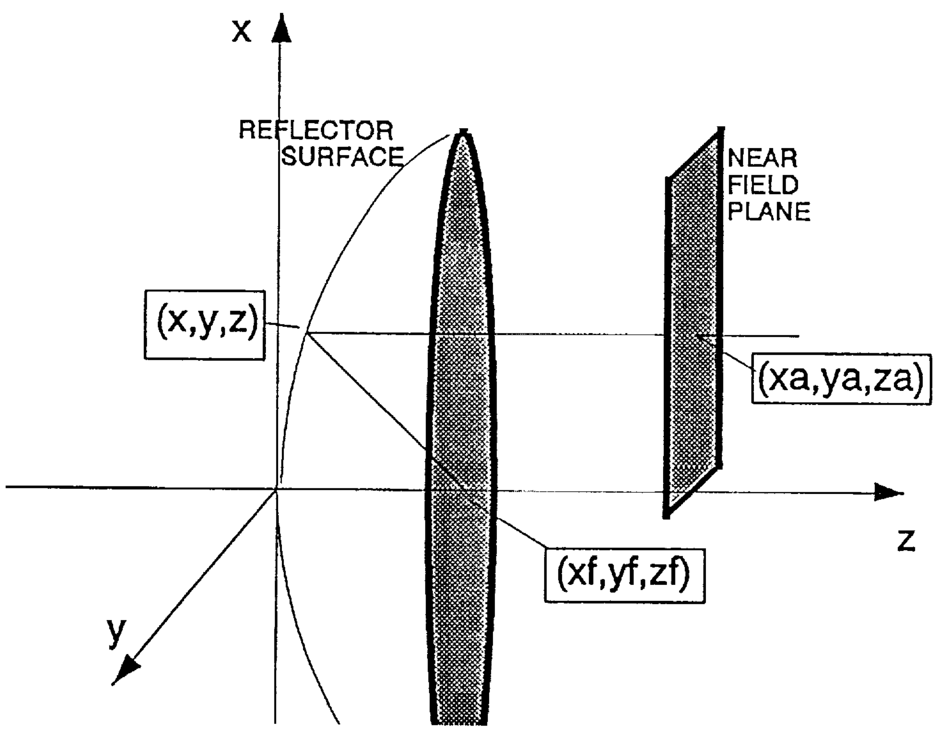

Figure 1. Geometry used to develop the RSE algorithm

Table1. Operations and timing. Times shown are for a Motorola 68881 co-processor operating at 40 mHz. Time is given in microseconds.

	ADD	SUBTRACT	MULTIPLY	DIVIDE	SQ.ROOT	TOTAL
$\partial\phi/\partial x$		2		1		3.0
$\partial\phi/\partial x$		2		1		3.0
m_x				1		1.0
m_y				1		1.0
m_z		2	2	1	1	6.0
C_1				1		1.0
C_2				1		1.0
d_1	1	1	1			3.0
d_2	1	1	1			3.0
f_1	2		3			5.0
f_2	1	1	3			5.0
f_3	2		2			4.0
z	1	4	9	1	2	17.0
x	1	1	1	1		4.0
Total	9.0	14.0	22.0	9.0	3.0	57.0
Cycle/oper	51	51	71	103	107	
Total cycles	459.0	714.0	1,562.0	927.0	321.0	2,100.0
Time/cycle	0.025	0.025	0.025	0.025	0.025	
Total time	11.48	17.85	0.00	23.18	0.00	52.51

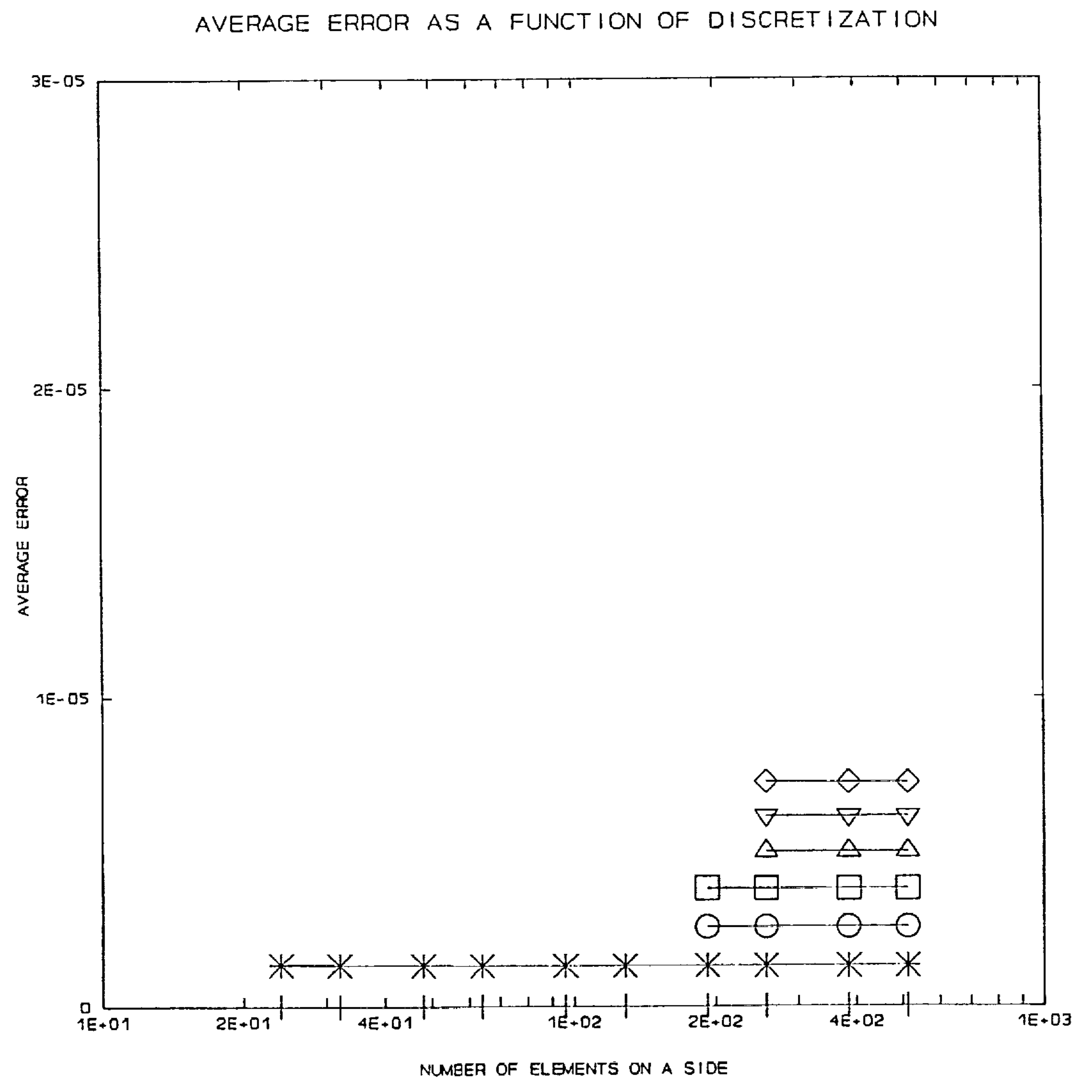

Figure 2. Error vs. Discretization. Curves are (from bottom) for 1, 2, 3, 4, 5, and 6mm distortion.

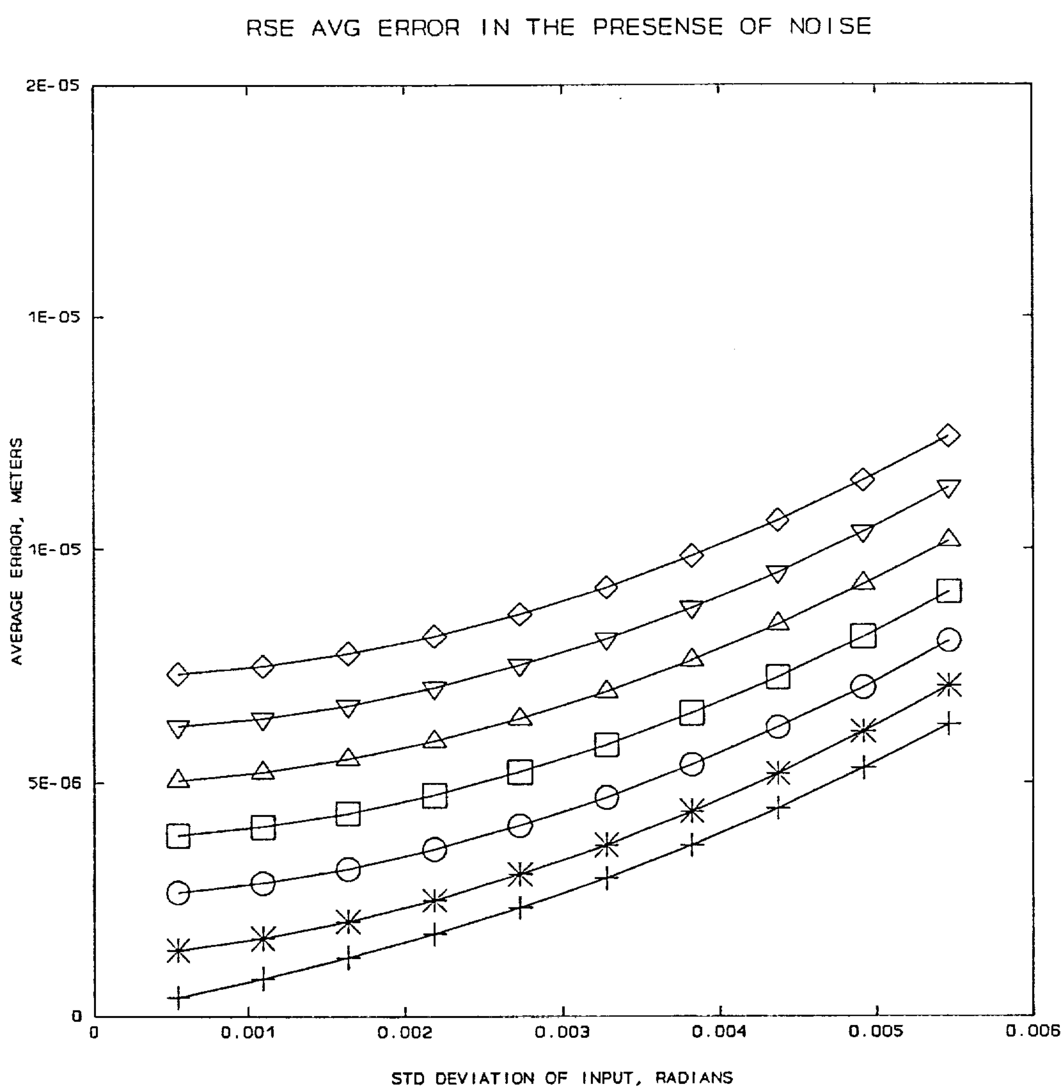

Figure 3 RSE error in the presence of noise. Curves are (from bottom) for 1, 2, 3, 4, 5, and 6mm distortion

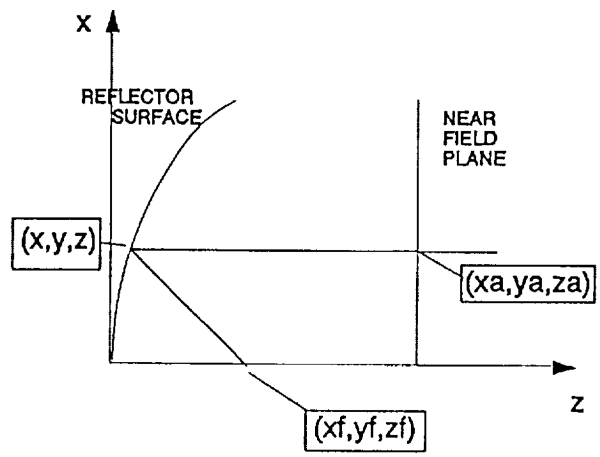

Figure 4. Problem geometry in the plane y=0

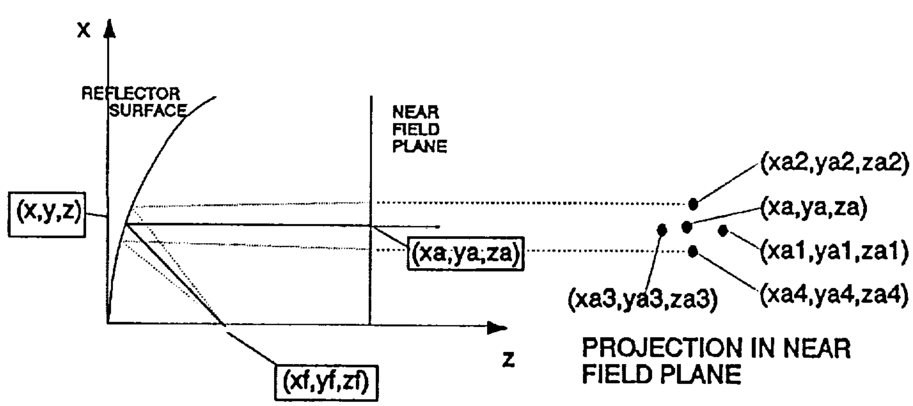

Figure 5.rnfpg ray tracing

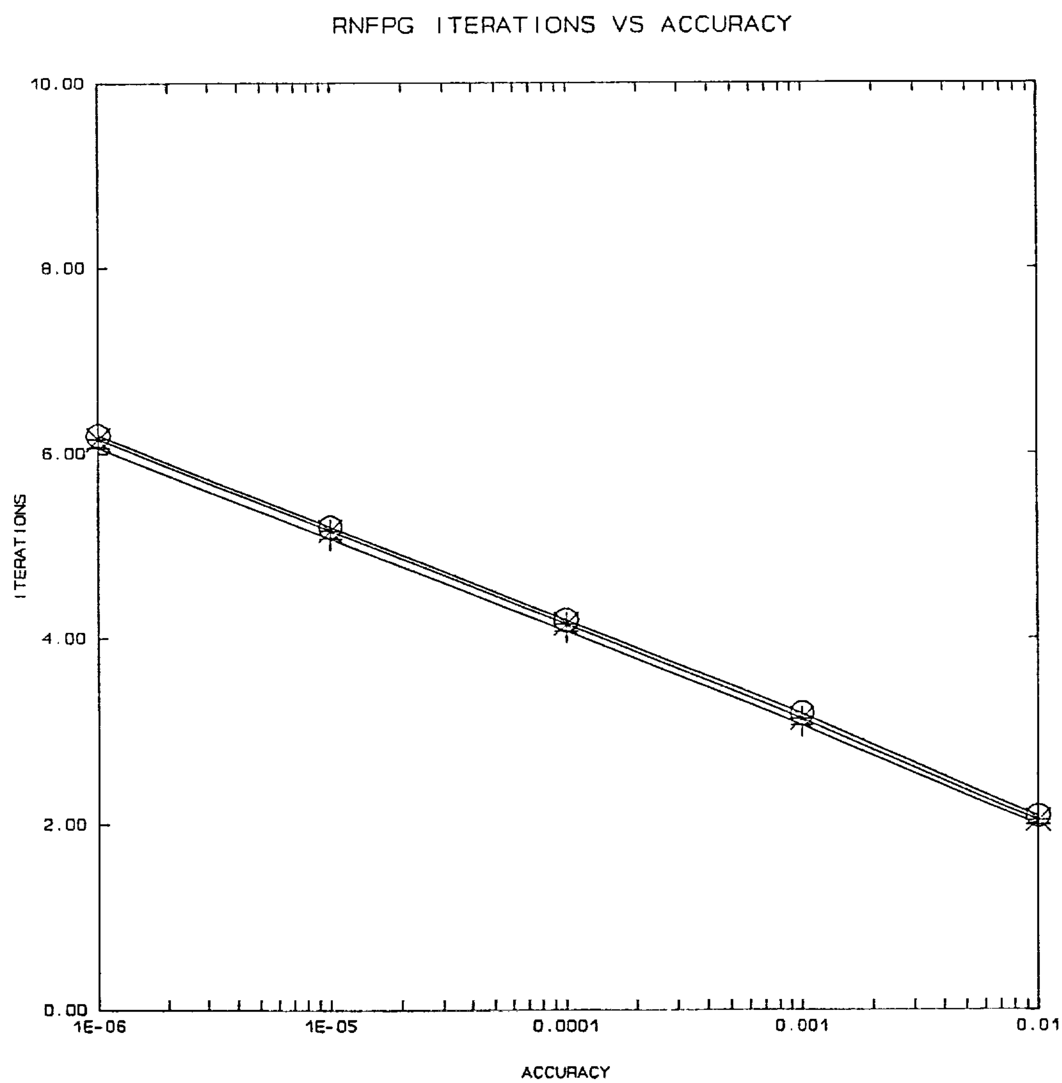

Figure 6. rnfpg iterations versus accuracy. Curves are (from bottom) for 1, 2, and 3mm distortion.

COMPUTER PROGRAM LISTINGS

The programs used with this algorithm are listed on the following pages.

vary.f

```
c       vary.f compiles into a variety of programs depending on
c       the mode of compilation
c       Compilation must be done with the c preprocessor cpp.
c       One of the following may be defined
c
c               (none)          defaults to vary number of cells
c               ERROR           varies the allowable error of rnfpg
c


        program vary
        implicit double precision (a-h)
        implicit double precision (o-z)
        double precision xamp(8),yamp(8)
      common /partial/ pdfdx,pdfdy,pdfdz
      common /distort/ del,omega,xampl,yampl
      integer type
      common /phys/ f,xf,yf,zf,zp,xmin,xmax,ymin,ymax,freq,type
      real maxerr(20,20),avgerr(20,20),rmserr(20,20)
      integer error(20,20)
        integer npts(8),nerrors(8)
#define MAX_CURVES 8
#define MAX_POINTS 50
#ifdef ERROR
        character*80 filename,pltttl,xttl,yttl,zttl
        real xdata(50,8)
        real ydata(50,8)
        real avgitr
        common /perf/ avgitr
#else  ERROR
        real xdata(50,8)
        real adata(50,8)
        real rdata(50,8)
        real mdata(50,8)
        real sigma(50,8)
        real inacc(50,8)
#endif ERROR
      common /plot/ idist,igrid,
     1 maxerr(20,20),avgerr(20,20),rmserr(20,20),error(20,20)
        data xamp /0.00, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07/
        data yamp /0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00/


      fmaxerr=.000001

      ymax=1.1
      ymin=0.1
      xmax=0.5
      xmin=-0.5
      type = 1

#ifdef ERROR
      infile1=513
#else
      infile1=512
      infile2=(infile1*3)/4
      idiv = ifix(log(float(infile1))/log(2.0)-2.5)
      ngrids =2*ifix(log(float(infile1))/log(2.0)-2.5)
      infile1=infile1+1
      infile2=infile2+1
#endif ERROR

      f = 1.0

      xf = 0.0
```

vary.f

```
      yf = 0.0
      zf = 1.0

      zp = 1.0

      periods = 1.0
      omega = 2.0 * 3.14159265 * periods / (ymax-ymin)

      freq=30500000000.
c     Wave number
      k = 2.0 * 3.141592650 / (300000000/freq)


        open(7,file="results.dat")

        open(99,file="contour.dat")

        min_dist=1
        max_dist=7

        do 10 i=min_dist,max_dist
        idist=i
        xampl=xamp(i)
        yampl=yamp(i)
        write(7,105)
 105    format('avgerr.wpg')
        write(7,106) xampl,yampl
 106    format(' rnfpg error analysis'/
     1  ' amplitude of (x,y) distortion = ('
     2  ,d18.10,',',d18.10,')' )

        xlambda = 0.1
        t = xampl
        del = t * xlambda

#ifdef ERROR
        do 20 ierror=1,5
                ferrmax=.000001*(10**(ierror-1))
                write(6,*)"max err = ",ferrmax
                write(7,*)"max err = ",ferrmax
                open(4,file="ph1.dat",form='UNFORMATTED')
                call rnfpg(infile1,ferrmax)
                close(4)
                xdata(ierror,i) = ferrmax
                ydata(ierror,i) = avgitr
   20           continue
                npts(i) = ierror-1
   10   continue
                filename = "iter2.wpg"
                pltttl = "RNFPG ITERATIONS VS ACCURACY"
                xttl = "ACCURACY"
                yttl = "ITERATIONS"
                zttl = "DISTORTION"
       call plotwpg(filename,pltttl,xttl,yttl,zttl,
     1 .000001,.01,.001,0.,10.,2.,1,0,
     2 max_dist-min_dist+1,
     2 npts,xdata,ydata)
#else
        open(4,file="ph1.dat",FORM='unformatted')
        call rnfpg(infile1,fmaxerr)
        close(4)
        open(3,file="ph1.dat",FORM='unformatted')
        open(4,file="ph1f.dat",FORM='unformatted')
        call fixphi(infile1)
        close(4)
        close(3)
```

vary.f

```
      open(4,file="ph2.dat",FORM='unformatted')
      call rnfpg(infile2,fmaxerr)
      close(4)
      open(3,file="ph2.dat",FORM='unformatted')
      open(4,file="ph2f.dat",FORM='unformatted')
      call fixphi(infile2)
      close(4)
      close(3)

      igrid=0
      do 20 j=1,idiv
              igrid=igrid+1
              xdata(igrid,i) =infile1/(2**(j-1))
              open(4,file='ph1f.dat',FORM='unformatted')
              call rsegrid(infile1,1+(infile1-1)/(2**(j-1)))
              adata(igrid,i)=avgerr(igrid,i)
              mdata(igrid,i)=maxerr(igrid,i)
              rdata(igrid,i)=rmserr(igrid,i)
              if(error(igrid,i).ne.0)go to 21
              close(4)
              igrid=igrid+1
              xdata(igrid,i)=infile2/(2**(j-1))
              open(4,file='ph2f.dat',FORM='unformatted')
              call rsegrid(infile2,1+(infile2-1)/(2**(j-1)))
              adata(igrid,i)=avgerr(igrid,i)
              mdata(igrid,i)=maxerr(igrid,i)
              rdata(igrid,i)=rmserr(igrid,i)
              if(error(igrid,i).ne.0)go to 21
              close(4)
20            continue
21    npts(i) = j-1

      do 30 j=1,10
              nerrors(j) = 10
        sigma(j,i)=((300000000./freq)/(360.*.01*5.)*float(j))
              open(4,file='ph1f.dat',FORM='unformatted')
              rewind(4)
              call rseerr(infile1,sigma(j,i),inacc(j,i),ierrflg)
              close(4)
 30   continue

10    continue


     call plotwpg("maxerr.wpg",
    1 "MAXIMUM ERROR AS A FUNCTION OF DISCRETIZATION",
    2 "NUMBER OF ELEMENTS ON A SIDE","MAXIMUM ERROR","DISTORTION",
    3 10.,1000.,10.,0.,.00003,.00001,1,0,
    4 max_dist-min_dist+1,
    5 npts,xdata,mdata)

      call plotwpg("avgerr.wpg",
    1 "AVERAGE ERROR AS A FUNCTION OF DISCRETIZATION",
    2 "NUMBER OF ELEMENTS ON A SIDE","AVERAGE ERROR","DISTORTION",
    3 10.,1000.,10.,0.,.00003,.00001,1,0,
    4 max_dist-min_dist+1,
    5 npts,xdata,adata)

      call plotwpg("rmserr.wpg",
    1 "ROOT MEAN SQUARE ERROR AS A FUNCTION OF DISCRETIZATION",
    2 "NUMBER OF ELEMENTS ON A SIDE","RMS ERROR","DISTORTION",
    1 10.,1000.,10.,0.,.00003,.00001,1,0,
    2 max_dist-min_dist+1,
    3 npts,xdata,rdata)

      call plotwpg("randerr.wpg",
```

vary.f

```
     1 "RSE AVG ERROR IN THE PRESENSE OF NOISE",
     2 "STD DEVIATION OF INPUT, RADIANS","AVERAGE ERROR, METERS",zttl,
     3 0.,.006,.001,0.,.00002,.000005,0,0,
     4 max_dist-min_dist+1,
     5 nerrors,sigma,inacc)

#endif  ERROR
        stop
        end
```

reffun.f

```
c     Calculate the z coordinate of the reflector surface

      real*8 function reffun(x,y)

      implicit real*8 (a-h)
      implicit real*8 (o-z)
      real argx, sinfunx, cosfunx
      real argy, sinfuny, cosfuny
      integer type
      real*8 x,y,f,del,omega,ymax
      real*8 temp

      common /partial/ pdfdx,pdfdy,pdfdz
      common /distort/ del,omega
      common /phys/ f,xf,yf,zf,zp,xmin,xmax,ymin,ymax,freq,type

c     x and y are the x and y positions of the point
c     del is the distortion amplitude factor
c     omega is the distortion wave number
c     ymax is the maximum y value


      if(type .eq. 1) then

c     This type is for an antenna with sinusoidal distortion that varies
c     only with the y variable

          argy = omega * (ymax-y)
              sinfuny = sin(argy)
              cosfuny = cos(argy)
          temp = -0.5/f
          pdfdx = x*temp
          pdfdy = y*temp - del * omega * sinfuny
          pdfdz = 1.0
          reffun = (x**2+y**2)/(4.0*f)+del*cosfuny
          return
      else if (type .eq. 2 ) then

c     This type is for an antenna with sinusoidal distortion that varies
c     only with the x variable

          argx = omega * (xmax-x)
              sinfunx = sin(argx)
              cosfunx = cos(argx)
          temp = -0.5/f
          pdfdx = x*temp - del * omega * sinfunx
          pdfdy = y*temp
          pdfdz = 1.0
          reffun = (x**2+y**2)/(4.0*f)+del*cosfunx
          return
      else if (type .eq. 3 ) then

c     This type is for an antenna with sinusoidal distortion that varies
c     with both the x and the y variable

          argx = omega * (xmax-x)
          argy = omega * (ymax-y)
              sinfunx = sin(argx)
              cosfunx = cos(argx)
          temp = -0.5/f
          pdfdx = x*temp - del * omega * sinfunx
          pdfdy = y*temp - del * omega * sinfuny
          pdfdz = 1.0
          reffun = (x**2+y**2)/(4.0*f)+del*cosfunx +del*cosfuny
          return
      endif
      end
```

rnfpg.f

```
      subroutine rnfpg(nphasegp,ferrmax)

c     The purpose of this program is to detect an antenna
c     reflector surface from the near field phase distribution.
c     The near field phase distribution is defined on an nxn
c     rectangular grid system.
c     Feeder location(xf,yf,zf), value of lambda, diameter of
c     the reflector aperture(d)
c*****************************************************************

      implicit real*8 (a-h)
      implicit real*8 (o-z)
      real*8 xa, ya
          real avgitr
      common /partial/ pdfdx,pdfdy,pdfdz
      integer type
      common /phys/ f,xf,yf,zf,zp,xmin,xmax,ymin,ymax,freq,type
          common /perf/ avgitr
      dimension phi(1026)
      open(11,file='res')

      pi = 3.1415926

c     Diameter of the region of interest on the reflector
      d=1.
c     Near field grid spacing
      delnf = d/(nphasegp-1)

        k = 2*3.1415926 /(300000000. / freq)

      fctrmin = 0.1
      fctrmax = 0.9

      nfirst=1
                write(6,998)ferrmax
  998   format("max err =",g8.3)

                nloops=0

c     Scan through the x axis of the near field
      do 10 i=1,nphasegp

c               call tick(i)
c           Scan through the y axis of the near field
            do 20 j=1,nphasegp

c                 Define the x,y,z  coordinates in the near field
                  xa = xmin + delnf * (i-1)
                  ya = ymin + delnf * (j-1)
                  za = zp

c                 set up for search in reflector plane
                  xr=xa
                  yr=ya
                  delta=dmax1(dabs(xmax-xmin),dabs(ymax-ymin))/10.

c                 reflector plane iteration loop
c
 601              continue
                nloops=nloops+1
c
c                 launch four rays
                  call qray(xr+delta,yr,xa1,ya1)
                  call qray(xr,yr+delta,xa2,ya2)
                  call qray(xr-delta,yr,xa3,ya3)
                  call qray(xr,yr-delta,xa4,ya4)
```

```
                    if(xa3.gt.xa)goto 649
                    if(xa.gt.xa1)goto 649
                    if(ya4.gt.ya)goto 649
                    if(ya.gt.ya2)goto 649
                    go to 650

 649                continue
                    write(6,648)
 648                format(' got outside of bundle of rays')
                    delta=delta*2
                    goto 601

 650                continue
                    if(xa1.ne.xa3)
     1                 xr=2*delta/(xa1-xa3)*(xa-xa3) + xr - delta
                    if(ya2.ne.ya4)
     1                 yr=2*delta/(ya2-ya4)*(ya-ya4) + yr - delta
 652                continue


                    skewx = 0
                    skewy = 0
                    if((xa1 - xa3) .eq. 0)goto 680
                       skewx = dabs(((xa1+xa3)-(xa2+xa4))/(xa1-xa3))
680                 if((ya2 - ya4) .eq. 0)goto 681
                       skewy = dabs(((ya2+ya4)-(ya1+ya3))/(ya2-ya4))
681                 factor=2*dmax1(skewx,skewy)
                    factor=dmax1(fctrmin,factor)
                    factor=dmin1(fctrmax,factor)

                    delta=delta*factor
                    if((abs((xa1-xa3)*pdfdxf)+abs((ya2-ya4)*pdfdy))
     1          .gt.ferrmax) goto 601
                    zr=reffun(xr,yr)
                    dist=dsqrt((xf-xr)**2+(yf-yr)**2+(zf-zr)**2)+
     1                     dsqrt((xa-xr)**2+(ya-yr)**2+(za-zr)**2)
                    phi(j)=k*dist
                    nphi=phi(j)/(2*pi)
                    phi(j)=phi(j)-nphi*2*pi
                    if(nfirst.ne.1)goto 200
                    write(4)dist,phi(j)
199                 format(f14.8/f14.8)
                    nfirst=0
200                 continue

  20                continue
             write(4)(phi(j),j=1,nphasegp)
 555         format(1026f14.8)
  10         continue
      close(11)
          avgitr = float(nloops)/float(nphasegp**2)
          write(7,997)avgitr
          write(6,997)avgitr
 997  format("avg iter = "f8.3)
      return
      end


      subroutine qray(xr,yr,xa,ya)
      implicit real*8 (a-h)
      implicit real*8 (o-z)
      common /partial/ pdfdx,pdfdy,pdfdz
      integer type
      common /phys/ f,xf,yf,zf,zp,xmin,xmax,ymin,ymax,freq,type
```

rnfpg.f

```
      real*8 l1x, l2x, l1y, l2y, l1z, l2z
      zr = reffun(xr,yr)
      l1x = xf - xr
      l1y = yf - yr
      l1z = zf - zr
      absnrml = pdfdx**2 + pdfdy**2 + pdfdz**2
      r = 2.0*(pdfdx * l1x + pdfdy * l1y + pdfdz * l1z)/absnrml
      l2x = l1x - r * pdfdx
      l2y = l1y - r * pdfdy
      l2z = l1z - r * pdfdz
      quick = (zp-zr)/l2z
      xa = quick * l2x + xr
      ya = quick * l2y + yr
      return
      end
```

fixphi.f

```
      subroutine fixphi(ninfile)
      implicit double precision (a-h)
      implicit double precision (o-z)
      double precision phi(1026)

      pi = 3.14159265
      twopi = 2*3.14159265

      read(3)dref,phiref
      write(4)dref,phiref
  199 format(f14.8/f14.8)
        ifirst = 1
        irow = 0

        do 10 i=1,ninfile
                read(3)(phi(n),n=1,ninfile)
                if(ifirst .eq. 1) phirow = phi(1)
                ifirst = 0
                philast = phirow
                if((phi(1)-phirow) .lt. (-1*pi)) then
                        irow = irow + 1
                        philast=phi(1)
                        phirow=phi(1)
                else if((phi(1)-phirow) .gt. pi) then
                        irow = irow - 1
                        philast=phi(1)
                        phirow=phi(1)
                endif
                icol = irow
                do 20 j=1,ninfile
                        if((phi(j)-philast) .lt. (-1*pi)) then
                                icol = icol + 1
                        else if((phi(j)-philast) .gt. pi) then
                                icol = icol - 1
                        endif
                        philast=phi(j)
                        phi(j) = phi(j) + icol*twopi
   20                   continue
                write(4)(phi(n),n=1,ninfile)
   10           continue
    7 format(1026f14.0)
    8 format(1026f14.8)
      return
      end
```

rsegrid.f

```
      subroutine rsegrid(ninfile,ntouse)
c
c     the purpose of this program is to detect an antenna reflector
c     surface from the near field phase distribution.  The near field
c     phase distribution is defined on an nxn rectangular grid system
c
c     surface detection
      implicit double precision (a-h)
      implicit double precision (o-z)
      double precision lambda,mx,my,mz,k
      double precision phi(3,1026)
      double precision dummy(1026)
        real*8 reffun
        integer contour(1026)
      common /partial/ pdfdx,pdfdy,pdfdz
      common /distort/ del,omega,xampl,yampl
      integer type
      common /phys/ f,xf,yf,zf,zp,xmin,xmax,ymin,ymax,freq,type
      real maxerr(20,20),avgerr(20,20),rmserr(20,20)
      integer error(20,20)
      common /plot/ idist,igrid,
     1  maxerr(20,20),avgerr(20,20),rmserr(20,20),error(20,20)
        character*4 comment

        error(igrid,idist)=0
        comment = '    '
        errmax = 0.
        sumerr = 0.
        sumsq = 0.
        nsum = 0

        if(ntouse.lt.100) write(99,801)ntouse,idist
 801    format(' e',i4.4,i2.2,'.dat')
                if(ntouse.lt.100) write(99,802)ntouse,del
 802    format(' ALGORITHM ERROR WITH ',I4.4,' GRIDS AND ',
     1  F9.3,' MM MAX SIN DISTORTION')

                if(ntouse.lt.100) write(99,803)1,'err<.00003'
                if(ntouse.lt.100) write(99,803)2,'err<.00002'
                if(ntouse.lt.100) write(99,803)3,'err<.00001'
                if(ntouse.lt.100) write(99,803)4,'err<.00000'
                if(ntouse.lt.100) write(99,803)5,'err>.00000'
                if(ntouse.lt.100) write(99,803)6,'err>.00001'
                if(ntouse.lt.100) write(99,803)7,'err>.00002'
                if(ntouse.lt.100) write(99,803)8,'err>.00003'
 803    format(' ',i3,' ',a)

                if(ntouse.lt.100) write(99,804)ntouse-2
 804    format(' ',i3)

        za = zp

       pi = 3.14159265

       read(4)dref,phiref

       lambda=300000000./freq
       k = (2*pi)/lambda
       delx=(xmax-xmin)/(ntouse-1)
       dely=(ymax-ymin)/(ntouse-1)

       ntoskip=(ninfile/(ntouse-1))-1
         write(6,*)' ntoskip = ',ntoskip

c
c      read in two lines of input to start the process
```

```
      if(ntoskip.eq.0) then
      read(4)(phi(2,n),n=1,ntouse)
      else
      read(4)phi(2,1),
     1  ((dummy(iskip),iskip=1,(ntoskip)),phi(2,n),n=2,ntouse)
      endif

        if(ntoskip.ne.0) then
        do 880 iskip=1,(ntoskip)
 880    read(4)dummy(1)
        endif

      if(ntoskip.eq.0) then
      read(4)(phi(3,n),n=1,ntouse)
      else
      read(4)phi(3,1),
     1  ((dummy(iskip),iskip=1,(ntoskip)),phi(3,n),n=2,ntouse)
      endif

        if(ntoskip.ne.0) then
        do 881 iskip=1,ntoskip
 881    read(4)dummy(1)
        endif

      do 400 n=1,ntouse
      phi(2,n)=phi(2,n)-phiref
  400 phi(3,n)=phi(3,n)-phiref
      do 10 i=2,ntouse-1
c      write(7,*)sumerr
      xa=xmin+(i-1)*delx
c           prepare to read in a new row
            do 11 i1=1,ntouse
                  phi(1,i1)=phi(2,i1)
   11             phi(2,i1)=phi(3,i1)


      if(ntoskip.eq.0) then
              read(4)(phi(3,n),n=1,ntouse)
      else
      read(4)phi(3,1),
     1  ((dummy(iskip),iskip=1,(ntoskip)),phi(3,n),n=2,ntouse)
      endif


              if(ntoskip.ne.0.and.i.ne.(ntouse-1)) then
              do 882 iskip=1,ntoskip
 882          read(4)dummy(1)
              endif

            do 401 n=1,ntouse
  401       phi(3,n)=phi(3,n)-phiref
            do 20 j=2,ntouse-1
            ya=ymin+(j-1)*dely
            d=dref+1/k*(phi(2,j))
c           dref and phiref are measured quantities

            dphidx=phi(3,j)-phi(1,j)
            if(dphidx.gt.pi)dphidx=dphidx-2*pi
            if(dphidx.lt.(-1*pi))dphidx=dphidx+2*pi
            if((dphidx.lt.-1).or.(dphidx.gt.1))then
                if(error(igrid,idist).eq.0)then
                        error(igrid,idist)=1
                        comment=' ?? '
                        write(6,570)
```

```
                      endif
              endif
 570      format(' grid size too large')
          dphidx=dphidx/(2*delx)

          dphidy=phi(2,j+1)-phi(2,j-1)
          if(dphidy.gt.pi)dphidy=dphidy-2*pi
          if(dphidy.lt.(-1*pi))dphidy=dphidy+2*pi
          if((dphidy.lt.-1).or.(dphidy.gt.1))then
              if(error(igrid,idist).eq.0)then
                      error(igrid,idist)=1
                      comment=' ?? '
                      write(6,570)
                      endif
              endif
          dphidy=dphidy/(2*dely)
          mx = dphidx/k
          my = dphidy/k
          mz = dsqrt (1.0-mx**2-my**2)
          c1 = mx/mz
          c2 = my/mz
          d1 = -c1*za + xa - xf
          d2 = -c2*za + ya - yf
          f1 = d1**2 + d2**2 + zf**2
          f2 = 2.0 * (d1*c1 + d2*c2 - zf)
          f3 = c1**2 + c2**2 + 1.0

          znu = -f1 + (f3*za**2 - 2.0 * dsqrt(f3)*d*za
     1          + d**2)
          znd = (f2 + f3*2.0*za - 2.0 * dsqrt(f3)*d)
          z=znu/znd
          x=c1*(z-za) + xa
          y=c2*(z-za) + ya
c          write(6,991)x,y,z
 991      format(" x= ",f18.10,", y= ",f18.10, ", z= ",f18.10)
              err = z - reffun(x,y)
              if(err.gt.0)then
                      contour(j)=5
                      if(err.gt..0001)contour(j)=6
                      if(err.gt..0002)contour(j)=7
                      if(err.gt..0003)contour(j)=8
              else
                      contour(j)=4
                      if(err.lt.-.0001)contour(j)=3
                      if(err.lt.-.0002)contour(j)=2
                      if(err.lt.-.0003)contour(j)=1
              endif
           derror= dabs(err)
c          write(6,556)x,y,z,derror
 556      format(5f14.8)
          if(derror .gt. errmax) errmax=derror
          sumerr = sumerr + derror
          sumsq  = sumsq  + derror**2
          nsum = nsum + 1
  20          continue
              if(ntouse.lt.100) write(99,819)(contour(j), j=2,ntouse-1)
 819  format(1026i1);
  10      continue
    rms = dsqrt(sumsq/nsum)
    average = sumerr/nsum
    maxerr(igrid,idist)=errmax
    avgerr(igrid,idist)=average
    rmserr(igrid,idist)=rms
    write(7,557)errmax,average,rms
 557 format(' maxerr = ', d14.8,
   1 ' sumerr/n = ', d14.8,
   2 ' sumsq/n =  ', d14.8)
    write(7,558)ntouse,average
 558 format(' ',i4,'  ',f14.8)
    return
    end
```

rseerr.f

```
      subroutine rseerr(ninfile,sigma,inacc,error)
c
c
c     surface detection
      implicit double precision (a-h)
      implicit double precision (o-z)
      real sigma, inacc,gasdev
          integer error
      double precision lambda,mx,my,mz,k
      double precision phi(3,1026)
        real*8 reffun
      common /partial/ pdfdx,pdfdy,pdfdz
      common /distort/ del,omega,xampl,yampl
      integer type
      common /phys/ f,xf,yf,zf,zp,xmin,xmax,ymin,ymax,freq,type
          logical exist,opened
          integer ios,nr
          integer arg1,arg2,arg3,arg4,arg5


    1 continue
        error=0
        errmax = 0.
        sumerr = 0.
        sumsq = 0.
                nsum = 0

                za = zp


      pi = 3.14159265

      read(4,end=699,err=698)dref,phiref

      lambda=300000000./freq
      k = (2*pi)/lambda
      delx=(xmax-xmin)/(ninfile-1)
      dely=(ymax-ymin)/(ninfile-1)
c
c     read in two lines of input to start the process
c
      read(4,end=699,err=698)(phi(2,n),n=1,ninfile)
      read(4,end=699,err=698)(phi(3,n),n=1,ninfile)

      do 400 n=1,ninfile
          perturb = sigma * gasdev()
      phi(2,n)=phi(2,n)-phiref + perturb
          perturb = sigma * gasdev()
          phi(3,n)=phi(3,n)-phiref + perturb
  400 continue

      do 10 i=2,ninfile-1
      xa=xmin+(i-1)*delx
c          prepare to read in a new row
           do 11 i1=1,ninfile
                 phi(1,i1)=phi(2,i1)
   11            phi(2,i1)=phi(3,i1)

                read(4,end=699,err=698)(phi(3,n),n=1,ninfile)

           do 401 n=1,ninfile
           perturb = sigma * gasdev()
           phi(3,n)=phi(3,n)-phiref + perturb
  401   continue
           do 20 j=2,ninfile-1
           ya=ymin+(j-1)*dely
```

```
          d=dref+1/k*(phi(2,j))
c         dref and phiref are measured quantities

          dphidx=phi(3,j)-phi(1,j)
          if(dphidx.gt.pi)dphidx=dphidx-2*pi
          if(dphidx.lt.(-1*pi))dphidx=dphidx+2*pi
          if((dphidx.lt.-1).or.(dphidx.gt.1))then
                    if(error.eq.0)then
                            error=1
                            write(6,510)
 510                        format('grid size too large')
                            endif
                    endif
          dphidx=dphidx/(2*delx)

          dphidy=phi(2,j+1)-phi(2,j-1)
          if(dphidy.gt.pi)dphidy=dphidy-2*pi
          if(dphidy.lt.(-1*pi))dphidy=dphidy+2*pi
          if((dphidy.lt.-1).or.(dphidy.gt.1))then
                    if(error.eq.0)then
                            error=1
                            write(6,510)
                            endif
                    endif
          dphidy=dphidy/(2*dely)
          mx = dphidx/k
          my = dphidy/k
          mz = dsqrt (1.0-mx**2-my**2)
          c1 = mx/mz
          c2 = my/mz
          d1 = -c1*za + xa - xf
          d2 = -c2*za + ya - yf
          f1 = d1**2 + d2**2 + zf**2
          f2 = 2.0 * (d1*c1 + d2*c2 - zf)
          f3 = c1**2 + c2**2 + 1.0
          znu = -f1 + (f3*za**2 - 2.0 * dsqrt(f3)*d*za
     1          + d**2)
          znd = (f2 + f3*2.0*za - 2.0 * dsqrt(f3)*d)
          z=znu/znd
          x=c1*(z-za) + xa
          y=c2*(z-za) + ya
              err = z - reffun(x,y)
           derror= dabs(err)
          if(derror .gt. errmax) errmax=derror
          sumerr = sumerr + derror
          sumsq  = sumsq  + derror**2
          nsum = nsum + 1
 20             continue
 819  format(1026i1);
 10       continue
      rms = dsqrt(sumsq/nsum)
      average = sumerr/nsum
          inacc = average
      write(7,555)sigma,inacc,errmax,average,rms
 555  format('sigma=',g14.8,',inacc=',g14.8,',errmax=', g14.8,
     1 ',average=', d14.8,',rms= ', d14.8)
      write(6,557)sigma,inacc
 557  format('sigma=',g14.8,',inacc=',d14.8)
      write(7,558)ninfile,average
 558  format(' ',i4,'  ',f14.8)
      return
 698    write(6,*)"---error---"
            return
 699    write(6,*)"---end---"
          call ERRSNS(arg1,arg2,arg3,arg4,arg5)
          write(6,*)arg1,arg2,arg3,arg4,arg5
          inquire(4,EXIST=exist,NEXTREC=nr,IOSTAT=ios,OPENED=opened)
          write(6,*)"exist",exist,"iostat",ios,"opened",opened
          write(6,*)"next record = ",nr
          write(6,*)"n=",n,",nsum=",nsum,",i=",i
          call flush(6)
          rewind (4)
          goto 1
      end
```

vctr.f

```
      program vctr
c     this program reads the coordinates of any point on the reflector
c     and determines the cooresponding coordinates on the near field
c     plane
      implicit real*8 (a-h)
      implicit real*8 (o-z)
      dimension x(9,9),y(9,9),z(9,9),k(81),l(81),xa(9,9),ya(9,9)
      real*8 l1x,l1y,l1z,l1,ul1x,ul1y,ul1z
      real*8 r,ul2x,ul2y,ul2z
      real*8 pdfdx,pdfdy,pdfdz,absnrml
      real*8 nrmlx,nrmly,nrmlz

      open(12,file='data')
      open(13,file='res',status='old')
      write(6,*) 'enter distortion factor t'
      read(5,*) t
      xlambda = 0.1
      nperiod = 1
      f = 1.0
      ymax = 1.1
      ymin = 0.1
      d = 1.
      delref = 0.125
      zp = 1.
      del = t * xlambda
      xf = 0.0
      yf = 0.0
      zf = 1.0
      pi = 3.14159265
      omega = 2.0 * pi * nperiod/(ymax-ymin)
      c= 2 * pi/xlambda
      zaa = zp
      nrefgp = d/delref + 1
      do 10 i=1,nrefgp
            do 20 j=1,nrefgp
                  read(13,3)k(i),l(j),x(i,j),xa(i,j),y(i,j),ya(i,j),z(i,j)
    3             format(2i3,5f11.7)
                  l1x = xf - x(i,j)
                  l1y = yf - y(i,j)
                  l1z = zf - z(i,j)
                  l1 = sqrt(l1x**2 + l1y**2 + l1z**2)
                  pdfdx = -x(i,j)/(2.0*f)
                  pdfdy = -y(i,j)/(2.0*f) - del*omega*sin(omega*(ymax
     1                          -y(i,j)))
                  pdfdz = 1.0
                  absnrml = sqrt(pdfdx**2 + pdfdy**2 + pdfdz**2)
                  nrmlx = pdfdx/absnrml
                  nrmly = pdfdy/absnrml
                  nrmlz = pdfdz/absnrml
                  ul1x = l1x / l1
                  ul1y = l1y / l1
                  ul1z = l1z / l1
                  r = 2.0*(nrmlx*ul1x+nrmly*ul1y+nrmlz*ul1z)
                  ul2x = ul1x - r*nrmlx
                  ul2y = ul1y - r*nrmly
                  ul2z = ul1z - r*nrmlz
c                 in original z(i,j) was z
                  xaa = (zaa-z(i,j))*(ul2x/ul2z) + x(i,j)
                  yaa = (zaa-z(i,j))*(ul2y/ul2z) + y(i,j)
                  write(12,5)k(i),l(j),x(i,j),xaa,y(i,j),yaa,z(i,j),zaa
    5             format(2i3,6f10.5)
   20             continue
   10       continue
      close(13)
      close(12)
      end
```

gasdev.f

```
      function gasdev()
c             returns a normally distributed deviate with zero mean
c             and unit variance
      double precision rand
        data iset/0/
        data gset/0.0/
      if(iset.eq.0) then
    1   v1 = 2. * rand() - 1.
        v2 = 2. * rand() - 1.
        r = v1**2 + v2**2
        if(r.ge.1)go to 1
        fac = sqrt(-2.*log(r)/r)
        gset = v1 * fac
        gasdev = v2 * fac
        iset = 1
      else
        gasdev = gset
        iset = 0
      endif
      return
        end
```

vary1k.f

```
        program vary
        implicit double precision (a-h)
        implicit double precision (o-z)
        double precision xamp(8),yamp(8)
      common /partial/ pdfdx,pdfdy,pdfdz
      common /distort/ del,omega,xampl,yampl
      integer type
      common /phys/ f,xf,yf,zf,zp,xmin,xmax,ymin,ymax,freq,type
      real maxerr(20,20),avgerr(20,20),rmserr(20,20)
      integer error(20,20)
        integer npts(8),nerrors(8)
#define MAX_CURVES 8
#define MAX_POINTS 50
c       character*80 filename,pltttl,xttl,yttl,zttl
#ifdef ERROR
        real xdata(50,8)
        real ydata(50,8)
        real avgitr
        common /perf/ avgitr
#else  ERROR
        real xdata(50,8)
        real adata(50,8)
        real rdata(50,8)
        real mdata(50,8)
        real sigma(50,8)
        real inacc(50,8)
#endif ERROR
      common /plot/ idist,igrid,
     1 maxerr(20,20),avgerr(20,20),rmserr(20,20),error(20,20)
        data xamp /0.00, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07/
        B
        data yamp /0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00/

          integer atime(3),iday,imonth,iyear
  500   format(i2.2,'/',i2.2,'/',i2.2,'  ', i2.2,':',i2.2'.'i2.2)

          call itime(atime)
          call idate(iday,imonth,iyear)
          write(6,500)imonth,iday,iyear,atime
      B
      fmaxerr=.000001

      ymax=1.1
      ymin=0.1
      xmax=0.5
      xmin=-0.5
      type = 1

      infile1=1024
      idiv = ifix(log(float(infile1))/log(2.0)-2.5)
      ngrids =2*ifix(log(float(infile1))/log(2.0)-2.5)
      infile1=infile1+1

      f = 1.0

      xf = 0.0
      yf = 0.0
      zf = 1.0

      zp = 1.0

      periods = 1.0
      omega = 2.0 * 3.14159265 * periods / (ymax-ymin)

      freq=30500000000.
c     Wave number
```

vary1k.f

```
      k = 2.0 * 3.141592650 / (300000000/freq)

        open(7,file="results.dat")

        open(99,file="contour.dat")

        min_dist=1
        max_dist=7

        read(5,997) i
 997    format(i3)
        if(i.lt.min_dist.or.i.gt.max_dist) then
            write(6,999)i
 999        format(" bad distortion selector:",i4);
            stop
        else
            write(6,998)i, xamp(i), yamp(i)
 998        format("doing iteration #",i2,"; ",g10.4,",",g10.4)
        endif


c       do 10 i=min_dist,max_dist
        idist=i
        xampl=xamp(i)
        yampl=yamp(i)
        write(7,105)
 105    format('avgerr.wpg')
        write(7,106) xampl,yampl
 106    format(' rnfpg error analysis'/
     1  ' amplitude of (x,y) distortion = ('
     2  ,d18.10,',',d18.10,')' )

        xlambda = 0.1
        t = xampl
        del = t * xlambda

        open(4,file="ph1.dat",FORM='unformatted')
        call rnfpg(infile1,fmaxerr)
        close(4)
        open(3,file="ph1.dat",FORM='unformatted')
        open(4,file="ph1f.dat",FORM='unformatted')
        call fixphi(infile2)
        close(4)
        close(3)
 21     npts(i) = j-1
 10     continue
          call itime(atime)
          call idate(iday,imonth,iyear)
          write(6,500)imonth,iday,iyear,atime
        stop
        end
```