GODDARD
GRANT
IN-74-CR
71415
p-68

ITERATIVE DECONVOLUTION OF X-RAY AND OPTICAL SNR IMAGES

NASA Grant NAG5-1364

Final Report

For the period 1 June 1990 through 30 November 1991

Principal Investigator

Peter Nisenson

February 1992

Prepared for

National Aeronautics and Space Administration

Greenbelt, Maryland 20771

Smithsonian Institution
Astrophysical Observatory
Cambridge, Massachusetts 02138

The Smithsonian Astrophysical Observatory
is a member of the
Harvard-Smithsonian Center for Astrophysics

The NASA Technical Officer for this Grant is Dr. Donald K. West, Code 684, Laboratory for Astronomy and Solar Physics, Space Sciences Directorate, Goddard Space Flight Center, Greenbelt, MD 20771.

(NASA-CR-189906) ITERATIVE DECONVOLUTION OF X RAY AND OPTICAL SNR IMAGES Final Report, 1 Jun. 1990 - 30 Nov. 1991 (Smithsonian Astrophysical Observatory) 68 p CSCL 20F

N92-18811

Unclas

G3/74 0071415

Iterative Deconvolution of X-ray and Optical SNR Images

Peter Nisenson Clive Standley, and John Hughes

Harvard-Smithsonian Center for Astrophysics
60 Garden Street
Cambridge, MA 01803

1 Introduction

Blind Iterative Deconvolution (BID) is a technique which was originally developed to correct the degrading effects of atmospheric turbulence on astronomical images from single short exposure, high signal-to- noise-ratio frames. At the Center for Astrophysics, we have implemented a version of BID following the general approach of Ayers and Dainty (1988), but extending the technique to use Wiener filtering (Nisenson el al, 1990) and developed it for application to high energy images from Einstein and ROSAT. In the optical, the point spread function (psf) that degrades the images is due to a combination of telescope and atmospheric aberrations. At high energies, the degrading function is the instrument response function, which is known to be time and energy level unstable. In both cases the psf is poorly known, so BID can be used to extract the psf from the image and then deconvolve the blurred image to produce a sharpened image.

BID follows the general approach of constrained iterative techniques such as those developed for phase retrieval (Gerchberg and Saxton, 1972; Fienup, 1978) with blind deconvolution (Lane and Bates, 1987). One starts with an image which is degraded by some blurring function (i.e. the point spread function or psf). A necessary condition for the algorithm to work is that the blurring function be invariant over the entire restored image field (stationarity), and we note that the degradation in purely a linear operation. The general approach in the algoritm is to find a pair of functions whose convolution gives the input data within a set of physical constraints. The most important constraint requires that both the image and the psf be positive. When the data is noisy, this will not be strictly true, but minimizing the negativity in the solution appears to be sufficient in almost all cases. Another constraint is applied by defining a "support" region in image space where the image (or psf) is non-zero, and then resetting that area to zero after each iteration. Finally, the signal-to-noise ratio in the Fourier transform (FT) of the image or psf constrains the dynamic range of the deconvolution operation by using a Wiener filter for the deconvolution operation. While it has not been proven that the restored images from BID are unique, complicated images appear to converge on only one sensible solution.

2 Implementation

A flow diagram for the technique is given in figure 1. One starts with a degraded image and an initial estimate of the point spread function (psf). The initial psf can be randomly chosen, however the number of iterations required for the algorithm to converge is highly dependant on how close the first estimate of the psf is to the actual psf. Both inputs are Fourier transformed and a deconvolution is performed by constructing a Wiener filter from the FT of the psf and an estimate of the noise in the image spectrum. The technique of Wiener (or Optimum) filtering damps the high frequencies and minimizes the mean square error between each estimate and the true spectrum.

The Weiner filtering spectrum, $o_f(u,v)$, usually takes the form:

$$o_f(u,v) = o(u,v) \cdot \frac{p^*(u,v)}{|p(u,v)|^2 + |n(u,v)|^2} \tag{1}$$

$o(u,v)$, $p(u,v)$ and $n(u,v)$ are the object, PSF and noise spectra respectively. For white noise, $n(u,v)$ usually can be replaced with a constant estimated as the rms fluctuation of the high frequency region in the object spectrum (i.e. at frequencies where the object power is negligible).

The result after Wiener filtering is transformed back to image space and positivity and support constraints are applied. The negatives in the image are set to zero and their total value summed. This value is divided by the number of pixels inside the support region and subtracted within that region in order to preserve the total power in the image. After subtraction, some areas of the image may become negative. If this is the case, the negatives are again truncated, summed and subtracted. This procedure is repeated until the negatives in the restored image are reduced to a negligible level. The original degraded image is then deconvolved by the restored image obtained from the first iteration. The result is transformed back to image space. Again, positivity and support constraints are enforced. The result is a new estimate of the psf. The iteration continues until a stable solution is found. A damping factor is used to stabilize the iteration, particularly important when the psf estimate is still inaccurate. About 20% of the image (or psf) from the previous cycle is averaged with the new image (or psf) in the early stages of the process. This percentage is reduced when the iteration has nearly converged.

Two criterion have been found to be very useful in determining the completion of the iterations: the ratio of positive power to negative power in the restored image and psf; and the rms difference between images or psfs from successive iterations. Both criteria drop irregularly in the first few cycles of the iteration, but they both level off and stabilize when the operation is close to convergence. After examination of the output image and the psf, the results may be fed back into the loop for continued iterations.

There are a number of parameters which must be chosen in order to ensure convergence and an optimum result. Probably the most important are accurately estimating the signal-to-noise ratio in the data, allowing construction of the Wiener filter, and carefully defining the region for the support constraint. It is also very important that the image and psf remain aligned with the support, since truncation seriously degrades the process. This is done by centering the initial image and psf, calculating the two support regions, and then recentering the psf after each cycle.

Appendix A contains listings of the software developed for BID. It consists of four packages: the main program (bid.f), a collection of subroutines used by BID (bidsubs.f), a set of subroutines used to interface with the IRAF data reduction package (irafsubs.f), and a multi-dimensional fast fourier transform (fourt.f). Each of the packages has internal documentation that describes the functions of each routine and aids the selection of parameters when running the program. This choice of parameters is highly dependant on the data that is to be deconvolved, and it must be tuned by the user through experience. However, suggested starting values are provided that should help initially. Also included is wiener deconvolution program that can be used after BID has found the psf. Reinsertion of the original data and the psf from BID into DCW (dcw.f) allows deconvolution of the image with an adjustable low-pass filter and a choice of wiener filter parameter to produce smoother or sharper reconstructions. Once the psf has been found, the signal-to-noise ratio in the data determines the frequency range and enhancement level acceptable in the deconvolved image. The programs must be compiled inside IRAF using the FC compliler command.

3 Einstein X-ray Images

We have applied BID to an extensive set of supernova remnants from the Large Magellenic Cloud. The results have varying success, mostly depending on the signal-to-noise ratios in the raw data. In the very low surface brightness SNR's, the result is not much more than a quasi-optimum smoothing. All deconvolved images were produced using the same parameters in BID (to find the psf) and then in DCW to produce the image: we used the parameters suggested in the BID program and we ran BID with the psf support-only option. In DCW, we used a frequency cutoff of 1.0 (the maximum value) and a wiener filter multiplier of 4.0. Some of the images could be restored with smaller wiener multipliers (producing a sharper image) but we show only the results for this one value to allow comparison between data sets.

Figures 1 and 2 are for Cyg X-1. The deconvolved image is point-like with the elongation seen in the raw data (figure 1), presumably due to aberrations removed. DEM71 is an extended low surface brightness remnant. The deconvolved image (Figure 4) is only a somewhat smoothed version of the input, unsurprising for the low signal-to-noise ratio in the raw image (Figure 3). N23 is another extended remnant. Here the deconvolved image (Figure 6) has sharpened and defined the bright knots, allowing a better feeling for the structural detail than in the raw data (Figure 5). The results for N49, N49B, N63A and N132 (Figures 7-14) are similar to N23: some improvement in definition of the brighter features and an overall smoothing of the images allowing easier interpretation.

Figures 15 and 16 show the data and deconvolution for the results obtained for N103B. In this case, the data are believed to suffer from image motion in the up-down direction, giving the image its double lobed appearance. The deconvolution has partially compensated for this, substantially brightening the lower "lobe". However, image motion is difficult to completely remove without very high signal-to-noise ratios since its transfer function drops rapidly to zero.

Figures 17 and 18 show the input image and deconvolution for SNR0102, a circular ring remnant, seen almost face-on. The deconvolution shows a number of interesting features including a number of bright knots and asymmetries in the ring. Particularly striking is the dark break in the ring near the top of the image. The raw data shows hints of this feature, but it is cleanly defined in the reconstruction.

Figures 19 and 20 produce what is perhaps the most perplexing result from the SNR0519. This is a rather small SNR, appearing rather amorphous in the raw data, however the deconvolution produces a very different result: a bright central peak with a small bright region, all sitting on a fainter extended remnant. This data was run for several hundred extra cycles and with different random starting psf's. In all cases the results wer the same. The image in figure 20 is after 200 iterations of BID, very nearly the same result as the 100 iteration output. Further analysis will be required to determine whether the result is real or whether some special problem with data produced it. This is the only result that does not appear to be consistent with the original data.

Figures 21 and 22 show the results for SNR0540, a very compact remnant. The data and the deconvolution show a very bright central core and some extend flux around the SNR. Figures 23 and 24 present two other data displays of the deconvolution, a contour plot and a surface plot. These displays hint at a flattening of the central core, possibly corresponding to the ring seen in optical images. Since it is only a few pixels wid, it would be difficult to conclude the exact nature of the flattening.

These result demonstrate the capabilities and the limitation of the BID program. In all cases, the results appear to be stable, and in all cases but one, consistent with original data. For data with higher signal-to- noise ratio, substantial sharpening of the image features is obtained. For the very low surface brightness images, the result is close to an optimally smoothed result.

4 Summary

BID appears to be a powerful new tool for high angular resolution astronomy. While the technique requires fairly high signal-to-noise ratios in the data, substantial improvement in image sharpness can be obtained even for low flux x-ray images. The programs are relatively easy to using, interfacing with the IRAF package to allow reading and writing IRAF images.

5 References

Ayers, G.R. and Dainty, J.C. 1988, *Opt. Lett.*, **13**, 547.

Brandt, P.N., Scharmer, G.B., Ferguson, S., Shine, R.A., Tarbell, T.D., and Title, A. M. 1988, *Nature*, **335**, 238.

Fienup, J.R. 1978, *Opt. Lett.*, **3**, 27.

Gerchberg, R.W. and Saxton, W.O. 1972, *Optik*, **35**, 237.

Hege, E.K., Cromwell, R.H., Blair, C.N., and Vokac, P.R., 1990, *Proceedings of the SPIE, "Amplitude and Spatial Interferometry"*, bf1234, 510.

Karovska, M. and Habbal, S. 1991, *Ap. J.*, **37**,

Lane, R.G. and Bates, R.H.T. 1987, *J. Opt. Soc. Am. A*, **4**, 180.

Nisenson, P., Standley, C. and Gay, D. 1990, *Proceedings of Space Telescope Science Institute Workshop on HST Image Processing*, Baltimore, Md.

Figure Captions

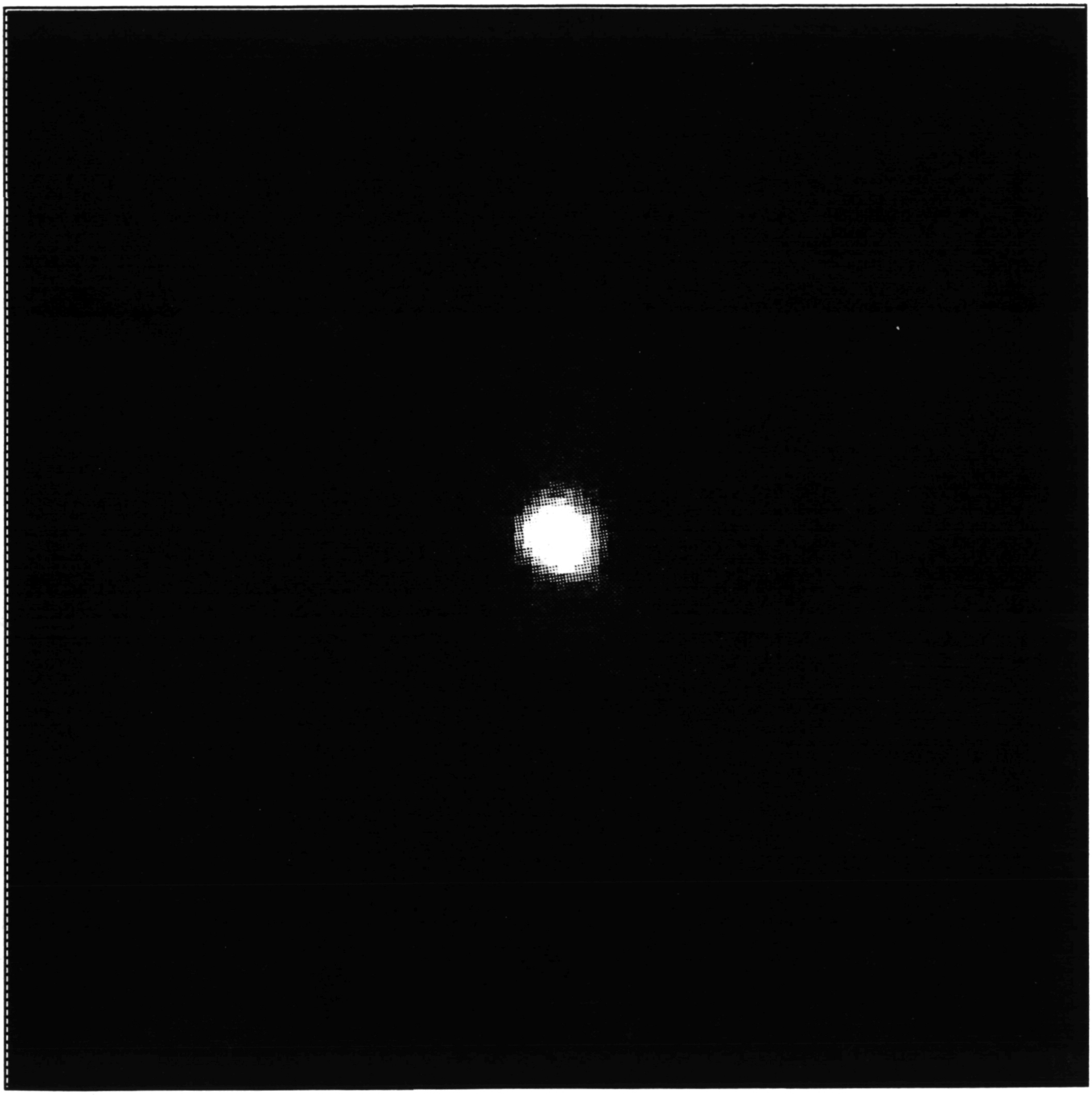

Figure (1) - Einstein Data for Cyg X-1

cygx1d300w4

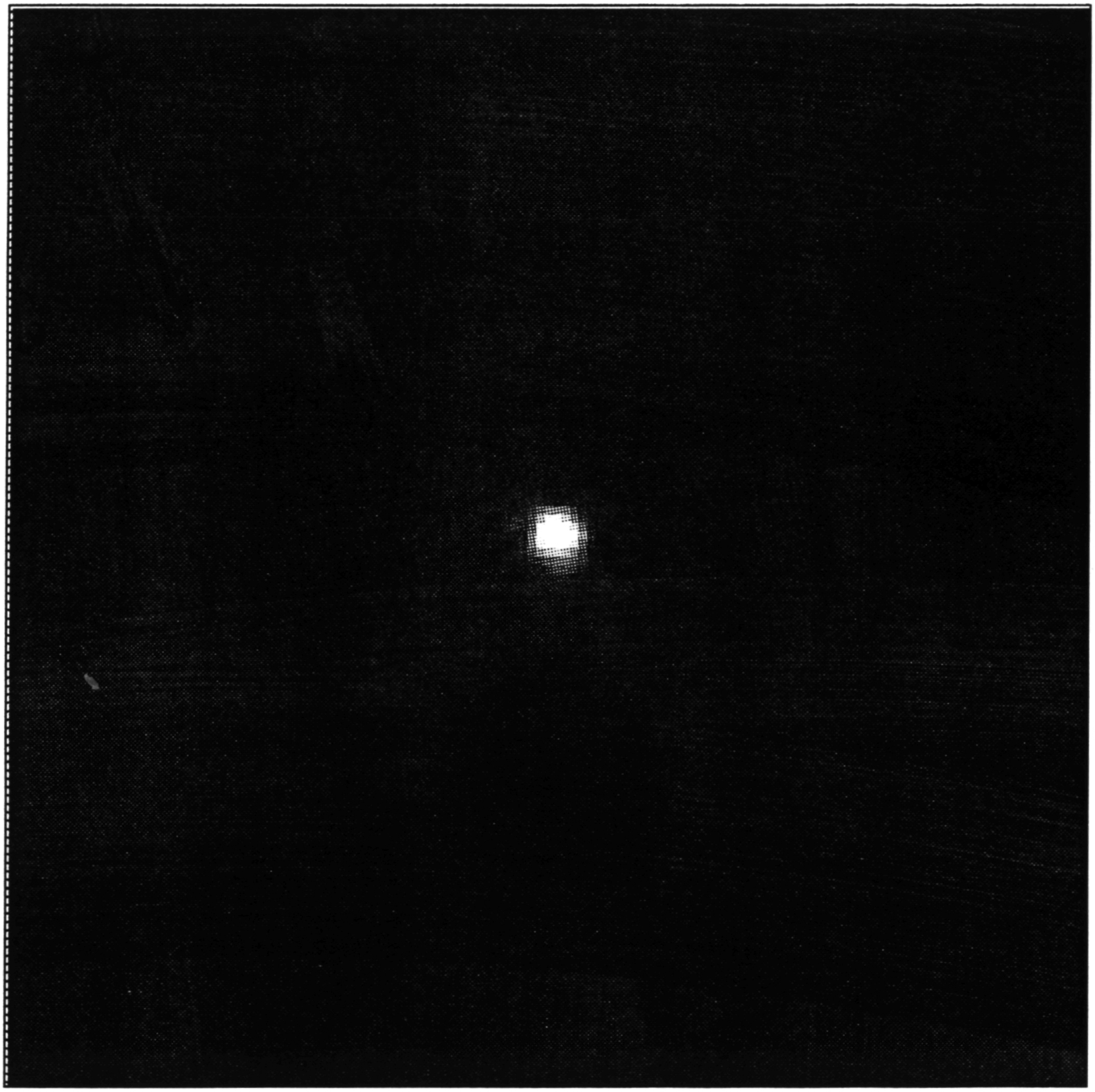

Figure (2) - BID Deconvolution for Cyg X-1

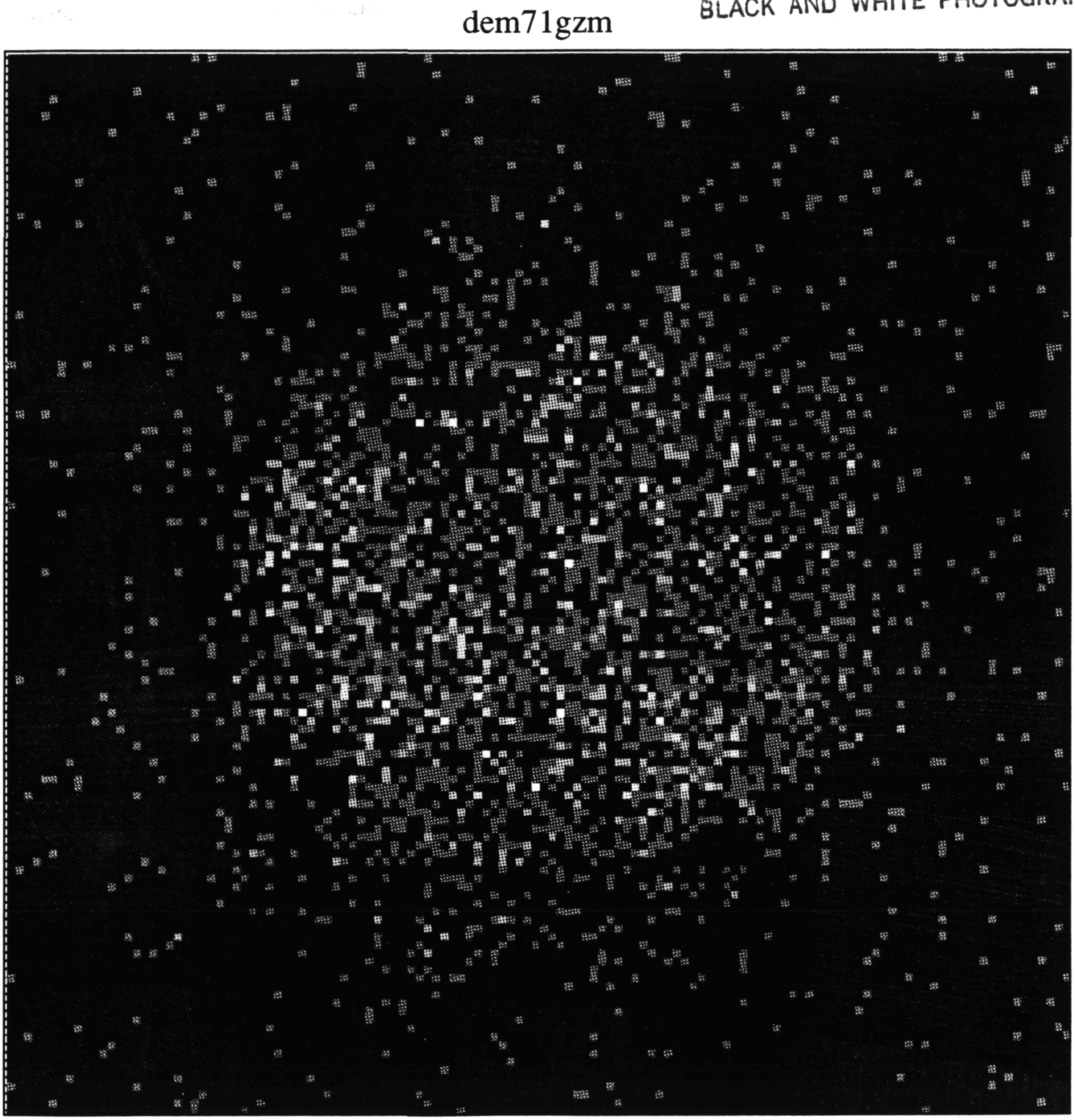

Figure (3) - Einstein data for DEM71

dem71d100w4

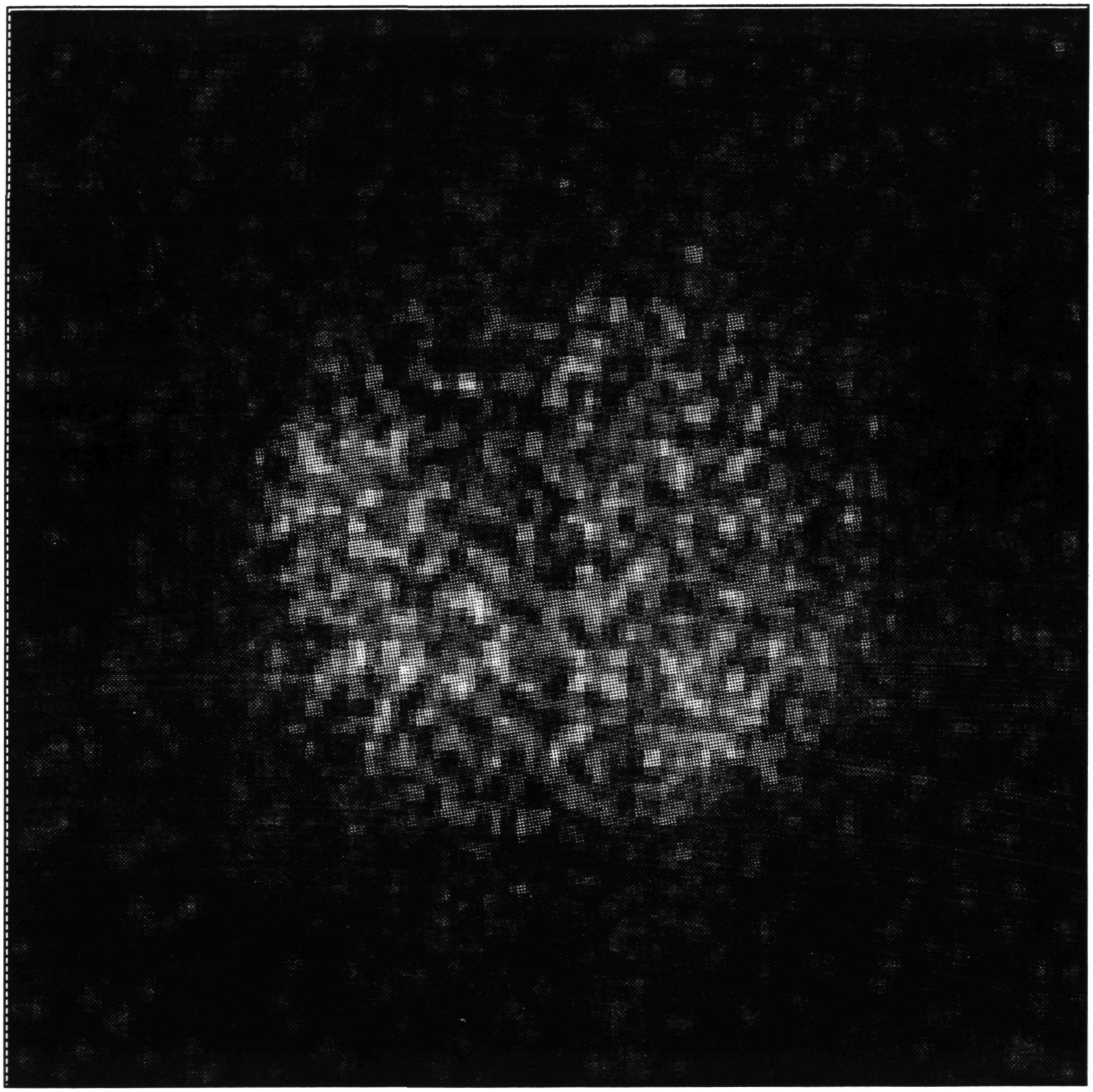

Figure (4) - Bid Deconvolution for DEM71 n23gzm

Figure (5) - Einstein Data for N23

n23d100w4

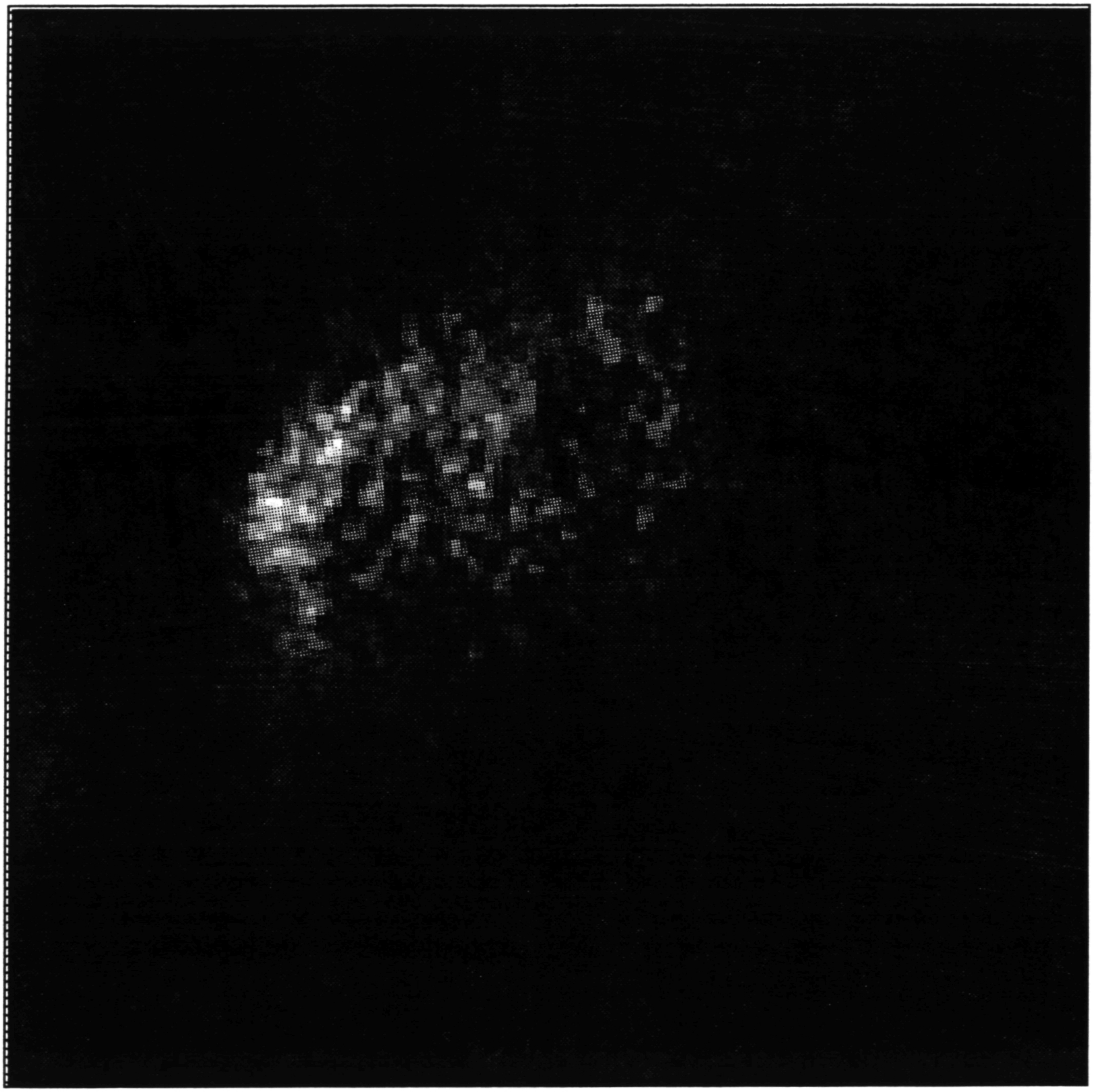

Figure (6) - BID Deconvolution for N23

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

n49gzm

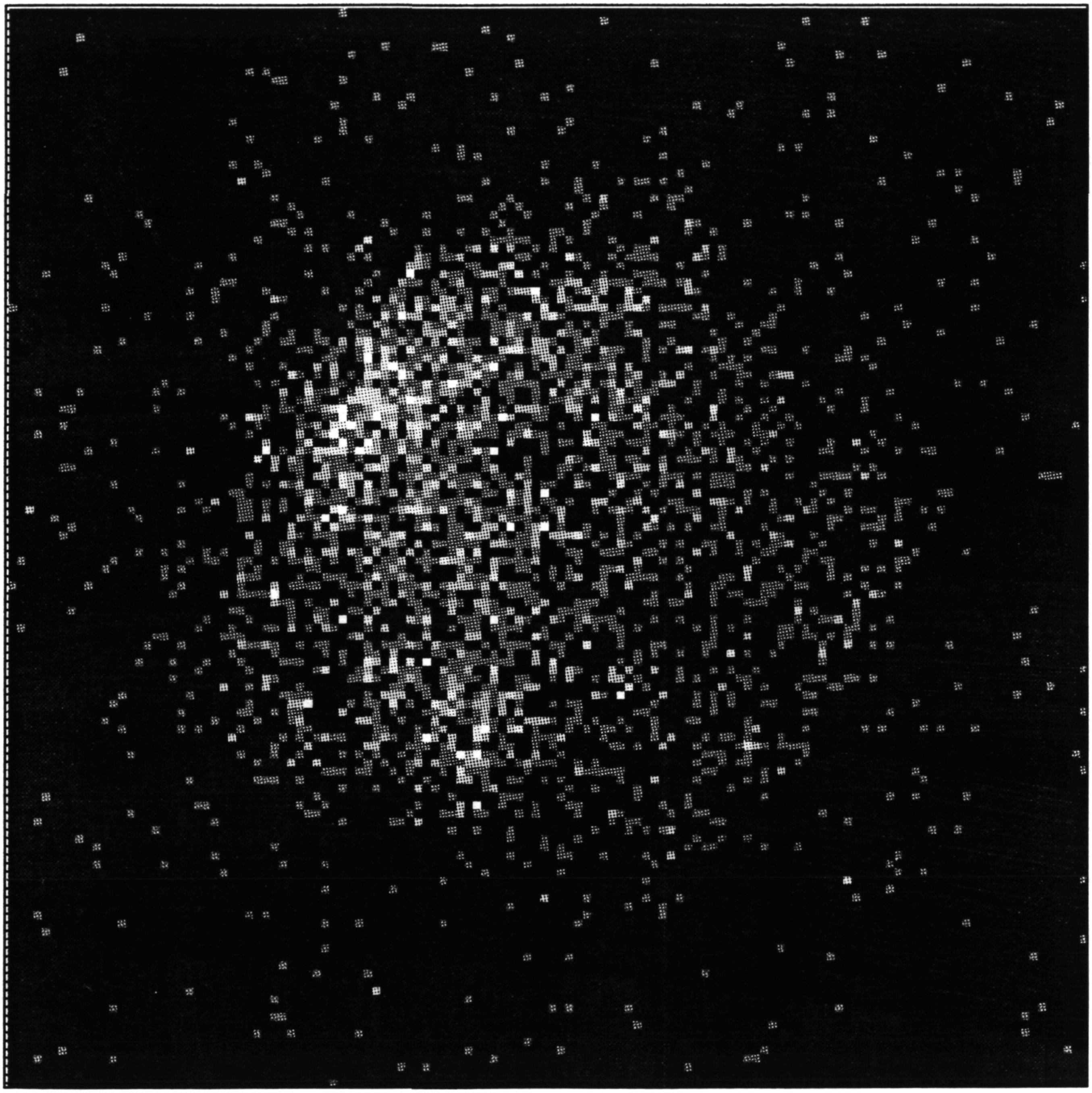

Figure (7) - Einstein data for N49

n49d100w4

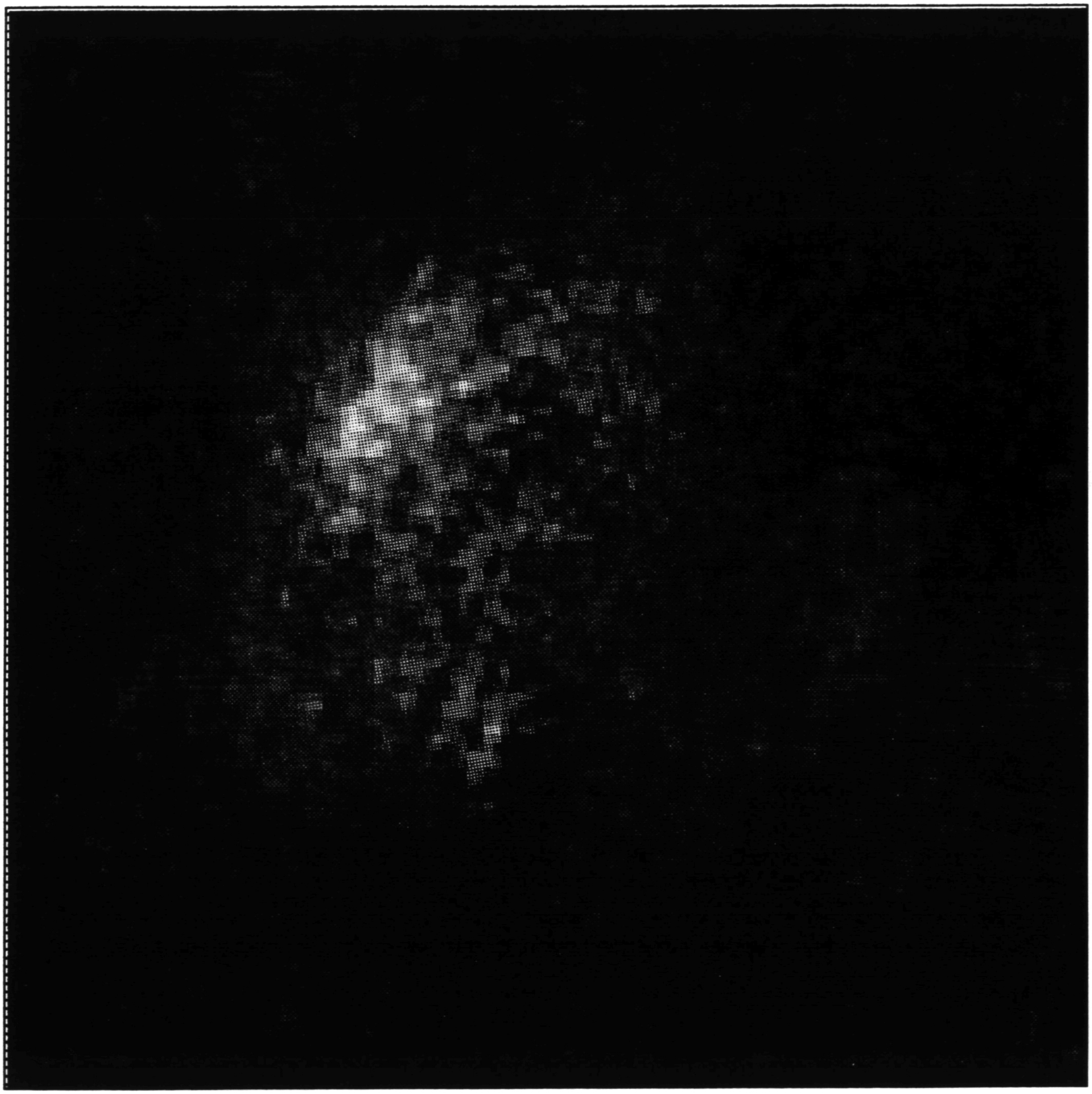

Figure (8) - Bid Deconvolution for N49

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

n49bgzm

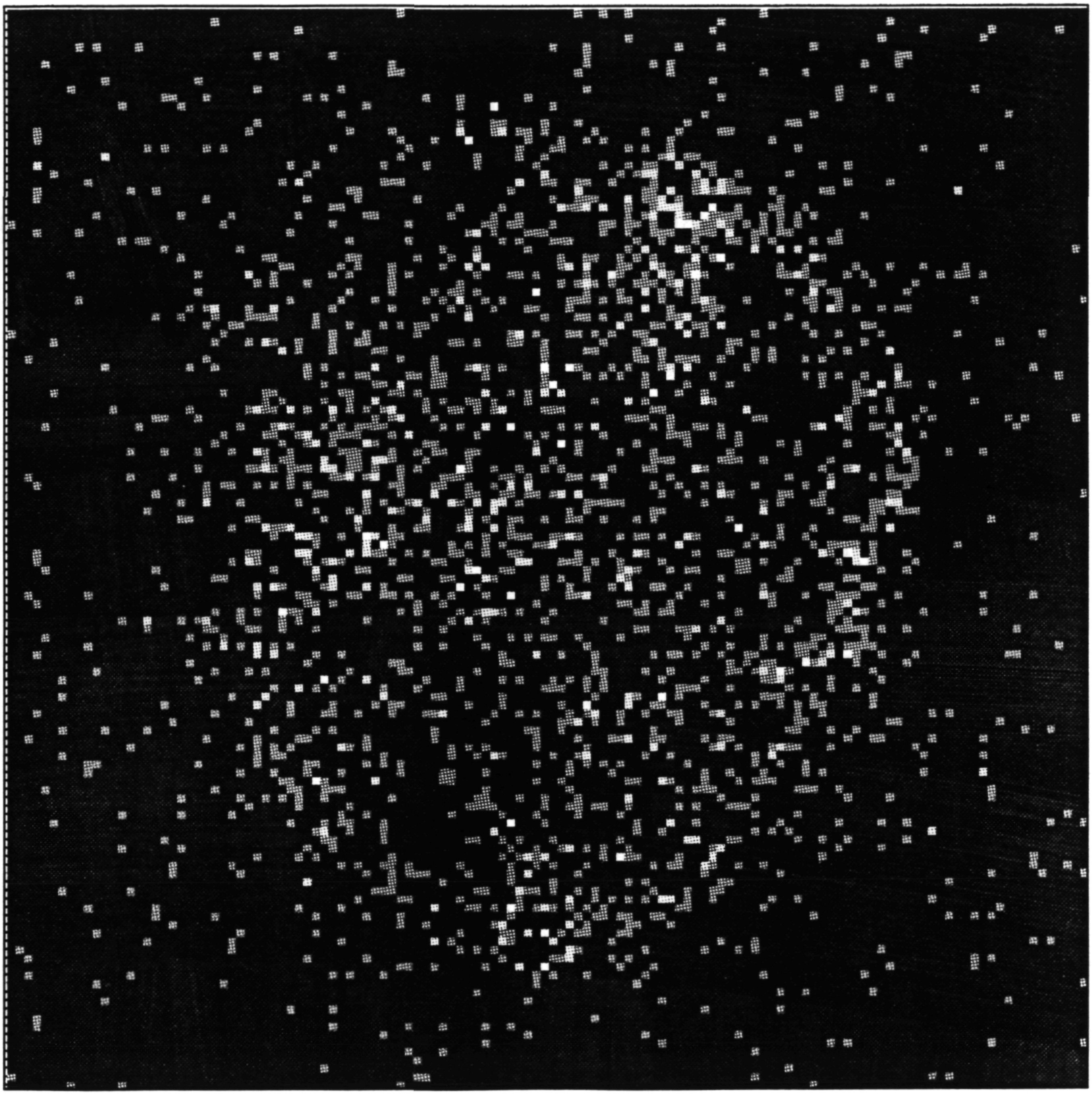

Figure (9) - Einstein Data for N49B

n49bd100w4

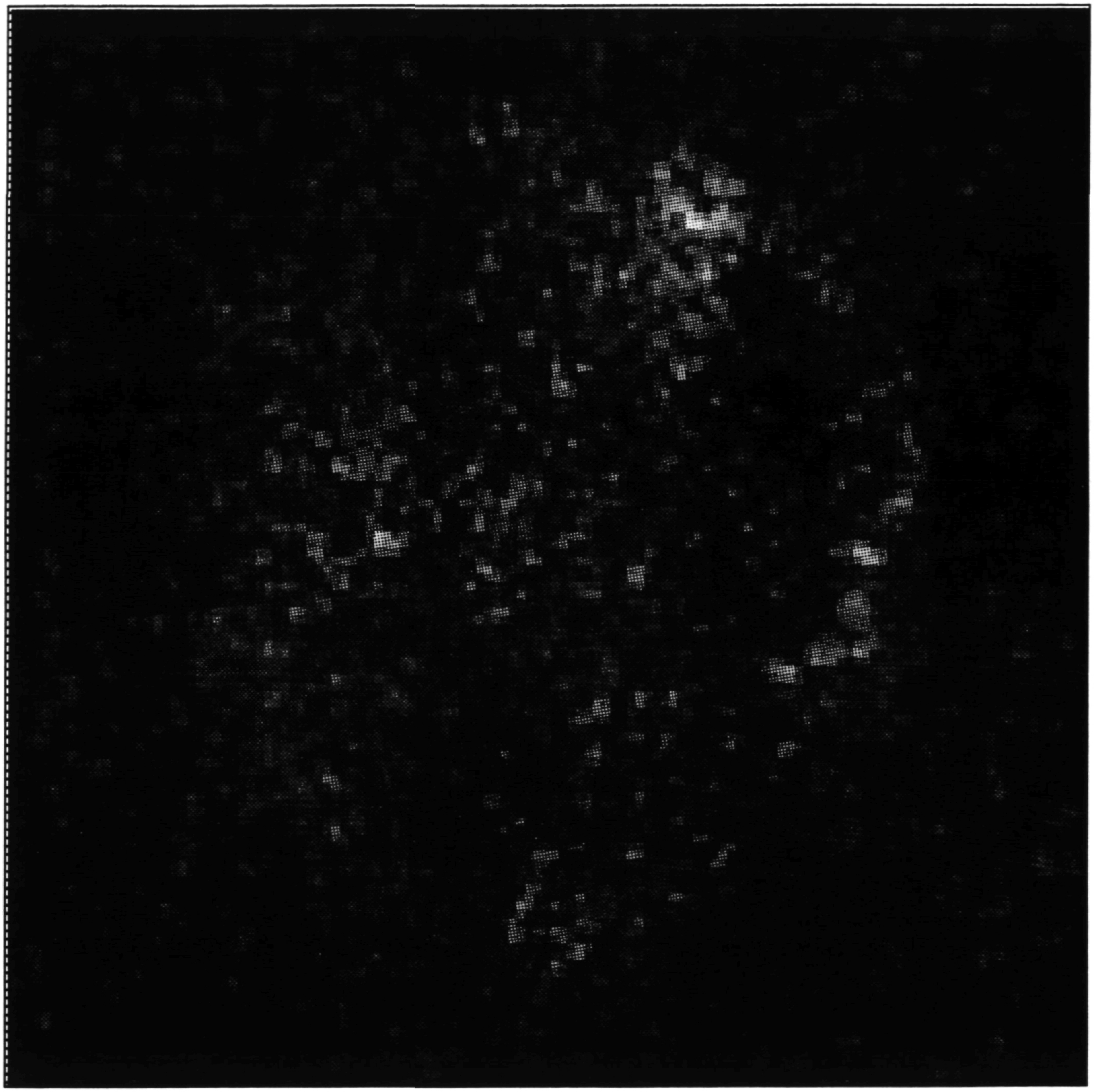

Figure (10) - BID Deconvolution for N49B n63agzm

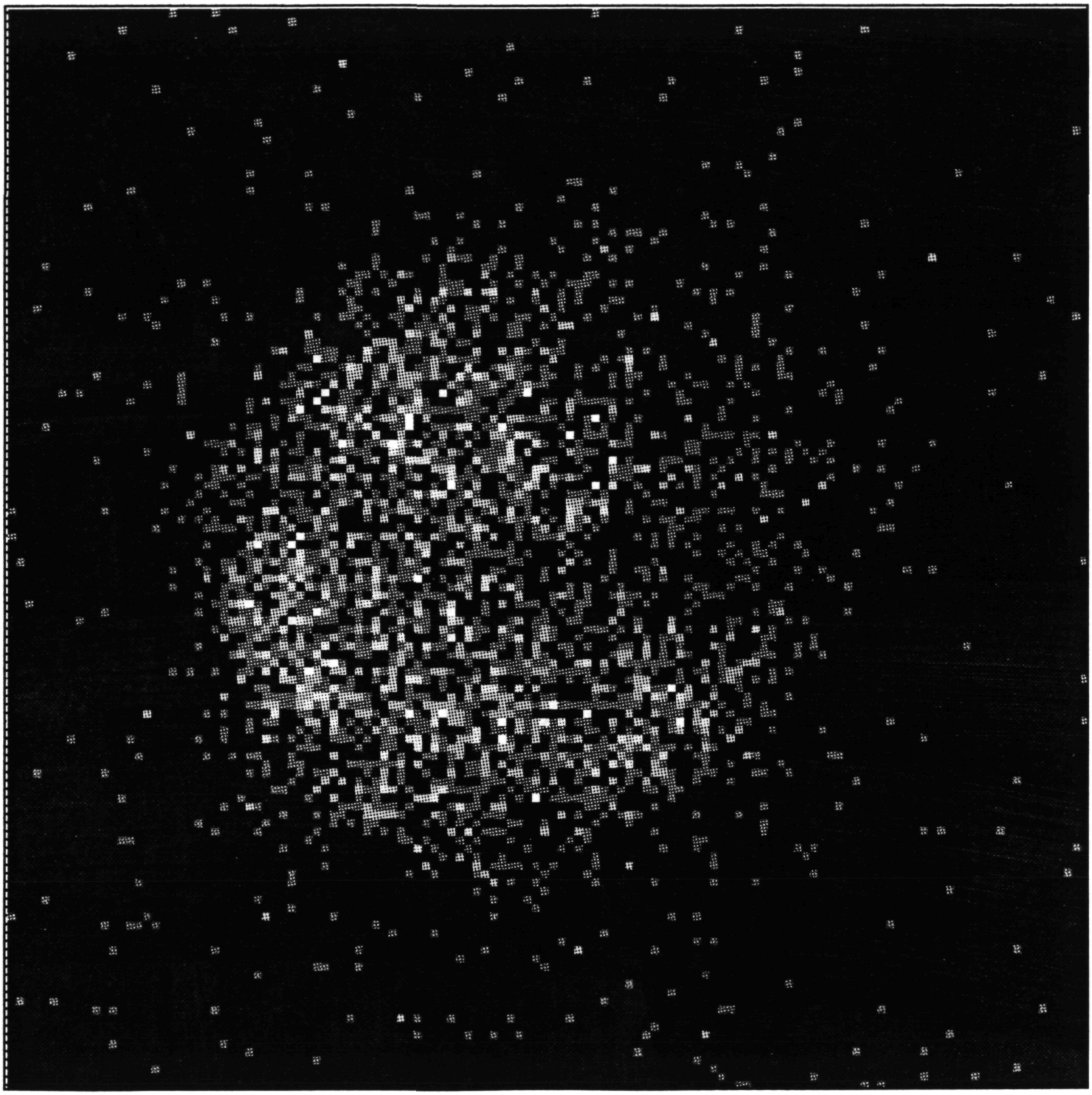

Figure (11) - Einstein data for N63A

n63ad100w4

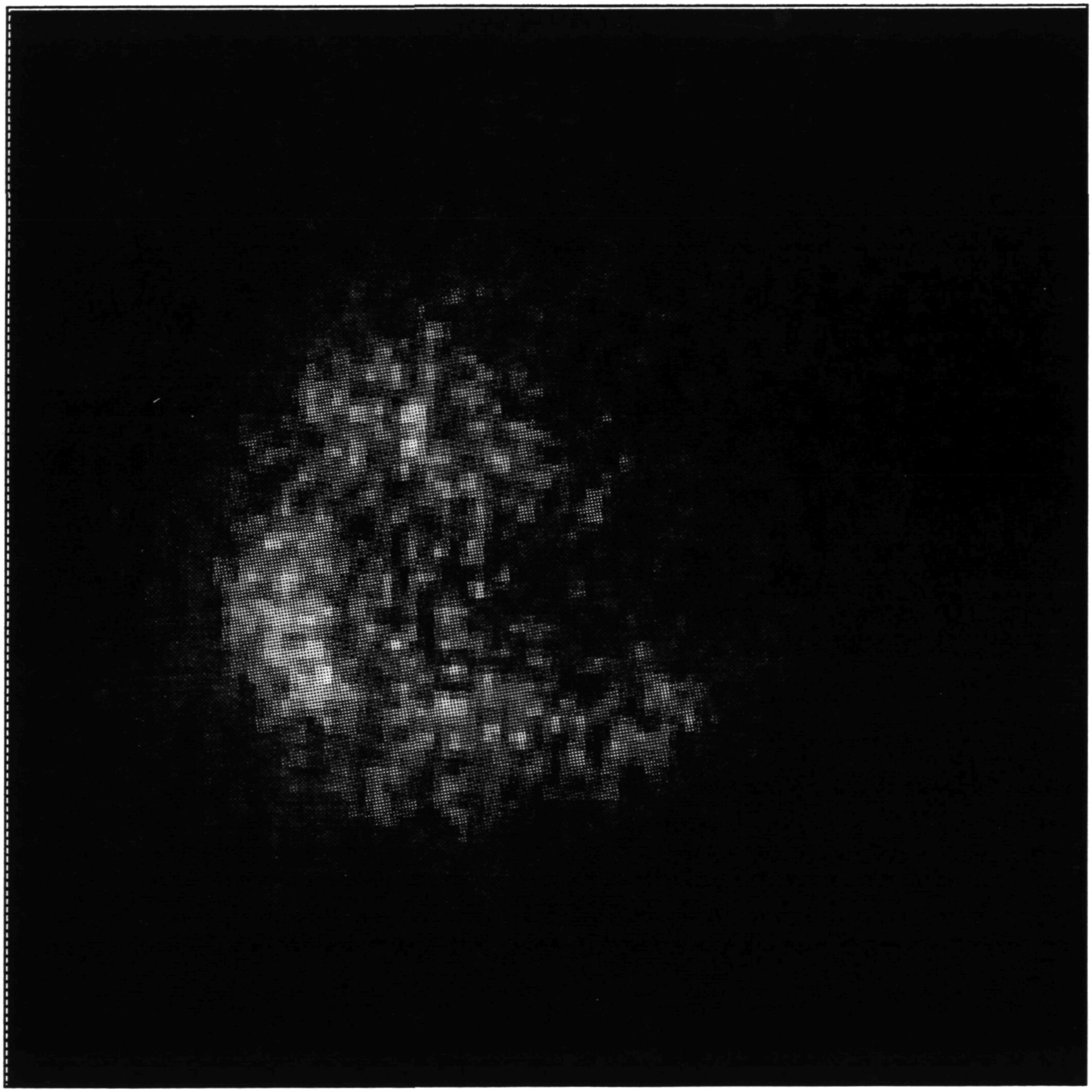

Figure (12) - Bid Deconvolution for N63A

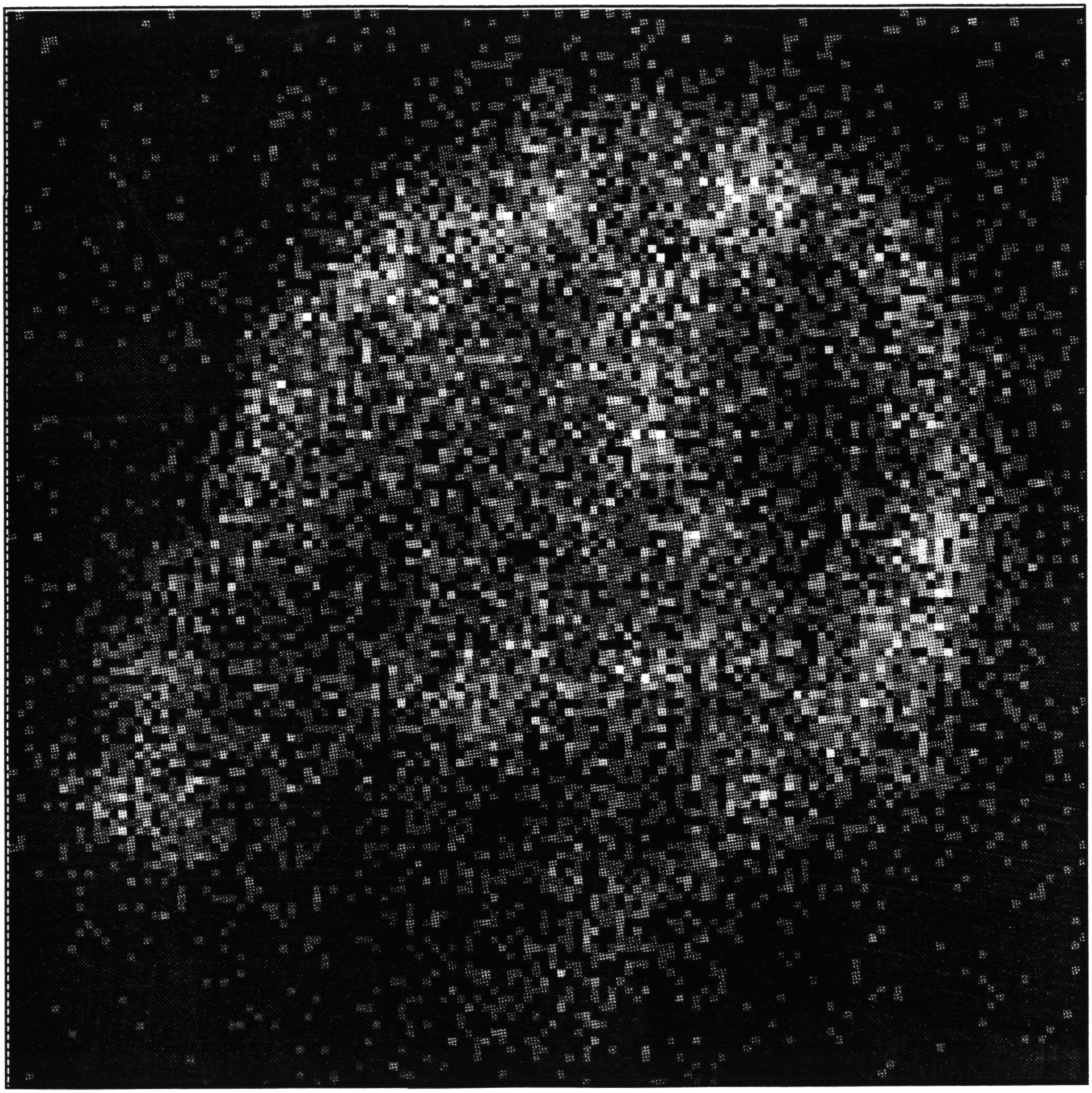

Figure (13) - Einstein data for N103B

n132d100w4

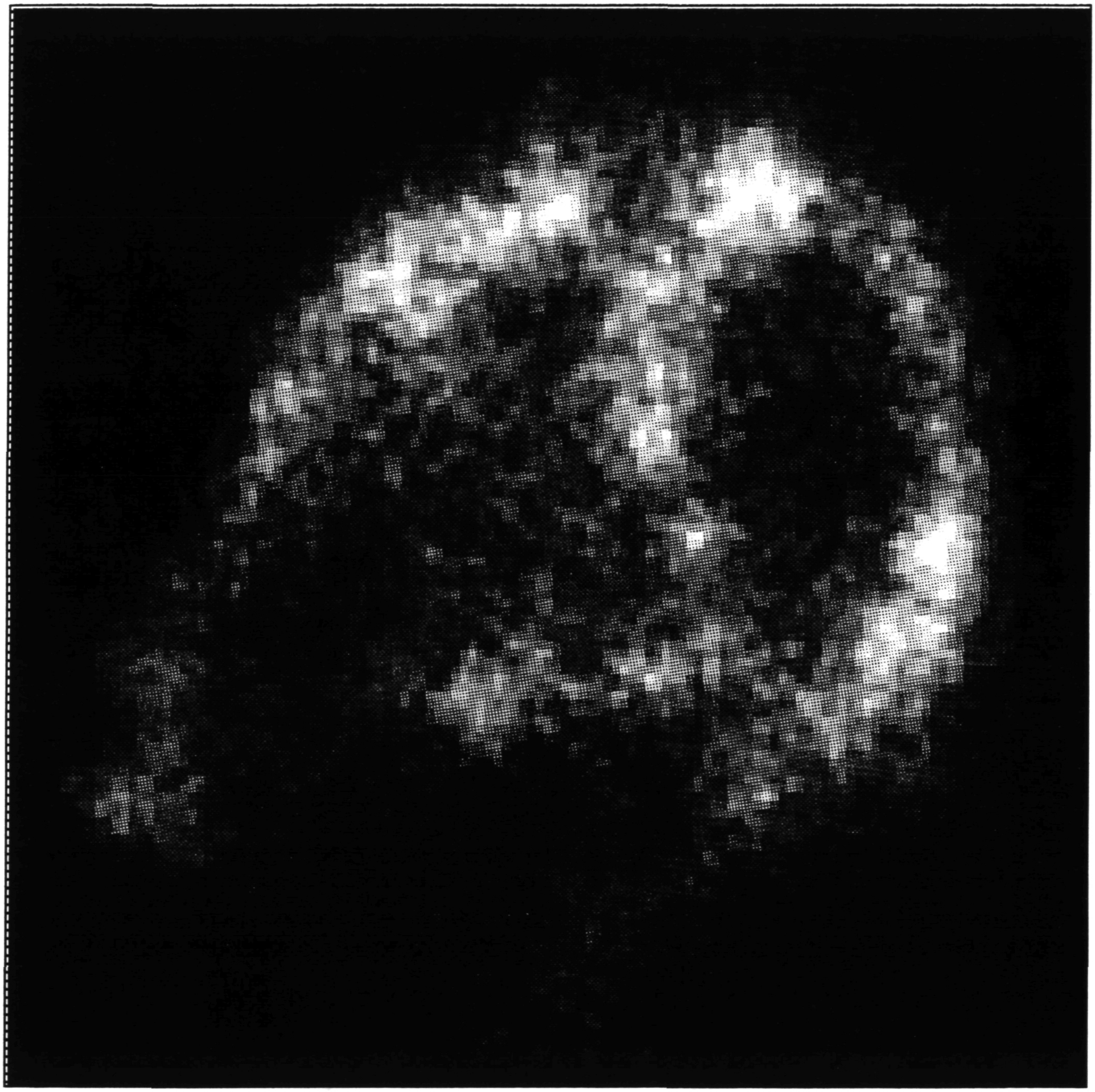

Figure (14) - Bid Deconvolution for N103B

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

n103bgzm

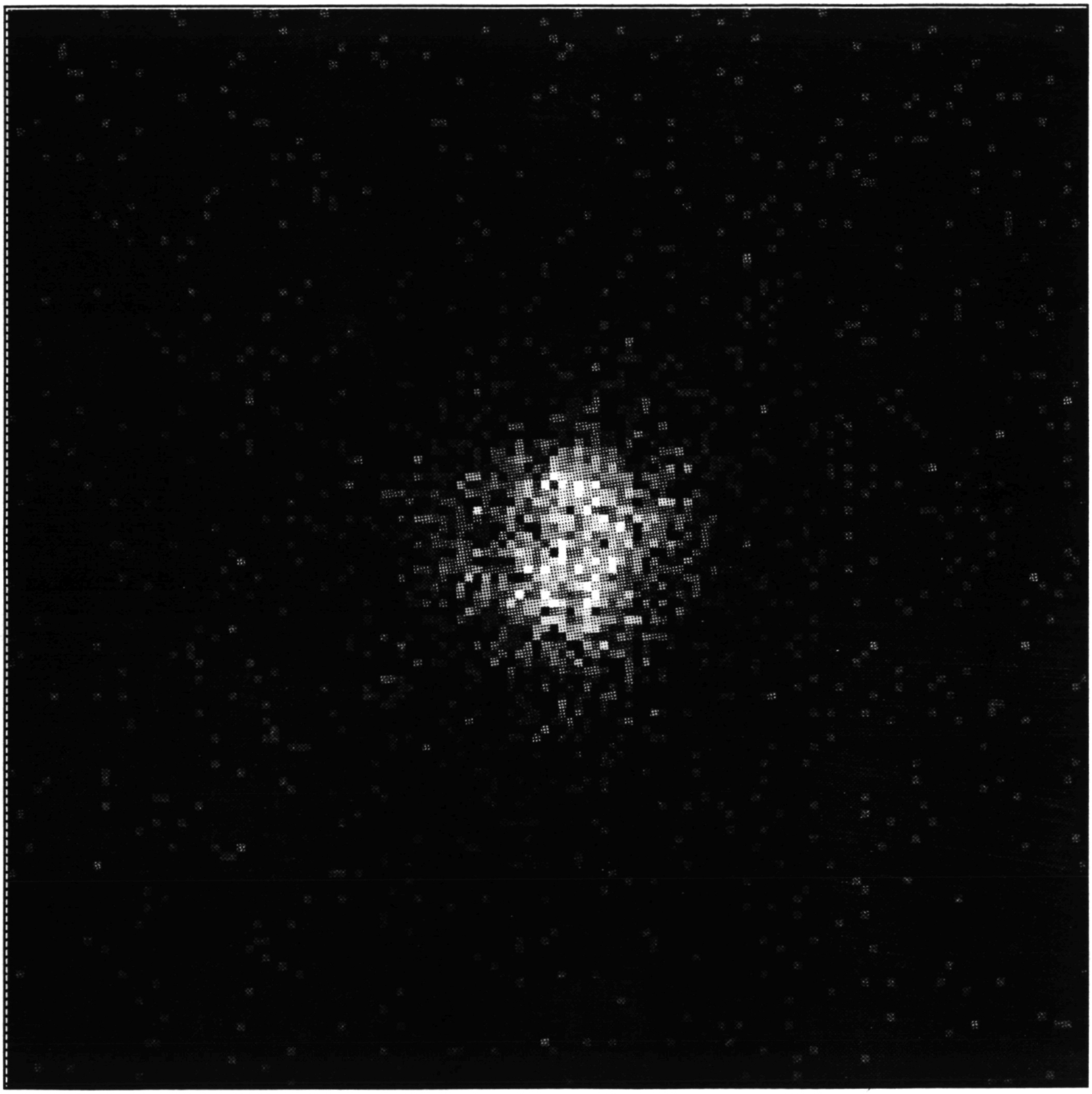

Figure (15) - Einstein Data for N132D

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

n103bd100w4

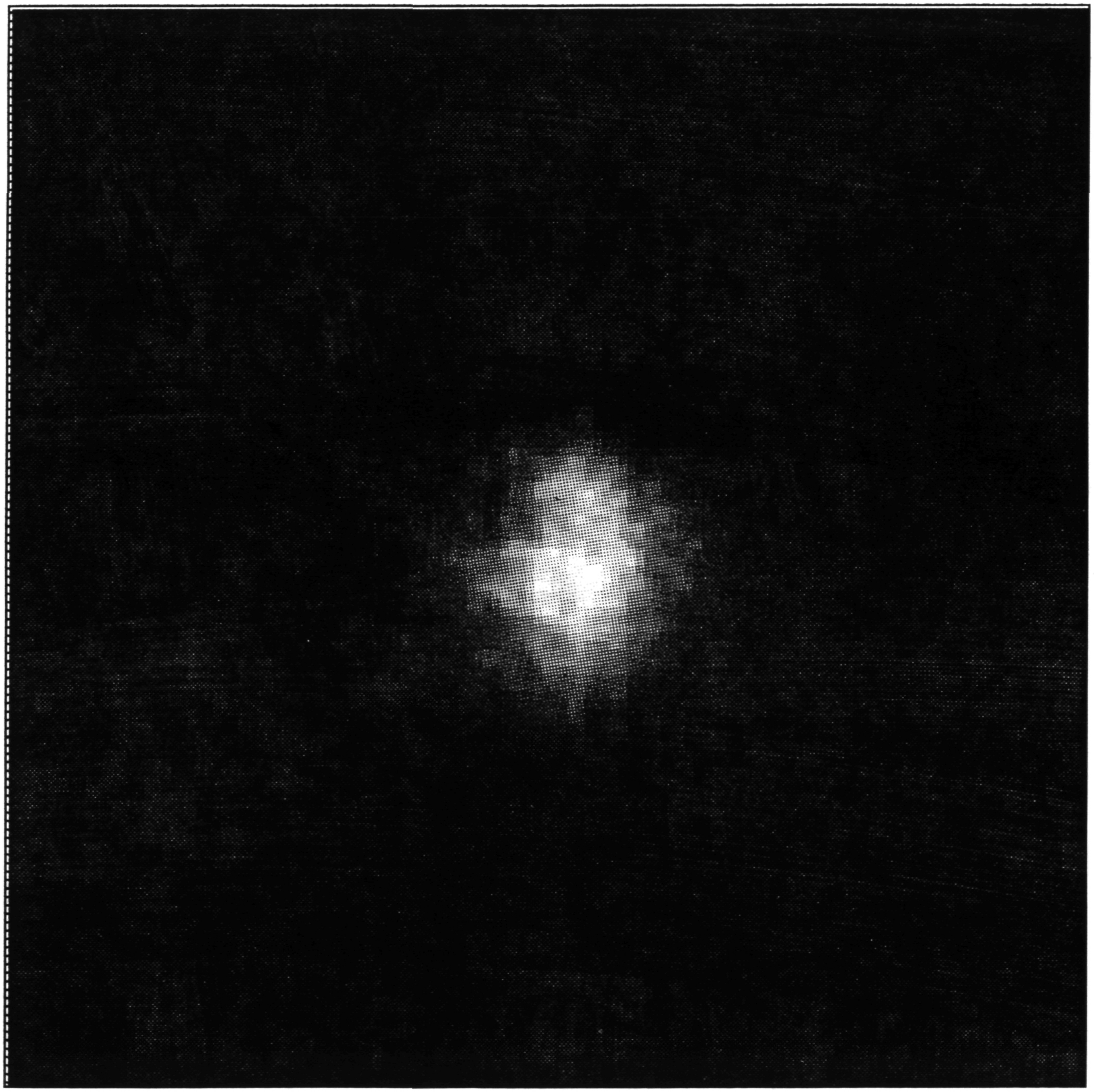

Figure (16) - BID Deconvolution for N132D snr0102gzm

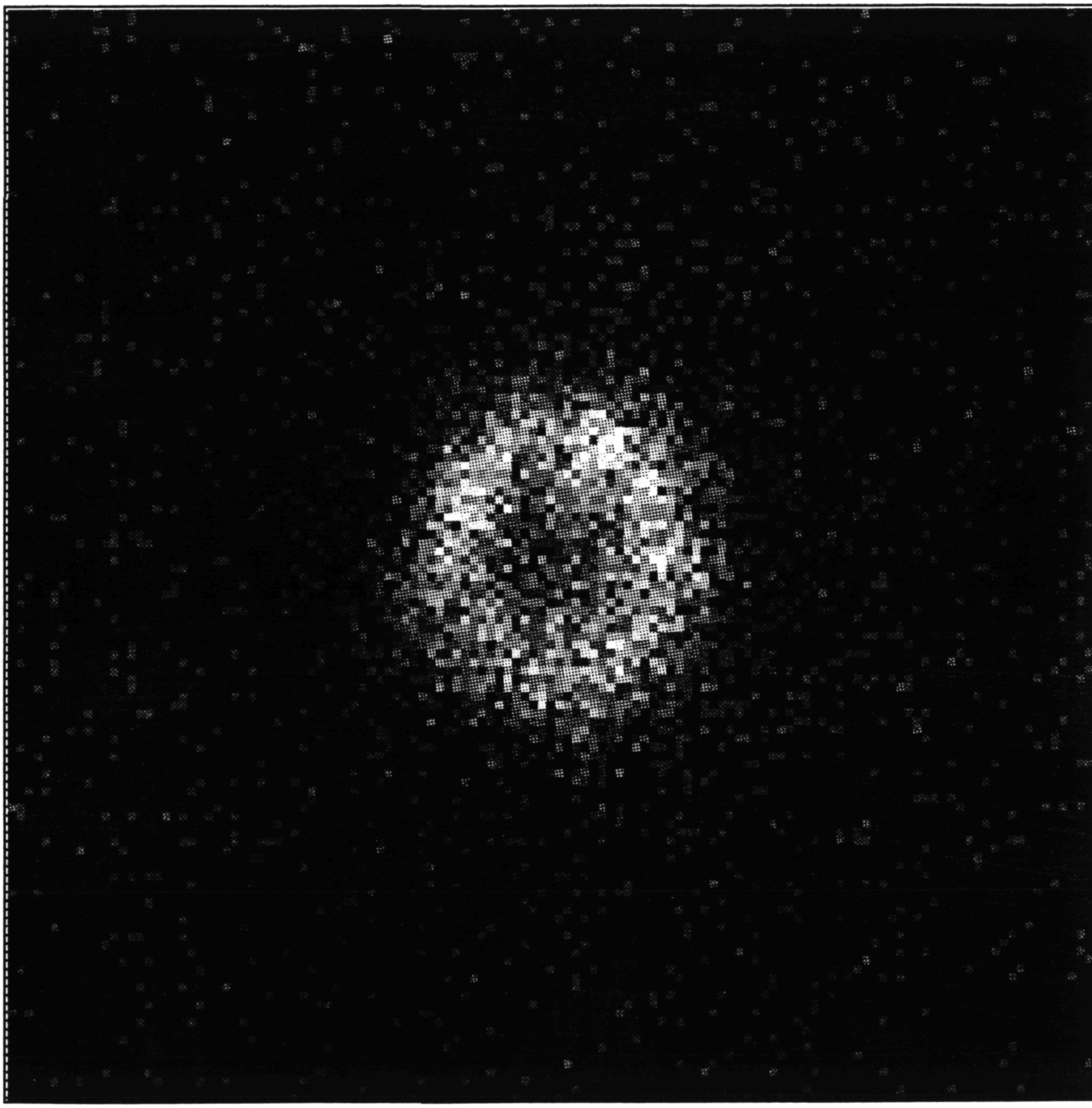

Figure (17) - Einstein Data for SNR0102

snr0102d100w4

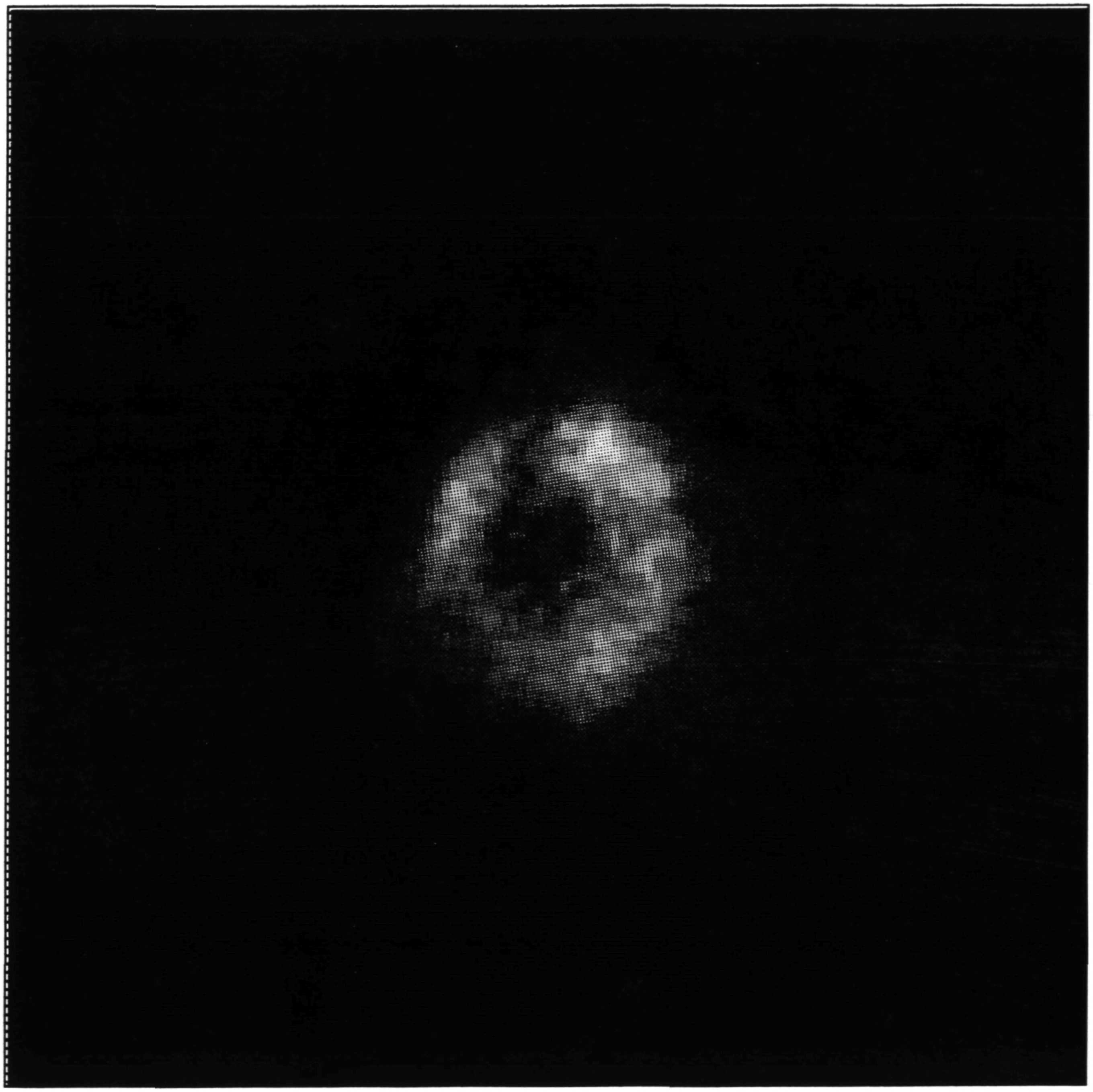

Figure (18) - BID Deconvolution for SNR0102

snr0519gzm

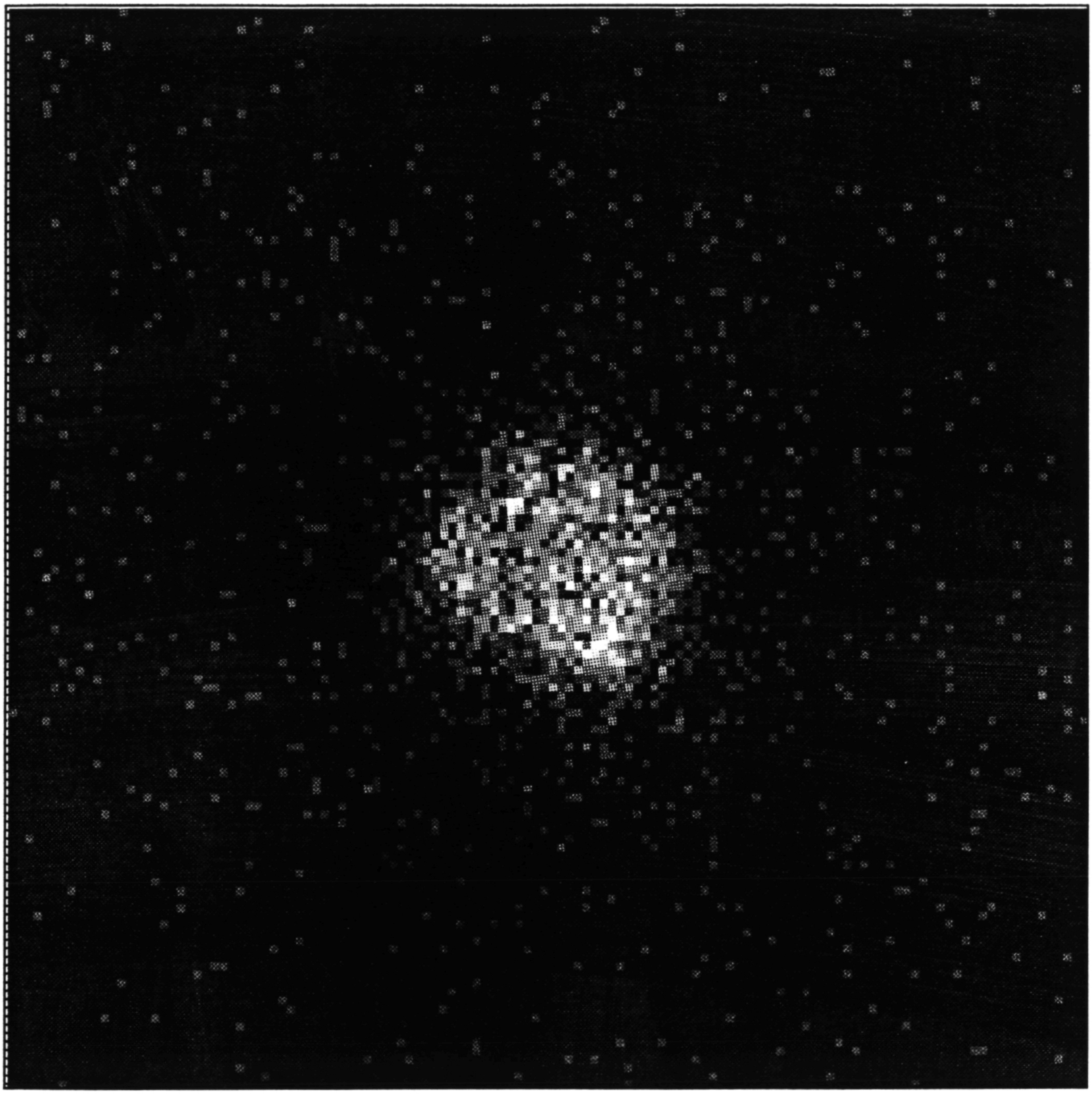

Figure (19) - Einstein data for SNR0519

snr0519d100w4

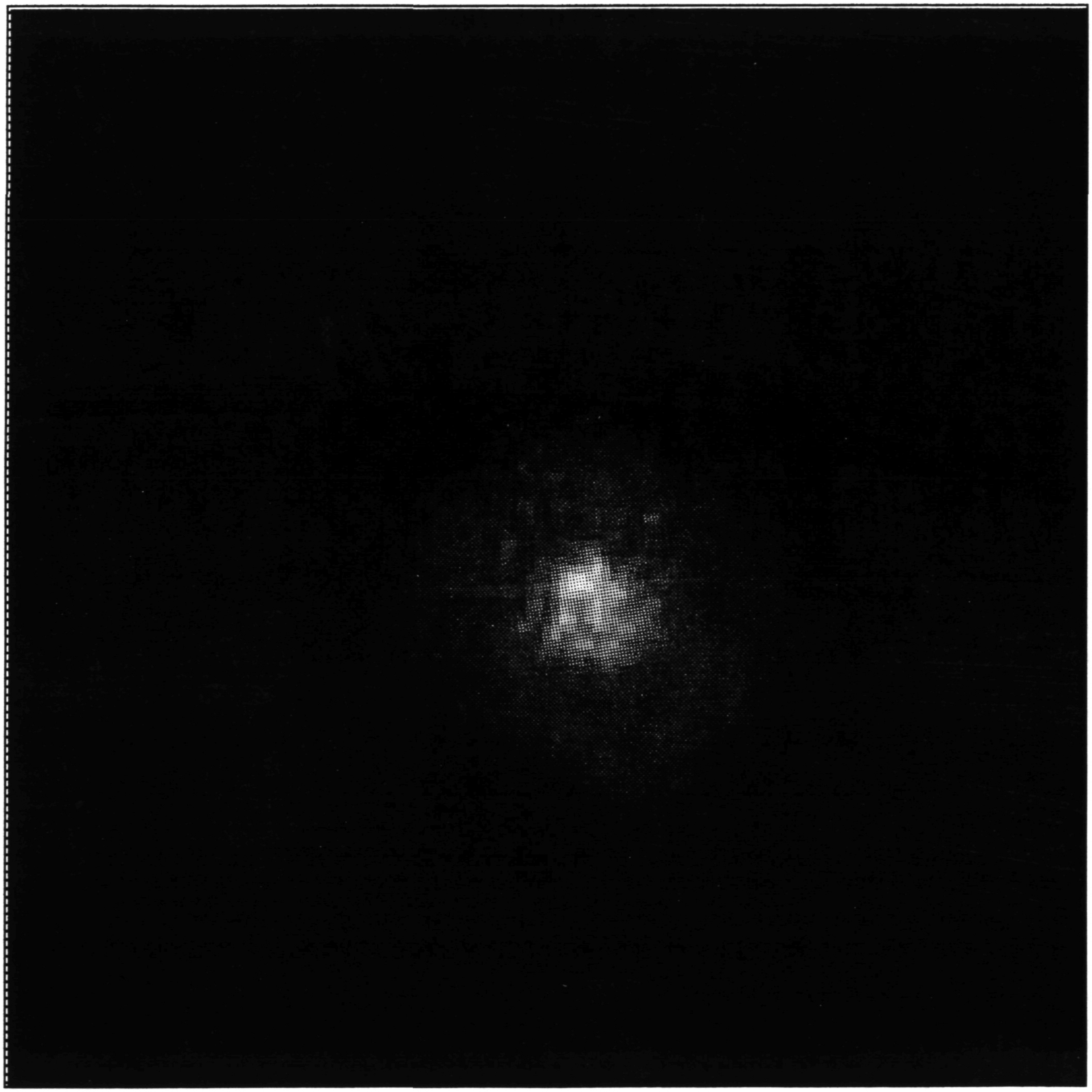

Figure (20) - Bid Deconvolution for SNR0519

snr0540Hgzm

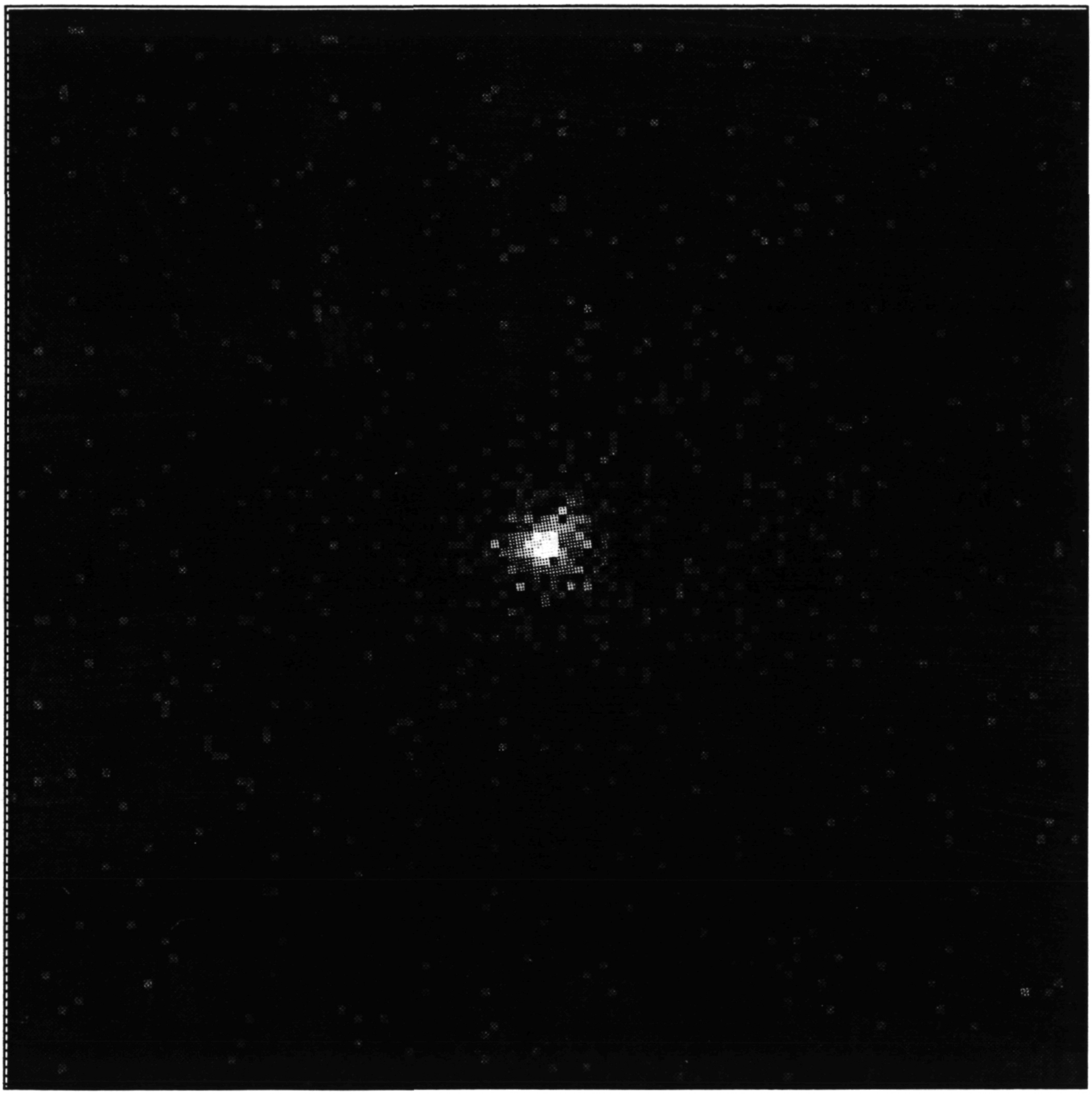

Figure (21) - Einstein Data for SNR0540

snr0540Hd100w4

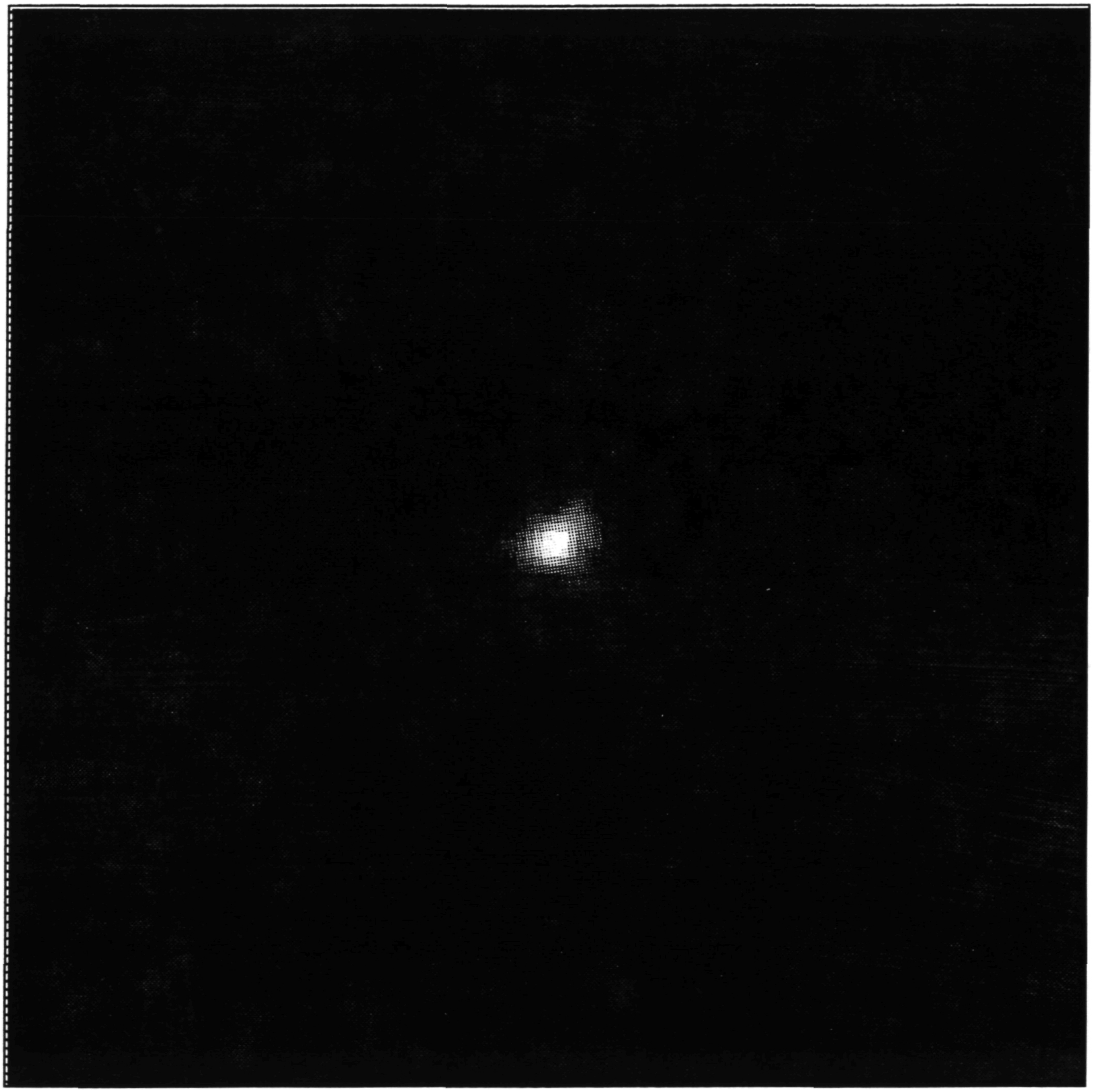

Figure (22) - BID Deconvolution for SNR0540

NOAO/IRAF nisenson@cfasp36.harvard.edu Wed Jan 15 09:35:22 1992

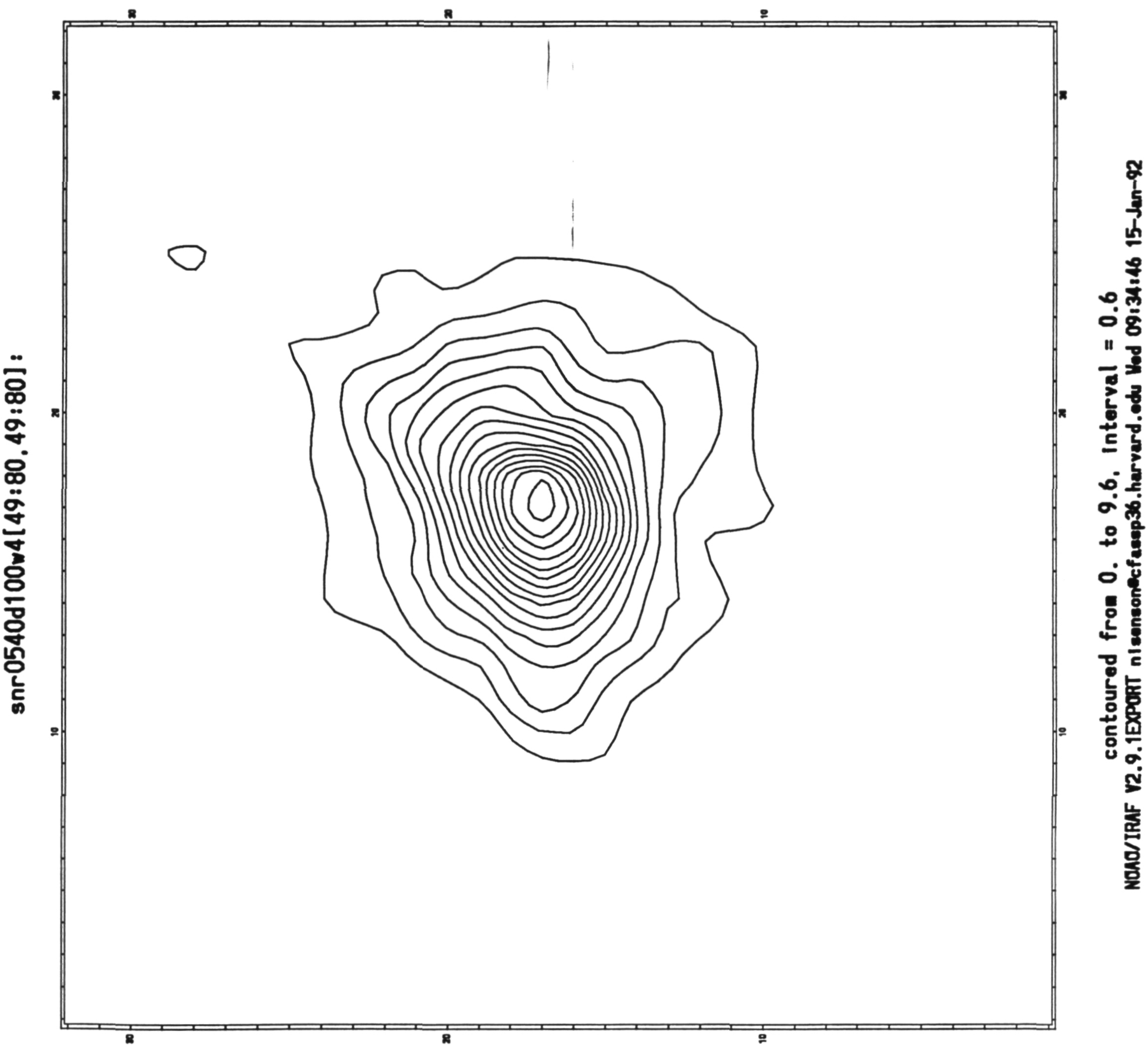

Figure (23) - Contour Plot of the BID Deconvolution of SNR0540

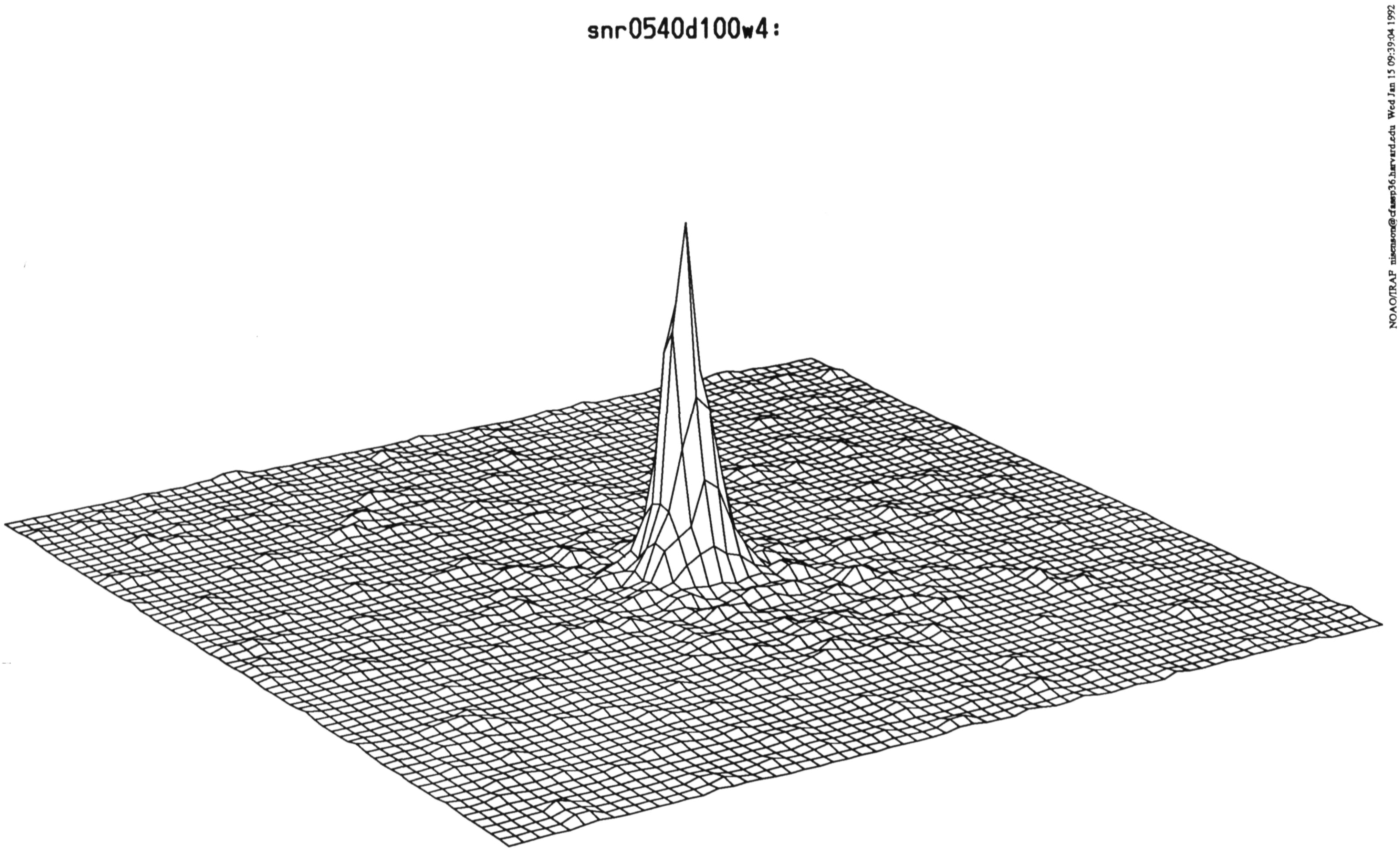

Figure (24) - Surface Plot of the BID Deconvolution of SNR0540

Appendix A

```
        program bid

c last edited: January 6, 1992
c Compile in IRAF using FC
c link with bidsubs, irafsubs, and fourt
c perform blind iterative deconvolution
c using wiener filters only
c needs input image, guess at psf
c
c*******************************************************************
c BID is set up to take square IRAF image arrays of up to 512x512
c Smaller arrays or smaller processing areas are chosen by
c  entering the parameter KSZ.
c Images smaller than KSZ are set in KSZ arrays and the outer
c regions are filled with either zero or and average constant.
c Images larger than KSZ (but smaller than 513x513) and
c  chunked to KSZ area, at an entered center.
c File names are for IRAF files without the .imh.
c 1st guess psf can be either a file or it can be generated with
c  PSFGEN.
c Program runs for MAXC cycles.
c BETA is the percentage (0 to 1.0) of the previous cycle averaged
c  with the current cycle.  Usually 0.8 near the start of the
c  process, but changed to 1.0 near convergence.
c Resulting psf's and images can be read back into the program
c  to perform additional cycles.
c CF is the radius in frequency space used for the Wiener filter
c  noise estimate. 1.0 is a good starting point.
c W1 is the Wiener filter scale factor.  1.0 is a good starting
c  value.  Program DCW allows generation of Wiener filtered
c  versions of the image using the psf from BID, but with different
c  values for W1.  Larger values for W1 give a smoother image,
c  smaller values enhance the high frequencies.
c Output file names should be given without the trailing .imh.
c Object and psf masks are generated using program MAKEMASK.
c Recommended starting masks should be of order twice the diameter
c  of the data and expected psf diameters.  For an object that nearly
c  fills the field, a constant mask (equivalent to no mask) should
c  be used for the object and only a psf mask is used.
c
c If you encounter problems in using this program, contact
c  Peter Nisenson, Center for Astrophysics, (617)495-7394
c

        parameter (isz=512,isz1=isz/2,isz2=isz1+1)

c complex fourier space arrays : c = g*f
        complex   c(isz*isz)
        complex   g(isz*isz)
        complex   f(isz*isz)

c real image space arrays, 1-d here but all routines think they are 2-d
        real    cobj(isz*isz),gref(isz*isz),fout(isz*isz),lastfout(isz*isz)
c masks and a temporary buffer
        real    maskr(isz*isz),mask(isz*isz),tempb(isz*isz)
c total power of input image
        real    power
c two working arrays, one for fourt, one for loadim
        real    work(2*isz),rwk(isz)
c testo,testr,rmsdiff array, for output to file
        real    stats(isz,3)

c temp integer array for loadim :
        integer         iph(isz,isz)
```

```
c nzero and nzeror are number of non-zeros (ones) in mask and maskr :
        integer         nzero,nzeror
c parameters for fourt :
        integer*4       nn(2),ndim,isign,iform
c file names :
        character       name1*80,name2*80,name3*80,name4*80,name5*80

        print *, ' BID : iterative deconvolution given initial '
        print *, ' guess at the psf, uses wiener deconvolution, '
        print *, ' support and positivity constraints to iteratively'
        print *, ' approximate the correct psf ...'
        print *, ''

        print *, ' Enter log file name'
        read (*,*) name5
        open(unit=1,name=name5,type='new',form='formatted')

c initialize ...
        print *, ' enter array dimension (ksz) '
        read (*,*) ksz
        write (1,*) ' array dimension ',ksz
        ksz1=ksz/2
        ksz2=ksz*ksz
        nn(1)=ksz
        nn(2)=ksz
        ndim=2
c
c read in image file, save total power for rescaling output
c
        print *, ' enter input image file name '
        read (*,'(a)') name1
        write (1,*) ' Input file name',name1
        call loadim (name1,cobj,iph,rwk,ksz,3)

        print *, ' 1 for border square mask, 2 for circular, 0 for none'
        read (*,*) isqm
        if (isqm.eq.1)then
          print *,'Enter the sqmask border width, 0 for none'
          read (*,*) iradm
        elseif(isqm.eq.2)then
          print *, ' Enter circular mask radius '
          read (*,*) radm
        endif

c apply square gaussian border
          if (isqm.eq.1)call sqmask(cobj,iradm,ksz)

c apply circular gaussian border
          if (isqm.eq.2)call circmask(cobj,radm,ksz)


        power=0.0
        do j=1,ksz2
            power=power+abs(cobj(j))
        enddo
        print *, ' Total power in image:',power
        write (1,*) ' Total power in image ',power
c
c read in the psf file and center it, scale total power to 1.0
c
        print *, '1 to enter psf file, 2 to generate gaussian'
        read(*,*) igss
        if(igss.eq.1)then
          print *, ' enter input psf file name'
```

```
          read (*,'(a)') name1
          write (1,*) 'Input psf file name ',name1
          call loadim(name1,gref,iph,rwk,ksz,3)
          call centre(gref,tempb,ksz,ksz1)
          call scale(gref,1.0,ksz)
        else
c generate 1st pass psf
c
          print *, 'Generate psf'
          print *, ' Enter 1/e radius for psf '
          read (*,*) wid
          write (1,*) ' Gaussian psf 1/e radius ',wid
          krn=1
          cf=1.0*ksz1
          call gauss(gref,wid,krn,ksz)
          iform=0
          isign=-1

          call copyr(g,gref,isign,ksz)
          call fourt(g,nn,ndim,isign,iform,work)

          call rolloff(g,cf,ksz)
          iform=1
          isign=1
          call fourt(g,nn,ndim,isign,iform,work)
          call copyr(g,gref,isign,ksz)
          call ffmax(amax,amin,imax,jmax,imin,jmin,gref,ksz)
          amm=amax-amin
          do j=1,ksz2
            gref(j)=(gref(j)-amin)/amm
          enddo
        endif
c
c enter reconstruction parameters
c
        print *, ' enter max # cycles for iteration '
        read (*,*) maxc
        write (1,*) ' max # iterations ',maxc

        print *, ' enter beta, the damping factor (try 0.8)'
        read (*,*) beta
        write (1,*) ' beta ',beta

        print *, ' enter radius for noise estimate (0 to 1.0)'
        read (*,*) cf
        write (1,*) ' noise radius ',cf
        cf = cf*ksz1
        icf = cf


        print *, ' enter object wiener filter parameter (Try 1.0)'
        read (*,*) wo1
        write (1,*) ' object Wiener filter multiplier ',wo1

        print *, ' enter psf wiener filter parameter (Try 1.0)'
        read (*,*) wp1
        write (1,*) ' psf Wiener filter multiplier ',wp1

        print *, '1 for circular object support, 2 for threshold,
     +   3 for none'
        read (*,*) ispp

        suppo=0.
        thresh=0.0
```

```
        if (ispp.eq.1)then
          print *,' enter the object support radius, 0 for none'
          read (*,*) suppo
          write (1,*) ' object support radius  ',suppo
        endif
        if (ispp.eq.2)then
          print *, ' enter threshold level for support '
          read (*,*) thresh
          write (1,*) ' object support threshold  ',thresh
        endif

        print *, ' 1 for psf positivity, 2 for just support'
        read (*,*) ipss

        print *,' Enter the psf circular support radius, 0 for none'
        read (*,*) suppr
        write (1,*) ' psf support radius  ',suppr

        print *, ' enter output image file name '
        read (*,'(a)') name2
        write (1,*) ' enter output image file name ',name2

        print *, ' enter output psf file name '
        read (*,'(a)') name3
        write (1,*) ' output psf file name ',name3

        print *, ' enter output statistics file name '
        read (*,'(a)') name4
        write (1,*) ' Output stat file name ',name4
c
c load previous output image or set f,ff to 0.0
c
        print *, ' 1 to enter output image from previous iteration'
        read (*,*) iobj
        if(iobj.eq.1)then
           print *, ' enter file name'
           read (*,'(a)') name1
           write (1,*) ' Old image file name  ',name1
           call loadim(name1,fout,iph,rwk,ksz,3)
           isign=-1
           iform=0
           call copyr(f,fout,isign,ksz)
           call fourt(f,nn,ndim,isign,iform,work)
           call normf(f,ksz)
c ft back to image space so lastfout will be scaled correctly
c for rmsdiff calculations ...
           isign=1
           iform=1
           call fourt(f,nn,ndim,isign,iform,work)
           call copyr(f,lastfout,isign,ksz)
           isign=-1
           iform=0
           call fourt(f,nn,ndim,isign,iform,work)
           call normf(f,ksz)
        else
          call zero(f,ksz)
          do j=1,ksz2
            lastfout(j)=0.0
          enddo
        endif

c
        print *, ' generate masks'
        if (ispp.eq.1.and.suppo.ne.0.)then
```

```
             call maskg(mask,suppo,ksz)
        elseif(ispp.eq.2)then
          do j=1,ksz2
            if(cobj(j).gt.thresh)then
                mask(j) = 1.0
            else
                mask(j)=0.0
            endif
          enddo
        else
              do j=1,ksz2
                mask(j) = 1.0
              enddo
        endif

c setting psf mask to radius of "supp"
        if(suppr.ne.0.)then
              call maskg(maskr,suppr,ksz)
        else
              do j=1,ksz2
                maskr(j) = 1.0
              enddo
        endif

c now count number of 1's in masks
        nzero = 0
        nzeror= 0
        do j=1,ksz2
          if(mask(j).ne.0.)nzero=nzero+1
          if(maskr(j).ne.0.)nzeror=nzeror+1
        enddo

        print *, 'nzero,nzeror',nzero,nzeror
c
c ft and normalize image for first deconvolution pass
c
        iform=0
        isign=-1
        call copyr(c,cobj,isign,ksz)
        call fourt(c,nn,ndim,isign,iform,work)
        call normf(c,ksz)
c
c perform deconvolution in loop
c
        print *, ' .....deconvolving'
        do 1000 iz=1,maxc
c ft, shift and normalize psf ...
          iform=0
          isign=-1
          call copyr(g,gref,isign,ksz)
          call fourt(g,nn,ndim,isign,iform,work)
          call shift(g,ksz)
          call normf(g,ksz)
c output = input deconvolved by psf :
c if first time through, deconvolve with beta=1.0
          if(iobj.ne.1.and.iz.eq.1)then
            call wienerf(c,g,f,ksz,cf,wo1,1.0)
          else
            call wienerf(c,g,f,ksz,cf,wo1,beta)
          endif
c ft output to image space, apply positivity, ft back, normalize
          iform=1
          isign=1
          call fourt(f,nn,ndim,isign,iform,work)
```

```
        call copyr(f,fout,isign,ksz)
        call posf(fout,mask,nzero,testo,testno,ksz)
        print *, ' '
        print *, ' Iteration #',iz
        write (1,*) ' '
        write (1,*) 'Iteration # ',iz
        print *, 'obj: summed neg/pos ',testo,' #neg/#pos ',testno
        write (1,*) 'obj: sum neg/pos ',testo,' #neg/#pos ',testno

c psf = input deconvolved by output ---
        iform=0
        isign=-1
        call copyr(f,fout,isign,ksz)
        call fourt(f,nn,ndim,isign,iform,work)
        call normf(f,ksz)
        call wienerf(c,f,g,ksz,cf,wp1,beta)
        call shift(g,ksz)
        iform=1
        isign=1
c ft psf to image space, centre, apply positivity
        call fourt(g,nn,ndim,isign,iform,work)
        call copyr(g,gref,isign,ksz)
        call centre(gref,tempb,ksz,ksz1)

        if(ipss.eq.1)then
          call posf(gref,maskr,nzeror,testr,testnr,ksz)
        else
          call support(gref,maskr,testr,testnr,ksz2)
        endif

        print *, 'psf: summed neg/pos ',testr,' #negs/#pos ',testnr
        write (1,*) 'psf: sum neg/pos ',testr,' #negs/#pos ',testnr
c
c testo is sum of negatives over sum of positives in output
c prior to positivity, testr is same for psf ...
c
        rmd = rmsdiff(lastfout,fout,ksz)
        print *, ' rmsdiff :',rmd
        write (1,*) ' rmsdiff :',rmd
        stats(iz,1)=testo
        stats(iz,2)=testr
        stats(iz,3)=rmd

1000    continue
c
c stop : scale total power of output to input power, psf to 1.0
c then save
c
2000    continue
        call scale(fout,power,ksz)
        rtot=0.0
        do j=1,ksz2
          rtot=rtot+fout(j)
        enddo
        print *, ' total power in image ',rtot
        call saveim(name2,fout,ksz,3)

        call scale(gref,1.0,ksz)
        call saveim(name3,gref,ksz,3)

        call saverect(name4,stats,ksz,3,ksz,3)

      stop
      end
```

```
      program dcw
c last edited: November 4, 1991
c Compile in IRAF using FC
c link with:  bidsubs, irafsubs, and fourt
c uses wiener deconvolution
c Allows application of a edge mask for images which extend
c beyond the frame boundary
c Best results usually obtained with square low-pass filter
c and cutoff of 1.0
c Smoothness of output image is adjusted by chnaging the wiener
c multiplier - greater value for smoother image.
c
c
      parameter (isz=512,isz1=isz/2,isz2=isz1+1)
      real*4 work(2*isz)
      real*4 ph(isz*isz),rwk(isz)
      complex snc(isz*isz),refc(isz*isz),sncft(isz*isz),sncdc(isz*isz)
      integer nn(2)
      integer iph(isz*isz)
      character name1*80,name2*80,name3*80

      print *, ''
      print *, ' ---- deconvolution using wiener filter ----'
      print *, ' ---- wiener parameter w1 : 0 => straight deconvolution ----'
      print *, ' ---- w1 >> 0 gives noise reduction ----'
      print *, ' ---- can truncate negatives at end or apply the ----'
      print *, ' ---- fienup algorithm to the output ----'
      print *, ''

      print *, ' enter input object file name '
      read (*,'(a)') name1
      print *, ' enter image dimension '
      read (*,*) ksz
      ksz2=ksz*ksz

      call loadim(name1,ph,iph,rwk,ksz,3)
      print *, ' 1 for border square mask, 2 for circular, 0 for none'
      read (*,*) isqm
      if (isqm.eq.1)then
        print *,'Enter the sqmask border width, 0 for none'
        read (*,*) iradm
      elseif(isqm.eq.2)then
        print *, ' Enter circular mask radius '
        read (*,*) radm
      endif

c apply square gaussian border
        if (isqm.eq.1)call sqmask(ph,iradm,ksz)

c apply circular gaussian border
        if (isqm.eq.2)call circmask(ph,radm,ksz)


      rintot=0.0
      do j=1,ksz2
              snc(j)=ph(j)
              rintot=rintot+ph(j)
      enddo
      print *, ' total power in image ',rintot
      print *, ' enter input reference file name '
      read (*,'(a)') name2
      call loadim(name2,ph,iph,rwk,ksz,3)
      do j=1,ksz2
```

```fortran
                refc(j)=ph(j)
        enddo

        print *, ' enter wiener filter multiplier, 1 is default'
        read (*,*) w1
        print *, ' enter radius for wiener noise estimate (0 to 1.0) '
        read (*,*) wr
        wr=wr*ksz/2.0
        print *, ' enter low frequency cutoff, (0 to 1.0) '
        read (*,*) cf
        cf=cf*ksz/2.0
        print *, ' 1 for circular low pass filter, 2 for square filter, or 0'
        read (*,*) iflt

c set up parameters for ft
        ndim=2
        iform=0
        nn(1)=ksz
        nn(2)=ksz
        isign=-1

c ft image
        print *, ' transforming image ...'
        call fourt(snc,nn,ndim,isign,iform,work)
c ft psf
        print *, ' transforming reference ...'
        call fourt(refc,nn,ndim,isign,iform,work)

        do j=1,ksz2
                sncft(j)=snc(j)
        enddo

c perform wiener deconvolution
600     print *, ' doing wiener deconvolution ...'
        call wienerd(snc,refc,ksz,wr,w1)
        do j=1,ksz2
                sncdc(j)=snc(j)
        enddo

c low pass filter
650     if(cf.ne.0.)call filt(snc,ksz,cf,iflt)

c shift image to center of field
        call shift(snc,ksz)

        iform=1
        isign=1
c ft back to image space
        print *, ' transforming back to image space ...'
        call fourt(snc,nn,ndim,isign,iform,work)

            do j=1,ksz2
                if(real(snc(j)).lt.0.0)snc(j)=0.0
            enddo

        do j=1,ksz2
            ph(j)=real(snc(j))
        enddo

        routtot=0.0
        do j=1,ksz2
                routtot=routtot+ph(j)
        enddo
        print *, 'output power before renorm ',routtot
```

```
        rsfact=rintot/routtot
        do j=1,ksz2
                ph(j)=ph(j)*rsfact
        enddo
        routtot=0.0
        do j=1,ksz2
                routtot=routtot+ph(j)
        enddo
        print *, ' total output power ', routout
        print *, ' enter output file name '
        read (*,'(a)') name3
        call saveim(name3,ph,ksz,3)

c again?
        print *, ' 0 to quit, 1 to redo with new w1, 2 to redo with new cf'
        read (*,*) ilp
        if (ilp .eq. 0) goto 99999
        if (ilp .eq. 1) then
           print *, ' enter wiener filter multiplier, 1 is default'
           read (*,*) w1
           print *, ' enter radius for wiener noise estimate (0 to 1.0) '
           read (*,*) wr
           wr = wr*ksz/2.0
           print *, ' enter low frequency cutoff, (0 to 1.0)'
           read (*,*) cf
           cf = cf*ksz/2.0
           print *, ' 1 for circular low pass filter, 2 for square filter '
           read (*,*) iflt
           do j=1,ksz2
                snc(j)=sncft(j)
           enddo
           goto 600
        endif
        if (ilp .eq. 2) then
           print *, ' enter low frequency cutoff (0 to 1.0)'
           read (*,*) cf
           cf = cf*ksz/2.0
           print *, ' 1 for circular low pass filter, 2 for square filter '
           read (*,*) iflt
           do j=1,ksz2
                snc(j)=sncdc(j)
           enddo
           goto 650
        endif


99999   stop
        end
```

```
c  Irafsubs.f
c        loadim : load an image (iraf, real or integer)
c        saveim : save an image (iraf or real)
c        saverect : save a rectangle chunked from corner of larger
c                rectangle to iraf file (for saving convergence statistics)

         subroutine loadim(name,a,itmp,rwk,isz,ftyp)

c last edited july30,1990
c loads iraf files into fortran program
c itmp is an integer array, iszxisz, for loading integer
c images, rwk is a one-row working real array,
c ftyp is file-type : 1 means integer, 2 means real, 3 means iraf.
c

         character name*80
         real a(isz,isz)
         integer itmp(isz,isz)
         real rwk(isz)
         integer ftyp
         integer im
         integer axl(7)
         integer naxis
         integer dtype
         integer ier
         integer xcentr, ycentr, i1, i2, j1, j2
         integer ist,iend,jst,jend,idim,jdim
         real bsum
         isz4=isz*isz*2
         isz6=isz4*2
c
c
       if (ftyp .eq. 1) then
         open (unit=1,file=name,access='direct',status='old',
     +         form='unformatted', recl=isz6)
         read (unit=1,rec=1) itmp

           open(unit=1, file=name, status='old', form='unformatted',
     +          access='direct',recl=isz6)
           read(unit=1) itmp
           do j=1,isz
             do i=1,isz
                   a(i,j) = itmp(i,j)
               end do
           end do
           close(unit=1)
           return
       else if (ftyp .eq. 2) then
         open (unit=1,file=name,access='direct',status='old',
     +         form='unformatted', recl=isz6)
         read (unit=1,rec=1) a
           close(unit=1)
           return
       else if (ftyp .eq. 3) then
           call imopen (name, 1, im, ier)
           if (ier .ne. 0) goto 99999
           call imgsiz (im, axl, naxis, dtype, ier)
           if (ier .ne. 0) goto 99999
           if ((axl(1) .gt. isz) .or. (axl(2) .gt. isz)) then
               print *, ' image bigger than array size :'
               print *, ' axl(1): ', axl(1),' ... axl(2): ', axl(2)
               write (*,*) 'input centre of extraction box'
               read (*,*) xcentr, ycentr
               xcentr = min ((axl(1) - isz/2), max (isz/2, xcentr))
```

```
          ycentr = min ((axl(2) - isz/2), max (isz/2, ycentr))
          i1 =  xcentr - isz / 2 + 1
          i2 =  i1 + isz - 1
          j1 = ycentr - isz / 2 + 1
          j2 = j1 + isz - 1
            call imgs2r (im, a, i1, i2, j1, j2, ier)
          if (ier .ne. 0) goto 99999
      else if ((axl(1) .lt. isz) .or. (axl(2) .lt.isz)) then
            idim = axl(1)
            jdim = axl(2)
            ist = (isz-idim)/2 + 1
            iend = ist+axl(1)-1
            jst = (isz-jdim)/2 + 1
            jend = jst+axl(2)-1
            do j=jst,jend
                call imgs2r(im,rwk,1,axl(1),1+j-jst,1+j-jst,ier)
                if (ier .ne. 0) goto 99999
                do i=ist,iend
                    a(i,j)=rwk(i-ist+1)
                end do
            end do
            bsum=0.0
            print *, ' image smaller than array size :'
            print *,
     &' 1 to calculate average background, 0 to set to zero'
            read (*,*) iback
            if (iback .eq. 0) goto 500
            do i=ist,iend
                bsum = bsum + a(i,jst)+a(i,jend)
            end do
            do j=jst,jend
                bsum = bsum + a(ist,j)+a(iend,j)
            end do
            bsum = bsum/(2.*jdim + 2.*idim)

            write (*,*) ' average background : ',bsum
500         if (jst .gt. 1) then
                do j=1,jst-1
                    do i=1,isz
                        a(i,j)=bsum
                    end do
                end do
            end if
            if (ist .gt. 1) then
                do i=1,ist-1
                    do j=jst,jend
                        a(i,j)=bsum
                    end do
                end do
            end if
            if (iend .lt. isz) then
                do i=iend+1,isz
                    do j=jst,jend
                        a(i,j)=bsum
                    end do
                end do
            end if
            if (jend .lt. isz) then
              do j=jend+1,isz
                do i=1,isz
                    a(i,j)=bsum
                end do
              end do
            end if
```

```
          else
              i1 = 1
              i2 = axl(1)
              j1 = 1
              j2 = axl(2)
                call imgs2r (im, a, i1, i2, j1, j2, ier)                    if (ier .ne. 0) go
to 99999
          end if
          call imclos (im, ier)
          if (ier .ne. 0) goto 99999
          return
      end if
c
99999   call imemsg (ier,name)
          write (*, 2222) name
2222      format('error: ',a80)
        stop
        end
c
c---------------------------------------------------------------------------
c
        subroutine saveim(name,a,isz,ftyp)

c last edited: July 30, 1991
c saves images as iraf files

        character name*80
        real a(isz,isz)
        integer isz
        integer ftyp
        integer axl(7),naxis,im,ier
        if (ftyp .eq. 3) then
                axl(1)=isz
                axl(2)=isz
                naxis=2
                call imcrea (name,axl,naxis,6,ier)
                if (ier .ne. 0) goto 99998
                call imopen (name,3,im,ier)
                if (ier .ne. 0) goto 99998
                call imps2r (im,a,1,isz,1,isz,ier)
                if (ier .ne. 0) goto 99998
                call imclos (im,ier)
                if (ier .ne. 0) goto 99998
                return
        else
                open (unit=1,file=name,status='new',form='unformatted',
     +            access='direct',recl=4*isz*isz)
                write (unit=1,rec=1) a
                close (unit=1)
                return
        end if
99998   call imemsg (ier,name)
          write (*, 2222) name
2222      format('error: ',a80)
        stop
        end

c *****************************************************************
c
        subroutine saverect(name,a,d1,d2,rd1,rd2)

c save a few lines of data for convergence plots
        character name*80
        integer d1,d2,rd1,rd2
```

```
      real a(rd1,rd2)
      integer axl(7),naxis,im,ier
      print *, 'd1:',d1,', d2:',d2,', rd1:',rd1,', rd2:',rd2
      axl(1)=d1
      axl(2)=d2
      naxis=2
      call imcrea (name,axl,naxis,6,ier)
      if (ier .ne. 0) goto 99997
      call imopen (name,3,im,ier)
      if (ier .ne. 0) goto 99997
      call imps2r (im,a,1,d1,1,d2,ier)
      if (ier .ne. 0) goto 99997
      call imclos (im,ier)
      if (ier .ne. 0) goto 99997
      return
99997 call imemsg (ier,name)
      write (*,'("error: ", a80)') name
      stop
      end
c
```

```
c
c bidsubs.f : subroutines for idconf and makemask, etc...
c
c last edited December 6, 1991
c
c ************************************************************************
c
c routines are :
c       cent :   Finds center of mass
c       centre : centres the object using ffmax to find psf center
c       circmask: multiply image by circular rolloff mask
c       copyr : copies real array into complex array or vice versa,
c               depending on value of isign
c       cwind : a rolloff window function in f-space
c       dtr:    removes trend from solar images before BID
c       ffmax : finds position and values of minimum and maximum pixels
c               in image
c       filt : apply square or circular rolloff filter using cwind at
c               given radius ...
c       maskg: Generates gaussian masked image
c       normf : normalizes in fourier space so a(1,1)=1
c       pcent : print array center
c       posf : the old version (and best working, it seems) of pos,
c               iteratively truncates negatives and adds them back in ...
c       rmsdiff : rmsdiff between two images (real), then stuffs second
c               image into first ...
c       rolloff : rolls off high frequencies using cwind
c       scale : scales image to given power level
c       shift : reverses sign of every other element in ft, shifts
c               object from center to corners or vice versa ...
c       sqmask: Generate edge rolloff mask for solar images
c       subtract:  Subtracts low frequencies for solar detrending
c       support:  Applies image plane support constraint for BID
c       wienerd : wiener filter deconvolution, with averaging in of
c               previous f  (f = (1.0-beta)*f+beta*c/g)
c       wienerf : another version of wienerd
c       zero : zeros a complex array
c
c ********************************************************************
c
      subroutine cent(ci,cj,tm,ph,isz)
c
c calculates center of mass of iszxisz image
c
c xm = x coordinate of centre of mass
c ym = y coordinate of centre of mass
c tm = running sum of values in image array
c ph() = image array
c isz = characteristic image dimension (ie. 128)
c
      dimension ph(isz,isz)
      real    copy

      call ffmax(amax, amin,imax,jmax,imin,jmin, ph, isz)
      amm=0.3*amax

      z1=isz/2
      z2=isz1+1
      tm=0.
      xc1=0.
      yc1=0.
      do j=1,isz
        do i=1,isz
          copy = ph(i,j)
```

```
          if (copy.ge.amm) then
              tm=tm+copy
              xc1=copy*i+xc1
              yc1=copy*j+yc1
          endif
        enddo
      enddo

      ci=xc1/tm
      cj=yc1/tm
      print *, ' center at ',ci,cj

      return
      end

c
c *****************************************************************
c
      subroutine centre(a, b, isz, isz1)
      real a(isz, isz), b(isz,isz)

      call cent(ci, cj, t, a, isz)

      do j = 1, isz
      do i = 1, isz
      b(i,j) = 0.0
      end do
      end do
      jr = (nint(cj) - isz1) - .5
      ir = (nint(ci) - isz1) - .5
      do j = 1, isz
      do i = 1, isz
      i1 = i + ir

c centre object

      j1 = j + jr
      if ((((i1 .le. isz) .and. (i1 .ge. 1)) .and. (j1 .le. isz)) .and.
     &(j1 .ge. 1)) then
      b(i,j) = a(i1,j1)
      end if
      end do
      end do
      do j = 1, isz
      do i = 1, isz
      a(i,j) = b(i,j)
      end do
      end do
      return
      end
c
c ********************************************************************
        subroutine circmask(obj,radm,ksz)
        real obj(ksz,ksz),radm

        ksz1=ksz/2
        ksz2=ksz1+1
        rr=ksz2-radm
        if(rr.lt.1)rr=1.
          do j=1,ksz
            do i=1,ksz
              r=sqrt(1.*(ksz2-i)**2+1.*(ksz2-j)**2)
                if(r.ge.radm)then
                  obj(i,j)=obj(i,j)*cwind(abs(r-radm)/rr)
```

```
                endif

            enddo
          enddo

        return
        end

c *******************************************************

c
        subroutine copyr(a,ar,isign,isz)
        complex a(isz,isz)
        real    ar(isz,isz)
        if(isign.eq.1)then
c copy into real array for pos
          do j=1,isz
            do i=1,isz
                ar(i,j)=real(a(i,j))
            enddo
          enddo
        else
c copy into real array for pos
          do j=1,isz
            do i=1,isz
                a(i,j)=ar(i,j)
            enddo
          enddo
        endif

        return
        end

c
c ********************************************************************
      function cwind(relr)
c       A window for fft's that has low sidelobes
      real c(4)
      data c / .074, .302, .233, .390 /
      if (relr .lt. 0.) stop 'cwind'
c careful ...
      if (relr .gt. 1.5) then
      cwind = 0.
      else
c (1-r**2), r=0... 1.0
      r = 1. - (relr * relr)
      r2 = r * r
      r3 = r2 * r
      cwind = ((c(1) + (r * c(2))) + (r2 * c(3))) + (r3 * c(4))
c let go all the way to zero
      if (cwind .lt. 0.) cwind = 0.
      end if
      return
      end
c
c************************************************************

        subroutine dtr(a,work,ac,acf,ksz,fc,dc)
c subtract out the low frequency trend in an image, then add dc to
c make image positive ...
        real a(ksz*ksz),work(2*ksz)
        complex ac(ksz*ksz),acf(ksz*ksz)
        integer*4 nn(2),ndim,isign,iform
```

```
        power=0.0
        ksz2=ksz*ksz
        do j=1,ksz2
            ac(j)=(0.,0.)
            ac(j)=a(j)
        enddo
c
        ndim=2
        nn(1)=ksz
        nn(2)=ksz
        iform=0
        isign=-1
        call fourt(ac,nn,ndim,isign,iform,work)
        do j=1,ksz2
            acf(j)=ac(j)
        enddo
        call filt(acf,ksz,fc,2)
        call subtract(ac,acf,ksz)
        iform=1
        isign=1
        call fourt(ac,nn,ndim,isign,iform,work)
        do j=1,ksz2
            a(j)=real(ac(j)/ksz2)
        enddo
        call ffmax(amax,amin,imax,jmax,imin,jmin,a,ksz)
        do j=1,ksz2
                a(j)=a(j)-amin
        enddo
        dc=-amin
        return
        end
c
c **************************************************************
c
      subroutine ffmax(amax, amin, imax, jmax, imin, jmin, ph, isz)
      dimension ph(isz, isz)
      amax = ph(1,1)
      amin = ph(1,1)
      do j = 1, isz
      do i = 1, isz
      if (ph(i,j) .gt. amax) then
      amax = ph(i,j)
      imax = i
      jmax = j
      end if
      if (ph(i,j) .lt. amin) then
      amin = ph(i,j)
      imin = i
      jmin = j
      end if
      end do
      end do
      return
      end

c
c ***********************************************
c
        subroutine filt(cobj,isz,cf,isq)
        complex cobj(isz,isz)
        isz1=isz/2

c low pass filter
c
```

```fortran
650     if(cf.ne.0.0.and.isq.eq.1)then
c circular filter
          print *, 'circular low pass filter ...'
         do 100 j=1,isz1
           do 100 i=1,isz1
               d1=sqrt((i-1.)**2+(j-1.)**2)/cf
               d2=sqrt((i-1.)**2+j**2)/cf
               d3=sqrt(i**2+(j-1.)**2)/cf
               d4=sqrt(1.*i**2+j**2)/cf
               cobj(i,j)=cwind(d1)*cobj(i,j)
               cobj(i,isz+1-j)=cwind(d2)*cobj(i,isz+1-j)
               cobj(isz+1-i,j)=cwind(d3)*cobj(isz+1-i,j)
               cobj(isz+1-i,isz+1-j)=cwind(d4)*cobj(isz+1-i,isz+1-j)
100      continue
        endif
c
        if(cf.ne.0.0.and.isq.eq.2)then
c square filter
         print *, 'square low pass filter'
         do 200 j=1,isz1
           do 200 i=1,isz1
               d1x=(i-1.)/cf
               d1y=(j-1.)/cf
               d2x=(i-1.)/cf
               d2y=j/cf
               d3x=i/cf
               d3y=(j-1.)/cf
               d4x=i/cf
               d4y=j/cf
               cobj(i,j)=cwind(d1x)*cwind(d1y)*cobj(i,j)
               cobj(i,isz+1-j)=cwind(d2x)*cwind(d2y)*cobj(i,isz+1-j)
               cobj(isz+1-i,j)=cwind(d3x)*cwind(d3y)*cobj(isz+1-i,j)
               cobj(isz+1-i,isz+1-j)=cwind(d4x)*cwind(d4y)
     +                  *cobj(isz+1-i,isz+1-j)
200      continue
        endif
c
c

        return
        end



c ******************************************************
        subroutine gauss(a,wid,krn,isz)
        real a(isz,isz)
        real wid
        integer seed
            seed=13972
        isz1=isz/2
        isz2=isz1+1

          if (krn .eq. 1) then
             x=rand(seed)
             do j=1,97
             x=rand(0)
             enddo
          endif

            wid2=wid*wid

            do j=1,isz
              do i=1,isz
```

```
            r2=((isz2-i)**2+(isz2-j)**2)
            r3=r2/wid2
            if (r3 .le. 50) then
              if(krn.eq.1) then
                rr=rand(0)
              else
                rr=1.
              endif
              a(i,j)=exp(-r2/wid)*rr
            endif
          enddo
        enddo
      return
      end

c ******************************************************
      subroutine maskg(m,rd,ksz)
      real    m(ksz,ksz)

      ksz1=ksz/2
      do j=1,ksz
        do i=1,ksz
            r=sqrt(float(ksz1-i)**2+float(ksz1-j)**2)
            if(r.le.rd)then
              m(i,j)=1.0
            else
              m(i,j)=0.0
            endif
        enddo
      enddo
      return
      end

c ********************************************************
c
      subroutine normf(a, isz)
      complex a(isz, isz)
      a1 = cabs(a(1,1))
      if (a1 .eq. 0.0) then
        print *, ' error in normf: a(1,1)=0.0'
        return
      endif
      do j = 1, isz
        do i = 1, isz
            a(i,j) = a(i,j) / a1
        end do
      end do
      return
      end
c
c*******************************************************
      subroutine pcent(a,ksz)
      real a(ksz,ksz)
      do j=ksz/2-3,ksz/2+4
        print 111, (a(i,j),i=ksz/2-3,ksz/2+4)
      enddo
111   format(8e10.2)
      return
      end


c ********************************************************************
c
      subroutine posf(a, b, nzero, test, testn, isz)
```

```
c modify 12/23/1991

      real a(isz, isz), b(isz, isz)
      integer nzero

c 1st apply support constraint
        if(nzero.ne.0)then
          do j = 1, isz
            do i = 1, isz
              a(i,j) = (a(i,j) * b(i,j)) / nzero
            end do
          end do
        else
          print *, ' mask is all zeros'
          return
        endif

c then truncate negatives
        do jz=1,3

          sump = 0.0
          sumn = 0.0
          npos = 0
          nneg = 0
          do j = 1, isz
            do i = 1, isz
              if (a(i,j) .lt. 0.0) then
                  sumn = sumn + a(i,j)
                  a(i,j) = 0.0
                  nneg=nneg+1
              elseif (a(i,j) .gt. 0.0)then
                  sump = sump + a(i,j)
                  npos = npos + 1
              end if
            end do
          end do
          if (jz .eq. 1)then
            if(sump.ne.0)then
             test = abs(sumn / sump)
            else
             print *,' Image all negs'
             return
            endif
            if(npos.ne.0)then
             testn=float(nneg)/npos
            else
             print *, ' No positives in image'
            endif
          endif

c now add in negatives to keep energy constant
          ss = sumn /nzero
          do j = 1, isz
            do i = 1, isz
              if (b(i,j) .eq. 1.0) then
                  a(i,j) = a(i,j) + ss
              end if
            end do
          end do

        enddo

c now  find min and max and rescale
```

```
c        call ffmax(amax,amin,imax,jmax,imin,jmin,a,isz)
c       amm=amax-amin
        do j=1,isz
          do i=1,isz
c           a(i,j)=(a(i,j)-amin)/amm
            if(a(i,j).lt.0.)a(i,j)=0.0
          enddo
        enddo

        return
        end
c
c ************************************************************
c
        function rmsdiff(a1,a2,isz)
        real a1(isz,isz), a2(isz,isz)
        fsum = 0.0
        do j = 1, isz
            do i = 1, isz
                 fsum = fsum + (a1(i,j) - a2(i,j))**2
                 a1(i,j) = a2(i,j)
            end do
        end do
        rmsdiff = sqrt (fsum)
        return
        end
c
c ****************************************************
c
        subroutine rolloff(a,cf,isz)
        complex a(isz,isz)
        real cf
        isz1=isz/2
        do j=1,isz1
          do i=1,isz1
            d1=sqrt((i-1.)**2+(j-1.)**2)/cf
            d2=sqrt((i-1.)**2+(j)**2)/cf
            d3=sqrt((i)**2+(j-1.)**2)/cf
            d4=sqrt((1.*i)**2+(1.*j)**2)/cf
            a(i,j)=a(i,j)*cwind(d1)
            a(i,isz+1-j)=a(i,isz+1-j)*cwind(d2)
            a(isz+1-i,j)=a(isz+1-i,j)*cwind(d3)
            a(isz+1-i,isz+1-j)=a(isz+1-i,isz+1-j)*cwind(d4)
          enddo
        enddo
        return
        end
c
c *****************************************************************
c
c
c
        subroutine scale(a,power,isz)
        real a(isz,isz), power
        real sum
        sum=0.0
        do j=1,isz
          do i=1,isz
            sum = sum + abs(a(i,j))
          enddo
        enddo
        rsc = power/sum
        do j=1,isz
          do i=1,isz
```

```
          a(i,j) = a(i,j)*rsc
        enddo
      enddo
      return
      end

c
c *****************************************************************
c
      subroutine shift(ph,nar)
c
c     shifts an image from origin to center by negating every other
c      frequency
      complex ph(nar,nar)
c
      if(mod(nar,2).eq.0)then
        n1=nar-1
        n2=nar
      else
        n1=nar
        n2=nar-1
      endif

      do j=2,n2,2
        do i=1,n1,2
              ph(i,j)=-ph(i,j)
              ph(j,i)=-ph(j,i)
        enddo
      enddo
c
      return
      end
c

c ****************************************************
      subroutine sqmask(obj,iradm,isz)
      real obj(isz,isz)
      integer iradm


      do j=1,isz
         do i=1,iradm
             clc=(1.*iradm-i)/(1.*iradm)
             obj(i,j)=obj(i,j)*cwind(clc)
             obj(j,i)=obj(j,i)*cwind(clc)
             obj(isz-i+1,j)=obj(isz-i+1,j)*cwind(clc)
             obj(j,isz-i+1)=obj(j,isz-i+1)*cwind(clc)
          enddo
       enddo
       return
      end

c ***************************************************

c
      subroutine subtract(ac,acf,ksz)
      complex ac(ksz,ksz),acf(ksz,ksz)
      do j=1,ksz
        do i=1,ksz
          ac(i,j)=ac(i,j)-acf(i,j)
        enddo
      enddo
c     ac(1,1)=cabs(acf(1,1))
      return
```

```
      end
c
c*******************************************************************

          subroutine support(a,mask,test,testn,ksz2)
c Apply support only to psf
          real a(ksz2), mask(ksz2)
          sump=0.
          sumn=0.
          npos=0
          nneg=0
          do j=1,ksz2
            a(j)=a(j)*mask(j)
            if(a(j).gt.0.)then
              sump=sump+a(j)
              npos=npos+1
            endif
            if(a(j).lt.0.)then
              sumn=sumn+a(j)
              nneg=nneg+1
            endif
          enddo
          test=abs(sumn/sump)
          testn=float(nneg)/npos

          return
          end



c ****************************************************
        subroutine wienerd(sig,ref,isz,cf,w1)
        complex sig(isz,isz)
        complex ref(isz,isz)
c
        isz1=isz/2
c 1st calc const level for wiener minimum
        icf=cf
        if(icf.gt.isz1)icf=isz1
        at=0.0
        nn=0
        do 100 j=1,isz1
          do 100 i=1,isz1
            id1=sqrt((i-1.)**2+(j-1.)**2)+0.5
            id2=sqrt((i-1.)**2+(j)**2)+0.5
            id3=sqrt((i)**2+(j-1.)**2)+0.5
            id4=sqrt((1.*i)**2+(1.*j)**2)+0.5
            if(id1.eq.icf)then
              at = at+cabs(ref(i,j))
              nn=nn+1
            endif
            if(id2.eq.icf)then
              at = at+cabs(ref(i,isz+1-j))
              nn=nn+1
            endif
            if(id3.eq.icf)then
              at = at+cabs(ref(isz+1-i,j))
              nn=nn+1
            endif
            if(id4.eq.icf)then
              at = at+cabs(ref(isz+1-i,isz+1-j))
              nn=nn+1
            endif
100     continue
```

```
c
        at=w1*at/nn
        print *, ' Wiener filter constant ',at
        at2=at*at
c
c perform wiener division
        do 200 j=1,isz
          do 200 i=1,isz
            bb1=cabs(ref(i,j))**2+at2
            if(bb1.ne.0.0)then
               sig(i,j)=sig(i,j)*conjg(ref(i,j))/(bb1)
            else
               sig(i,j) =0.0
            end if
200     continue
c
        return
        end

c
c ***********************************************************************

        subroutine wienerf(c,g,f,isz,cf,w1,beta)
c wiener filter for BID.  Adds fraction of previous iteration to
c wiener filtered result

        complex c(isz,isz)
        complex g(isz,isz),f(isz,isz)
        real beta,w1,cf

        isz1=isz/2
c 1st calc const level for wiener minimum
        icf=cf
        if(icf.gt.isz1)icf=isz1
        at=0.0
        nn=0
        do j=1,isz1
          do i=1,isz1
            id1=sqrt((i-1.)**2+(j-1.)**2)+0.5
            id2=sqrt((i-1.)**2+(j)**2)+0.5
            id3=sqrt((i)**2+(j-1.)**2)+0.5
            id4=sqrt((1.*i)**2+(1.*j)**2)+0.5
            if(id1.eq.icf)then
              at = at+cabs(g(i,j))
              nn=nn+1
            endif
            if(id2.eq.icf)then
              at = at+cabs(g(i,isz+1-j))
              nn=nn+1
            endif
            if(id3.eq.icf)then
              at = at+cabs(g(isz+1-i,j))
              nn=nn+1
            endif
            if(id4.eq.icf)then
              at = at+cabs(g(isz+1-i,isz+1-j))
              nn=nn+1
            endif
          enddo
        enddo

        at=w1*at/nn
        at2=at*at
```

```
c perform wiener division
        do j=1,isz
          do i=1,isz
            bb1=cabs(g(i,j))**2+at2
            if(bb1.ne.0.0)then
               f(i,j)=(1.0-beta)*f(i,j)+beta*c(i,j)*conjg(g(i,j))/(bb1)
            end if
          enddo
        enddo

        return
        end
c
c***********************************************************
c
c
        subroutine zero(a,isz)
        complex a(isz,isz)
        do j=1,isz
          do i=1,isz
            a(i,j)=0.0
          enddo
        enddo
        return
        end
```

```
c  Fourt.f
c   NN, ISIGN AND IFORM MUST ALL
c    BE DIMENSIONED INTEGER*4 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c     THE FAST FOURIER TRANSFORM IN USASI BASIC FORTRAN
c
c
c
c     TRANSFORM(J1,J2,...) = SUM(DATA(I1,I2,...)*W1**((I1-1)*(J1-1))
c
c                                 *W2**((I2-1)*(J2-1))*...),
c
c     WHERE I1 AND J1 RUN FROM 1 TO NN(1) AND W1=EXP(ISIGN*2*PI*
c
c     SQRT(-1)/NN(1)), ETC.  THERE IS NO LIMIT ON THE DIMENSIONALITY
c
c     (NUMBER OF SUBSCRIPTS) OF THE DATA ARRAY.  IF AN INVERSE
c
c     TRANSFORM (ISIGN=+1) IS PERFORMED UPON AN ARRAY OF TRANSFORMED
c
c     (ISIGN=-1) DATA, THE ORIGINAL DATA WILL REAPPEAR,
c
c     MULTIPLIED BY NN(1)*NN(2)*...  THE ARRAY OF INPUT DATA MAY BE
c
c     REAL OR COMPLEX, AT THE PROGRAMMERS OPTION, WITH A SAVING OF
c
c     UP TO FORTY PER CENT IN RUNNING TIME FOR REAL OVER COMPLEX.
c
c     (FOR FASTEST TRANSFORM OF REAL DATA, NN(1) SHOULD BE EVEN.)
c
c     THE TRANSFORM VALUES ARE ALWAYS COMPLEX, AND ARE RETURNED IN THE
c
c     ORIGINAL ARRAY OF DATA, REPLACING THE INPUT DATA.  THE LENGTH
c
c     OF EACH DIMENSION OF THE DATA ARRAY MAY BE ANY INTEGER.  THE
c
c     PROGRAM RUNS FASTER ON COMPOSITE INTEGERS THAN ON PRIMES, AND IS
c
c     PARTICULARLY FAST ON NUMBERS RICH IN FACTORS OF TWO.
c
c
c
c     TIMING IS IN FACT GIVEN BY THE FOLLOWING FORMULA.  LET NTOT BE THE
c
c     TOTAL NUMBER OF POINTS (REAL OR COMPLEX) IN THE DATA ARRAY, THAT
c
c     IS, NTOT=NN(1)*NN(2)*...  DECOMPOSE NTOT INTO ITS PRIME FACTORS,
c
c     SUCH AS 2**K2 * 3**K3 * 5**K5 * ...  LET SUM2 BE THE SUM OF ALL
c
c     THE FACTORS OF TWO IN NTOT, THAT IS, SUM2 = 2*K2.  LET SUMF BE
c
c     THE SUM OF ALL OTHER FACTORS OF NTOT, THAT IS, SUMF = 3*K3+5*K5+..
c
c     THE TIME TAKEN BY A MULTIDIMENSIONAL TRANSFORM ON THESE NTOT DATA
c
c     POINT ADD TIME = SIX MICROSECONDS), T = 3000 + NTOT*(600+40*SUM2+
c
c     IS T = T0 + NTOT*(T1+T2*SUM2+T3*SUMF).  ON THE CDC 3300 (FLOATING
c
c     175*SUMF) MICROSECONDS ON COMPLEX DATA.
c
c     IMPLEMENTATION OF THE DEFINITION BY SUMMATION WILL RUN IN A TIME
c
c
```

```
C
C     PROPORTIONAL TO NTOT*(NN(1)+NN(2)+...).  FOR HIGHLY COMPOSITE NTOT
C
C     THE SAVINGS OFFERED BY THIS PROGRAM CAN BE DRAMATIC.  A ONE-DIMEN-
C
C     SIONAL ARRAY 4000 IN LENGTH WILL BE TRANSFORMED IN 4000*(600+
C
C     40*(2+2+2+2+2)+175*(5+5+5)) = 14.5 SECONDS VERSUS ABOUT 4000*
C
C     4000*175 = 2800 SECONDS FOR THE STRAIGHTFORWARD TECHNIQUE.
C
C
C
C     THE FAST FOURIER ALGORITHM PLACES TWO RESTRICTIONS UPON THE
C
C     NATURE OF THE DATA BEYOND THE USUAL RESTRICTION THAT
C
C     THE DATA FORM ONE CYCLE OF A PERIODIC FUNCTION.  THEY ARE--
C
C     1.  THE NUMBER OF INPUT DATA AND THE NUMBER OF TRANSFORM VALUES
C
C     MUST BE THE SAME.
C
C     2. CONSIDERING THE DATA TO BE IN THE TIME DOMAIN,
C
C     THEY MUST BE EQUI-SPACED AT INTERVALS OF DT.  FURTHER, THE TRANS-
C
C     FORM VALUES, CONSIDERED TO BE IN FREQUENCY SPACE, WILL BE EQUI-
C
C     SPACED FROM 0 TO 2*PI*(NN(I)-1)/(NN(I)*DT) AT INTERVALS OF
C
C     2*PI/(NN(I)*DT) FOR EACH DIMENSION OF LENGTH NN(I).  OF COURSE,
C
C     DT NEED NOT BE THE SAME FOR EVERY DIMENSION.
C
C
C
C     THE CALLING SEQUENCE IS--
C
C     CALL FOURT(DATA,NN,NDIM,ISIGN,IFORM,WORK)
C
C
C
C     DATA IS THE ARRAY USED TO HOLD THE REAL AND IMAGINARY PARTS
C
C     OF THE DATA ON INPUT AND THE TRANSFORM VALUES ON OUTPUT.  IT
C
C     IS A MULTIDIMENSIONAL FLOATING POINT ARRAY, WITH THE REAL AND
C
C     IMAGINARY PARTS OF A DATUM STORED IMMEDIATELY ADJACENT IN STORAGE
C
C     (SUCH AS FORTRAN IV PLACES THEM).  THE EXTENT OF EACH DIMENSION
C
C     IS GIVEN IN THE INTEGER ARRAY NN, OF LENGTH NDIM.  ISIGN IS -1
C
C     TO INDICATE A FORWARD TRANSFORM (EXPONENTIAL SIGN IS -) AND +1
C
C     FOR AN INVERSE TRANSFORM (SIGN IS +).  IFORM IS +1 IF THE DATA AND
C
C     THE TRANSFORM VALUES ARE COMPLEX.  IT IS 0 IF THE DATA ARE REAL
C
C     BUT THE TRANSFORM VALUES ARE COMPLEX.  IF IT IS 0, THE IMAGINARY
C
C     PARTS OF THE DATA SHOULD BE SET TO ZERO.  AS EXPLAINED ABOVE, THE
```

```
c
c     TRANSFORM VALUES ARE ALWAYS COMPLEX AND ARE STORED IN ARRAY DATA.
c
c     WORK IS AN ARRAY USED FOR WORKING STORAGE.  IT IS NOT NECESSARY
c
c     IF ALL THE DIMENSIONS OF THE DATA ARE POWERS OF TWO.  IN THIS CASE
c
c     IT MAY BE REPLACED BY 0 IN THE CALLING SEQUENCE.  THUS, USE OF
c
c     POWERS OF TWO CAN FREE A GOOD DEAL OF STORAGE.  IF ANY DIMENSION
c
c     IS NOT A POWER OF TWO, THIS ARRAY MUST BE SUPPLIED.  IT IS
c
c     FLOATING POINT, ONE DIMENSIONAL OF LENGTH EQUAL TO TWICE THE
c
c     LARGEST ARRAY DIMENSION (I.E., NN(I) ) THAT IS NOT A POWER OF
c
c     TWO.  THEREFORE, IN ONE DIMENSION FOR A NON POWER OF TWO,
c
c     WORK OCCUPIES AS MANY STORAGE LOCATIONS AS DATA.  IF SUPPLIED,
c
c     WORK MUST NOT BE THE SAME ARRAY AS DATA.  ALL SUBSCRIPTS OF ALL
c
c     ARRAYS BEGIN AT ONE.
c
c
c
c     EXAMPLE 1.  THREE-DIMENSIONAL FORWARD FOURIER TRANSFORM OF A
c
c     COMPLEX ARRAY DIMENSIONED 32 BY 25 BY 13 IN FORTRAN IV.
c
c     COMPLEX ARRAY DIMENSIONED 32 BY 25 BY 13 IN FORTRAN IV.
c
c     DIMENSION DATA(32,25,13),WORK(50),NN(3)
c
c     COMPLEX DATA
c
c     DATA NN/32,25,13/
c
c     DO 1 I=1,32
c
c     DO 1 J=1,25
c
c     DO 1 K=1,13
c
c   1 DATA(I,J,K)=COMPLEX VALUE
c
c     CALL FOURT(DATA,NN,3,-1,1,WORK)
c
c
c
c     EXAMPLE 2.  ONE-DIMENSIONAL FORWARD TRANSFORM OF A REAL ARRAY OF
c
c     LENGTH 64 IN FORTRAN II.
c
c     DIMENSION DATA(2,64)
c
c     DO 2 I=1,64
c
c     DATA(1,I)=REAL PART
c
c   2 DATA(2,I)=0.
c
c     CALL FOURT(DATA,64,1,-1,0,0)
```

```
c
c
c
c     THERE ARE NO ERROR MESSAGES OR ERROR HALTS IN THIS PROGRAM.  THE
c
c     PROGRAM RETURNS IMMEDIATELY IF NDIM OR ANY NN(I) IS LESS THAN ONE.
c
c
c
c     PROGRAM BY NORMAN BRENNER FROM THE BASIC PROGRAM BY CHARLES
c
c     RADER (BOTH OF MIT LINCOLN LABORATORY).  MAY 1967.  THE IDEA
c
c     FOR THE DIGIT REVERSAL WAS SUGGESTED BY RALPH ALTER (ALSO MIT LL).
c
c     THIS IS THE FASTEST AND MOST VERSATILE VERSION OF THE FFT KNOWN
c
c     TO THE AUTHOR.  A PROGRAM CALLED FOUR2 IS AVAILABLE THAT ALSO
c
c     PERFORMS THE FAST FOURIER TRANSFORM AND IS WRITTEN IN USASI BASIC
c
c     FORTRAN.  IT IS ABOUT ONE THIRD AS LONG AND RESTRICTS THE
c
c     DIMENSIONS OF THE INPUT ARRAY (WHICH MUST BE COMPLEX) TO BE POWERS
c
c     OF TWO.  ANOTHER PROGRAM, CALLED FOUR1, IS ONE TENTH AS LONG AND
c
c     RUNS TWO THIRDS AS FAST ON A ONE-DIMENSIONAL COMPLEX ARRAY WHOSE
c
c     LENGTH IS A POWER OF TWO.
c
c
c
c     REFERENCE--
c
c     FAST FOURIER TRANSFORMS FOR FUN AND PROFIT, W. GENTLEMAN AND
c
c     G. SANDE, 1966 FALL JOINT COMPUTER CONFERENCE.
c
c
c
c     THE WORK REPORTED IN THIS DOCUMENT WAS PERFORMED AT LINCOLN LAB-
c
c     ORATORY, A CENTER FOR RESEARCH OPERATED BY MASSACHUSETTS INSTITUTE
c
c     OF TECHNOLOGY, WITH THE SUPPORT OF THE U.S. AIR FORCE UNDER
c
c     CONTRACT AF 19(628)-5167.
c
      subroutine fourt(data, nn, ndim, isign, iform, work)
      dimension data(1), nn(1), ifact(32), work(1)
# 126 "fourt.for"
      twopi = 6.283185307
      rthlf = .7071067812
      if (ndim - 1) 920, 1, 1
    1 ntot = 2
      do 2 idim = 1, ndim
      if (nn(idim)) 920, 920, 2
c
c
c     MAIN LOOP FOR EACH DIMENSION
c
c
c
```

```
# 132 "fourt.for"
    2 ntot = ntot * nn(idim)
# 136 "fourt.for"
      np1 = 2
      do 910 idim = 1, ndim
      n = nn(idim)
      np2 = np1 * n
c
c
c     IS N A POWER OF TWO AND IF NOT, WHAT ARE ITS FACTORS
c
c
c
# 140 "fourt.for"
      if (n - 1) 920, 900, 5
# 144 "fourt.for"
    5 m = n
      ntwo = np1
      if = 1
      idiv = 2
   10 iquot = m / idiv
      irem = m - (idiv * iquot)
      if (iquot - idiv) 50, 11, 11
   11 if (irem) 20, 12, 20
   12 ntwo = ntwo + ntwo
      ifact(if) = idiv
      if = if + 1
      m = iquot
      goto 10
   20 idiv = 3
      inon2 = if
   30 iquot = m / idiv
      irem = m - (idiv * iquot)
      if (iquot - idiv) 60, 31, 31
   31 if (irem) 40, 32, 40
   32 ifact(if) = idiv
      if = if + 1
      m = iquot
      goto 30
   40 idiv = idiv + 2
      goto 30
   50 inon2 = if
      if (irem) 60, 51, 60
   51 ntwo = ntwo + ntwo
      goto 70
   60 ifact(if) = m
c
c
c     SEPARATE FOUR CASES--
c
c        1. COMPLEX TRANSFORM
c
c        2. REAL TRANSFORM FOR THE 2ND, 3RD, ETC. DIMENSION.  METHOD--
c
c           TRANSFORM HALF THE DATA, SUPPLYING THE OTHER HALF BY CON-
c
c           JUGATE SYMMETRY.
c
c        3. REAL TRANSFORM FOR THE 1ST DIMENSION, N ODD.  METHOD--
c
c           SET THE IMAGINARY PARTS TO ZERO.
c
c        4. REAL TRANSFORM FOR THE 1ST DIMENSION, N EVEN.  METHOD--
c
```

```
c            TRANSFORM A COMPLEX ARRAY OF LENGTH N/2 WHOSE REAL PARTS
c
c            ARE THE EVEN NUMBERED REAL VALUES AND WHOSE IMAGINARY PARTS
c
c            ARE THE ODD NUMBERED REAL VALUES.  SEPARATE AND SUPPLY
c
c            THE SECOND HALF BY CONJUGATE SYMMETRY.
c
c
c
# 174 "fourt.for"
   70 non2p = np2 / ntwo
# 189 "fourt.for"
      icase = 1
      ifmin = 1
      i1rng = np1
      if (idim - 4) 74, 100, 100
   74 if (iform) 71, 71, 100
   71 icase = 2
      i1rng = np0 * (1 + (nprev / 2))
      if (idim - 1) 72, 72, 100
   72 icase = 3
      i1rng = np1
      if (ntwo - np1) 100, 100, 73
   73 icase = 4
      ifmin = 2
      ntwo = ntwo / 2
      n = n / 2
      np2 = np2 / 2
      ntot = ntot / 2
      i = 1
      do 80 j = 1, ntot
      data(j) = data(i)
c
c
c     SHUFFLE DATA BY BIT REVERSAL, SINCE N=2**K.  AS THE SHUFFLING
c
c     CAN BE DONE BY SIMPLE INTERCHANGE, NO WORKING ARRAY IS NEEDED
c
c
c
# 209 "fourt.for"
   80 i = i + 2
# 214 "fourt.for"
  100 if (non2p - 1) 101, 101, 200
  101 np2hf = np2 / 2
      j = 1
      do 150 i2 = 1, np2, np1
      if (j - i2) 121, 130, 130
  121 i1max = (i2 + np1) - 2
      do 125 i1 = i2, i1max, 2
      do 125 i3 = i1, ntot, np2
      j3 = (j + i3) - i2
      tempr = data(i3)
      tempi = data(i3 + 1)
      data(i3) = data(j3)
      data(i3 + 1) = data(j3 + 1)
      data(j3) = tempr
  125 data(j3 + 1) = tempi
  130 m = np2hf
  140 if (j - m) 150, 150, 141
  141 j = j - m
      m = m / 2
      if (m - np1) 150, 140, 140
```

```
  150 j = j + m
c
c
c     SHUFFLE DATA BY DIGIT REVERSAL FOR GENERAL N
c
c
c
# 235 "fourt.for"
      goto 300
# 239 "fourt.for"
  200 nwork = 2 * n
      do 270 i1 = 1, np1, 2
      do 270 i3 = i1, ntot, np2
      j = i3
      do 260 i = 1, nwork, 2
      if (icase - 3) 210, 220, 210
  210 work(i) = data(j)
      work(i + 1) = data(j + 1)
      goto 240
  220 work(i) = data(j)
      work(i + 1) = 0.
  240 ifp2 = np2
      if = ifmin
  250 ifp1 = ifp2 / ifact(if)
      j = j + ifp1
      if ((j - i3) - ifp2) 260, 255, 255
  255 j = j - ifp2
      ifp2 = ifp1
      if = if + 1
      if (ifp2 - np1) 260, 260, 250
  260 continue
      i2max = (i3 + np2) - np1
      i = 1
      do 270 i2 = i3, i2max, np1
      data(i2) = work(i)
      data(i2 + 1) = work(i + 1)
c     MAIN LOOP FOR FACTORS OF TWO.
c
c     W=EXP(ISIGN*2*PI*SQRT(-1)*M/(4*MMAX)).  CHECK FOR W=ISIGN*SQRT(-1)
c
c     AND REPEAT FOR W=W*(1+ISIGN*SQRT(-1))/SQRT(2).
c
c
c
# 265 "fourt.for"
  270 i = i + 2
# 270 "fourt.for"
  300 if (ntwo - np1) 600, 600, 305
  305 np1tw = np1 + np1
      ipar = ntwo / np1
  310 if (ipar - 2) 350, 330, 320
  320 ipar = ipar / 4
      goto 310
  330 do 340 i1 = 1, i1rng, 2
      do 340 k1 = i1, ntot, np1tw
      k2 = k1 + np1
      tempr = data(k2)
      tempi = data(k2 + 1)
      data(k2) = data(k1) - tempr
      data(k2 + 1) = data(k1 + 1) - tempi
      data(k1) = data(k1) + tempr
  340 data(k1 + 1) = data(k1 + 1) + tempi
  350 mmax = np1
  360 if (mmax - (ntwo / 2)) 370, 600, 600
```

```
370 lmax = max0(npltw,mmax / 2)
    do 570 l = npl, lmax, npltw
    m = l
    if (mmax - npl) 420, 420, 380
380 theta = - ((twopi * float(l)) / float(4 * mmax))
    if (isign) 400, 390, 390
390 theta = - theta
400 wr = cos(theta)
    wi = sin(theta)
410 w2r = (wr * wr) - (wi * wi)
    w2i = (2. * wr) * wi
    w3r = (w2r * wr) - (w2i * wi)
    w3i = (w2r * wi) + (w2i * wr)
420 do 530 i1 = 1, i1rng, 2
    kmin = i1 + (ipar * m)
    if (mmax - npl) 430, 430, 440
430 kmin = i1
440 kdif = ipar * mmax
450 kstep = 4 * kdif
    if (kstep - ntwo) 460, 460, 530
460 do 520 k1 = kmin, ntot, kstep
    k2 = k1 + kdif
    k3 = k2 + kdif
    k4 = k3 + kdif
    if (mmax - npl) 470, 470, 480
470 u1r = data(k1) + data(k2)
    u1i = data(k1 + 1) + data(k2 + 1)
    u2r = data(k3) + data(k4)
    u2i = data(k3 + 1) + data(k4 + 1)
    u3r = data(k1) - data(k2)
    u3i = data(k1 + 1) - data(k2 + 1)
    if (isign) 471, 472, 472
471 u4r = data(k3 + 1) - data(k4 + 1)
    u4i = data(k4) - data(k3)
    goto 510
472 u4r = data(k4 + 1) - data(k3 + 1)
    u4i = data(k3) - data(k4)
    goto 510
480 t2r = (w2r * data(k2)) - (w2i * data(k2 + 1))
    t2i = (w2r * data(k2 + 1)) + (w2i * data(k2))
    t3r = (wr * data(k3)) - (wi * data(k3 + 1))
    t3i = (wr * data(k3 + 1)) + (wi * data(k3))
    t4r = (w3r * data(k4)) - (w3i * data(k4 + 1))
    t4i = (w3r * data(k4 + 1)) + (w3i * data(k4))
    u1r = data(k1) + t2r
    u1i = data(k1 + 1) + t2i
    u2r = t3r + t4r
    u2i = t3i + t4i
    u3r = data(k1) - t2r
    u3i = data(k1 + 1) - t2i
    if (isign) 490, 500, 500
490 u4r = t3i - t4i
    u4i = t4r - t3r
    goto 510
500 u4r = t4i - t3i
    u4i = t3r - t4r
510 data(k1) = u1r + u2r
    data(k1 + 1) = u1i + u2i
    data(k2) = u3r + u4r
    data(k2 + 1) = u3i + u4i
    data(k3) = u1r - u2r
    data(k3 + 1) = u1i - u2i
    data(k4) = u3r - u4r
520 data(k4 + 1) = u3i - u4i
```

```
      kdif = kstep
      kmin = (4 * (kmin - i1)) + i1
      goto 450
  530 continue
      m = m + lmax
      if (m - mmax) 540, 540, 570
  540 if (isign) 550, 560, 560
  550 tempr = wr
      wr = (wr + wi) * rthlf
      wi = (wi - tempr) * rthlf
      goto 410
  560 tempr = wr
      wr = (wr - wi) * rthlf
      wi = (tempr + wi) * rthlf
      goto 410
  570 continue
      ipar = 3 - ipar
      mmax = mmax + mmax
c
c
c     MAIN LOOP FOR FACTORS NOT EQUAL TO TWO.
c
c     W=EXP(ISIGN*2*PI*SQRT(-1)*(J1+J2-I3-1)/IFP2)
c
c
c
# 369 "fourt.for"
      goto 360
# 374 "fourt.for"
  600 if (non2p - 1) 700, 700, 601
  601 ifp1 = ntwo
      if = inon2
  610 ifp2 = ifact(if) * ifp1
      theta = - (twopi / float(ifact(if)))
      if (isign) 612, 611, 611
  611 theta = - theta
  612 wstpr = cos(theta)
      wstpi = sin(theta)
      do 650 j1 = 1, ifp1, np1
      thetm = - ((twopi * float(j1 - 1)) / float(ifp2))
      if (isign) 614, 613, 613
  613 thetm = - thetm
  614 wminr = cos(thetm)
      wmini = sin(thetm)
      i1max = (j1 + i1rng) - 2
      do 650 i1 = j1, i1max, 2
      do 650 i3 = i1, ntot, np2
      i = 1
      wr = wminr
      wi = wmini
      j2max = (i3 + ifp2) - ifp1
      do 640 j2 = i3, j2max, ifp1
      twowr = wr + wr
      j3max = (j2 + np2) - ifp2
      do 630 j3 = j2, j3max, ifp2
      jmin = (j3 - j2) + i3
      j = (jmin + ifp2) - ifp1
      sr = data(j)
      si = data(j + 1)
      oldsr = 0.
      oldsi = 0.
      j = j - ifp1
  620 stmpr = sr
      stmpi = si
```

```
      sr = ((twowr * sr) - oldsr) + data(j)
      si = ((twowr * si) - oldsi) + data(j + 1)
      oldsr = stmpr
      oldsi = stmpi
      j = j - ifp1
      if (j - jmin) 621, 621, 620
  621 work(i) = (((wr * sr) - (wi * si)) - oldsr) + data(j)
      work(i + 1) = (((wi * sr) + (wr * si)) - oldsi) + data(j + 1)
  630 i = i + 2
      wtemp = wr * wstpi
      wr = (wr * wstpr) - (wi * wstpi)
  640 wi = (wi * wstpr) + wtemp
      i = 1
      do 650 j2 = i3, j2max, ifp1
      j3max = (j2 + np2) - ifp2
      do 650 j3 = j2, j3max, ifp2
      data(j3) = work(i)
      data(j3 + 1) = work(i + 1)
  650 i = i + 2
      if = if + 1
      ifp1 = ifp2
c
c
c     COMPLETE A REAL TRANSFORM IN THE 1ST DIMENSION, N EVEN, BY CON-
c
c     JUGATE SYMMETRIES.
c
c
c
# 430 "fourt.for"
      if (ifp1 - np2) 610, 700, 700
# 435 "fourt.for"
  700 goto (900, 800, 900, 701), icase
  701 nhalf = n
      n = n + n
      theta = - (twopi / float(n))
      if (isign) 703, 702, 702
  702 theta = - theta
  703 wstpr = cos(theta)
      wstpi = sin(theta)
      wr = wstpr
      wi = wstpi
      imin = 3
      jmin = (2 * nhalf) - 1
      goto 725
  710 j = jmin
      do 720 i = imin, ntot, np2
      sumr = (data(i) + data(j)) / 2.
      sumi = (data(i + 1) + data(j + 1)) / 2.
      difr = (data(i) - data(j)) / 2.
      difi = (data(i + 1) - data(j + 1)) / 2.
      tempr = (wr * sumi) + (wi * difr)
      tempi = (wi * sumi) - (wr * difr)
      data(i) = sumr + tempr
      data(i + 1) = difi + tempi
      data(j) = sumr - tempr
      data(j + 1) = (- difi) + tempi
  720 j = j + np2
      imin = imin + 2
      jmin = jmin - 2
      wtemp = wr * wstpi
      wr = (wr * wstpr) - (wi * wstpi)
      wi = (wi * wstpr) + wtemp
  725 if (imin - jmin) 710, 730, 740
```

```
  730 if (isign) 731, 740, 740
  731 do 735 i = imin, ntot, np2
  735 data(i + 1) = - data(i + 1)
  740 np2 = np2 + np2
      ntot = ntot + ntot
      j = ntot + 1
      imax = (ntot / 2) + 1
  745 imin = imax - (2 * nhalf)
      i = imin
      goto 755
  750 data(j) = data(i)
      data(j + 1) = - data(i + 1)
  755 i = i + 2
      j = j - 2
      if (i - imax) 750, 760, 760
  760 data(j) = data(imin) - data(imin + 1)
      data(j + 1) = 0.
      if (i - j) 770, 780, 780
  765 data(j) = data(i)
      data(j + 1) = data(i + 1)
  770 i = i - 2
      j = j - 2
      if (i - imin) 775, 775, 765
  775 data(j) = data(imin) + data(imin + 1)
      data(j + 1) = 0.
      imax = imin
      goto 745
  780 data(1) = data(1) + data(2)
      data(2) = 0.
c
c
c     COMPLETE A REAL TRANSFORM FOR THE 2ND, 3RD, ETC. DIMENSION BY
c
c     CONJUGATE SYMMETRIES.
c
c
c
# 496 "fourt.for"
      goto 900
# 501 "fourt.for"
  800 if (i1rng - np1) 805, 900, 900
  805 do 860 i3 = 1, ntot, np2
      i2max = (i3 + np2) - np1
      do 860 i2 = i3, i2max, np1
      imax = (i2 + np1) - 2
      imin = i2 + i1rng
      jmax = ((2 * i3) + np1) - imin
      if (i2 - i3) 820, 820, 810
  810 jmax = jmax + np2
  820 if (idim - 2) 850, 850, 830
  830 j = jmax + np0
      do 840 i = imin, imax, 2
      data(i) = data(j)
      data(i + 1) = - data(j + 1)
  840 j = j - 2
  850 j = jmax
      do 860 i = imin, imax, np0
      data(i) = data(j)
      data(i + 1) = - data(j + 1)
c
c
c     END OF LOOP ON EACH DIMENSION
c
c
```

```
c
# 520 "fourt.for"
  860 j = j - np0
# 524 "fourt.for"
  900 np0 = np1
      np1 = np2
  910 nprev = n
  920 return
      end
```