

the
Optical
Sciences
Company

P.O. Box 1329, Placentia, California 92670 ■ Phone (714) 524-3622

Report No. DR-514

AD-A228 774

Final Report on ONR Contract No. N00014-88-C-0692
Phased Array Imaging

David L. Fried
and
Douglas T. Sherwood
the Optical Sciences Company
380 S. Placentia, Suite J
Placentia, CA 92670

September 1990

Final Report for Period: 1 September 88-30 April 90

Prepared for: Office of Naval Research
Department of The Navy
800 N. Quincy Street
Arlington, Va 22217-5000

DTIC
ELECTE
NOV 19 1990
S B D

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

D. L. FRIED ASSOCIATES, INC. dba the OPTICAL SCIENCES COMPANY

90 11 16 034

Final Report on ONR Contract No. N00014-88-C-0692
Phased Array Imaging

David L. Fried
and
Douglas T. Sherwood
the Optical Sciences Company
380 S. Placentia, Suite J
Placentia, CA 92670

September 1990

Final Report for Period: 1 September 88-30 April 90

Prepared for: Office of Naval Research
Department of The Navy
800 N. Quincy Street
Arlington, Va 22217-5000

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS None	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DR-514			5. MONITORING-ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION the Optical Sciences Company		6b. OFFICE SYMBOL (If applicable) 9D674	7a. NAME OF MONITORING ORGANIZATION Department of the Navy Office of Naval Research	
6c. ADDRESS (City, State, and ZIP Code) P. O. Box 1329 Placentia, CA 92670			7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, VA 22217-5000	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research		8b. OFFICE SYMBOL (If applicable) N00014	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-88-C-0692	
8c. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, VA 22217-5000			10. SOURCE OF FUNDING NUMBERS	
			PROGRAM ELEMENT NO.	PROJECT NO.
			TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Final Report on ONR Contract N00014-88-C-0692, Phased Array Imaging				
12. PERSONAL AUTHOR(S) David L. Fried, Douglas T. Sherwood				
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM 880901 TO 900430	14. DATE OF REPORT (Year, Month, Day) 9009	
15. PAGE COUNT				
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	Super resolution, sparse array, CLEAN, image resolution, noise, image processing. (RH)	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
The problem of recoverable image resolution is investigated for the case where an imaging array is used which array has an optical transfer function that may be described as consisting of "islands" of nonzero value in a sea of zero values. ~~We wish to know if~~ ^{Can} the missing spatial frequency information can be provided--~~if~~, in effect, a form of (interpolative) super resolution, ~~can be achieved~~. The CLEAN algorithm used by radio astronomers suggests that this should be possible. The results developed here indicate that this can be done, with no significant price in terms of signal-to-noise ratio to be paid, and further show that a nonlinear algorithm, like CLEAN, is not required. The results show that the feasibility of doing this depends on the angular size of the object being imaged. We find that its size must be less than the inverse of the largest gap between "islands" in the array's optical transfer function. <u>Keywords:</u>				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Marilyn E. Kruger			22b. TELEPHONE (Include Area Code) 714-524-3622	22c. OFFICE SYMBOL 9D674

Table of Contents

<u>Chapter</u>	<u>Section</u>	<u>Title</u>	<u>Page</u>
		ABSTRACT	3
1		Introduction and Summary	4
	1.1	Introduction and Summary	4
2		Super-Resolution with Sparse Arrays of Optical Apertures:	15
		A Preliminary Investigation	
	2.1	Introduction	16
	2.2	Discrete Optical Model	16
	2.2.1	One-Dimensional Model	17
	2.2.2	Two-Dimensional Model	23
	2.3	Minimum-Variance Processor	26
	2.3.1	Minimum-Variance Results for the One-Dimensional Case	34
	2.3.2	Minimum-Variance Results for the Two-Dimensional Case	35
	2.4	Least-Squares Processor and Results, One-Dimensional	35
		Case, Positivity Constraint	
3		Super-Resolution with Sparse Arrays Revisited—Conclusion of	60
		Two Aperture Case	
	3.1	Introduction	61
	3.2	Some Changes to the One-Dimensional Model	61
	3.3	Results Using the Minimum-Variance Method	70
	3.4	Results Using the Least Squares Method	86
4		Object Reconstruction with Sparse Arrays of Optical	102
		Apertures. Part I: Linear Methods	
	4.1	Introduction	103
	4.2	Discrete Optical Model (One-Dimensional)	105
	4.3	Performance Measure	111
	4.4	Minimum-Variance Processor	113
	4.5	Unweighted Least-Squares Processor	121
	4.6	Discussion	122
5		A Random Process with Finite Support: Authorization	129
		Function of Its Fourier Transform and Energy and Power	
		Spectral Densities	
	5.1	Introduction	130
	5.2	Continuous Case	130
	5.2.1	Autocorrelation Function of Fourier Transform	130
	5.2.2	Energy Spectral Density and Power Spectral Density of $x(r)$	131
	5.3	Discrete Case	132
	5.3.1	Autocorrelation Function of Fourier Transform	132
	5.3.2	Energy Spectral Density and Power Spectral Density of $x(n)$	134
6		Object Reconstruction with Sparse Arrays of Optical	135
		Apertures. Part II: Nonlinear Methods	
	6.1	Introduction	136
	6.2	Discrete Optical Model (One-Dimensional)	136
	6.3	A Performance Measure	142
	6.4	Unweighted Least-Squares, Finite Support and Positivity	144
		Constraints	

6.5

CLEAN	146
REFERENCES	164
APPENDIX A for Chapter 2	166
APPENDIX B for Chapter 3	192
APPENDIX C for Chapter 4	221
APPENDIX D for Chapter 6	232
APPENDIX E for Chapter 6	234
APPENDIX F for Chapter 6	235

ABSTRACT

The problem of recoverable image resolution is investigated for the case where an imaging array is used which array has an optical transfer function that may be described as consisting of "islands" of nonzero value in a sea of zero values. We wish to know if the missing spatial frequency information can be provided—if, in effect, a form of (interpolative) super resolution can be achieved. The CLEAN algorithm used by radio astronomers suggests that this should be possible. The results developed here indicate that this can be done, with no significant price in terms of signal-to-noise ratio to be paid, and further show that a nonlinear algorithm, like CLEAN, is not required. The results show that the feasibility of doing this depends on the angular size of the object being imaged. We find that its size must be less than the inverse of the largest gap between "islands" in the array's optical transfer function.



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Chapter 1

Introduction and Summary

1.1. Introduction and Summary

The objective of the work reported here has its origin in the desire to form a high resolution image of a satellite in a geosynchronous orbit with ground based optical imaging equipment. Underlying the willingness to express such a desire is the presumption that by the exploitation of some technique or combination of techniques—drawn from the methodologies of adaptive optics, of white light speckle imagery, and of other possible relevant approaches—it will be possible to circumvent the seeming limits to resolution that are imposed by atmospheric turbulence. Once we grant such a presumption we next encounter the problem posed by the sheer physical size of the instrument required for the task.

The nominal range involved is $R = 4 \times 10^7$ m. If the imaging is to be carried out utilizing $\lambda = 5 \times 10^{-7}$ m wavelength light, and the desired linear resolution on the satellite is $\delta x = 0.1$ m, then the span, $S_{\max}$ of the imaging system's aperture must be

$$\begin{aligned} S_{\max} &= R\lambda/\delta x \\ &= (4 \times 10^7)(5 \times 10^{-7})/(0.1) \\ &= 200 \text{ m}, \end{aligned} \tag{1.1}$$

This dimension is much too great for us to be able to even toy with the idea that imaging system might be a conventional sort of "full aperture" imaging telescope. Clearly the imaging system design has to be based on some sort of array concept—and a sparse array concept at that.

Once we accepted the position that the instrument design should be based on a sparse array concept, the next matter for consideration has to do with just how sparse an array we might be able to use. The kind of instrument and its cost would be strongly influenced by the degree of sparsity we could tolerate.

We had initially maintained that since the resolution we sought, δx , corresponded to a spatial frequency, $\kappa_{\max}$, such that

$$\kappa_{\max} = S_{\max}/\lambda, \tag{1.2}$$

then we would need an array pattern which provided for "coverage" of all spatial frequencies, κ , such that

$$|\kappa| \leq \kappa_{\max}. \tag{1.3}$$

By the term "coverage" we meant that the array's optical transfer function for that spatial frequency, $\tau(\kappa)$, has a non-zero magnitude. This in turn means that for any spatial frequency, κ , satisfying Eq. (1.3), there must be at least one pair of points in the "composite aperture" associated with the entire array which points are separated by a distance S , where

$$S = \kappa\lambda. \tag{1.4}$$

Put differently, the requirement is that for all S such that

$$|S| \leq S_{\max}, \tag{1.5}$$

there must be at least one pair of points in the array's composite aperture whose separation is equal to S . If such a pair of points do not exist then for the spatial frequency $\kappa = S/\lambda$, the optical transfer function, $\tau(\kappa)$, of the array will be zero. This would seem to imply that any information about the satellite that is represented in its image by this spatial frequency, κ , would be missing in the image eventually developed from the array's measurements.

The consequence of this is that the minimum acceptable size of the individual elements of the array is "tied" to the spacing of the array elements. If for example, the spacing of array elements is in integer multiples of 2 m, then the array elements must have diameters in excess of 1 m. If they do not, then there will be spatial frequencies for which $S = \kappa\lambda$ has a magnitude just larger

than 1 m, for which the optical transfer function would be zero—and this, we argued, would result in images of little utility—strongly “defaced” by the absence of significant portions of the spatial frequency spectrum. The implications of this seemed to be two-fold. First, a great deal of physical aperture area (i.e., a great deal of glass) would have to be provided. Second, unless we were prepared to consider a very large number of separate array elements, the aperture diameter of the individual array element would have to be considerably larger than the effective coherence diameter, r_0 , imposed by atmospheric turbulence considerations. This carried with it the implication that just as we probably would have to be able to phase the array elements to each other, so also we would probably have to provide a wavefront distortion compensation/accommodation procedure for each array element’s aperture.

It was at this point in the consideration of this matter that Ken Johnston of NRL introduced a quite different position. He argued that the radio astronomer’s experience with the CLEAN algorithm demonstrated that it was possible to recover very acceptable, high resolution images starting with data from an array so sparse that for many of the spatial frequencies that were present in the recovered image, the value of the array’s optical transfer function was equal to zero. It was not necessarily clear how/why this could be the case, but it nonetheless appeared to be true. The practical implication of this was that with reasonably large spacing between array elements, and thus with only a moderate number of such elements, it would be possible to use a quite small aperture diameter for each element—providing of course that light gathering requirements did not force the use of larger aperture diameters. In fact from a missing spatial frequencies point of view, the individual element’s aperture diameter could be small enough so that there would not be any significant wavefront distortion within the element’s aperture. This would close out the question of whether or not any adaptive optics were unavoidably needed to compensate for wavefront distortion in each aperture element—leaving the system design with the only unavoidable turbulence/phase compensation problem being that of how to handle the relative phase difference between the individual array elements. And of course, the permitted smallness of the individual array elements offered an advantage in terms of cost and mechanical complexity.

But before we could accept all of these advantages it was necessary to understand if the CLEAN algorithm does actually work as well as it seems to and also how/why it does so. We had to consider the possibility that CLEAN works as well as it does for the radio astronomers only because it has available to it an immense signal-to-noise ratio in the starting data. The available publications do not clearly rule out this possibility. If it did rely on a very large signal-to-noise ratio it would be of little potential utility to us in designing a sparse optical array for imaging a geosynchronous range satellite. Also we had to understand just how sparse the array could be and still produce useful images, i.e., how far apart from each other could the array elements be (or how few array elements we needed to use). It was to investigate these questions that the work reported here was undertaken.

This work is reported in the various chapters following this introduction. The balance of this introduction explains how we viewed/approached this problem, the methods we used in carry out our investigations, and finally what we found. As a convenience to the reader we will first take up a discussion of our work in a bit more detail.

To sum up, we found that it is possible to recover a quite satisfactory image even though the array is so sparse that the value of its optical transfer function is equal to zero over much of the spatial frequency domain covered by the image eventually produced. We found that this was possible even with quite modest signal-to-noise ratio values for the basic data produced by the array; we found that the image recovery process seems to impose little penalty in terms of signal-to-noise ratio. Moreover, we found that there was no strict requirement that the image recovery process be nonlinear—even though the CLEAN algorithm itself is nonlinear. We found that the size of the individual aperture elements did not matter (except as that effects the total light gathering/shot noise) but that what was of critical importance was the relationship between the size of the object being imaged and the spacing between array elements. The spacing between array elements set a critical angular dimension and the angular size of the object being imaged had to be smaller

than that if a satisfactory image was to be recovered without imposing a severe signal-to-noise ratio penalty.

Our conceptual approach to this problem was through the idea of super resolution. More than two decades ago there was a flurry of activity around the concept of super resolution—a term which can be taken as meaning, at that time, the possibility of obtaining an image containing valid data for spatial frequency whose magnitude is above D/λ (cycles/rad f.o.v) when collecting the optical information using an imaging system whose aperture diameter is D and when working in a (narrow) spectral band characterized by a wavelength λ . The value of the optical transfer function is zero for such large spatial frequencies, and the ability to obtain valid image data at such spatial frequencies with such an instrument is in a rather obvious sense “super resolution”. It seemed to us that, as a practical matter, the term “super resolution” referred not so much to the magnitude of the spatial frequency but more so to the fact that the value of the optical transfer function was equal to zero for the spatial frequencies being considered. In that sense what was being accomplished by the CLEAN algorithm working with data from a very sparse array was super resolution. The spatial frequencies in question were not, in general, as large as the largest covered by the array’s span, but the value of the array’s optical transfer function was equal to zero for the spatial frequencies in question—and yet valid image data was being produced by the CLEAN algorithm for these spatial frequencies.

With this putative equating of the sparse array problem to the concept of super resolution, we were able to draw on the insight provided by Jim Harris more than 25 years ago, that to the extent that super resolution works, it works because the object being imaged is of finite size, on a uniform (black) background.* He showed that no two distinctly different objects of *finite angular size* can have identical images. This caused us to recognize the potential significance of the finite size of the satellite we were interested in imaging. But Harris’ work was followed by a number of others that showed that while super resolution could be achieved—in fact could be achieved by quite linear signal processing methods—there was a tremendous signal-to-noise ratio penalty to be paid for what was achieved.

Based on consideration of these various results we formulated an approach to investigating what could be accomplished by processing sparse array data, which approach restricted attention to finite size objects and to linear signal processing algorithms (at least initially). We chose to use a spatially quantized (i.e., a sample value) formulation of the problem, rather than one based on the use of analytic functions. (This assured us that, though we might burden the computer, the most complex analytic operations we would have to consider were matrix manipulations.) We assumed as a given, the matrix representing the imaging system’s (i.e., the array’s) point spread function—a matrix which when multiplied by the data array (of sample values) representing the object, would result in a data set (of sample values) representing the raw or initial image formed by the imaging system/array. We assumed that there would be some set of random (noise) values added to this initial image formed by the imaging system/array. (We always considered a gaussian noise with an rms level that was the same for all pixels in the initial image.) We then posed our analysis in terms of a linear process (i.e., an image processing matrix) to be applied to (i.e., to be multiplied by) the data set constituting the initial image so as to produce a set of estimated values for the data set representing the object being imaged.

When we assumed that we knew nothing of the statistics of the object being imaged then we imposed a least square error criteria in the selection of our linear process (i.e., in the selection of our image processing matrix). Our least square criteria was that the recovered object (image), if imaged by the array (i.e., if multiplied by the array’s optical transfer function matrix) would produce a set of data matching the row (initial) image values as closely as possible, in a least square sense. This led in a direct manner from knowledge of the array’s optical transfer function to formulation of a matrix for linear processing, (i.e. matrix multiplication) of the raw (initial) image data to produce the enhanced image. It allowed us to develop results for the mean square error in the recovered

* J. L. Harris, “Diffraction and Resolving Power” J. Opt. Soc. Am. 54 931-936 (1964).

image as a function of the rms noise and the nominal signal level in the raw image.

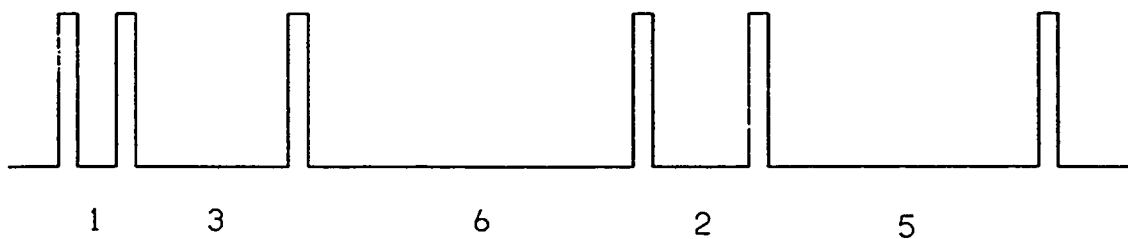
We also considered the case in which we assumed that we knew something about the statistics of the object being imaged. In this case we were able to use a minimum variance formulation of the image processing problem. In this case also we obtained an image enhancement matrix to be multiplied by the raw (initial) image data produced by the array, and thereby produce an enhanced image. In this case also we developed an expression for the mean square error in the recovered image as a function of the mean square error in the raw image and of the nominal signal level in the raw image.

In developing this line of investigation we recognized that it might not be desirable to attempt to recover an image containing all spatial frequencies up to the highest that the sample density (or pixel size) could support. Accordingly we considered spatial frequency analysis of the recovered image. We developed expressions allowing us to calculate the signal-to-noise ratio to be associated with each spatial frequency component of the recovered/enhanced image, and the signal-to-noise ratio to be associated with an image enhanced for all spatial frequencies up to some cut-off frequency, and with zero content for higher spatial frequencies. Such results were developed for both minimum variance and least square error formulations.

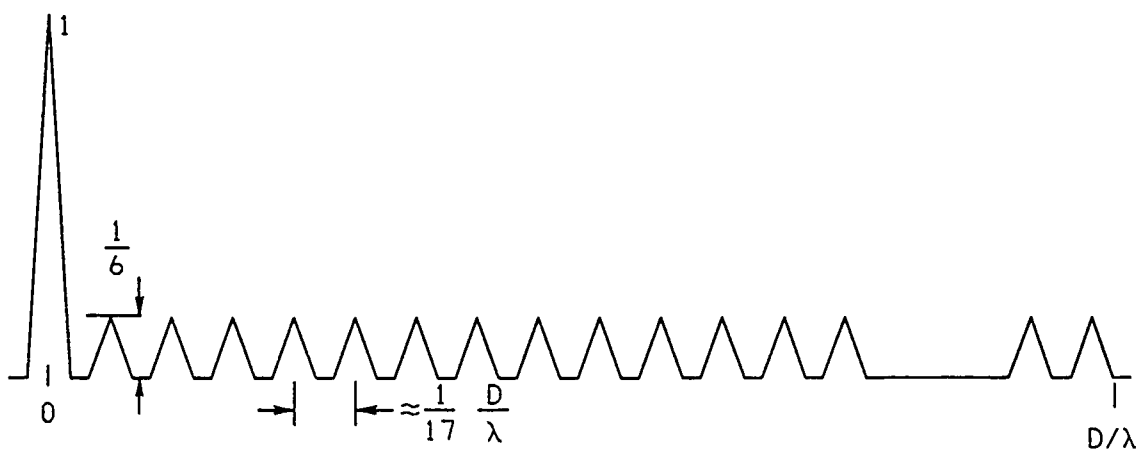
Our initial work started with a two-aperture one-dimensional array, and with a three-aperture two-dimensional array. This work along with our basic analytic formulation of the problem is presented in Chapters 2 and 3 of this report. Unfortunately we were able to develop virtually no useful insight from this work, and the documentation is included here partly for completeness, but mostly as a reference for the detail of the analytic formulation—which we continued to use. We now realize that our failure to obtain useful insight from the results of this work has to do with the fact that the problem was not "rich" enough. The two-aperture one-dimensional array and the three-aperture two-dimensional arrays are too simple. In subsequent work we considered multi-aperture one-dimensional arrays. For these multi-aperture arrays the imaging process/problem was sufficiently rich that we were able to develop considerable insight into the image enhancement/super resolution process. (We did not pursue the two-dimensional array problem any further, considering that the insight we got from the one dimensional array problem was equally applicable to the two-dimensional array.) After a brief discussion of the one-dimensional array results that we obtained with linear processing we will discuss work done with nonlinear processing algorithm; such results are developed in Chapter 4.

As an instructive example of the results we obtained with a linear processing image enhancement technique we present the minimum variance results obtained with the one-dimension six aperture (non redundant) sparse array whose array pattern and optical transfer function are shown in Fig. 1.1. We have a sparse array span denoted by D , with an optical transfer function that has non-zero values out to a spatial frequency of D/λ . The optical transfer function has islands of nonzero value with spacing of $\frac{1}{17}D/\lambda$ cycles/rad_{f.o.v.} in spatial frequency—out to $\frac{13}{17}D/\lambda$ cycles/rad_{f.o.v.}, but then has a gap of $\frac{3}{17}D/\lambda$ cycles/rad_{f.o.v.} to the next island. Between the islands, in the gaps of $\frac{1}{17}D/\lambda$ and $\frac{3}{17}D/\lambda$, the value of the optical transfer function is zero. We call particular attention to this transition from a gap of $\frac{1}{17}D/\lambda$ to $\frac{3}{17}D/\lambda$ cycles/rad_{f.o.v.} at a spatial frequency of $\frac{13}{17}D/\lambda$ cycles/rad_{f.o.v.}. We also note that the value of the optical transfer function is zero for all spatial frequencies above D/λ cycles/rad_{f.o.v.}.

The analysis associated with this array was formulated utilizing pixel size of $\frac{1}{4}\lambda/D$ both to define the object and its recovered image, and to represent the (noisy) raw (initial) image formed by the array. With this sample size we are able to consider spatial frequencies up to $2D/\lambda$. We took as our knowledge of the object being imaged, the assumption that it was a zero-mean white noise pattern, i.e., that there was no correlation between pixel values, that the average value of each pixel was zero, and that the standard deviation in the value of each pixel was the same—a value denoted by σ_x . We considered the noise in the array's image plane to also be zero mean white noise with a standard deviation denoted by σ_n . We relate σ_x and σ_n by presume a scale factor in associating numerical values with the raw image data such that the matrix representing the array's



(a)



(b)

Figure 1.1 Array Pattern and Optical Transfer Function for a Six-Element One-Dimensional Non-Redundant Array. The spacing of the array elements, indicated by the numbers 1, 3, 6, 2, and 5 in Fig. 1.1a total to 17, which we associate with the array span, to be denoted by D . It is to be understood as significant that the array elements are small enough compared to the array spacing that the value of the optical transfer function is equal to zero over much of the spatial frequency domain of interest. We see in Fig. 1.1b that the space between the individual "islands" of nonzero OTF value is $\frac{1}{17} D/\lambda$, out to $\frac{13}{17} D/\lambda$, with only small zero-value regions between the islands—but there is a significantly larger gap of zero-value OTF between the $\frac{13}{17} D/\lambda$ spatial frequency island and the $\frac{16}{17} D/\lambda$ island, a gap that will be of considerable significance in interpreting our results.

point spread function would image an everywhere equal to unity infinite extent object as an everywhere equal to unity infinite extent raw image. For our analysis we considered the case where the raw data "signal-to-noise ratio" was,

$$\text{SNR}_{\text{RAW}} = 20 \log_{10}(\sigma_x/\sigma_n), \quad (1.6)$$

equal to 50.

When we calculated mean square error in each spatial frequency component of the processed image produced by the minimum variance estimator for various size objects, we obtained the results shown in Fig. 1.2. These results are for object sizes of $3\lambda/D$, $6\lambda/D$, $9\lambda/D$, ..., $21\lambda/D$. Examining this figure we can see that only for the two largest objects, $18\lambda/D$ and $21\lambda/D$, do we make much error in estimating the spatial frequency components for *all* spatial frequencies below $\frac{13}{17}D/\lambda$, even for spatial frequencies for which the arrays optical transfer function is equal to zero. For spatial frequencies between $\frac{13}{17}D/\lambda$ and $\frac{16}{17}D/\lambda$, it is only for the two smallest objects, $3\lambda/D$ and $6\lambda/D$, that acceptable results are obtained. The quality of the estimate for spatial frequencies beyond D/λ falls off quite rapidly, going out the farthest for the smallest objects.

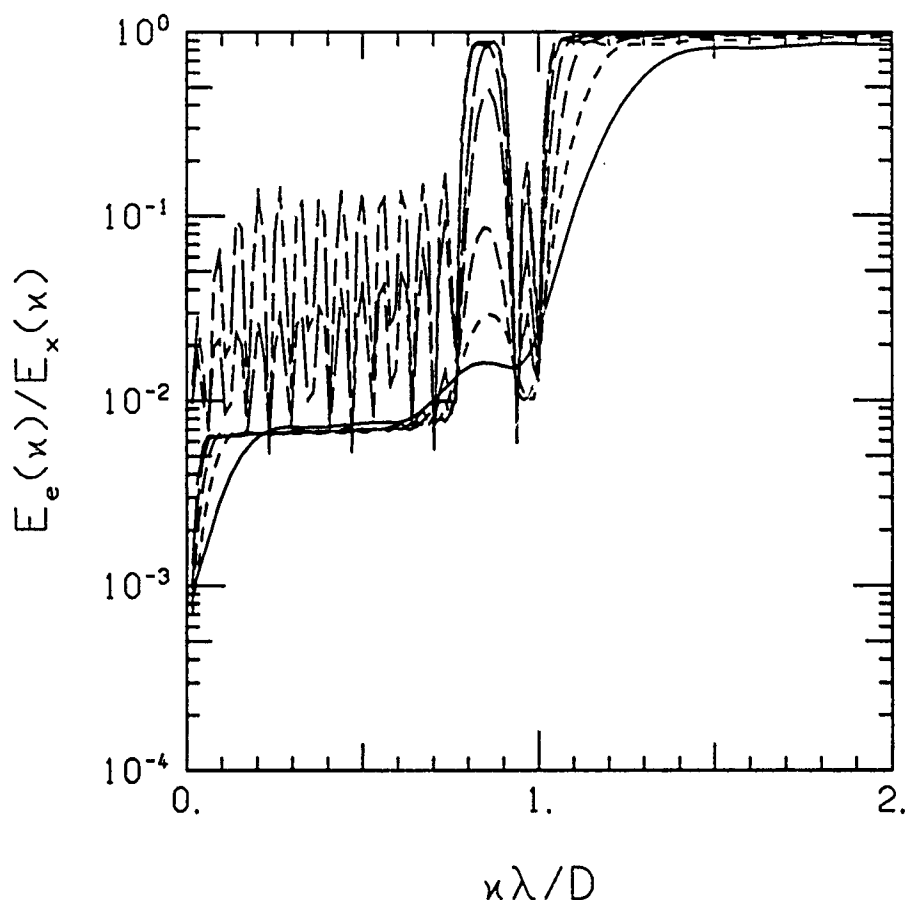


Figure 1.2. Spatial Frequency Component Estimation Error.

Results are shown here for the six-element sparse array depicted in Fig. 1.1, for object of size $3\lambda/D$, $6\lambda/D$, $9\lambda/D$, ..., $21\lambda/D$. Each curve represents the mean square error, $E_e(\kappa)$, in our estimate of a spatial frequency component, κ , normalized with respect to the mean square value expected for that component, $E_x(\kappa)$. Minimum variance estimation with $\frac{1}{4}\lambda/D$ sized pixels and an initial signal-to-noise ratio, SNR_{RAW} , of 50 dB is utilized in developing these results. It should be noted in studying the results depicted here, that only for the $18\lambda/D$ and $21\lambda/D$ cases do the curves show any "excess" noise for spatial frequencies below $\frac{13}{17}D/\lambda$ —even for those spatial frequencies for which the MTF, as shown in Fig. 1.1b, is equal to zero.

In Fig. 1.3 we show related results, only for all object sizes from $1\lambda/D$, $2\lambda/D$, ..., to $32\lambda/D$, and in terms of the rms error in the enhanced image. The results are shown as a function of spatial frequency cut-off, κ_0 , in the enhanced image. The signal to noise ratio is defined by the equation

$$\text{SNR}_{\text{EST}} = 20 \log_{10}(\sigma_{\hat{x}}/\sigma_n), \quad (1.7)$$

where $\sigma_{\hat{x}}$ denotes the standard deviation in the pixel values of the enhanced image. The results shown in Fig. 1.3 may be considered as being obtained as a spatial frequency integral over (an extended set of) the results shown in Fig. 1.2.

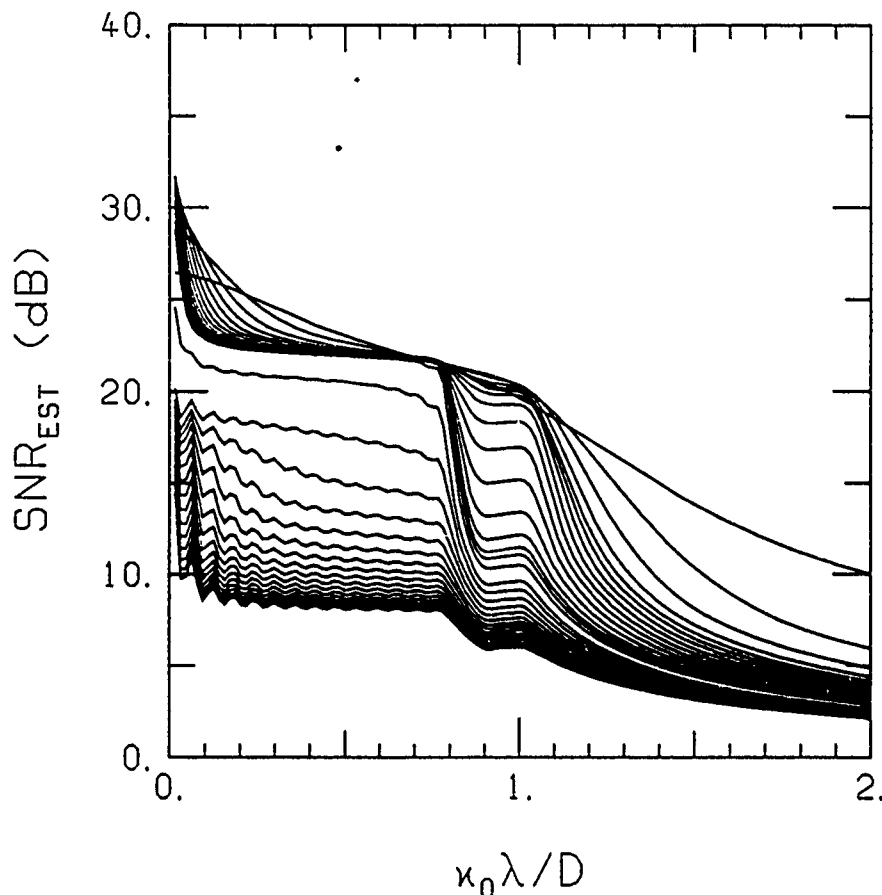


Figure 1.3. Enhanced Image Signal-to-Noise Ratio as a Function of Spatial Frequency Cut-off. The results shown here may be thought of as being developed from those in Fig. 1.2 (except that here we consider object sizes of $1\lambda/D$, $2\lambda/D$, $3\lambda/D$, ..., $32\lambda/D$), by integrating over spatial frequency up to the cut-off frequency, κ_0 . These results correspond to the mean square error in the estimated pixel value of the enhanced image, normalized by the variance (associated with the object's statistics). It is to be noted that good results are obtained for object sizes up to $17\lambda/D$ (the inverse in the basic optical transfer function gap shown in Fig. 1.1b, for spatial frequency cut-offs up to $\frac{13}{17}D/\lambda$). For cut-offs up to D/λ good results are obtained only for object sizes about up to $6\lambda/D$, about equal to the inverse of the largest gap in the optical transfer function when considering spatial frequencies up to D/λ .

It is clear from an examination of these results that there is a break in enhanced image quality at a spatial frequency of D/λ and an earlier break at $\frac{13}{17}D/\lambda$ for all objects except those whose size is greater than $6\lambda/D$. Most important of all, we note that performance is seriously degraded for any spatial frequency cut-off for objects larger than $17D/\lambda$.

All of these results suggest to us that the critical consideration is the size of the object relative to the inverse in the largest gap between "islands" in the optical transfer function. If we take $\frac{13}{17}D/\lambda$ as our cut-off spatial frequency, then the largest gap is $\frac{1}{17}D/\lambda$ —and we get good results for all objects whose size is no greater than $17\lambda/D$. This is made quite explicit by the presentation of the data shown in Fig. 1.4. If we take D/λ as our cut-off spatial frequency, then the largest gap is $\frac{13}{17}D/\lambda = \frac{1}{5.67}D/\lambda$, and the only good results are for objects whose size is no greater than $6\lambda/D$. We conclude from these results and others like it, that super resolution, i.e., the ability to develop image data for spatial frequencies for which the value of the imaging system's optical transfer function is zero, can be obtained by linear processing of the raw image data. The (only) requirement is that the object be smaller than the inverse of the largest gap in the modulation transfer function, and that we formulate our image processing algorithm with this knowledge in hand.

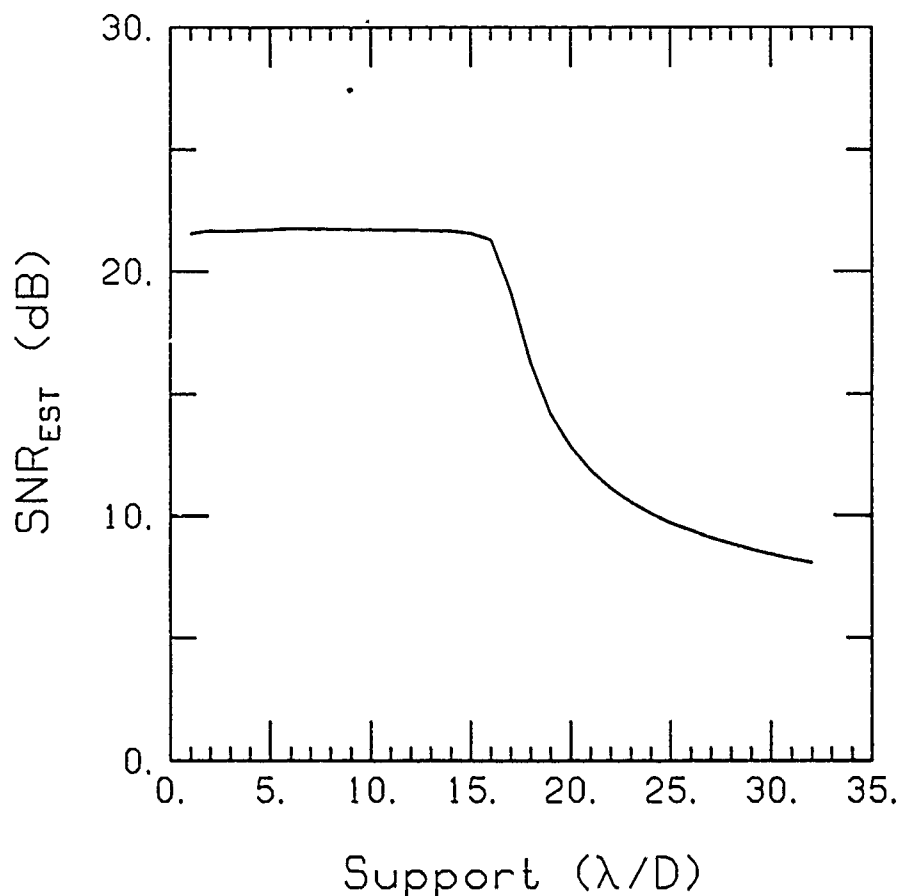


Figure 1.4. Enhanced Image Signal-to-Noise Ratio as a Function of Object Size.

The results shown here are taken directly from Fig. 1.3, for a cut-off spatial frequency of $\frac{13}{17}D/\lambda$. For this cut-off spatial frequency the largest gap between islands of nonzero value in the optical transfer function shown in Fig. 1.1b, is $\frac{1}{17}D/\lambda$. We consider it significant that the signal-to-noise ratio does not fall-off, i.e., apparently super resolution has been achieved so long as the object size is no greater than the inverse of this gap.

To understand why it should be that we can apparently develop the missing data for spatial frequencies for which the optical transfer function is zero—in essence achieving super resolution—we undertook the study reported in Chapter 5. In this work we show that if we start with an infinite extent white noise pattern—a pattern for which there is zero correlation between the amplitudes of the components at two distinct spatial frequencies, no matter how close the spatial frequencies—and

if we slice out a limited size sample of this random pattern and embed it in an infinite extent of zero values, then we now have an infinite random pattern for which there is non-zero correlation between different spatial frequency components! However, only if the components are for spatial frequency components that are close enough together is the correlation substantial. We find that the concept of spatial frequencies that are close enough together corresponds to the inverse of the difference of the two frequencies being greater than the size of the random sample that we extracted from the original random pattern.

We interpret this as meaning that in any finite support random pattern, spatial frequency components are necessarily correlated if the spatial frequencies are close enough, where close enough has to do with the inverse of the objects size. We believe that the minimum variance estimator naturally takes advantage of this correlation. This allows it to estimate the amplitude of the unmeasured spatial frequency components, using the amplitude of the measured components as a basis for the estimates—providing that the object is not too large to allow accurate estimation to be done this way. Put in a different way, we would say that the correlation indicates that the missing spatial frequencies do not contain any significant amount of new information; it's just a matter of "spreading out" the information we do get from the measurement in a somewhat different way, and this is what the algorithm does. It's still "super resolution" according to our definition of that term as providing image data for spatial frequencies for which the optical transfer function is equal to zero—but somehow the wonder of it, of *super* resolution seems to be gone. Nonetheless, we shall call it "super resolution".

It may be recalled that the CLEAN algorithm and other related algorithms possibly promising super resolution were nonlinear in nature. We have thus far restricted our attention to linear algorithms. This would seem to leave open the possibility that if we had used a suitable nonlinear algorithm we would have been able to process larger objects. To investigate this possibility, since the nonlinear algorithms do not lend themselves to closed form analytic treatment, as were used in Chapter 4, we switched to a Monte Carlo approach. We developed results using a Monte Carlo approach, and then compared those results (i.e., the rms errors) with the rms error for a linear algorithm—namely the least square algorithm. These results are presented in Chapter 6.

We carried out Monte Carlo runs for both the CLEAN algorithm and for a least square augmented by positivity requirement. In general we got essentially the same performance from the nonlinear as from the linear algorithm. This is illustrated in Fig. 1.5. Based on this and the other results developed in Chapter 6 we have concluded that there is no special virtue in the use of a nonlinear image enhancement algorithm—though there may be practical computational advantages—and that for analysis of expected performance we may rely on results developed with a linear algorithm being used for image enhancement.

But most important of all, we have concluded that Ken Johnston's suggestion is valid—that we could use an imaging array with widely spaced array elements and small element size to image an object in a geosynchronous orbit. Exactly how small an aperture diameter each array element can have is no doubt going to be strongly influenced by light gathering/signal strength considerations, and it may be possible to use aperture diameters small enough to avoid wavefront distortion. Certainly we now know that there is no reason to rule out the use of such small apertures on the basis of the consequent presence of zeros in the array's optical transfer function.

Regarding the spacing between array elements we note that if we were interested in imaging a 10 m diameter satellite in geosynchronous orbit, working at $0.5\ \mu\text{m}$ wavelength, then the basic array element spacing parameter would be 2.0 m. If light gathering considerations did not preclude it there is no reason why the size of the array elements could not be as small as 10 cm, or less.

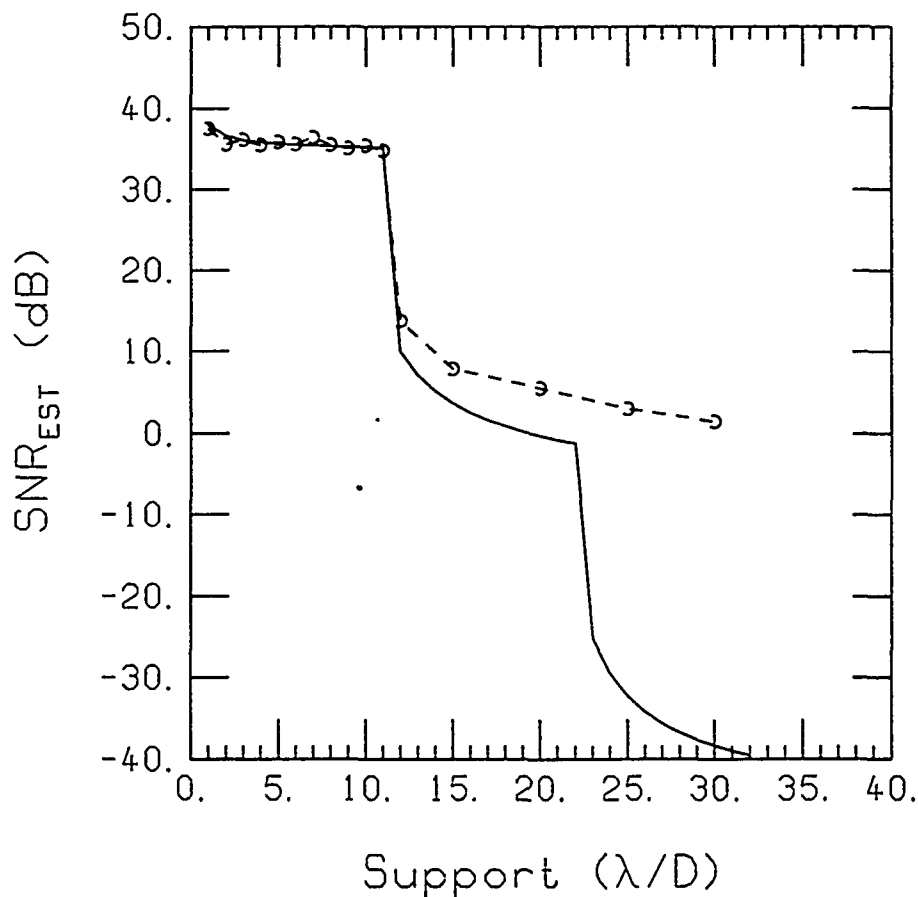


Figure 1.5. Comparison of Image Enhancement Results Using Least Square Estimation and Using CLEAN. The continuous curve represents the analytic results obtained for least square estimation. The dots (connected by the broken line) represent the results obtained from 40 Monte Carlo runs for each dot, for the CLEAN algorithm. It is obvious from consideration of these results that there is no significant advantage obtained by use of a nonlinear algorithm, i.e., CLEAN, rather than the linear image enhancement algorithm associated with least square error processing. (For more details regarding the results shown here the readers attention is directed to Fig. 6.33 and the associated discussion in Chapter 6.)

Chapter 2

Arrays of Optical Apertures: A Preliminary Investigation

(originally issued as TR-1012)

2.1. Introduction

This chapter presents the results of a preliminary investigation into the feasibility of obtaining useful images of objects viewed through a sparse array of optical apertures. If an array of apertures is sufficiently sparse, then the modulation transfer function (MTF) of the array will contain holes, or regions of zero response. Thus, the Fourier transform of a noise-free image of an object will also have corresponding regions of zero frequency response. To obtain a useful estimate of the object intensity distribution from the image, one needs to "fill in" the missing frequency regions. We refer to this process as (interpolative) super-resolution*. To do so, one must have constraint information about the object. For example, if one knows the support region of the object, then theoretically, knowledge of the Fourier transform of the object intensity distribution in one region can be used to fill in the missing information in another region (analytic continuation). A positivity constraint on the estimate of the object intensity distribution can also be used in addition to, or instead of, the support constraint to extend object frequency information. In this report, our primary emphasis is on the object support constraint.

We begin in Section 2.2 by developing a discrete model of an optical system so that the system response can be approximated with a matrix equation.

In Section 2.3 we assume we have first and second moment information about the object intensity distribution and observation noise, and we develop a minimum-variance estimate of the object using the support constraint only. The results indicate that the achievement of useful super-resolution with significantly sparse arrays requires very large signal-to-noise ratios.

In Section 2.4 we investigate the implication of the addition of a positivity constraint to the finite-support constraint. Here we assume no statistical knowledge and use a least-squares estimator instead. Unfortunately, the nonlinear nature of the positivity constraint greatly complicates the computation of useful object estimates, and our results to date are consequently rather limited. However, these results indicate that positivity, when used in addition to finite support, has the potential of significantly improving super-resolution performance.

2.2. Discrete Optical Model

The intensity distribution of a monochromatic image of an object can be described mathematically as the two-dimensional convolution of the intensity distribution of the object with the point-spread function of the optical system. Since the spatial frequency response of any optical system has finite support, the point-spread function is infinite in extent. In order to study super resolution quantitatively, we have chosen to use a discrete matrix model of convolution. This necessitates not only discretizing the optical model, but assuming that the point-spread function is of finite extent. Thus, the spatial frequency response of the model will not have finite support. However, we assume that if the extent of the point-spread function is chosen sufficiently large, truncation of the point-spread function will have a minimal effect on the results. We will use both one and two dimensional discrete models. The one-dimensional model, being computationally less complex, will enable the generation of a large body of results that are easily displayed. The two-dimensional model will be used to verify that the conclusions drawn from consideration of the one-dimensional results extend to two dimensions.

* The term "super resolution" has customarily been used to refer to the development of image/object information associated with spatial frequencies higher than those that go with the largest dimension of the aperture of the imaging system—spatial frequencies for which the imaging system's transfer function has zero value. It is entirely consistent with this to generalize the concept of "super resolution" to the development of image/object information for any spatial frequency for which the imaging system's transfer function has zero value—even if the spatial frequency is less than that associated with the largest dimension of the aperture. For a filled aperture this generalization is an empty one, but for a sparse array imaging system this generalization represents a significant "extension" of the basic concept of "super resolution". We may use the terms "interpolative super resolution" and "extrapolative super resolution" to distinguish between the case 1) where we are "filling in" missing data at spatial frequencies less than the highest the aperture can nominally provide, and 2) when we are "filling in" missing data at spatial frequencies greater than the highest the aperture can nominally provide, respectively.

2.2.1 One-Dimensional Model

Let the components of the $L \times 1$ vector $\mathbf{z}$ be the intensity pixels of the object line, let the $N \times 1$ vector $\mathbf{y}$ represent the image line intensity pixels, and let the $N \times L$ matrix B be the transformation from object line to image line (its columns contain shifted versions of the system point-spread-function). Then an image can be represented in the matrix equation

$$\mathbf{y} = B\mathbf{z} + \mathbf{n}, \quad (2.1)$$

where $\mathbf{n}$ is a $N \times 1$ noise vector. The product $B\mathbf{z}$ in (1) represents convolution of the object line with the point spread-function. Let the point spread function be $h(k)$, where $k = -K, -K + 1, \dots, 0, +1, +2, \dots, +K$. Then the B matrix is

$$B = \begin{bmatrix} h(-K) & 0 & \dots & 0 \\ h(-K+1) & h(-K) & \dots & 0 \\ \vdots & h(-K+1) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \dots & 0 \\ h(0) & \vdots & \dots & 0 \\ \vdots & h(0) & \dots & h(-K) \\ \vdots & \vdots & \dots & h(-K+1) \\ h(K-1) & \vdots & \dots & \vdots \\ h(K) & h(K-1) & \dots & \vdots \\ 0 & h(K) & \dots & h(0) \\ 0 & 0 & \dots & \vdots \\ \vdots & 0 & \dots & \vdots \\ \vdots & \vdots & \dots & h(K-1) \\ 0 & 0 & \dots & h(K) \end{bmatrix}. \quad (2.2)$$

Since B has N rows, from (2) we must have that $N = L + 2K$.

Now let the $M \times 1$ vector $\mathbf{x}$ represent an object of finite contiguous support of length $M \leq L$ pixels located somewhere in the object line. We can write

$$\mathbf{z} = W\mathbf{x}, \quad (2.3)$$

where the $L \times M$ matrix W is of the form

$$W = \begin{bmatrix} & & & 0 & & \\ \text{---} & & & & & \text{---} \\ & 1 & & & & 0 \\ & & 1 & & & \\ & & & \ddots & & \\ & 0 & & & & 1 \\ \text{---} & & & & & \text{---} \\ & & & 0 & & \end{bmatrix}. \quad (2.4)$$

Thus, W comprises an $M \times M$ identity matrix embedded in a $L \times M$ matrix of zeros. Combining Eq.'s (2.3) and (2.1) yields

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}. \quad (2.5)$$

To complete the model, we need a point-spread function. To derive a point-spread function, we first need to choose an aperture function. To represent a sparse array of apertures in one dimension we will use a simple model of two small apertures of length d with outside spacing D , as shown in Fig. 2.1. The corresponding modulation transfer function (MTF) is given below, where κ is the spatial frequency variable and λ is wavelength;

$$MTF(\kappa) = \frac{1}{A} \int dx w(x + \frac{1}{2}\kappa\lambda) w(x - \frac{1}{2}\kappa\lambda). \quad (2.6)$$

Normally A is chosen to be the area (in this case the length) of the aperture, so that the MTF at zero spatial frequency is unity. However, we wish to "penalize" the MTF of a sparse aperture to reflect the fact that the total energy in the image plane of a sparse array will be reduced from that of a full array in proportion to the ratio $2d/D$. We will thus set A equal to D instead of the more conventional $2d$. The resulting MTF is shown in Fig. 2.2a. Fig. 2.2b shows the MTF when $d/D = \frac{1}{2}$ (the full aperture case). We note from Fig. 2.2a that, for $d < D/3$, there is a region of zero MTF over the range $d/\lambda \leq \kappa \leq D/\lambda - 2d/\lambda$. It is this region that will bear scrutiny in performance evaluations. Fig.'s 2.3-2.7 show the MTF on a log-log scale for various values of d/D .

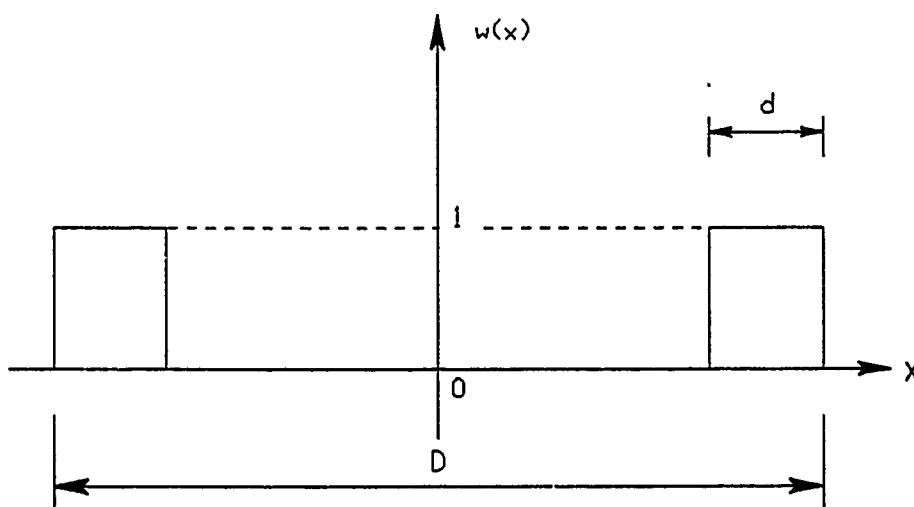


Figure 2.1. One-Dimensional Aperture Function

The point-spread function of an optical system is the inverse Fourier transform of the MTF function. Without going into the details, one can show that the inverse Fourier transform of the MTF shown in Fig. 2.2a is

$$h_c(x) = \frac{4d^2}{D\lambda} \cos^2 \left[\frac{\pi(D-d)}{\lambda} x \right] \left[\frac{\sin(\pi x d/\lambda)}{\pi x d/\lambda} \right]^2. \quad (2.7)$$

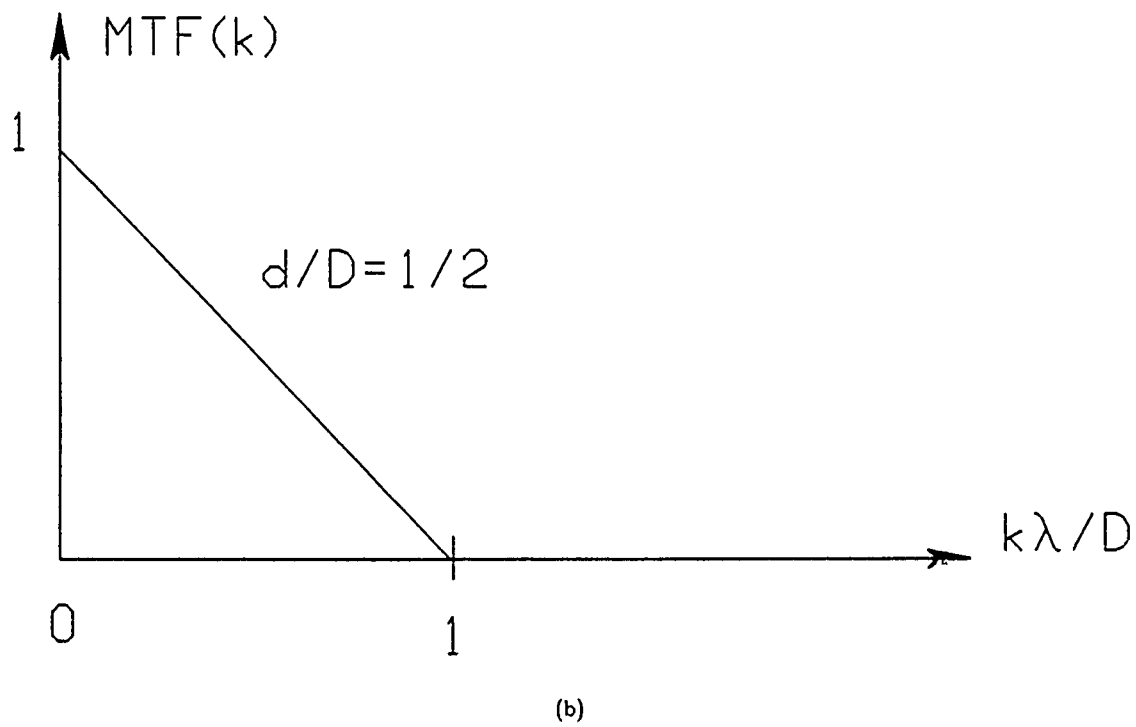
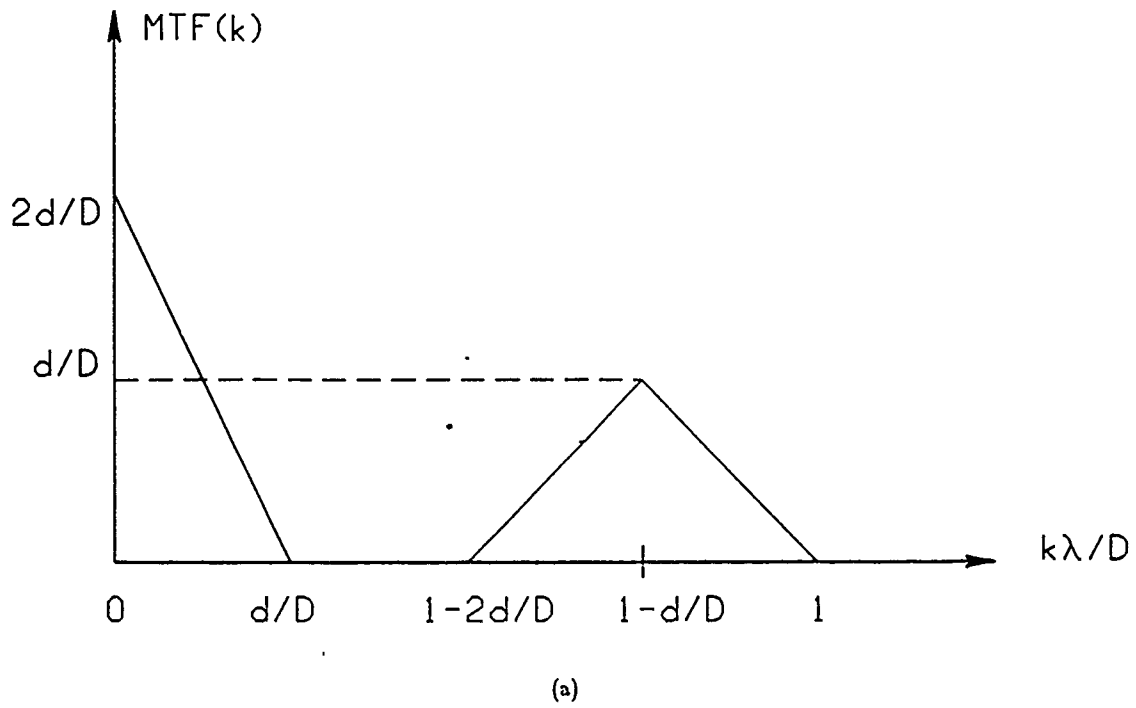


Figure 2.2. Modulation Transfer Function (MTF) for Aperture Function of Fig. 2.1.

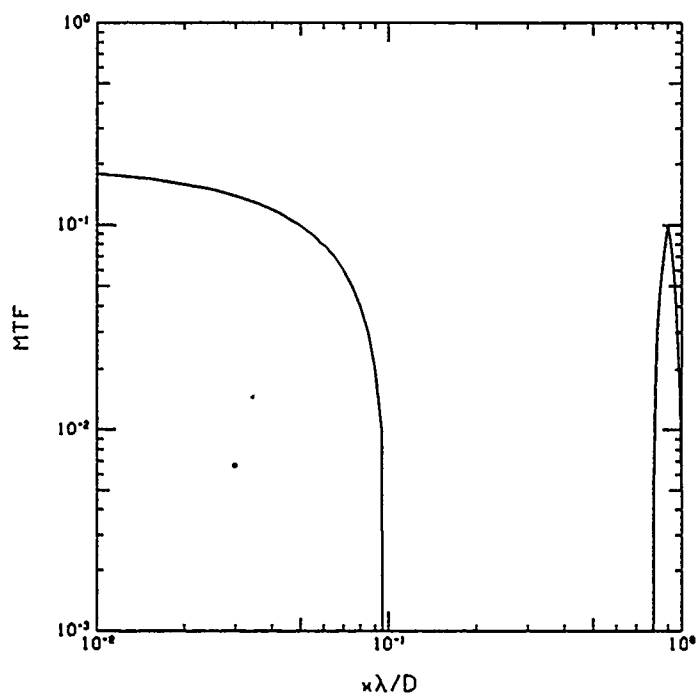


Figure 2.3. MTF of the Aperture of Fig. 2.1 with $d/D = 0.1$.
The region of zero MTF is $0.7 D/\lambda$ wide.

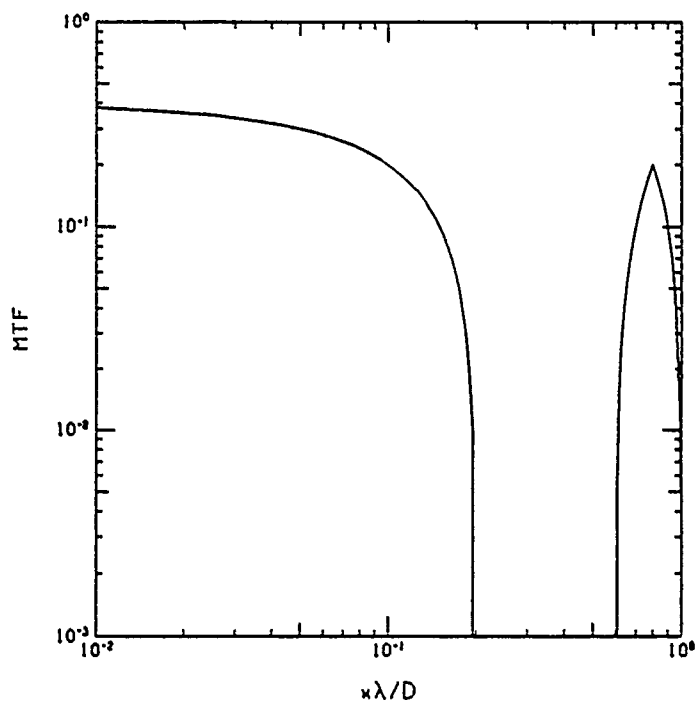


Figure 2.4. MTF of the Aperture of Fig. 2.1 with $d/D = 0.2$.
The region of zero MTF is $0.4 D/\lambda$ wide.

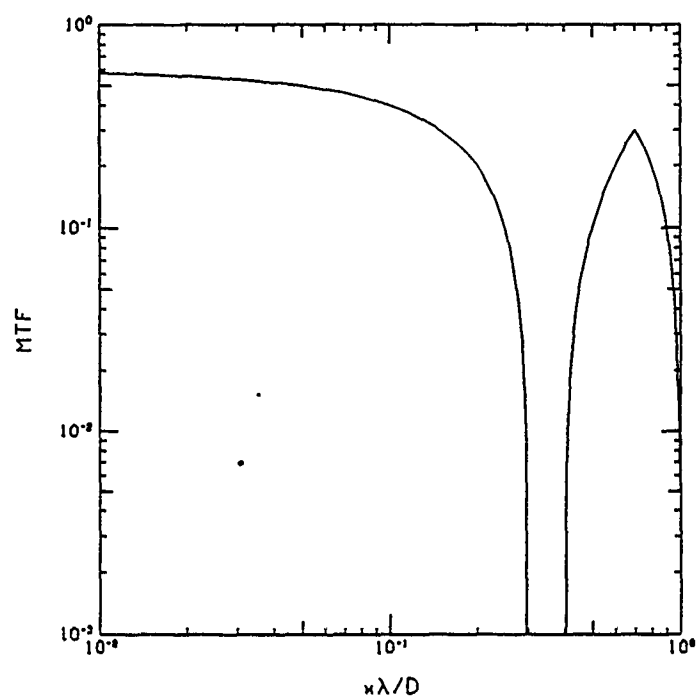


Figure 2.5. MTF of the Aperture of Fig. 2.1 with $d/D = 0.3$.
The region of zero MTF is $0.1 D/\lambda$ wide.

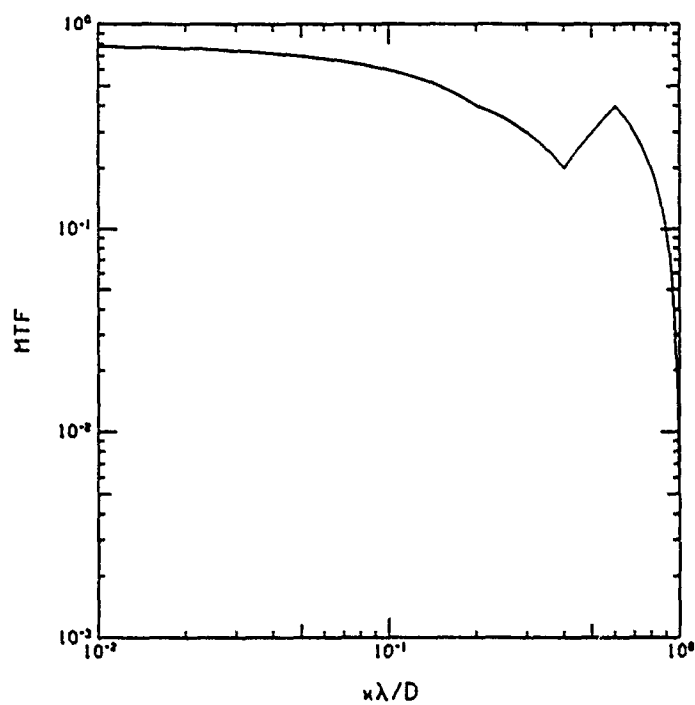


Figure 2.6. MTF of the Aperture of Fig. 2.1 with $d/D = 0.4$.

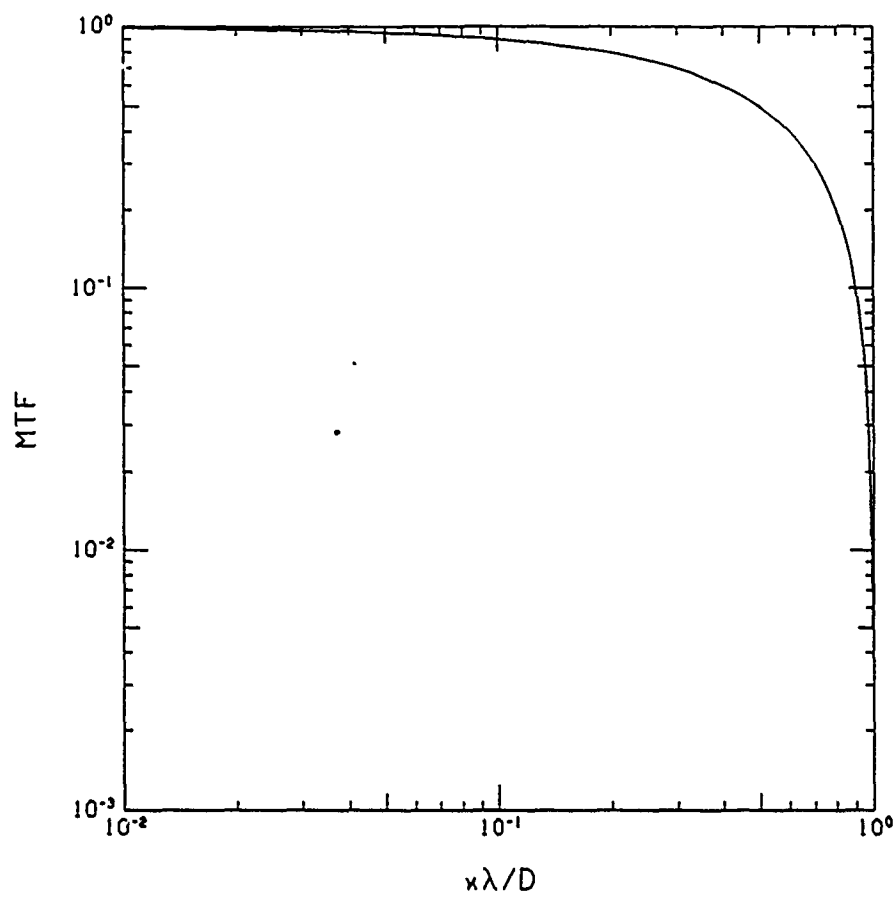


Figure 2.7. MTF of the Aperture of Fig. 2.1 with $d/D = 0.5$ (full aperture).

The subscript c denotes that this point-spread function is the continuous version. The discrete version of the point-spread function is found by sampling and truncating the continuous version. Let Δ denote the pixel spacing in the image line. Then the discrete point spread function used in the B matrix given by Eq. (2.2) is

$$h(n) = \begin{cases} \Delta h_c(n\Delta), & -K \leq n \leq K \\ 0, & \text{else.} \end{cases} \quad (2.8)$$

Before we go on to the two-dimensional model, we wish to discuss the evaluation of the performance of a processor. Any processor we chose will have, as an observable, the image vector $\mathbf{y}$. Using $\mathbf{y}$ and whatever *a priori* information is available, the processor will form an estimate of the object vector $\mathbf{x}$. Let us call this estimate $\hat{\mathbf{x}}$. Let the error in the estimate be

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}. \quad (2.9)$$

One common measure of performance is the mean-square-error, $\langle \mathbf{e}^T \mathbf{e} \rangle$. For our purposes this measure is not useful. It gives us no indication of how well the processor performs as a function of spatial frequency. The performance measure we will use is the energy spectrum of the error vector. The Fourier transform of the error vector is

$$\tilde{e}(\kappa) = \sum_n e(n) \exp[-i2\pi n\Delta\kappa], \quad (2.10)$$

where $e(n)$ is the n^{th} component of the error vector. The energy spectrum of the error vector, which we denote by $E_e(\kappa)$, is the expectation of the square of the magnitude of $\tilde{e}(\kappa)$ as given by Eq. (2.10):

$$\begin{aligned} E_e(\kappa) &= \langle |\tilde{e}(\kappa)|^2 \rangle \\ &= \sum_m \sum_n \langle e(n)e(m) \rangle \exp[-i2\pi\Delta\kappa(n-m)]. \end{aligned} \quad (2.11)$$

By plotting $E_e(\kappa)$ versus κ , we will determine how well the processor performs at each spatial frequency, thus determining its super-resolution capability. We note from Eq. (2.11) that we will need all entries of the error correlation matrix $\langle \mathbf{e} \mathbf{e}^T \rangle$ in the evaluation of the energy spectrum of the error, not just the diagonal terms that are required to evaluate mean-square-error. We further note that mean-square-error, $\langle \mathbf{e}^T \mathbf{e} \rangle$, is the integral of the error energy spectrum:

$$\langle \mathbf{e}^T \mathbf{e} \rangle = \Delta \int_{-\frac{1}{2\Delta}}^{\frac{1}{2\Delta}} d\kappa E_e(\kappa). \quad (2.12)$$

2.2.2 Two-Dimensional Model

We will generalize the one-dimensional model of the previous section to develop a two-dimensional model. We now have object and image planes, rather than lines, and a two-dimensional point spread function. Following our notation of the previous section, let the components of the vector $\mathbf{z}_n$ be the intensity pixels of the n^{th} row of the object plane, let the components of the vector $\mathbf{y}_n$ be the intensity pixels of the n^{th} row of the image plane, and let the matrix B_{nl} represent the influence that the l^{th} row of the object plane has on the n^{th} row of the image plane. Then we can write:

$$\mathbf{y}_n = \sum_l B_{nl} \mathbf{z}_l + \mathbf{n}_n, \quad (2.13)$$

where $\mathbf{n}_n$ is an additive noise vector for the n^{th} row of the image plane. Assume we truncate the point-spread function with a square window of dimension $2K+1$ on edge, and let the discrete point-spread function be $h(m,n)$, where m is the row index and n is the column index. Then the B_{nl} matrix of Eq. (2.13) has the form

$$B_{nl} = \begin{bmatrix} h(n-l, -K) & 0 & \dots & 0 \\ h(n-l, -K+1) & h(n-l, -K) & \dots & \vdots \\ \vdots & h(n-l, -K+1) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ h(n-l, 0) & \vdots & \dots & 0 \\ \vdots & h(n-l, 0) & \dots & h(n-l, -K) \\ \vdots & \vdots & \dots & h(n-l, -K+1) \\ \vdots & \vdots & \ddots & \vdots \\ h(n-l, K) & \vdots & \dots & \vdots \\ 0 & h(n-l, K) & \dots & \vdots \\ \vdots & 0 & \dots & h(n-l, 0) \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h(n-l, K) \end{bmatrix} \quad (2.14)$$

Now let us assume that the object has a contiguous region of support within the object plane. Let the components of $\mathbf{x}_l$ be the object pixels contained within the support region on the l^{th} line of the object plane. Then we can write

$$\mathbf{z}_l = W_l \mathbf{x}_l \quad (2.15)$$

We point out that the length of $\mathbf{x}_l$ can be, in general, different for each l (some may be of zero length). The form of the W_l matrix is the same as that shown in Eq. (2.4). Substituting Eq. (2.15) into Eq. (2.13), we have,

$$\mathbf{y}_n = \sum_l B_{nl} W_l \mathbf{x}_l + \mathbf{n}_n \quad (2.16)$$

now assume we have L rows in the object plane, N rows in the image plane, and let

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \quad (2.17)$$

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_L \end{bmatrix}, \quad (2.18)$$

$$\mathbf{n} = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_N \end{bmatrix}, \quad (2.19)$$

$$W = \begin{bmatrix} W_1 & & & 0 \\ & W_2 & & \\ & & \ddots & \\ 0 & & & W_L \end{bmatrix}, \quad (2.20)$$

and

$$B = \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1L} \\ B_{21} & B_{22} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ B_{N1} & \vdots & \dots & B_{NL} \end{bmatrix}, \quad (2.21)$$

Then Eq. (2.16) can be written in matrix form:

$$\mathbf{y} = B W \mathbf{x} + \mathbf{n}. \quad (2.22)$$

We need a point-spread function to complete our model. The aperture function we will use comprises three small circular apertures each of diameter d inscribed within a circle of diameter D . This aperture function is shown in Fig. 2.8. The corresponding MTF is the two-dimensional auto-correlation of the aperture function, which we can write as

$$\text{MTF}(\kappa_x, \kappa_y) = \frac{1}{A} \iint dx dy w(x + \frac{1}{2}\kappa_x \lambda, y + \frac{1}{2}\kappa_y \lambda) \times w(x - \frac{1}{2}\kappa_x \lambda, y - \frac{1}{2}\kappa_y \lambda). \quad (2.23)$$

As in the one-dimensional model, we take A to be the area of the "full aperture", i.e. of the circumscribing circle, rather than the area of the actual aperture. We write,

$$A = \frac{1}{4}\pi D^2. \quad (2.24)$$

The continuous two-dimensional point-spread function is the inverse Fourier transform of the MTF:

$$h_c(x, y) = \iint d\kappa_x d\kappa_y \text{MTF}(\kappa_x, \kappa_y) \exp[i2\pi(k_x x + k_y y)]. \quad (2.25)$$

Finally, the discrete point-spread function, $h(m, n)$, used in Eq. (2.14) is a sampled and truncated version of Eq. (2.25), which we can write as

$$h(m, n) = \begin{cases} \Delta^2 h_c(m\Delta, n\Delta), & -K \leq m, n \leq K \\ 0, & \text{else.} \end{cases} \quad (2.26)$$

Eq.'s (2.23) through (2.26) have been evaluated numerically for various values of d/D . Some examples are shown in Fig.'s 2.9 through 2.18.

To complete the model, we will compute the energy spectrum of the error. Let the estimate of the object intensity distribution be denoted by $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_L \end{bmatrix}, \quad (2.27)$$

where $\hat{x}_l$ is the estimate of the intensity distribution be denoted of the l^{th} row of the object. The error vector is

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - \hat{\mathbf{x}} \\ &= \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_L \end{bmatrix}. \end{aligned} \quad (2.28)$$

It is more convenient to express the error as a two-dimensional array of scalars and accordingly write

$$e(m, n) = (e_m)_n. \quad (2.29)$$

The Fourier transform of the error array is

$$\tilde{e}(\kappa_x, \kappa_y) = \sum_m \sum_n e(m, n) \exp[-i2\pi(m\kappa_x + n\kappa_y)]. \quad (2.30)$$

The energy spectrum of the error array is

$$\begin{aligned} E_e(\kappa_x, \kappa_y) &= \langle |\tilde{e}(\kappa_x, \kappa_y)|^2 \rangle \\ &= \sum_k \sum_l \sum_m \sum_n \langle e(k, l) e(m, n) \rangle \exp\{i2\pi[(m-k)\kappa_x + (n-l)\kappa_y]\}. \end{aligned} \quad (2.31)$$

2.3. Minimum-Variance Processor

The first processor (estimator) we will investigate is the so-called minimum variance processor. We assume that we have first and second moment information on the object vector $\mathbf{x}$ and the noise vector $\mathbf{n}$ of Eq.'s (2.5) and (2.22), i.e., we know the mean vector and covariance matrices of $\mathbf{x}$ and $\mathbf{n}$. Since we know the mean values of $\mathbf{x}$ and $\mathbf{n}$, and we can compute the mean value of $\mathbf{y}$, we will assume that the mean values have been subtracted out of Eq.'s (2.5) and (2.22), and our estimate $\hat{\mathbf{x}}$ is the deviation of $\mathbf{x}$ from its mean value. In other words, we have the observation model

$$\mathbf{y} = B\mathbf{W}\mathbf{x} + \mathbf{n}, \quad (2.32)$$

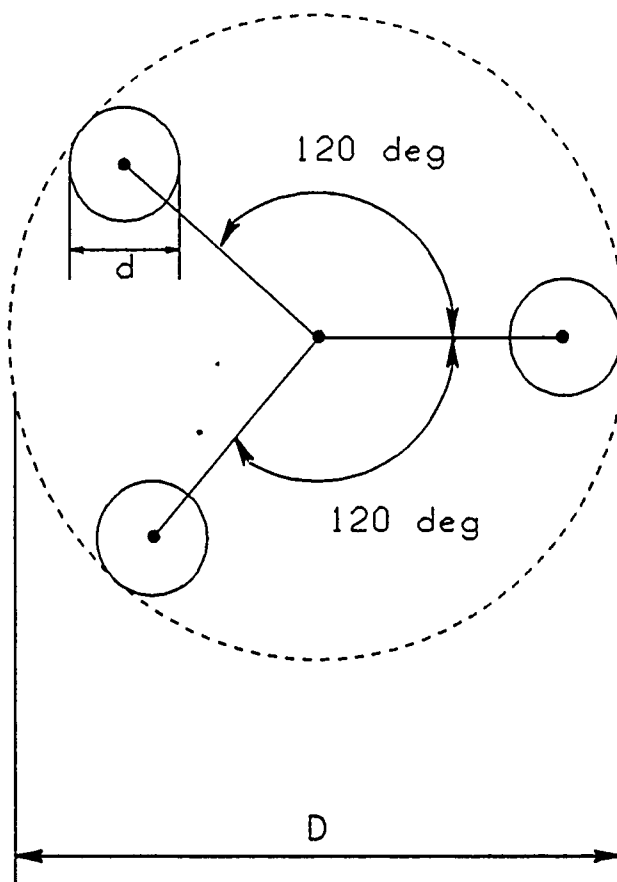


Figure 2.8. Aperture Function For Two-Dimensional Model.

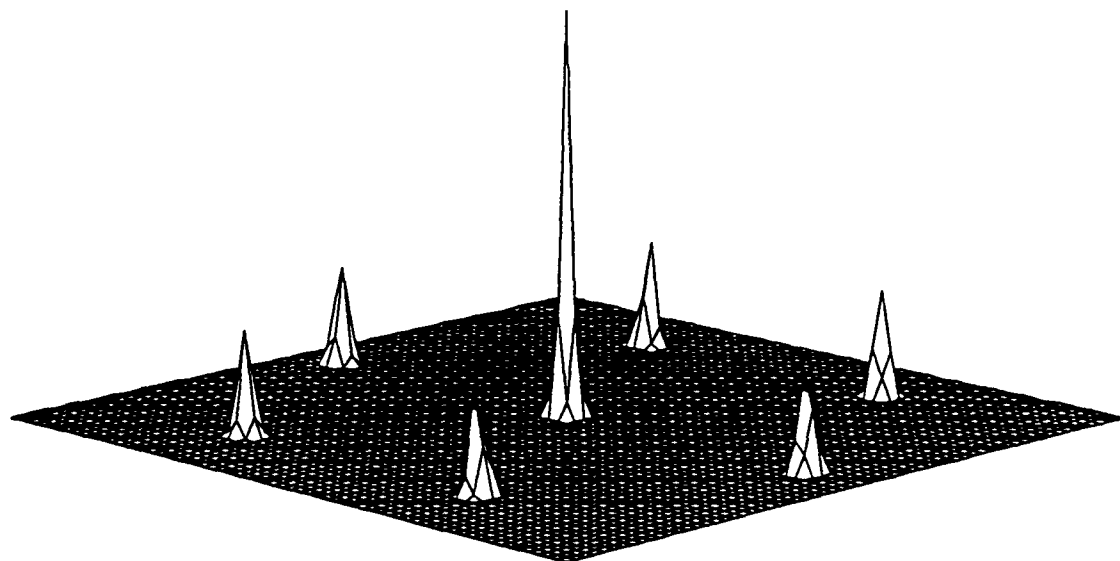


Figure 2.9. MTF of the Aperture of Fig. 2.8 with $d/D = 0.05$.
The image is $2 D/\lambda$ on edge. The height of the largest peak is $3(d/D)^2 = 0.0075$.

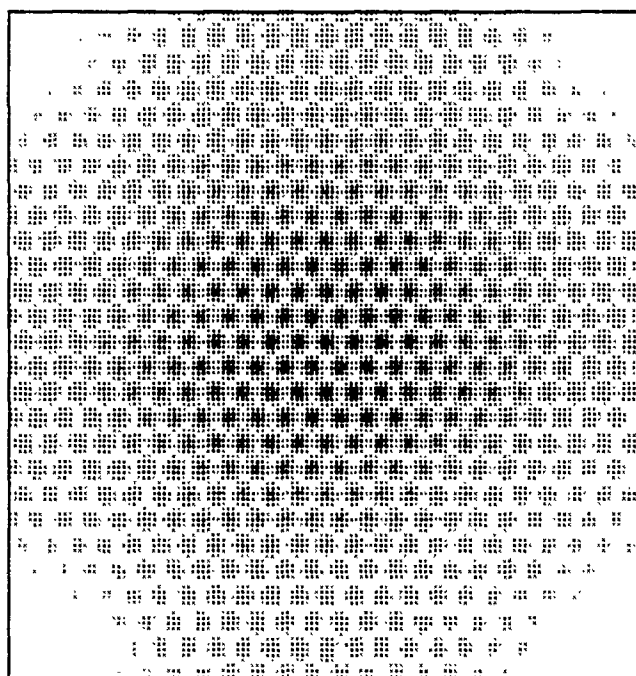


Figure 2.10. Greyscale Image of the Point-Spread Function of the Aperture of Fig. 2.8 With $d/D = 0.05$.
The image is $32 \lambda/D$ on edge.

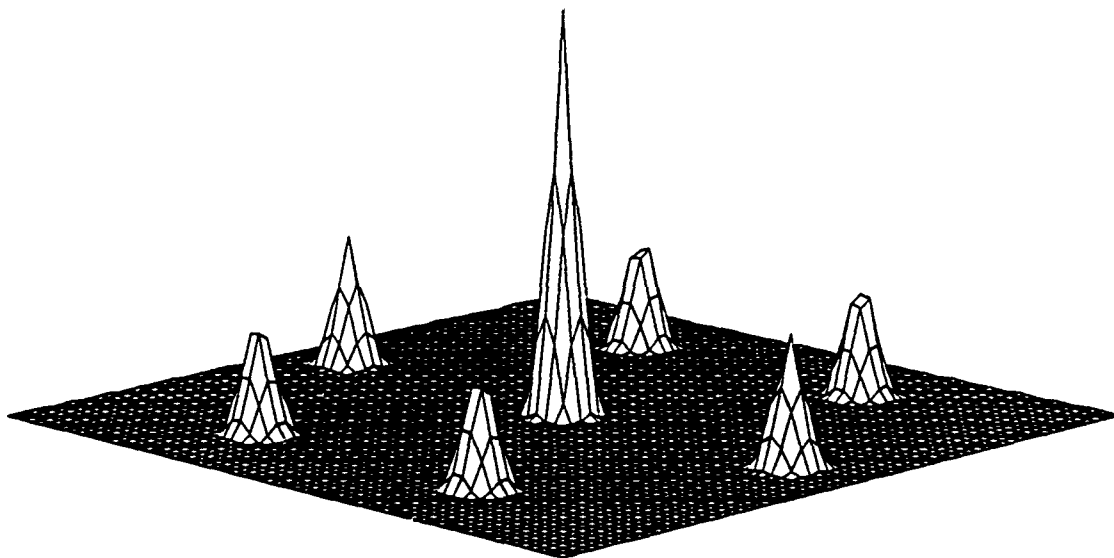


Figure 2.11. MTF of the Aperture of Fig. 2.8 With $d/D = 0.1$
 The image is $2d/\lambda$ on edge. The height of the highest peak is $3(d/D)^2 = 0.03$.

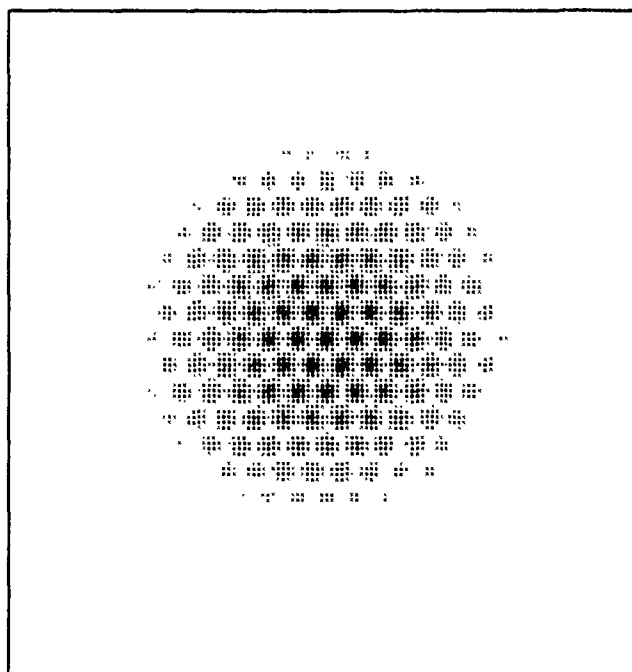


Figure 2.12. Grey-Scale Image of the Point-Spread Function of the Aperture of Fig. 2.8 With $d/D = 0.1$.
 The image is $32 \lambda/D$ on edge.

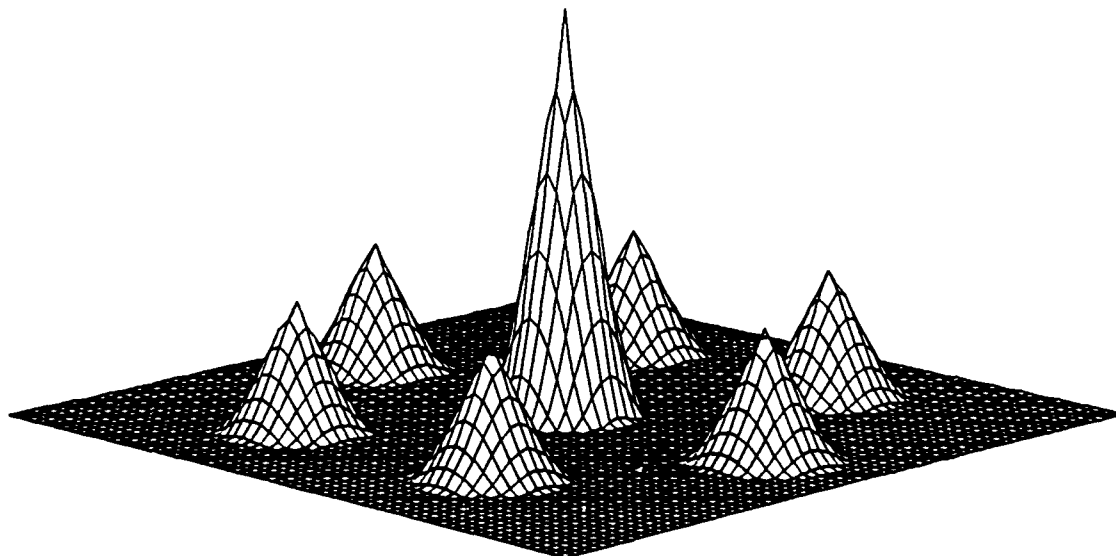


Figure 2.13. MTF of the Aperture of Fig. 2.8 With $d/D = 0.2$.
 The image is $2D/\lambda$ on edge. The height of the highest peak is $3 (d/D)^2 = 0.12$.

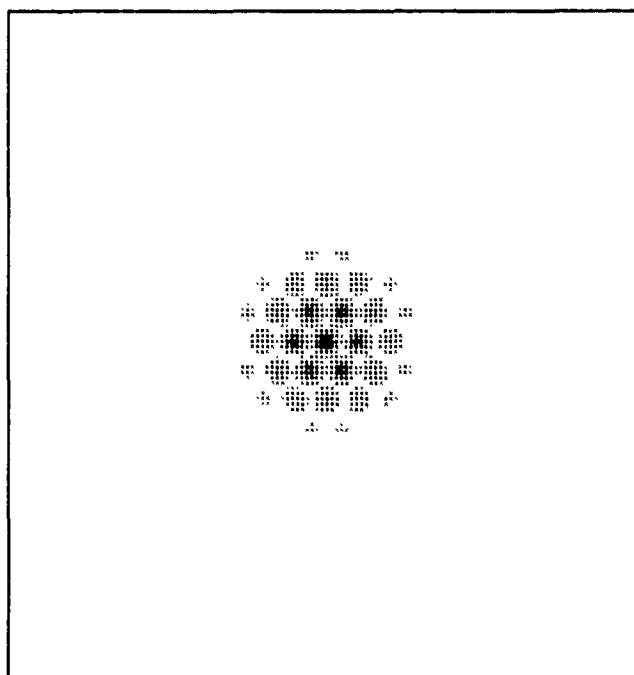


Figure 2.14. Grey-Scale Image of the Point-Spread Function of the Aperture of Fig. 2.8 With $d/D = 0.2$.
 The image is $32\lambda/D$ on edge.

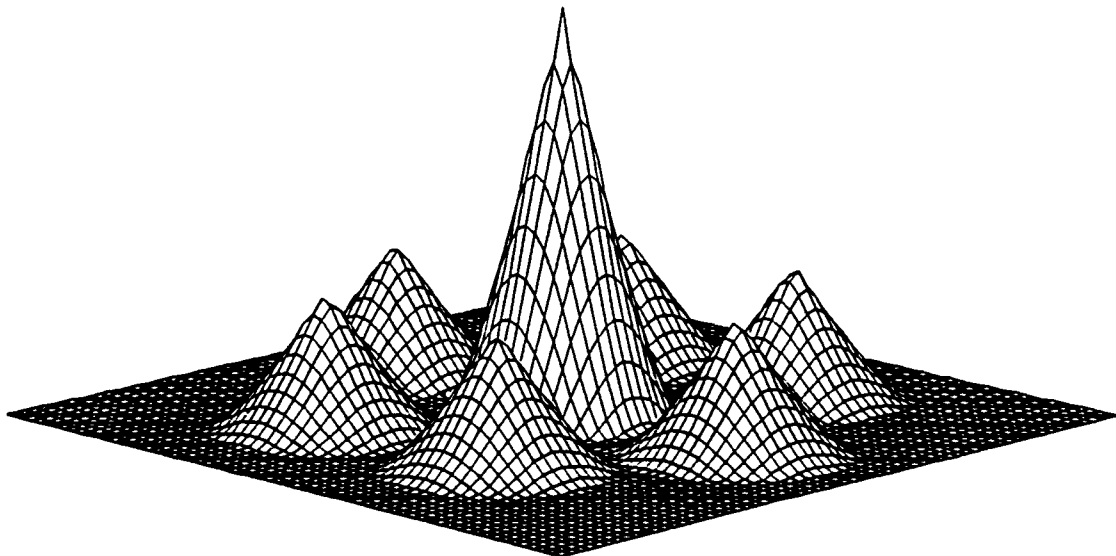


Figure 2.15. MTF of the Aperture of Fig. 2.8 With $d/D = 0.3$.
 The image is $2D/\lambda$ on edge. The height of the highest peak is $3(d/D)^2 = 0.27$.

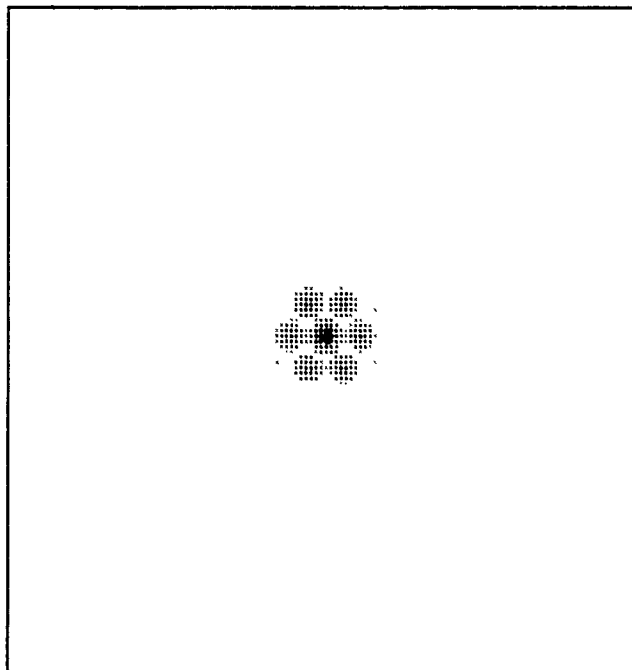


Figure 2.16. Grey-Scale Image of the Point-Spread Function of the Aperture of Fig. 2.8 With $d/D = 0.3$.
 The image is $32\lambda/D$ on edge.

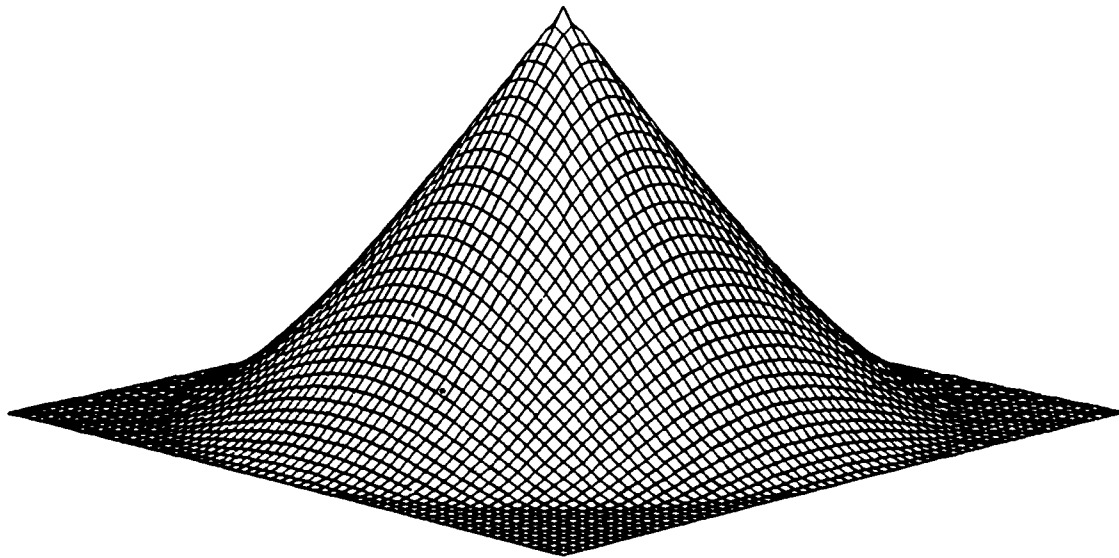


Figure 2.17. MTF of the Aperture of Fig. 2.8 With $d/D = 1$, (Full Aperture)
The image is $2D/\lambda$ on edge. The height at the peak is unity.

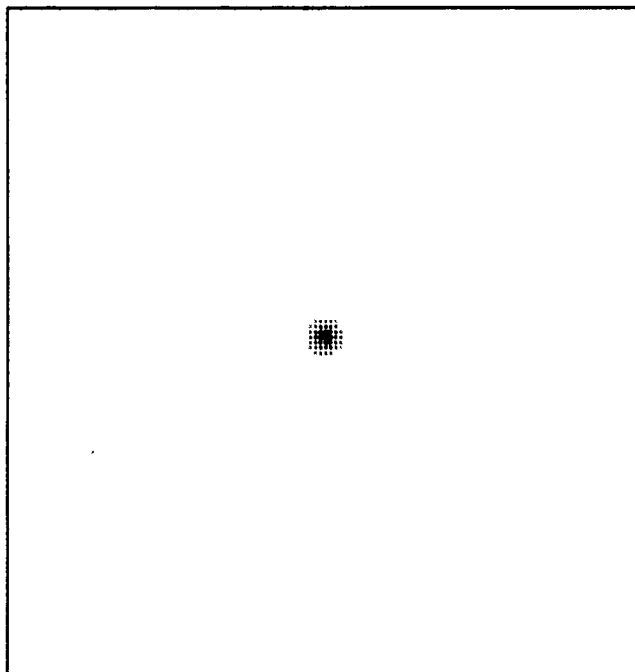


Figure 2.18. Grey-Scale Image of the Point Spread Function of the Full Aperture (Circle of Diameter D).
The image is $32\lambda/D$ on edge.

where all vectors in Eq. (2.32) have zero mean. We will use as an estimator a linear transformation of the observation vector:

$$\hat{\mathbf{x}} = H\mathbf{y}. \quad (2.33)$$

The error vector is

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - \hat{\mathbf{x}} \\ &= \mathbf{x} - H\mathbf{y}. \end{aligned} \quad (2.34)$$

We wish to choose H so as to minimize the variance of the error. Since $\mathbf{e}$ is zero-mean, this is equivalent to minimizing the mean-square-error, given by

$$\begin{aligned} \xi &= \langle \mathbf{e}^T \mathbf{e} \rangle \\ &= \text{Tr} \langle \mathbf{e} \mathbf{e}^T \rangle. \end{aligned} \quad (2.35)$$

Let e_n and x_n be the n^{th} components of the vectors $\mathbf{e}$ and $\mathbf{x}$, and let $\mathbf{h}_n$ be the n^{th} row of H . Then we can write:

$$e_n = x_n - \mathbf{y}^T \mathbf{h}_n. \quad (2.36)$$

We can see that choosing $\mathbf{h}_n$ to minimize $\langle e_n^2 \rangle$, for every n , minimizes ξ . To minimize $\langle e_n^2 \rangle$, we invoke the orthogonality principle (also known as the projection theorem): we choose $\mathbf{h}_n$ so that

$$\langle e_n \mathbf{y} \rangle = 0. \quad (2.37)$$

Using Eq. (2.36) in Eq. (2.37) yields the equation

$$\langle x_n \mathbf{y} \rangle = \langle \mathbf{y} \mathbf{y}^T \rangle \mathbf{h}_n. \quad (2.38)$$

Solving for $\mathbf{h}_n$ we get

$$\mathbf{h}_n = \langle \mathbf{y} \mathbf{y}^T \rangle^{-1} \langle x_n \mathbf{y} \rangle. \quad (2.39)$$

Thus, the optimum H matrix, H_o , has a value given by the expression

$$H_o = \langle \mathbf{x} \mathbf{y}^T \rangle \langle \mathbf{y} \mathbf{y}^T \rangle^{-1}. \quad (2.40)$$

The covariance matrices in Eq. (2.40) can be computed using Eq. (2.32), yielding

$$\langle \mathbf{x} \mathbf{y}^T \rangle = \langle \mathbf{x} \mathbf{x}^T \rangle (B\mathbf{W})^T, \quad (2.41)$$

$$\langle \mathbf{y} \mathbf{y}^T \rangle = (B\mathbf{W}) \langle \mathbf{x} \mathbf{x}^T \rangle (B\mathbf{W})^T + \langle \mathbf{n} \mathbf{n}^T \rangle. \quad (2.42)$$

In computing Eq. (2.41) and (2.42), it was assumed that this object and noise vector are uncorrelated. Using Eq. (2.41) and (2.42) in Eq. (2.40) yields

$$\begin{aligned} H_o &= \langle \mathbf{x} \mathbf{x}^T \rangle (B\mathbf{W})^T \left[(B\mathbf{W}) \langle \mathbf{x} \mathbf{x}^T \rangle (B\mathbf{W})^T + \langle \mathbf{n} \mathbf{n}^T \rangle \right]^{-1} \\ &= R_{xx} G^T [G R_{xx} G^T + R_{nn}]^{-1}, \end{aligned} \quad (2.43)$$

where

$$R_{xx} = \langle \mathbf{x} \mathbf{x}^T \rangle, \quad (2.44)$$

$$R_{nn} = \langle \mathbf{n} \mathbf{n}^T \rangle, \quad (2.45)$$

$$G = B\mathbf{W}. \quad (2.46)$$

One can apply a matrix inversion lemma to Eq. (2.43) to show that

$$H_o = (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1} G^T R_{nn}^{-1}. \quad (2.47)$$

In order to compute the energy spectrum of the error, we need the error covariance matrix. Using Eq. (2.34) we can write

$$\langle ee^T \rangle = \langle ex^T \rangle - \langle ey^T \rangle H_o^T. \quad (2.48)$$

Since we have chosen H_o so that each component of the error vector is orthogonal to the observation vector y , the second term in Eq. (2.48) must be zero. Therefore, we have

$$\begin{aligned} \langle ee^T \rangle &= \langle ex^T \rangle \\ &= \langle x\alpha^T \rangle - H_o \langle yx^T \rangle. \end{aligned} \quad (2.49)$$

Using Eq.'s (2.41) and (2.44)-(2.47) in Eq. (2.49) yields

$$\begin{aligned} \langle ee^T \rangle &= R_{xx} - (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1} G^T R_{nn}^{-1} G R_{xx} \\ &= (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1} [(G^T R_{nn}^{-1} G + R_{xx}^{-1}) R_{xx} - G^T R_{nn}^{-1} G R_{xx}] \\ &= (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1}. \end{aligned} \quad (2.50)$$

To simplify computation of Eq. (2.50), we will assume that both the object and noise vectors are white, i.e.,

$$R_{xx} = \sigma_x^2 I, \quad (2.51)$$

$$R_{nn} = \sigma_n^2 I, \quad (2.52)$$

where σ_x^2 and σ_n^2 are the variances of object and noise pixels, respectively. If we define a signal-to-noise ratio as

$$\text{SNR} = \sigma_x^2 / \sigma_n^2, \quad (2.53)$$

then Eq. (2.50) can be written

$$\langle ee^T \rangle = (G^T G (\text{SNR}) + I)^{-1} \sigma_x^2. \quad (2.54)$$

The numerator of the SNR defined by Eq. (2.53) is referenced to the object plane and the denominator is referenced to the image plane. Can we relate this to a similarly-defined SNR totally referenced to the image plane? In general, the answer is no. Such an SNR would vary from pixel to pixel. Rather than attempting to develop a suitable definition for SNR in the image plane, we will instead simply make the following observation. Given a very large object of uniform intensity, the intensity of the image is also uniform and is equal to the intensity of the object scaled by the MTF evaluated at zero spatial frequency. One can show that the scale factor for the one-dimensional model is $2d/D$ and for the two-dimensional model is $3(d/D)^2$ (evaluate Eq. (2.6) and (2.23), respectively, at zero spatial frequency).

2.3.1 Minimum-Variance Results for the One-Dimensional Case.

In order to proceed, we must choose several parameters. Two of the parameters that strongly interact are pixel spacing Δ and the length of the point-spread function $(2K+1)\Delta$. Since the MTF of Fig. 2.2 is band-limited to the range $|\kappa| \leq D/\lambda$, a pixel spacing of $\Delta \leq \frac{1}{2} \frac{\lambda}{D}$ (reciprocal of the Nyquist rate) would be sufficient to avoid aliasing if we were not truncating the point-spread function. However, truncation of the point-spread function means that the actual MTF is no longer band-limited, introducing the possibility of significant distortion of the MTF from aliasing. We thus

pick a pixel spacing of $\Delta = \frac{1}{4} \frac{\lambda}{D}$, which corresponds to a spatial sampling frequency of $4D/\lambda$, which is twice the nyquist rate for the MTF of Fig. 2.2. We next choose $K = 63$, which yields a point-spread function length of $31.75 \lambda/D$, sufficient to reduce aliasing to negligible levels. We will use an object line of $32 \lambda/D$ long ($L = 128$), allowing us to study support lengths up to $32 \lambda/D$. Having chosen L and K , the number of pixels in the image line is fixed at $N = 2L + K = 254$, yielding an image line length of $63.5 \lambda/D$.

We are now in a position to compute the energy spectrum of the error vector for various values of SNR, support length, and d/D , using Eq. (2.54) in Eq. (2.11). The results are shown in Fig.'s 2.19 through 2.23. The energy spectrum in each figure is normalized by the total energy of the object vector ($M\sigma_x^2$) to remove the dependence on the variance of an object pixel (see Eq. (2.54)).

The results are rather discouraging. For example, with $d/D = 0.1$ and $\text{SNR} = 10^4$ (Fig. 2.23c), the only support length which shows a significant amount of super-resolution is $1\lambda/D$ (roughly the size of the image of a point source through a full aperture of length D). We need to move to an aperture with $d/D = 0.3$ (a not very sparse aperture) to achieve super-resolution with a significant improvement in support length (Fig. 2.21c). Even in the case, a usable support length would be less than $8\lambda/D$. It is clear from these results, that, despite *a priori* statistical knowledge, achieving significant super-resolution performance using only a finite-support constraint on the object requires an enormously large signal-to-noise ratio. As we shall see in the next section, the two-dimensional case is somewhat more encouraging.

2.3.2 Minimum-Variance Results for the Two-Dimensional Case.

We will use the same parameters in the two-dimensional case as we used in the one-dimensional case: object and image pixels $\frac{1}{4} \frac{\lambda}{D}$ on edge ($\Delta = \frac{1}{4} \frac{\lambda}{D}$), a square object plane of $32 \lambda/D$ on edge, a square image plane of $63.5 \lambda/D$ on edge, and a square point-spread function of $31.75 \lambda/D$ on edge. Object support regions will be square with M pixels on edge ($\frac{M}{4} \frac{\lambda}{D}$ on edge). We can now compute the energy spectrum for the two-dimensional model for various values of SNR, support size, and d/D using Eq. (2.54) in Eq. (2.31), where the error vector in Eq. (2.54) is given by Eq. (2.28). Since the energy spectrum of the error vector is now a surface, we have chosen to display the results as a set of grey-scale images in Fig.'s 2.24-2.37. Each figure corresponds to a different aperture sparsity (d/D), support size, and SNR. As in the one-dimensional case, the energy spectrum in each figure is normalized by the total energy in the object ($M^2\sigma_x^2$). The worst case in each figure is a normalized error-vector energy-spectrum of unity, corresponding to black. Normalized energy spectra below 10^{-2} is displayed as white. Each figure is $4D/\lambda$ on edge. The region of most interest to us is the portion of the figures in a circle of diameter $2D/\lambda$, centered at the center of the square. To see how the energy surface relates to the aperture MTF, refer to Fig. 2.24a. This figure corresponds to $d/D = 0.1$, a support size of $7 \frac{\lambda}{D} \times 7 \frac{\lambda}{D}$, and $\text{SNR} = 10^2$. If we compare this figure to Fig. 2.11, which shows the MTF corresponding to $d/D = 0.1$, the source of the white diamond-shaped regions in Fig. 2.24a becomes clear. Although there is some super-resolution displayed in the figure, the resulting object estimates would probably not be useful. With an array of this sparsity, and a support region of $7\lambda/D$ on edge, we would probably need an SNR somewhere between 10^4 and 10^5 (Fig.'s 2.24c and 2.24d) to yield useful object estimates. To see how sensitive the results are to aperture sparsity, refer to Fig. 2.28. Here, useful object estimates appear to be achievable with a much-reduced $\text{SNR} = 10^2$ if we use an aperture with $d/D = 0.2$ (the MTF of which is shown in Fig. 2.13).

2.4. Least-Squares Processor and Results, One-Dimensional Case, Positivity Constraint.

In computing the results of Section 2.3, we assumed knowledge of the mean and covariance of the object and noise vectors, and we assumed only a finite-support constraint on the object. A probably more realistic situation is one where we are presented with an image and given the MTF of the optical system, and we are required to find the object which best explains the image in some sense, given no statistical knowledge. Furthermore, we may wish to use both finite support and

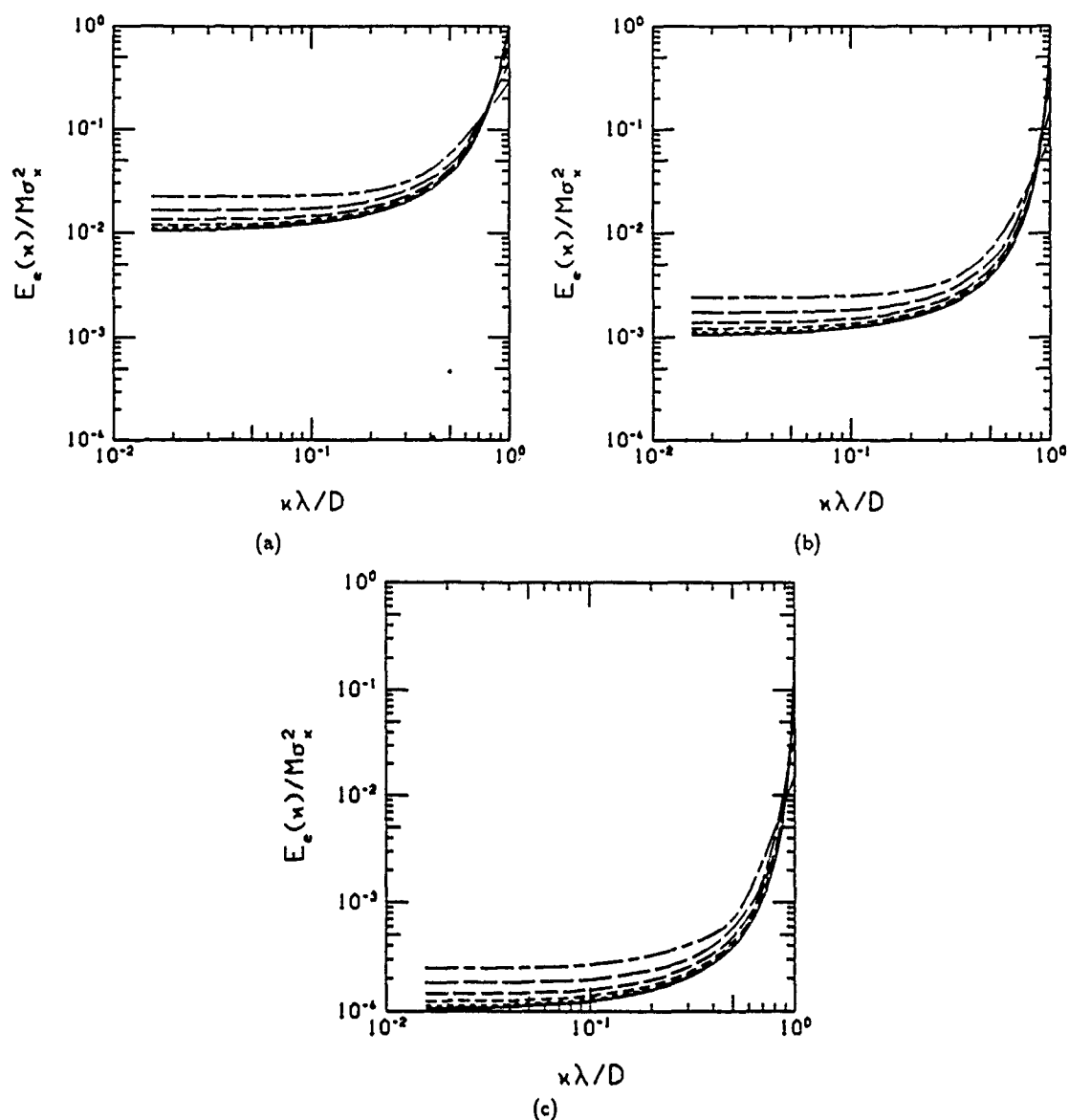


Figure 2.19. Normalized Energy Spectrum of The Error Vector with $d/D = 0.5$ (full aperture). Curve sets (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and 32 λ/D .

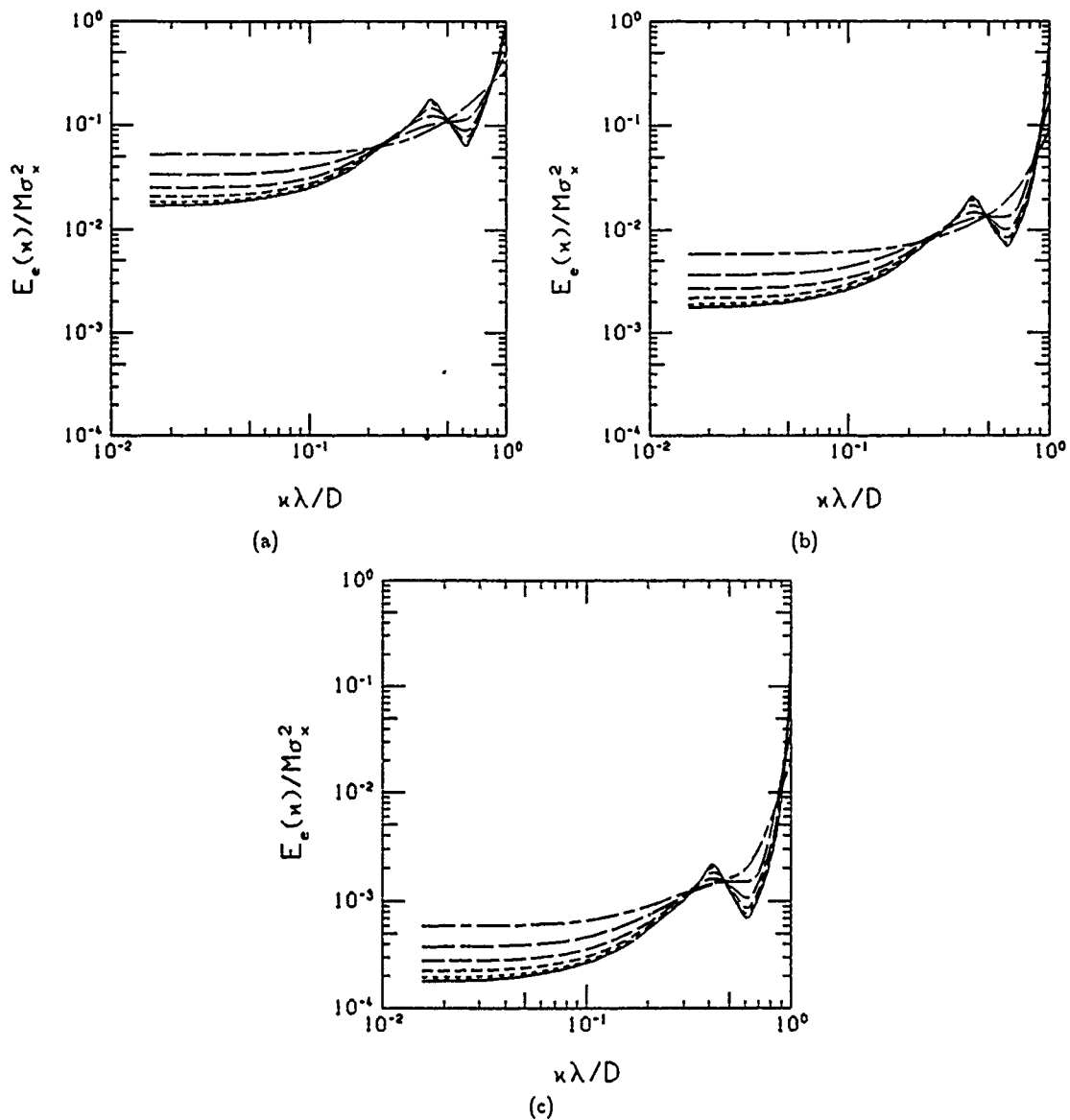


Figure 2.20. Normalized Energy Spectrum of The Error Vector with $d/D = 0.4$.
 Curve sets (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and $32 \lambda/D$.

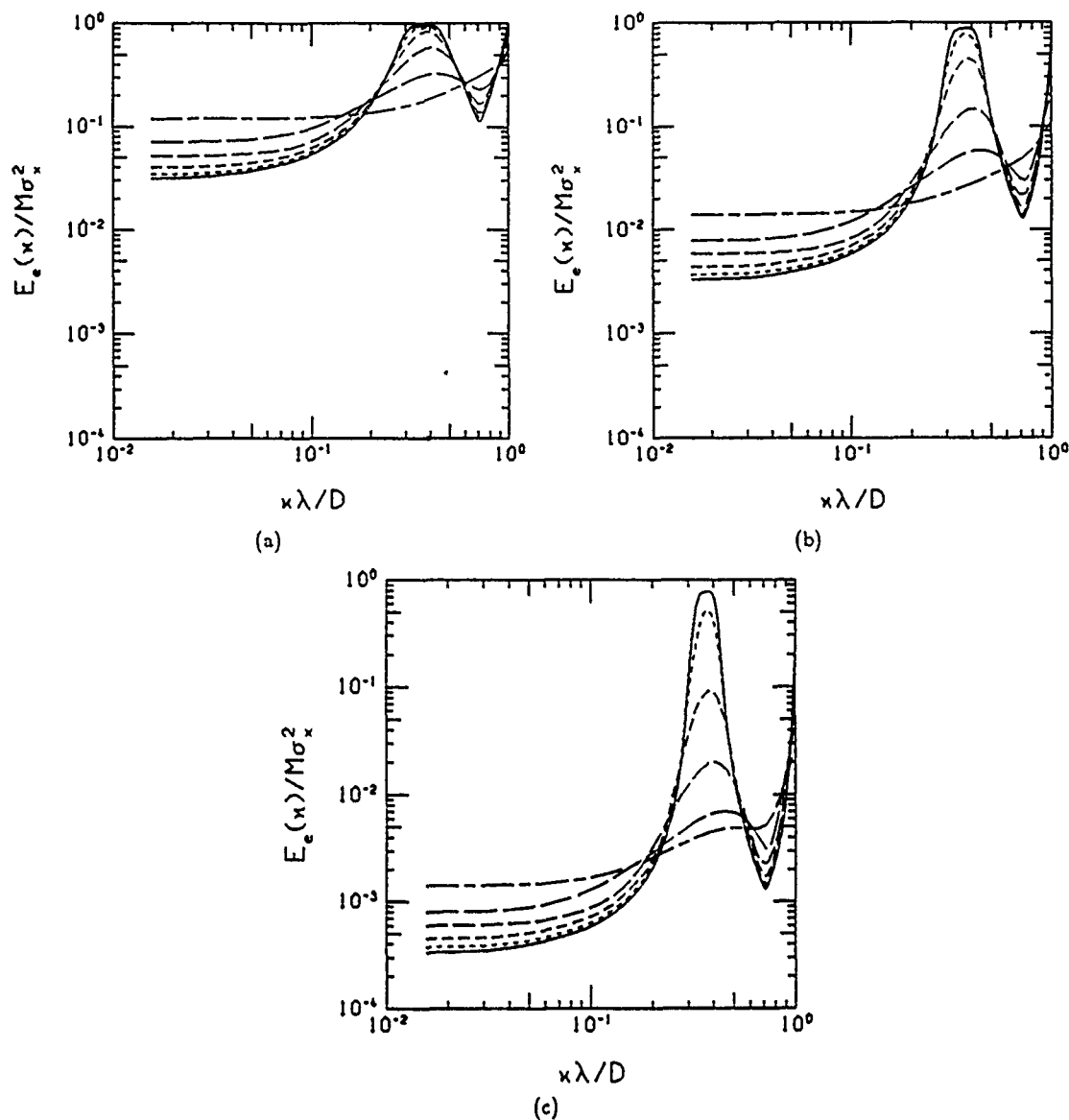


Figure 2.21. Normalized Energy Spectrum of The Error Vector with $d/D = 0.3$.
 Curve sets (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and $32 \cdot \lambda/D$.

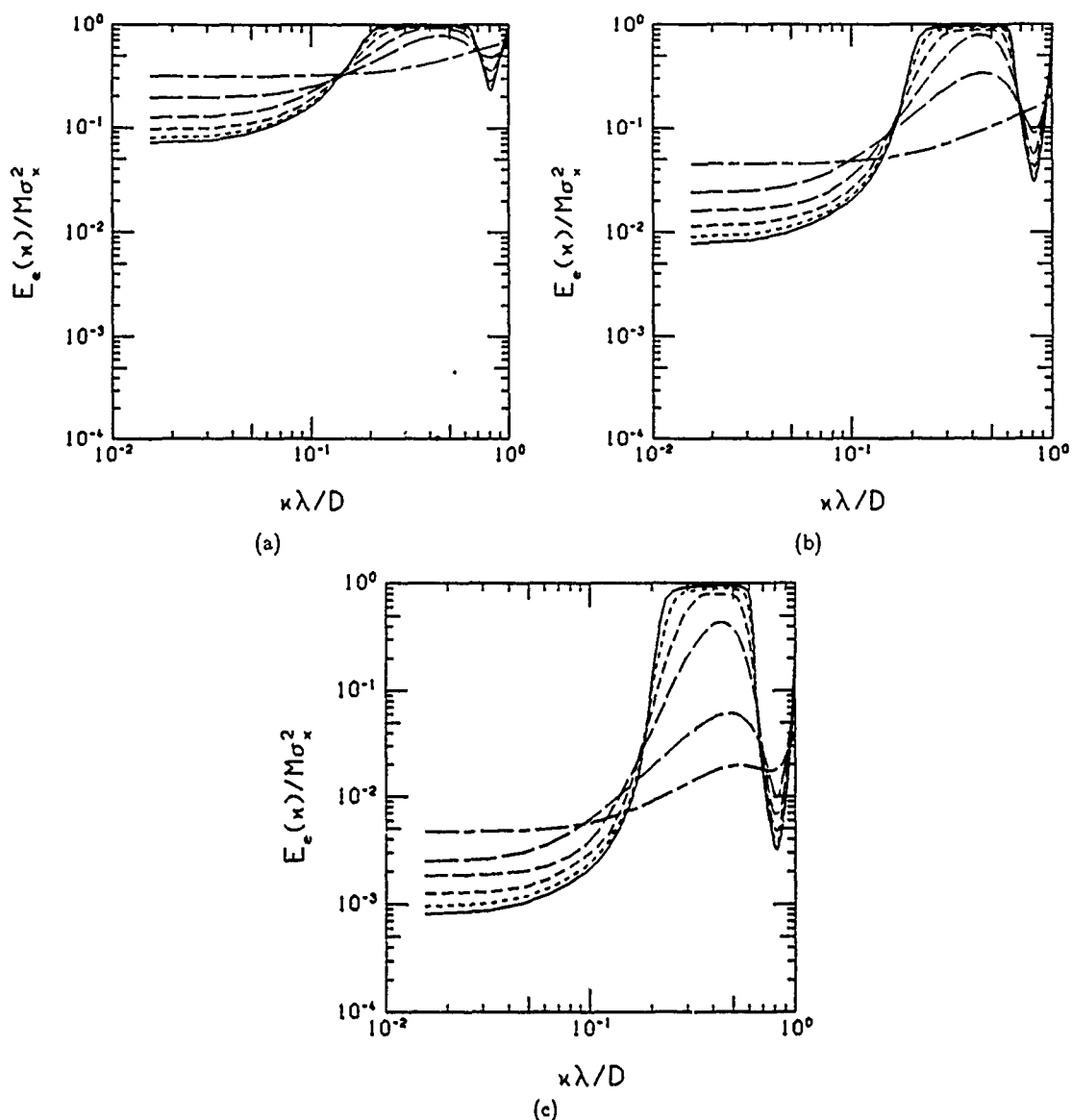


Figure 2.22. Normalized Energy Spectrum of The Error Vector with $d/D = 0.2$. Curve sets (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and 32 λ/D .

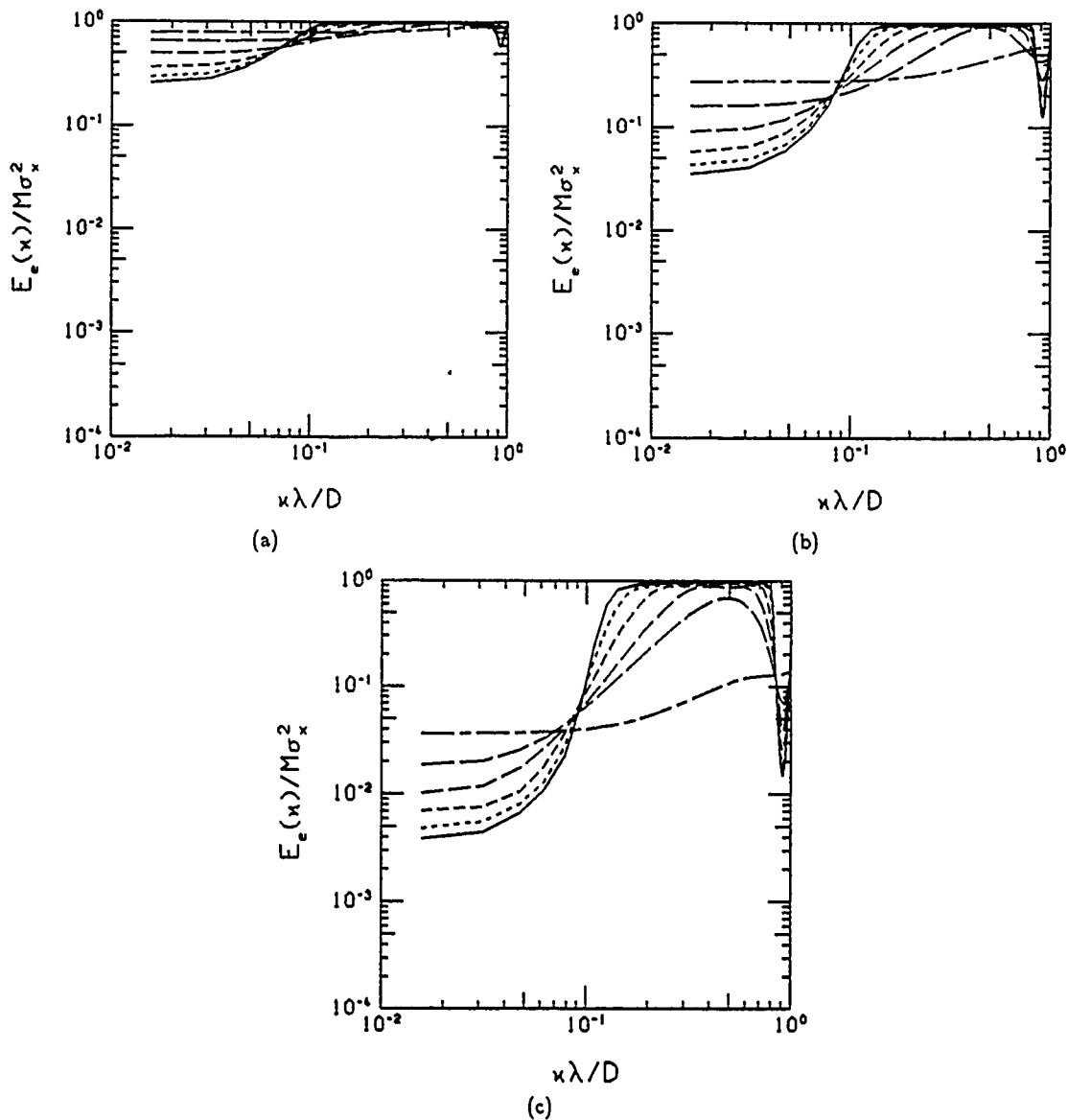
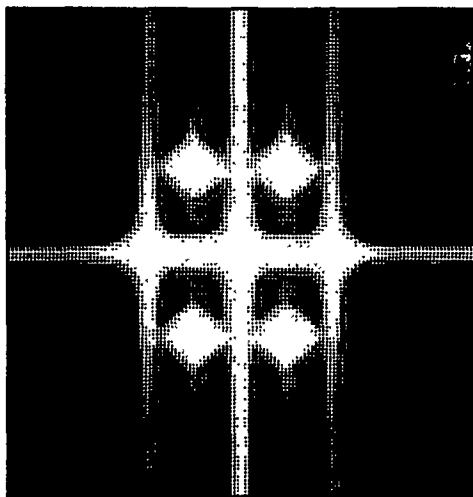
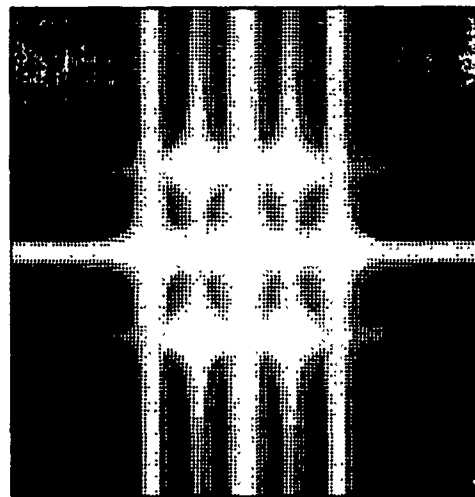


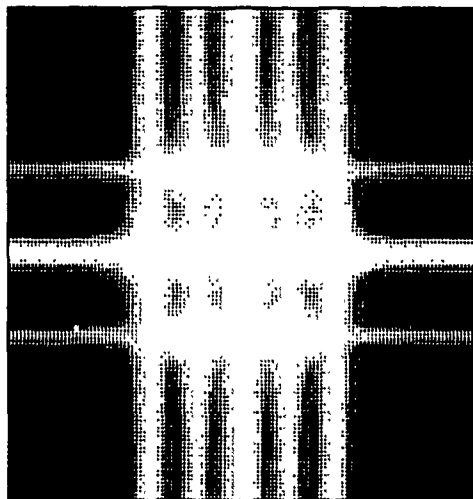
Figure 2.23. Normalized Energy Spectrum of The Error Vector with $d/D = 0.1$. Curve sets (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and 32 λ/D .



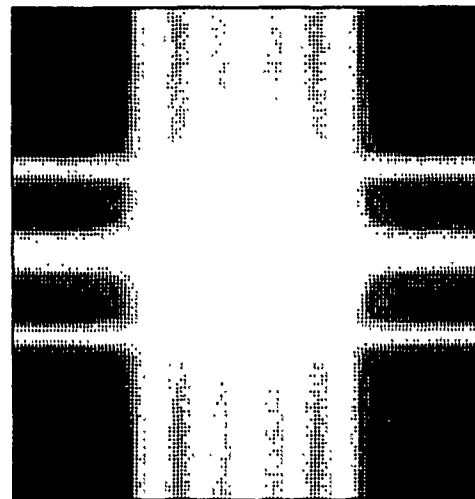
(a)



(b)



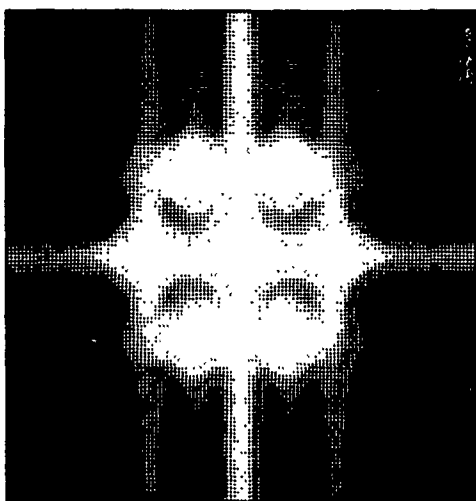
(c)



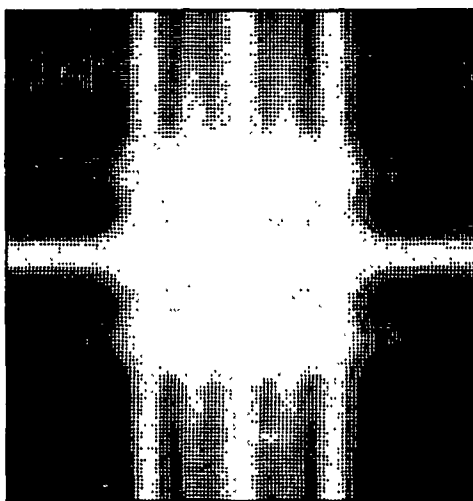
(d)

Figure 2.24. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

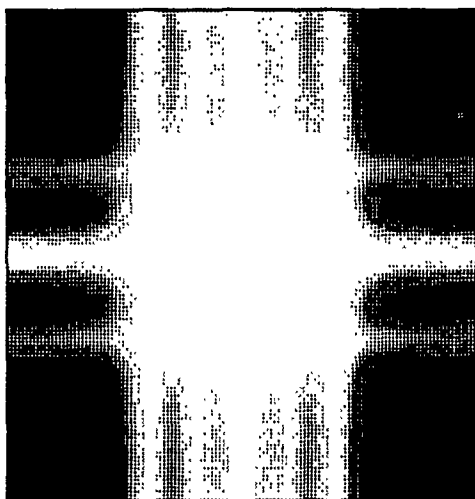
Support size is $7\lambda/D \times 7\lambda/D$ and $d/D = 0.1$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), (c), and (d) correspond to SNR's of 10^2 , 10^3 , 10^4 , and 10^5 , respectively.



(a)



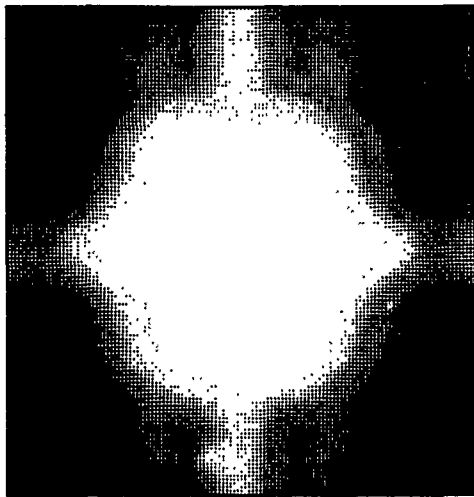
(b)



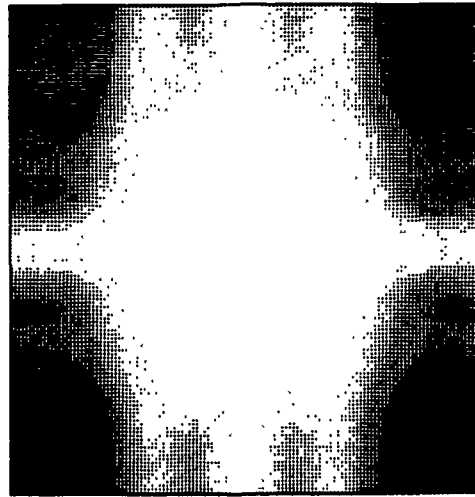
(c)

Figure 2.25. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

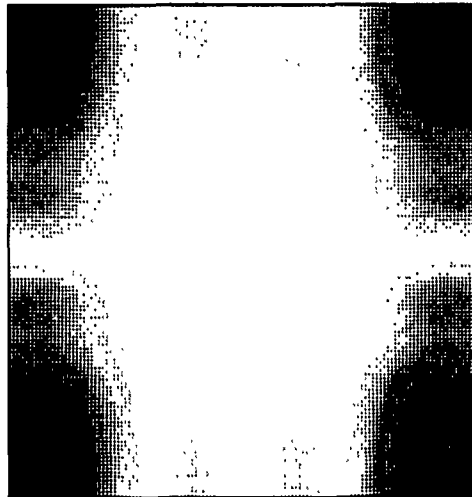
Support size is $4\lambda/D \times 4\lambda/D$ and $d/D = 0.1$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)



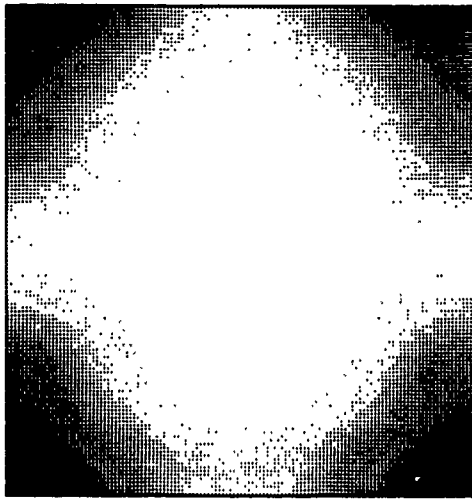
(b)



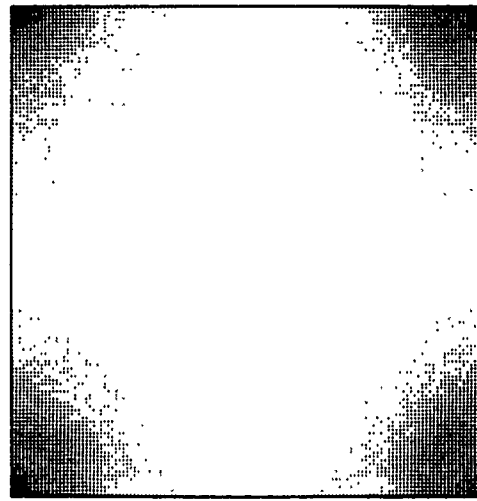
(c)

Figure 2.26. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

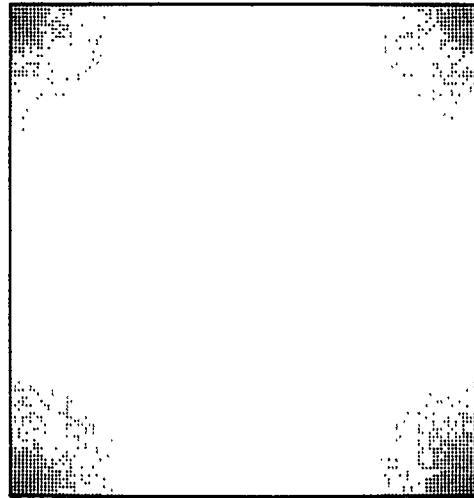
Support size is $2\lambda/D \times 2\lambda/D$ and $d/D = 0.1$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)



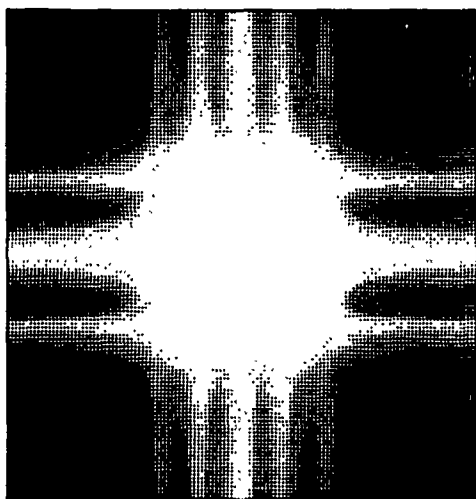
(b)



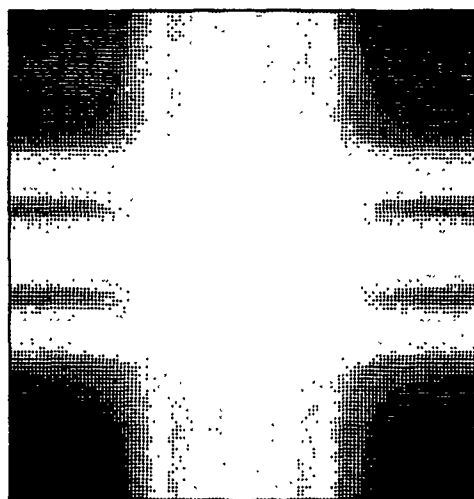
(c)

Figure 2.27. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

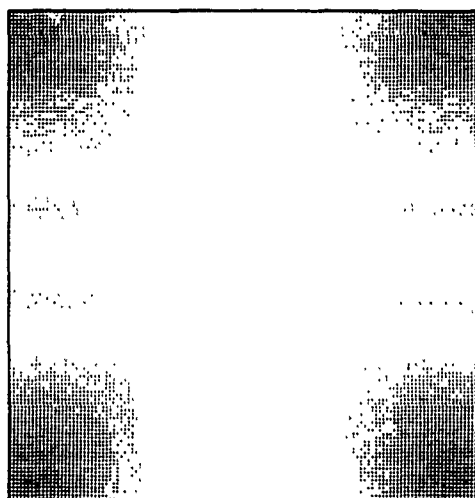
Support size is $1\lambda/D \times 1\lambda/D$ and $d/D = 0.1$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)



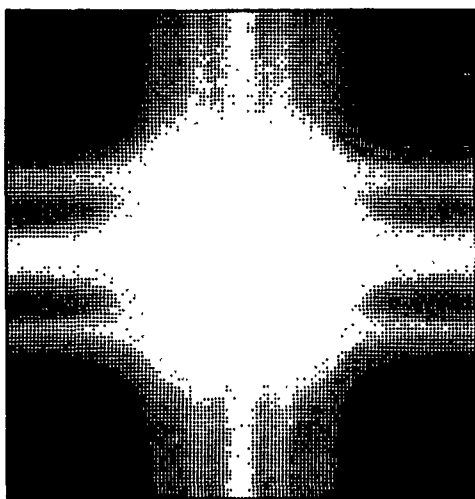
(b)



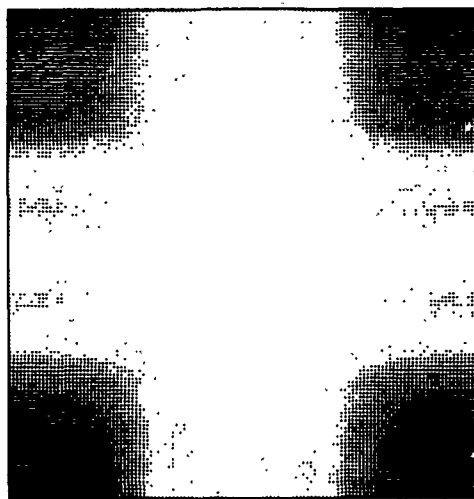
(c)

Figure 2.28. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

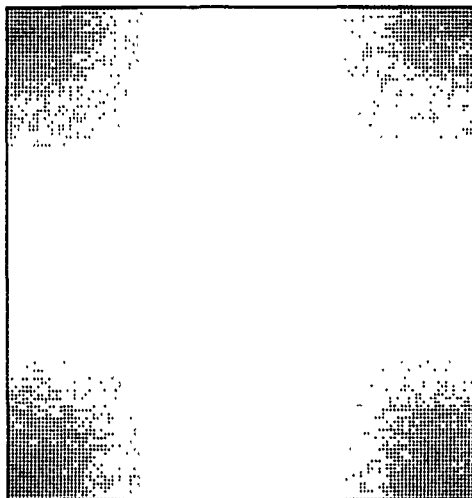
Support size is $7\lambda/D \times 7\lambda/D$ and $d/D = 0.2$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)



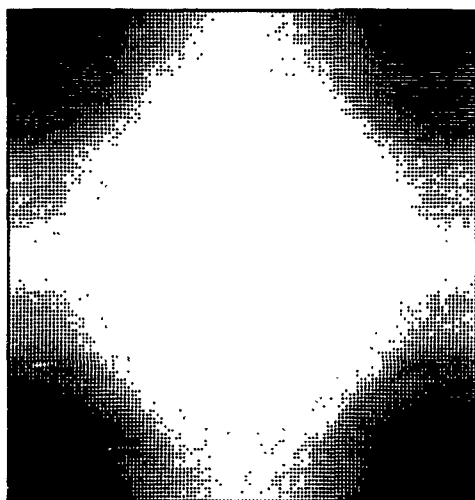
(b)



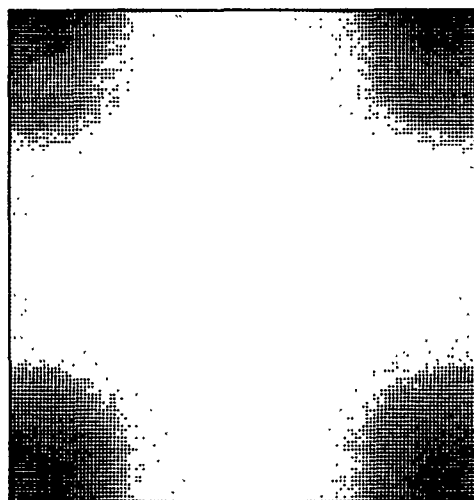
(c)

Figure 2.29. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

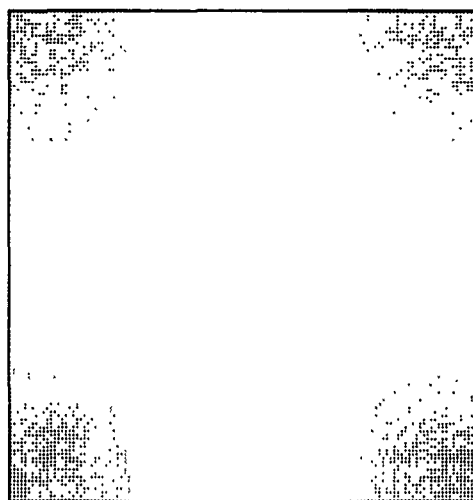
Support size is $4\lambda/D \times 4\lambda/D$ and $d/D = 0.2$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)



(b)



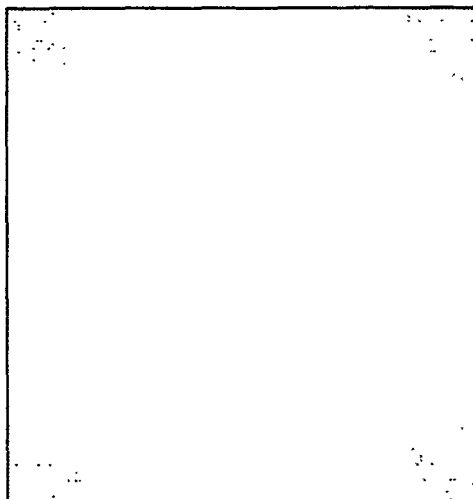
(c)

Figure 2.30. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

Support size is $2\lambda/D \times 2\lambda/D$ and $d/D = 0.2$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



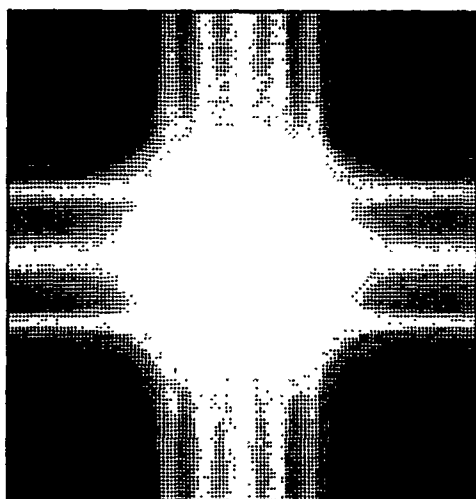
(a)



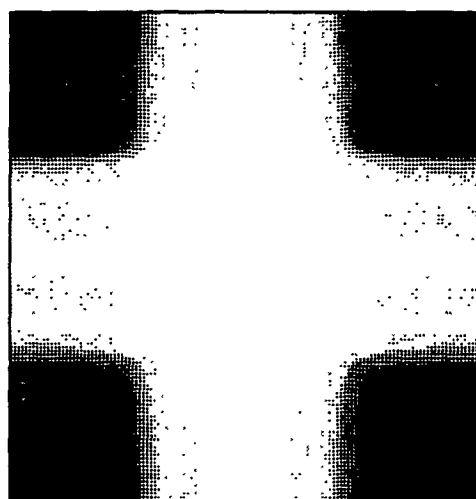
(b)

Figure 2.31. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

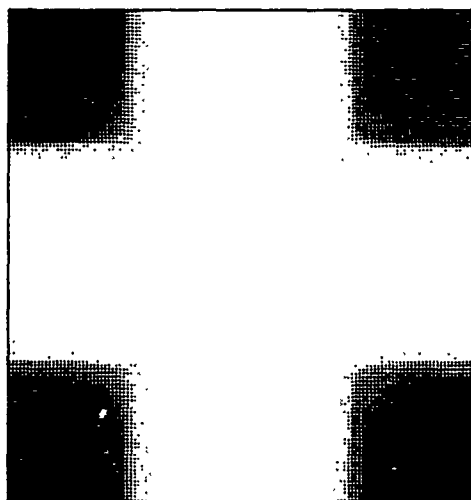
Support size is $1\lambda/D \times 1\lambda/D$ and $d/D = 0.2$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a) and (b) correspond to SNR's of 10^2 and 10^3 , respectively.



(a)



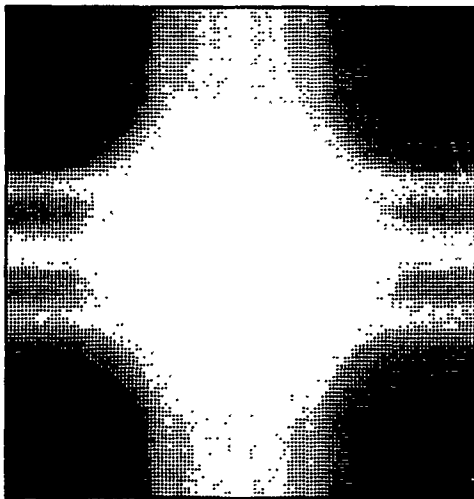
(b)



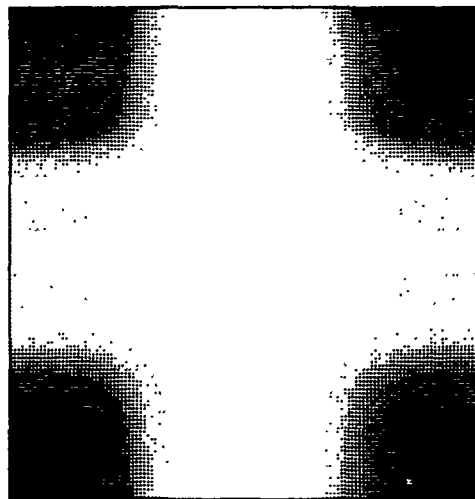
(c)

Figure 2.32. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

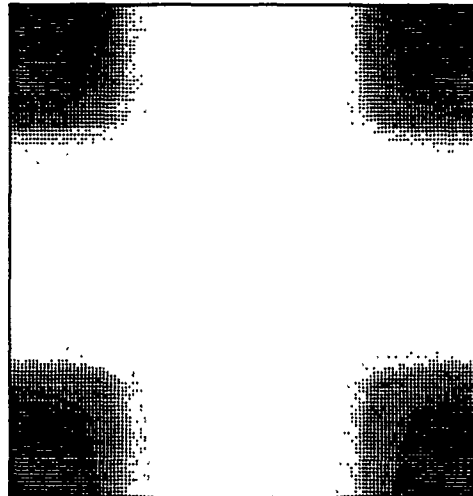
Support size is $7\lambda/D \times 7\lambda/D$ and $d/D = 0.3$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)



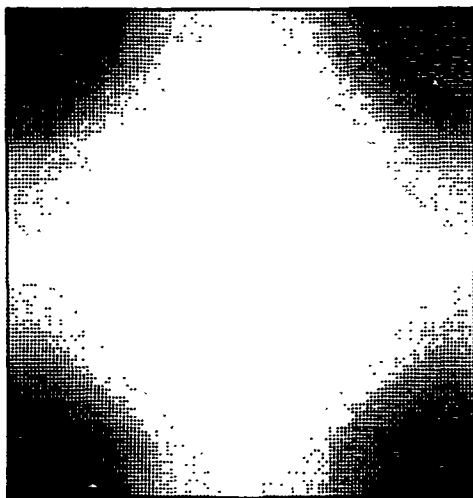
(b)



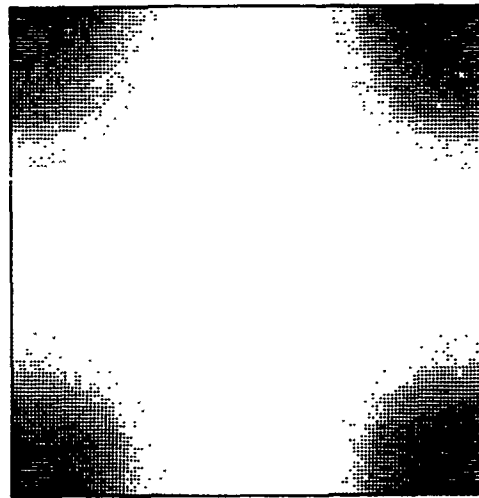
(c)

Figure 2.33. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

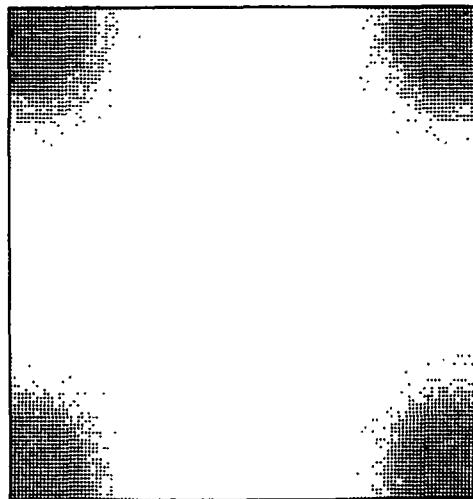
Support size is $4\lambda/D \times 4\lambda/D$ and $d/D = 0.3$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)



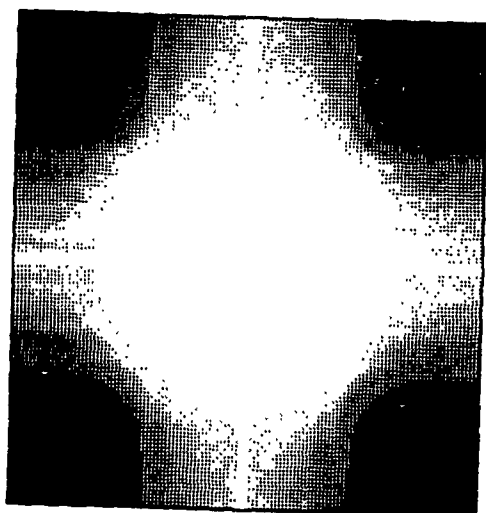
(b)



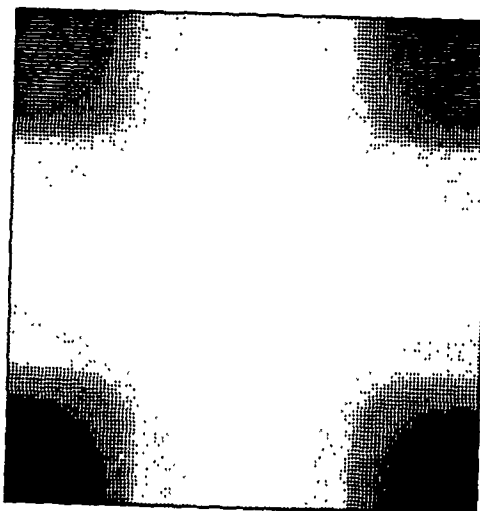
(c)

Figure 2.34. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

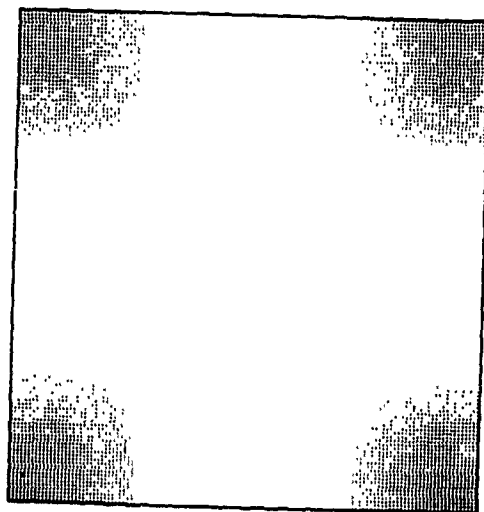
Support size is $2\lambda/D \times 2\lambda/D$ and $d/D = 0.3$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)

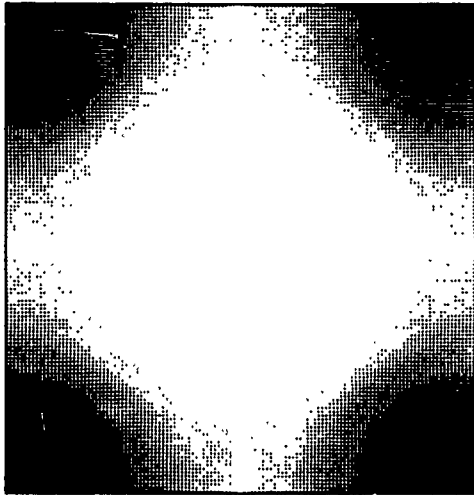


(b)

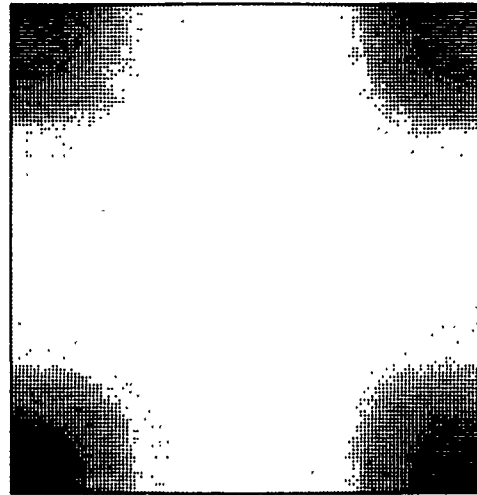


(c)

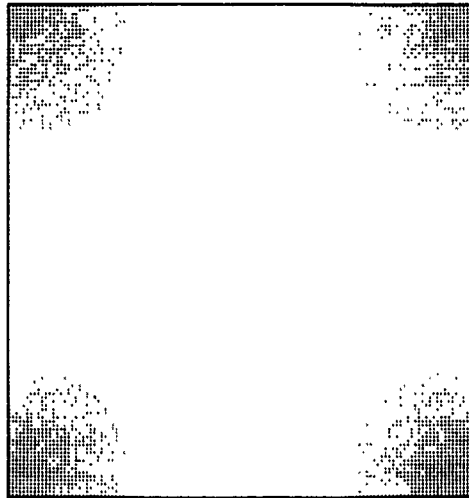
Figure 2.35. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $7\lambda/D \times 7\lambda/D$ and $d/D = 1.0$ (full aperture). Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)



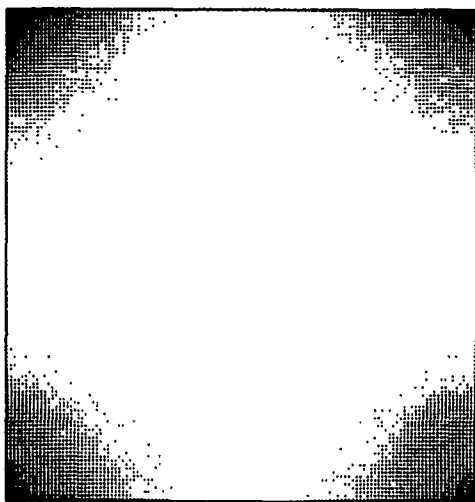
(b)



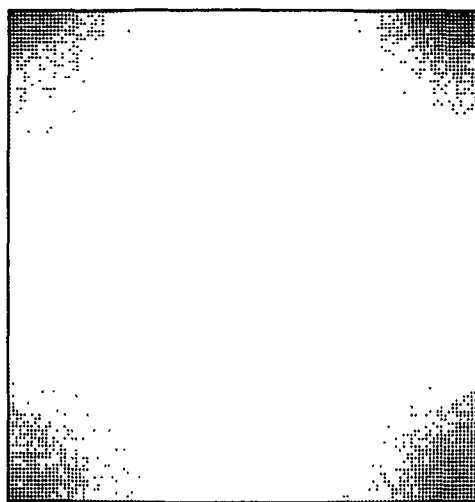
(c)

Figure 2.36. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

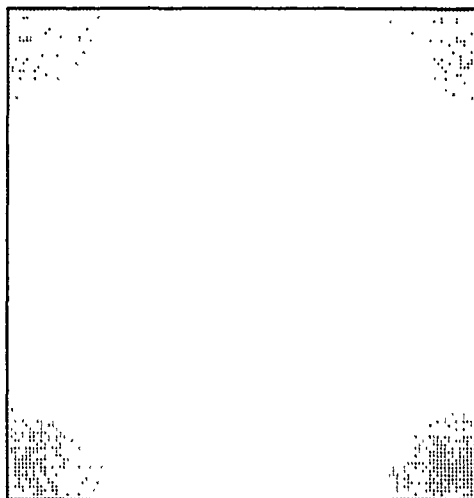
Support size is $4\lambda/D \times 4\lambda/D$ and $d/D = 1.0$ (full aperture). Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.



(a)



(b)



(c)

Figure 2.37. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy.

Support size is $2\lambda/D \times 2\lambda/D$ and $d/D = 1.0$ (fullaperture). Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2} . Fig.'s (a), (b), and (c) correspond to SNR's of 10^2 , 10^3 , and 10^4 , respectively.

positivity as constraints on the allowable objects. In this section we will use a least-squares criterion for deciding which object "best" explains the image, i.e., we pick the object vector estimate $\hat{\mathbf{x}}$ which minimizes

$$\epsilon = \|\mathbf{y} - G\hat{\mathbf{x}}\|^2. \quad (2.55)$$

Here we use the same one-dimensional optical model as that described in Section 2.2, with $\mathbf{y}$ the vector of image pixels, $\mathbf{x}$ the vector of object pixels, and $G = BW$ a matrix containing the system point-spread function and the object support information. We will distinguish between two object estimators: $\hat{\mathbf{x}}_s$, which minimizes Eq. (2.55) using only finite support as a constraint on the object, and $\hat{\mathbf{x}}_p$, which minimizes Eq. (2.55) when the object is constrained by both finite support and positivity. We can obtain $\hat{\mathbf{x}}_s$ in closed form. Carrying out the operation indicated in Eq. (2.55) yields

$$\begin{aligned} \epsilon &= (\mathbf{y} - G\hat{\mathbf{x}})^T (\mathbf{y} - G\hat{\mathbf{x}}) \\ &= \mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T G\hat{\mathbf{x}} + \hat{\mathbf{x}}^T G^T G\hat{\mathbf{x}}. \end{aligned} \quad (2.56)$$

The gradient of ϵ with respect to the components of $\hat{\mathbf{x}}$ is

$$\nabla \epsilon = -2G^T \mathbf{y} + 2G^T G\hat{\mathbf{x}}, \quad (2.57)$$

where here the gradient is taken to be a column vector. Setting Eq. (2.57) to zero and solving for $\hat{\mathbf{x}}$ yields

$$\hat{\mathbf{x}}_s = (G^T G)^{-1} G^T \mathbf{y}. \quad (2.58)$$

Since G has full column rank [see Eq. (2.14)], $G^T G$ is nonsingular and its inverse is well defined. The error vector with this estimate is

$$\begin{aligned} \mathbf{e}_s &= \mathbf{x} - \hat{\mathbf{x}}_s \\ &= \mathbf{x} - (G^T G)^{-1} G^T \mathbf{y}. \end{aligned} \quad (2.59)$$

Using $\mathbf{y}$ from Eq. (2.5) in Eq. (2.59), with $G = BW$, yields

$$\begin{aligned} \mathbf{e}_s &= \mathbf{x} - (G^T G)^{-1} G^T (G\mathbf{x} + \mathbf{n}) \\ &= (G^T G)^{-1} G^T \mathbf{n}. \end{aligned} \quad (2.60)$$

The covariance matrix of the error vector is

$$\begin{aligned} \langle \mathbf{e}_s \mathbf{e}_s^T \rangle &= (G^T G)^{-1} G^T \langle \mathbf{n} \mathbf{n}^T \rangle G (G^T G)^{-1} \\ &= (G^T G)^{-1} G^T R_{nn} G (G^T G)^{-1}. \end{aligned} \quad (2.61)$$

If we again assume that the noise vector is zero-mean and white, i.e.

$$R_{nn} = \sigma_n^2 I, \quad (2.62)$$

then

$$\langle \mathbf{e}_s \mathbf{e}_s^T \rangle = \sigma_n^2 (G^T G)^{-1}. \quad (2.63)$$

Unfortunately, one cannot obtain $\hat{\mathbf{x}}_p$ in closed form; one must use some numerical iteration method for each case of an object vector and a noise vector. Therefore, to evaluate the error vector covariance matrix when using positivity as a constraint requires a simulation. To generate object vectors, we generated M independent Rayleigh random variables, one for each pixel. Thus, object intensity pixels are always positive. Noise vector were generated using independent, zero-mean, Gaussian random variable for each pixel. An image vector $\mathbf{y}$ was generated using Eq. (2.5) for each

sample object and noise vector generated. The object vector estimate, $\hat{\mathbf{x}}_p$, for each case, was found using a modified gradient-search method where the components in the object vector were not allowed to go negative. If we let $\mathbf{x}(n)$ be the object vector and $\hat{\mathbf{x}}_p(n)$ be the object vector estimate for the n^{th} trial, then the error covariance matrix was estimated using

$$\langle \widehat{\mathbf{e}_p}, \widehat{\mathbf{e}_p}^T \rangle = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}(n) - \hat{\mathbf{x}}_p(n)) (\mathbf{x}(n) - \hat{\mathbf{x}}_p(n))^T. \quad (2.64)$$

Using Eq. (2.63) and (2.64) in Eq. (2.11), we can compute the energy spectrum of the error vector with and without the positivity constraint, and compare results.

Unfortunately, at the time of this report, our results are rather meager due to difficulties with the modified gradient search method. At this time we have results only for an object support length of four pixels ($1\lambda/D$). The results are shown in Fig.'s 2.38-2.40. Again, we have normalized the error-vector energy spectra with the total energy in the object vector (sum of the mean-square intensity values of each pixel). Each figure contains a set of curves corresponding to one d/D ratio (0.1 for Fig. 2.38, 0.2 for Fig. 2.39, and 0.3 for Fig. 2.40). Within each figure are three sets of two curves each, each set corresponding to a different SNR (from bottom to top, 10^4 , 10^3 , and 10^2 , respectively). Here, SNR is defined as the ratio of the second moment of an object intensity pixel to the second moment of a noise pixel. The solid curve in each curve-pair corresponds to the finite-support constraint only, and the dashed curve is the result when the positivity constraint is added. With these very limited results, it would appear that the use of a positivity constraint in addition to the finite-support constraint may result in a significant improvement in performance. However, we note that the amount of improvement seems to decrease as SNR increases.

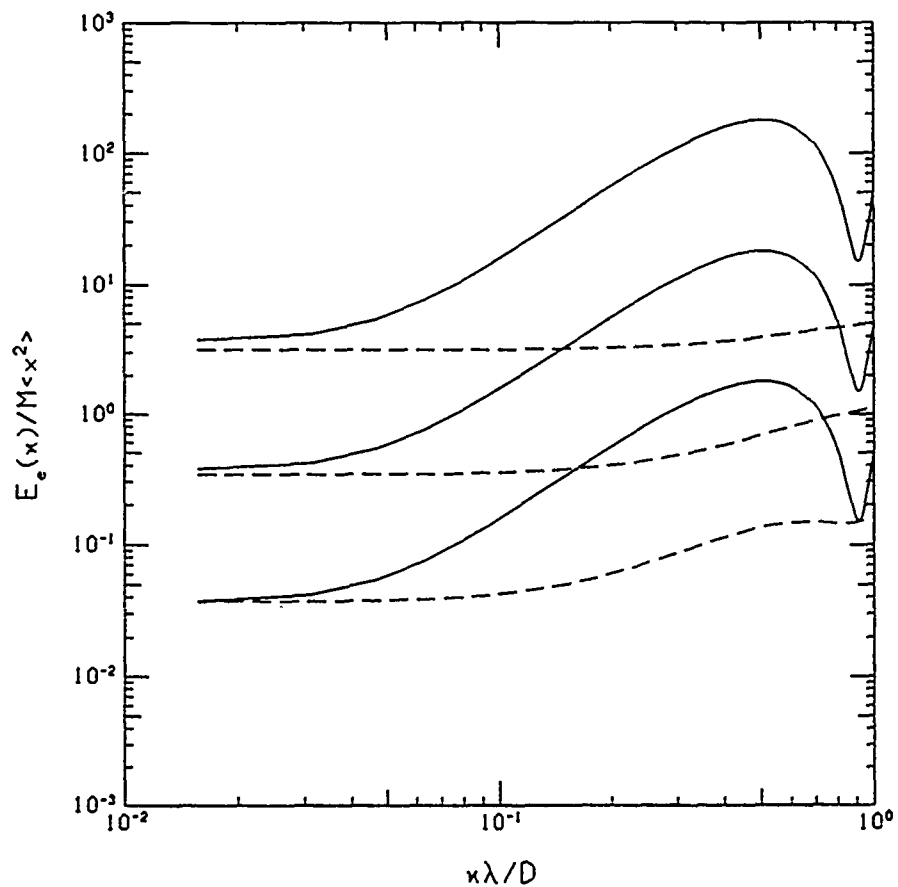


Figure 2.38. Normalized Error Vector Energy Spectrum for $d/D = 0.1$ and a Support Length of $1\lambda/D$. The three sets of curves correspond to SNR's of (from top to bottom) 10^2 , 10^3 , and 10^4 . The solid and dashed curves are without and with positivity constraint, respectively.

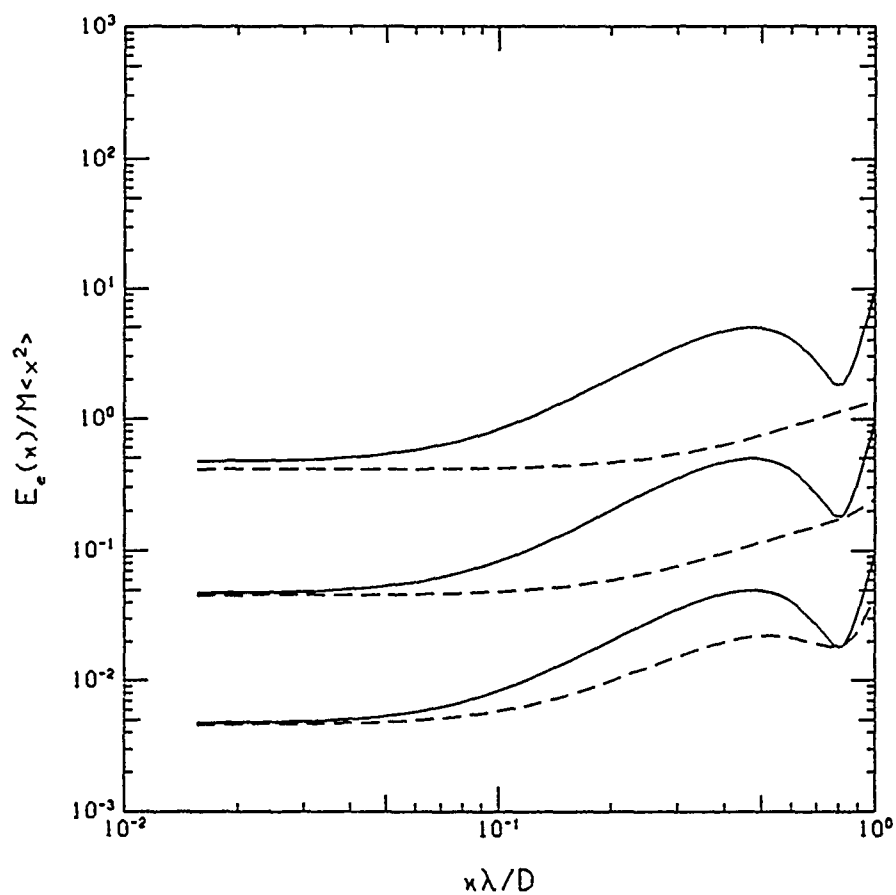


Figure 2.39. Normalized Error Vector Energy Spectrum for $d/D = 0.2$ and a Support Length of $1\lambda/D$. The three sets of curves correspond to SNR's of (from top to bottom) 10^2 , 10^3 , and 10^4 . The solid and dashed curves are without and with positivity constraint, respectively.

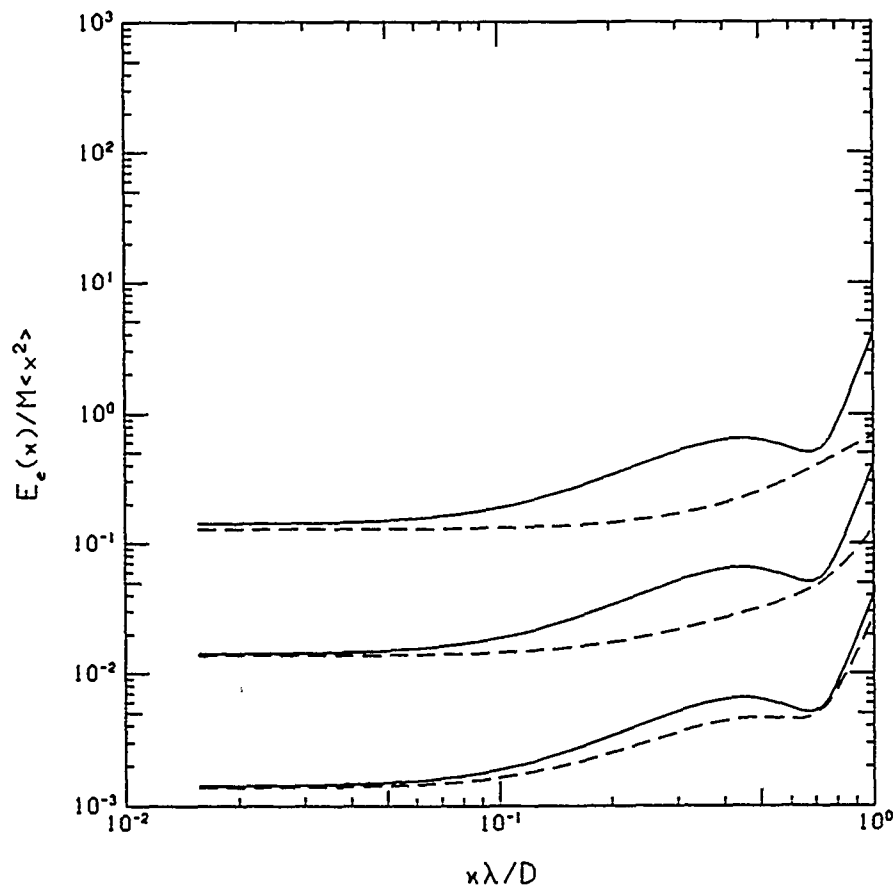


Figure 2.40. Normalized Error Vector Energy Spectrum for $d/D = 0.3$ and a Support Length of $1\lambda/D$. The three sets of curves correspond to SNR's of (from top to bottom) 10^2 , 10^3 , and 10^4 . The solid and dashed curves are without and with positivity constraint, respectively.

Chapter 3

Super-Resolution with Sparse Arrays Revisited—Conclusion of Two Aperture Case

(originally issued as TR-1052)

3.1. Introduction

In Chapter 2 we reported on some results of an investigation into the feasibility of obtaining useful reconstruction of objects viewed through a sparse array of apertures. A requirement placed on the reconstruction was that the highest useful spatial frequency in the reconstruction be commensurate with the outer diameter of the sparse array. All of the one-dimensional results contained in Chapter 2 were restricted to the case of two fixed subapertures, a case that we now know to be of little interest. In this chapter we present some miscellaneous two-aperture results that were obtained subsequent to the work of Chapter 2. We report these results, without comment, for purposes of historical completeness and the remote chance that they may be of use.

All of the results in this chapter are for the one-dimensional model described in Section 2.2.1 of Chapter 2. Section 3.2 of this chapter describes some changes made to the model subsequent to publication of Chapter 2. Sections 3 and 4 of this chapter give results for the minimum variance and least squares methods, respectively. Appendix B contains listings of fortran programs that, in addition to those given in Chapter 2, were used to generate the data of this chapter.

3.2. Some Changes to the One-dimensional Model

All of the one-dimensional results of Chapter 2 were generated using the point spread function (PSF) given by Eq.'s (2.7) and (2.8) of Chapter 2. For purposes of this chapter, we replace that earlier PSF with two point spread functions, called $h_1(n)$ and $h_2(n)$.

$$h_1(n) = \Delta w_\ell(x) h_c(x)|_{x=n\Delta}, \quad (3.1)$$

where Δ is the sample interval,

$$h_c(x) = \frac{4d^2}{D\lambda} \cos^2 \left[\frac{\pi(D-d)}{\lambda} x \right] \left[\frac{\sin(\pi x d/\lambda)}{\pi x d/\lambda} \right]^2, \quad (3.2)$$

and $w_\ell(x)$ is the ℓ^{th} of five window functions given below.

Window One (Rectangular)

$$w_1(x) = \begin{cases} 1, & |x| \leq k\Delta \\ 0, & \text{else.} \end{cases} \quad (3.3)$$

Window Two (Bartlett)

$$w_2(x) = \begin{cases} 1 - |x|/k\Delta, & |x| \leq k\Delta \\ 0, & \text{else.} \end{cases} \quad (3.4)$$

Window Three (Hanning)

$$w_3(x) = \begin{cases} \frac{1}{2} + \frac{1}{2} \cos(\pi x/k\Delta), & |x| \leq k\Delta \\ 0, & \text{else.} \end{cases} \quad (3.5)$$

Window Four (Hamming)

$$w_4(x) = \begin{cases} 0.54 + 0.46 \cos(\pi x/k\Delta), & |x| \leq k\Delta \\ 0, & \text{else.} \end{cases} \quad (3.6)$$

Window Five (Blackman)

$$w_5(x) = \begin{cases} 0.42 + 0.5 \cos(\pi x/k\Delta) + 0.08 \cos(2\pi x/k\Delta), & |x| \leq k\Delta \\ 0, & \text{else.} \end{cases} \quad (3.7)$$

The PSF $h_2(n)$ is similar to $h_1(n)$, but without a window and scaled differently.

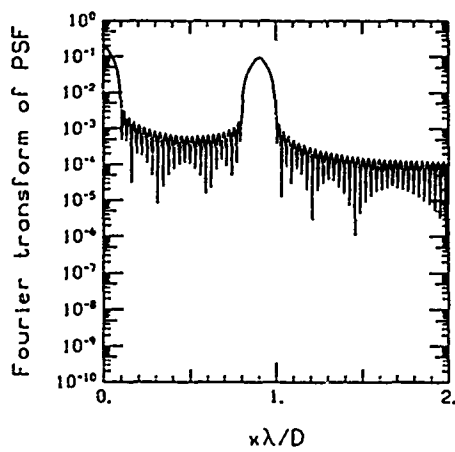
$$h_2(n) = \frac{D}{2d} \Delta h_c(n\Delta), \quad |n| < \infty. \quad (3.8)$$

We point out without proof that the Fourier transform of $h_1(n)$, evaluated at the origin, is equal to $2d/D$, whereas $h_2(n)$ is normalized so that its Fourier transform is unity at the origin for all values of d/D .

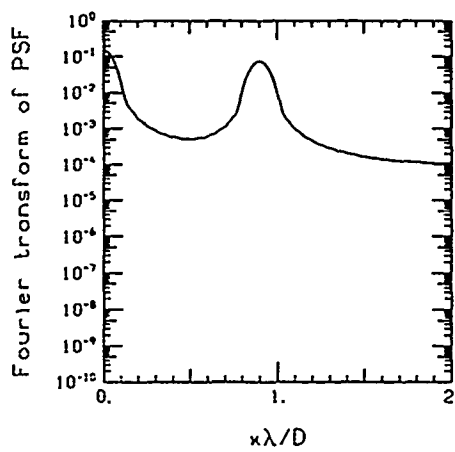
All results in this report are for a sample interval Δ of

$$\Delta = \frac{1}{4} \frac{\lambda}{D} \quad (3.9)$$

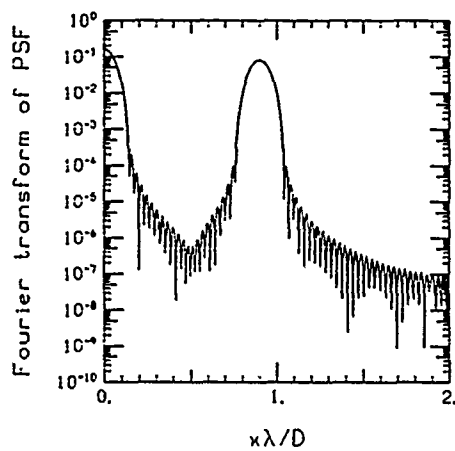
The Fourier transform of the PSF given by $h_1(n)$ for each of the windows and various values of k and d/D is shown in Fig.'s 3.1 to 3.7.



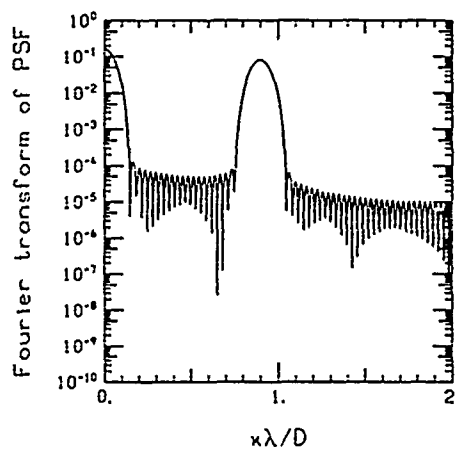
(a)



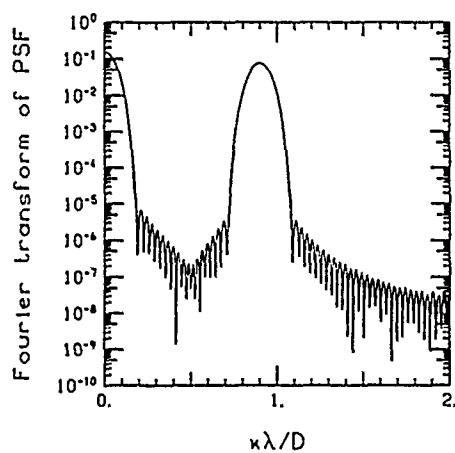
(b)



(c)

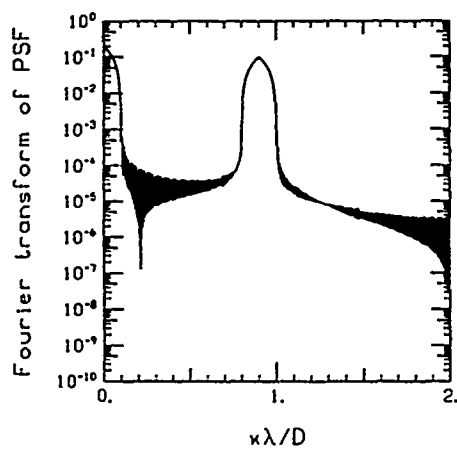


(d)

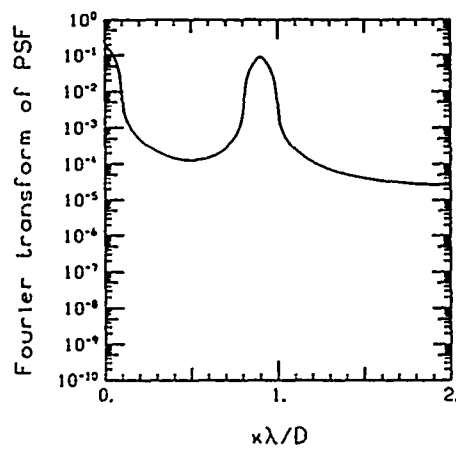


(e)

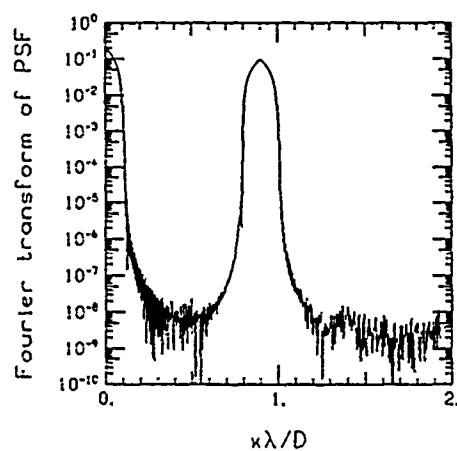
Figure 3.1. Fourier transform of the PSF of Eq. (3.1), $k = 63, d/D = 0.1$. Figures (a)-(e) correspond to window number 1-5, respectively.



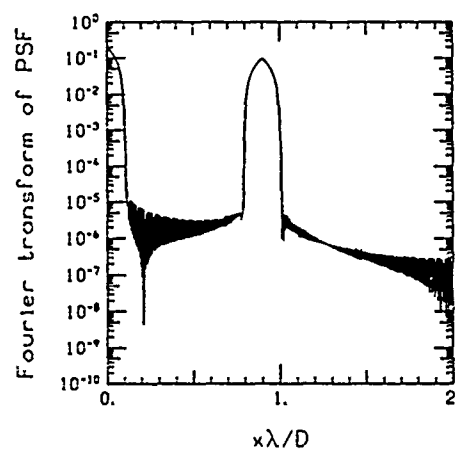
(a)



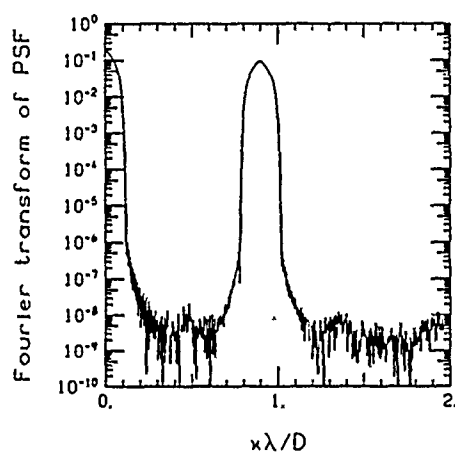
(b)



(c)

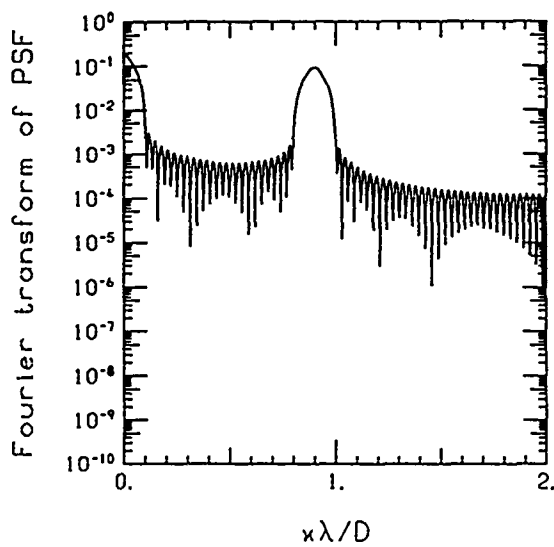


(d)

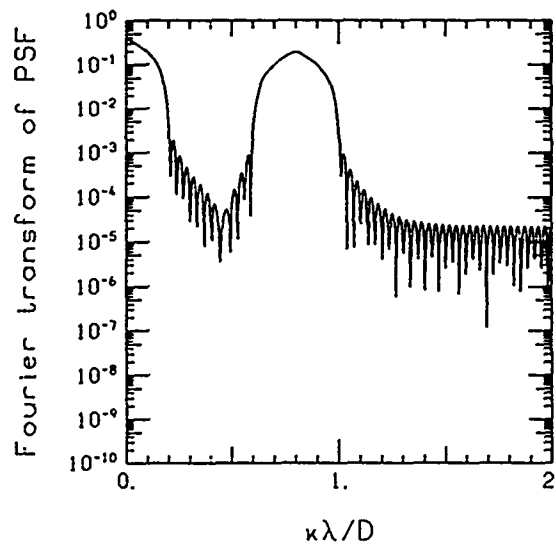


(e)

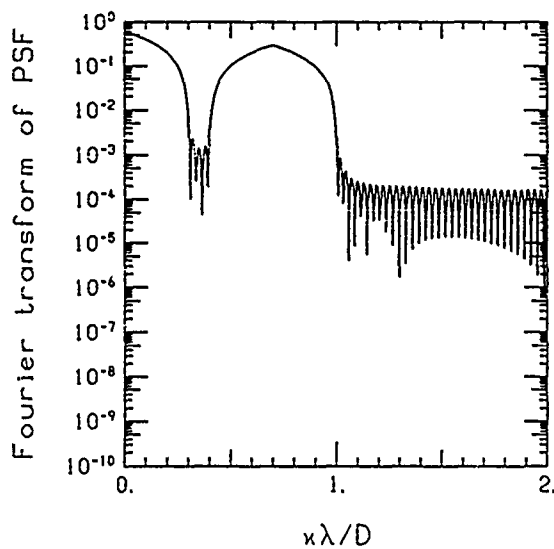
Figure 3.2. Fourier Transform the PSF of Eq. (3.1), $k = 255$, $d/D = 0.1$. Figures (a)-(e) correspond to window number 1-5, respectively.



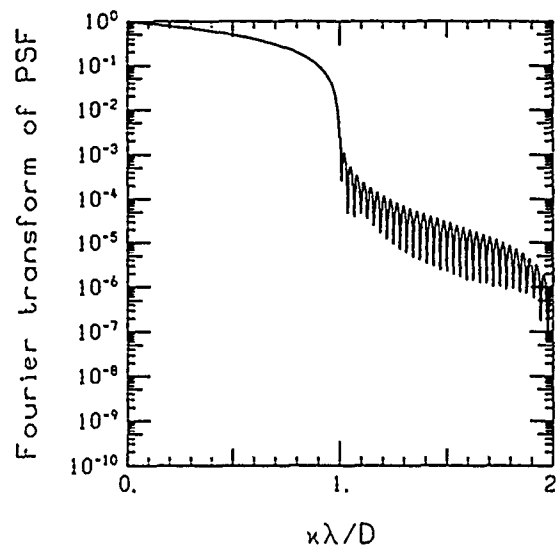
(a)



(b)



(c)



(d)

Figure 3.3. Fourier transforms of the PSF of Eq. (3.1), $k = 63$, window number 1. Figures (a)-(d) correspond to $d/D = 0.1, 0.2, 0.3$, and 0.5 , respectively.

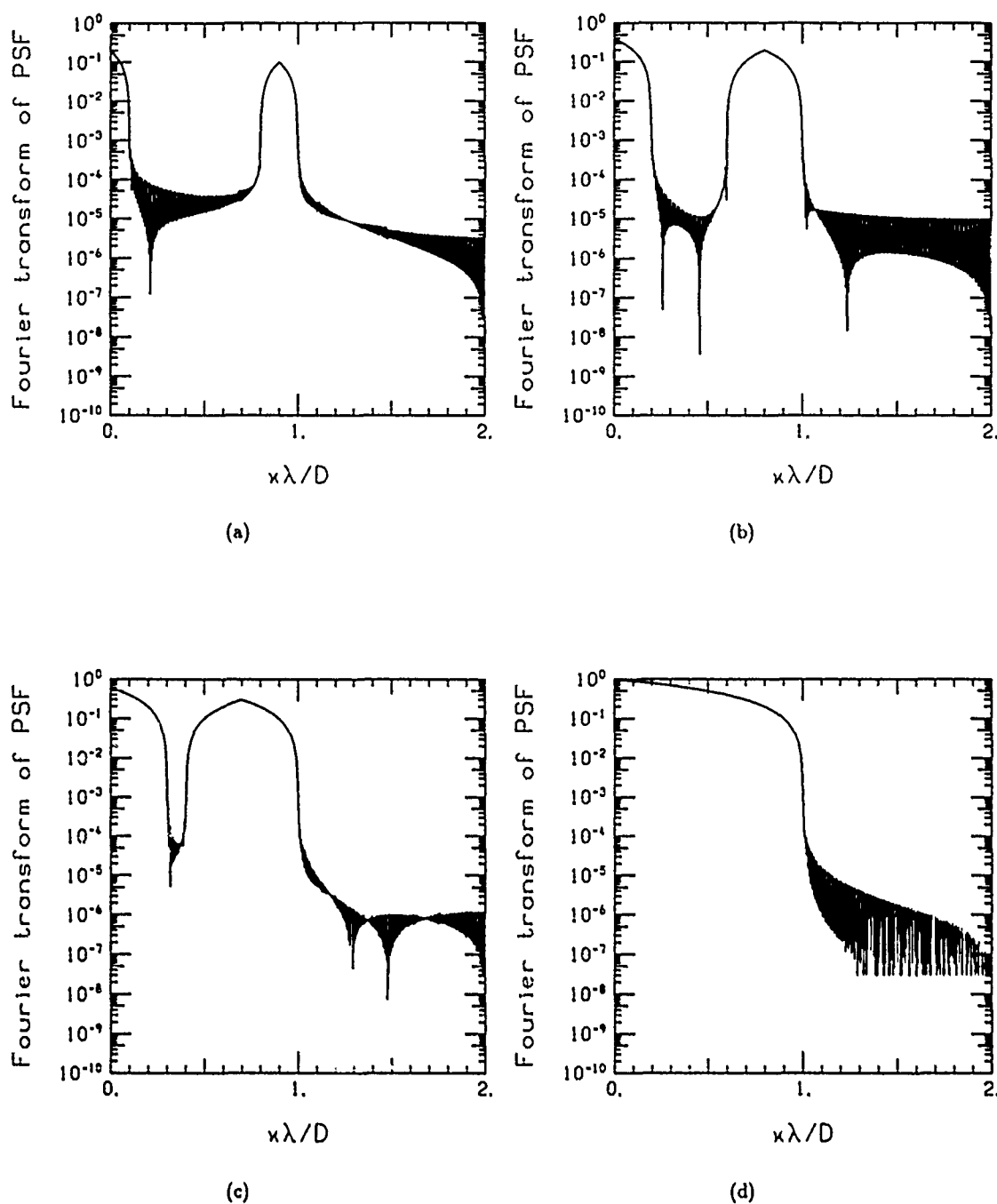
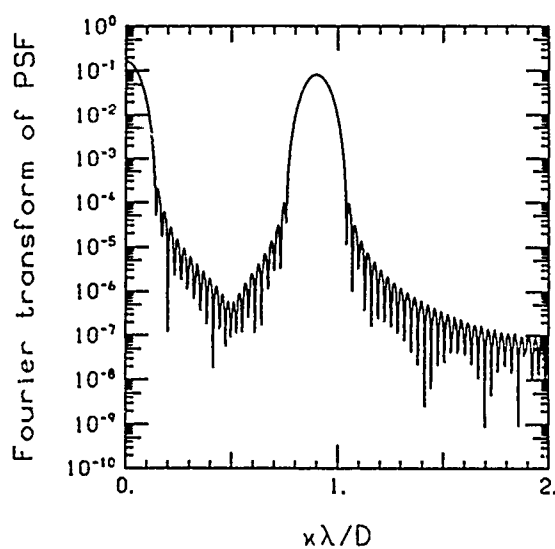
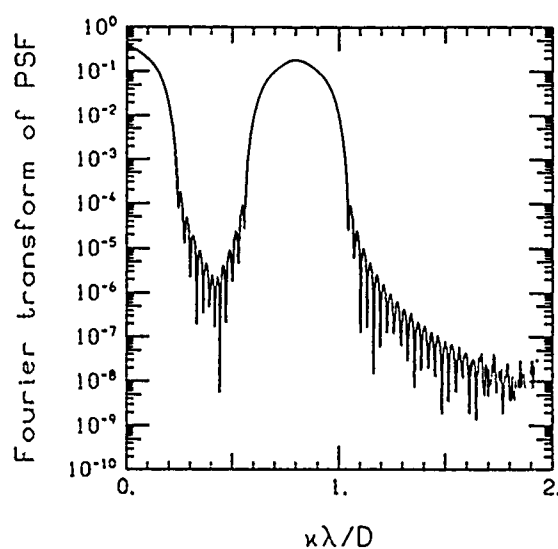


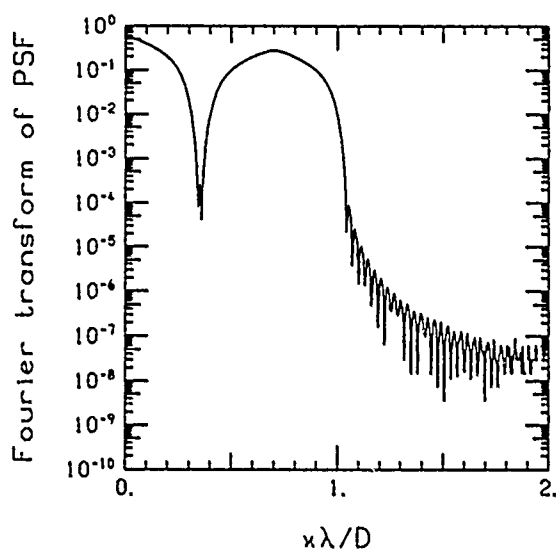
Figure 3.4. Fourier transforms of the PSF of Eq. (3.1), $k = 255$, window number 1. Figures (a)-(d) correspond to $d/D = 0.1, 0.2, 0.3$, and 0.5 , respectively.



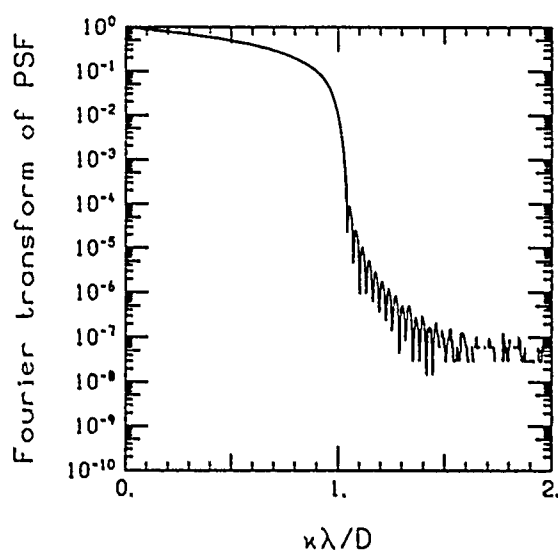
(a)



(b)

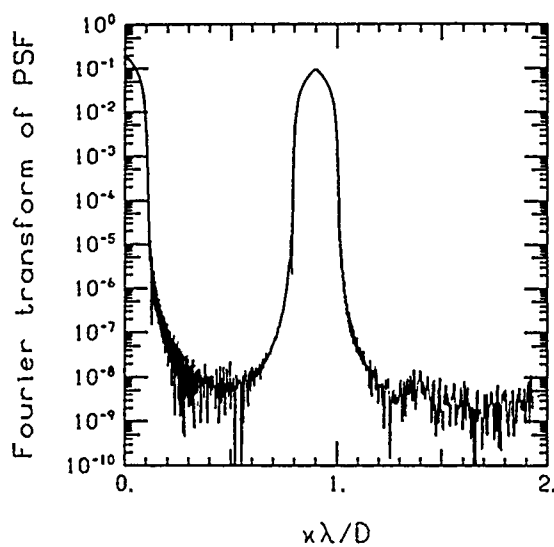


(c)

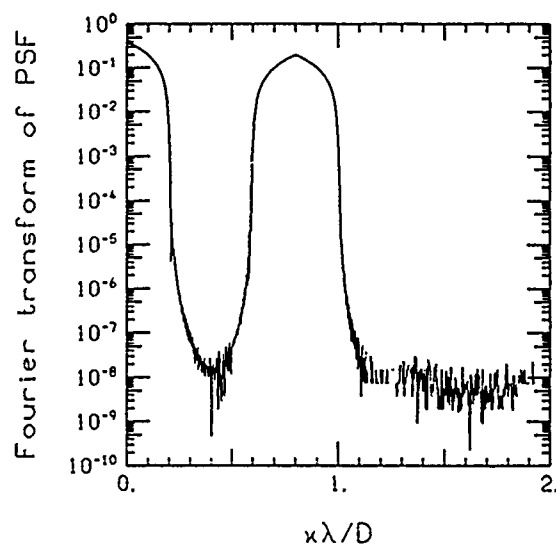


(d)

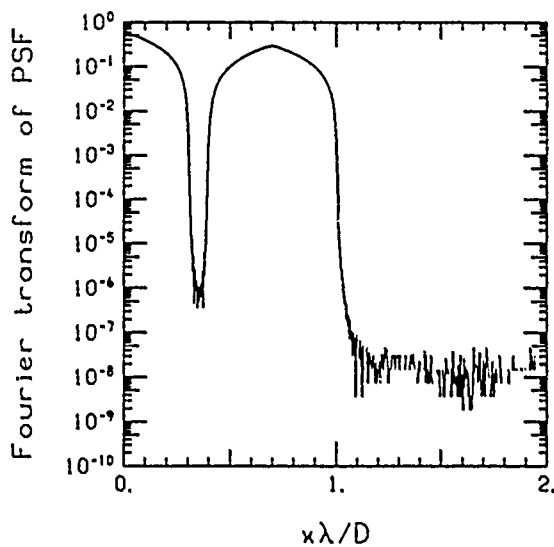
Figure 3.5. Fourier transforms of the PSF of Eq. (3.1), $k = 63$, window number 3. Figures (a)-(d) correspond to $d/D = 0.1, 0.2, 0.3$, and 0.5 , respectively.



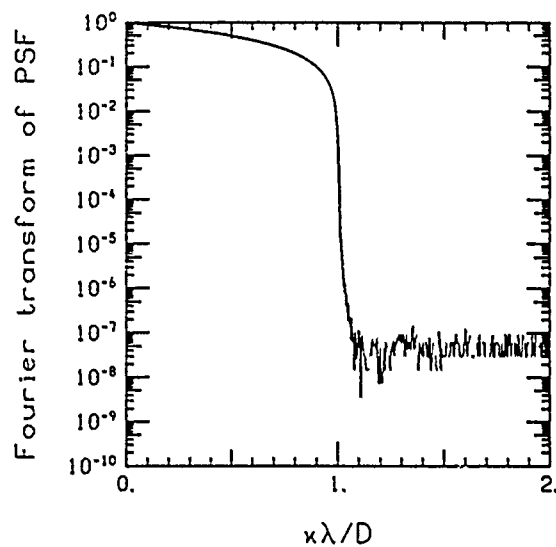
(a)



(b)



(c)



(d)

Figure 3.6. Fourier transforms of the PSF of Eq. (3.1), $k = 255$, window number 3. Figures (a)-(d) correspond to $d/D = 0.1, 0.2, 0.3$, and 0.5 , respectively.

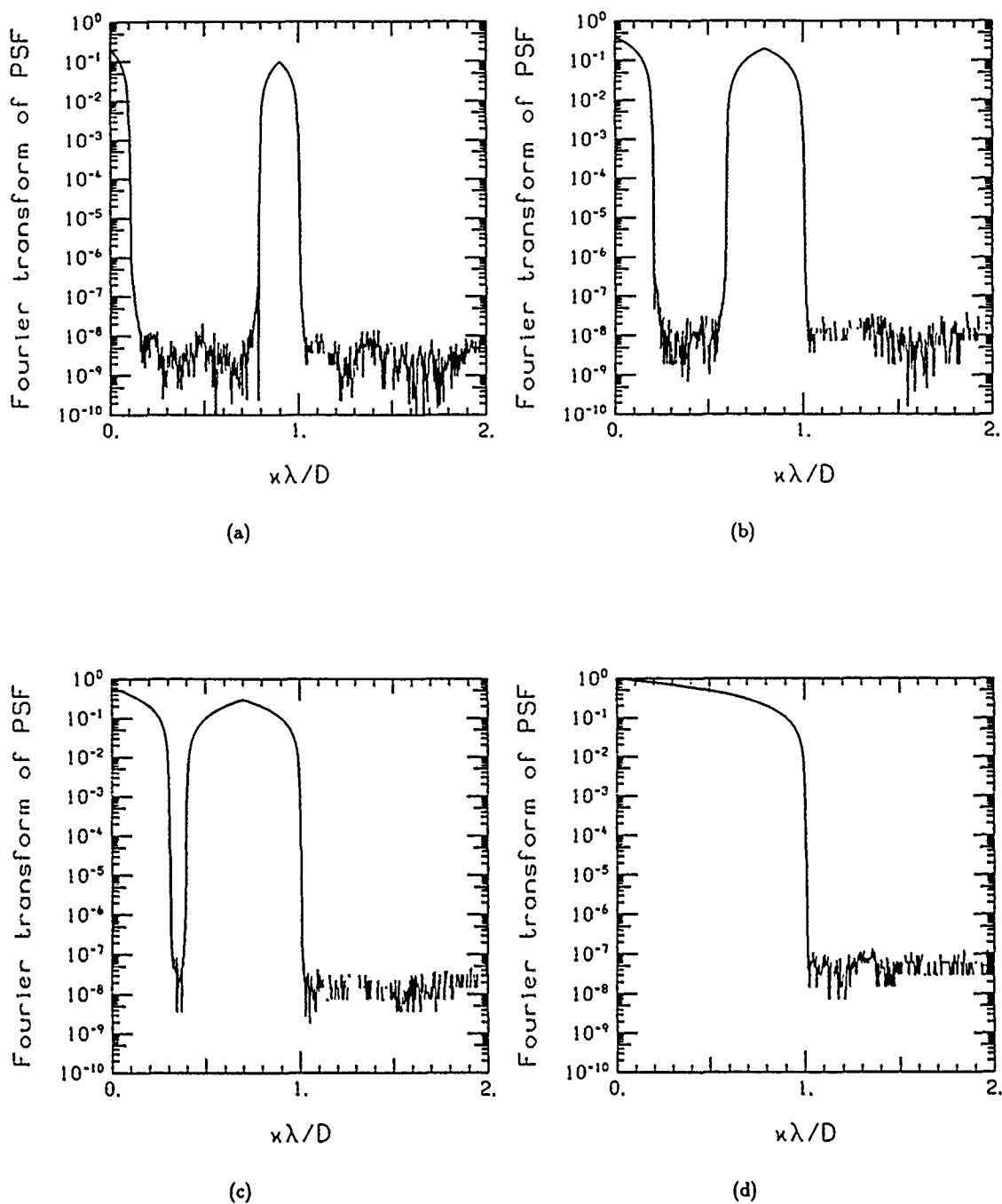


Figure 3.7. Fourier transforms of the PSF of Eq. (3.1), $k = 511$, window number 5. Figures (a)-(d) correspond to $d/D = 0.1, 0.2, 0.3,$ and 0.5 , respectively.

3.3. Results Using the Minimum-Variance Method

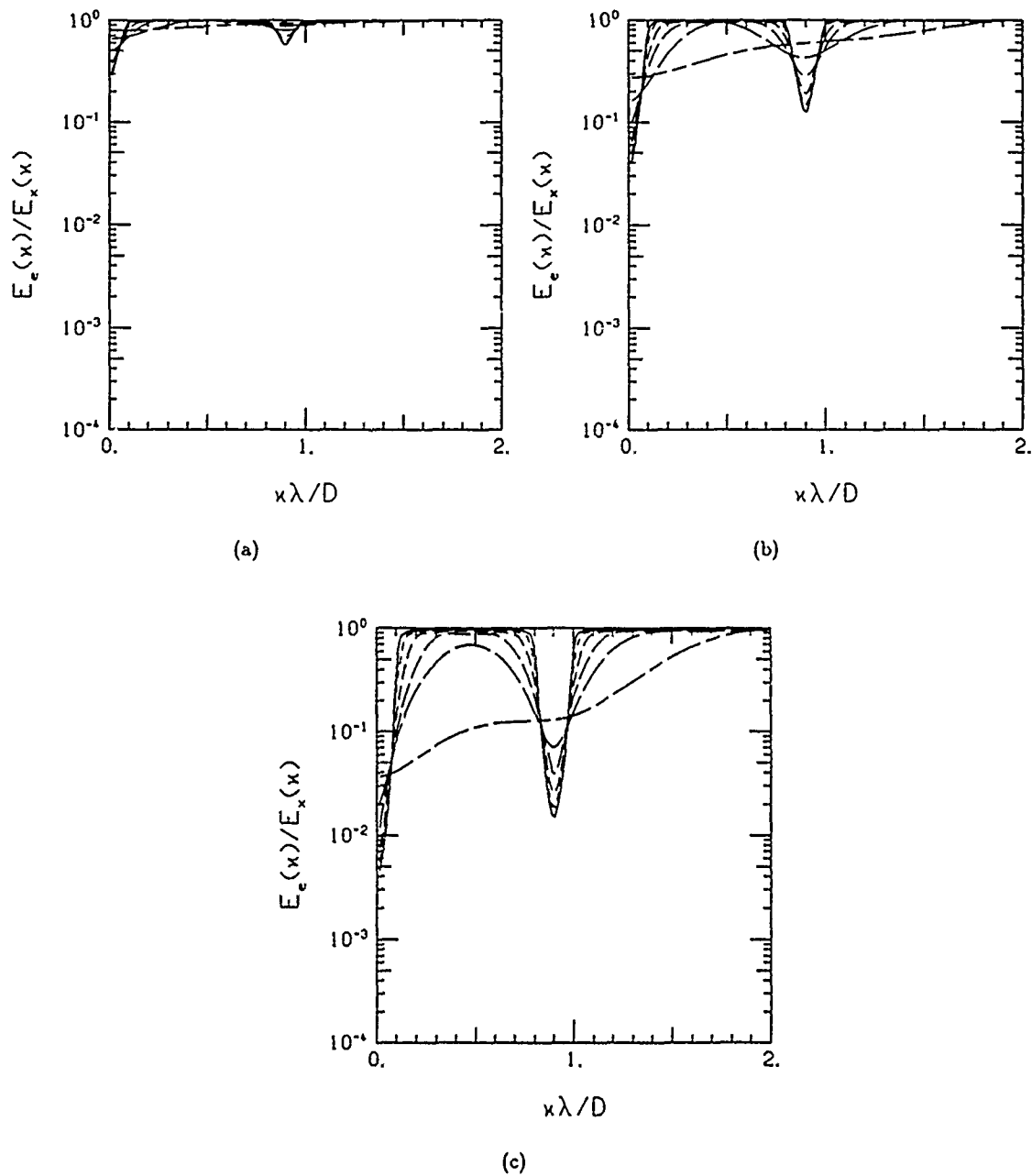
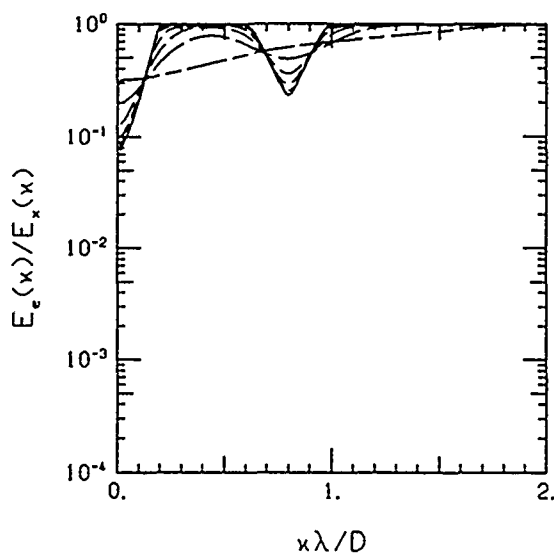
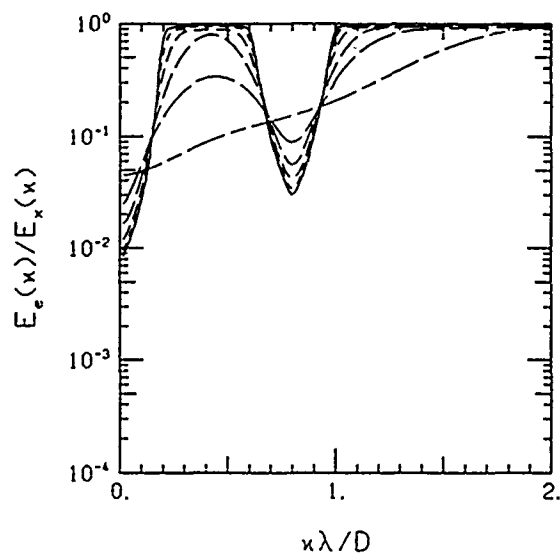


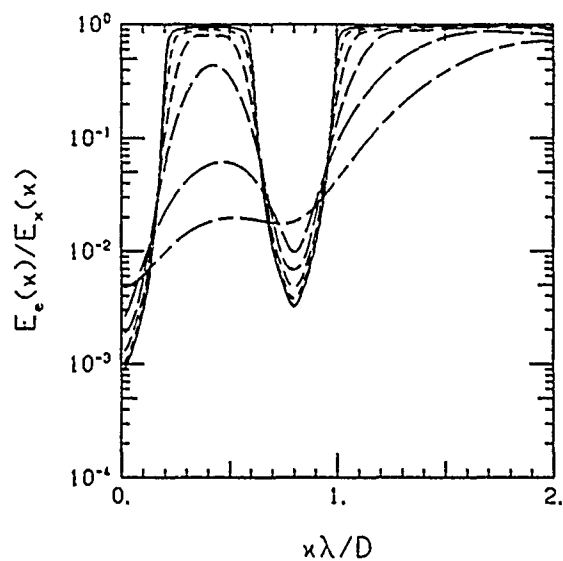
Figure 3.8. Energy spectra ratios using the PSF of Eq. (3.1) with window number three, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and $d/D = 0.1$. Figures (a)-(c) correspond to SNR_{REF} of 20 dB, 30 dB, and 40 dB, respectively. The six curves in each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$.



(a)

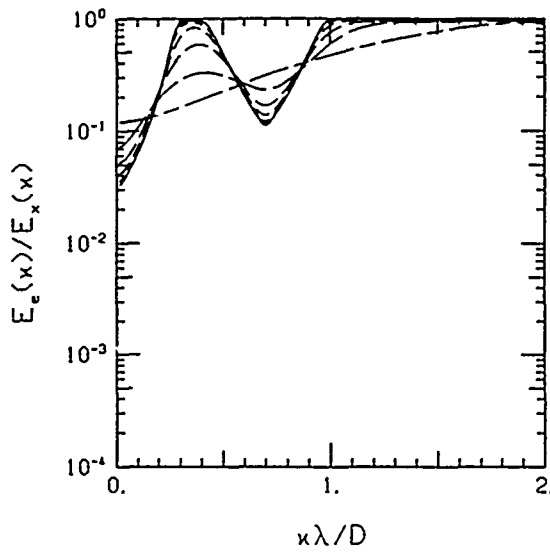


(b)

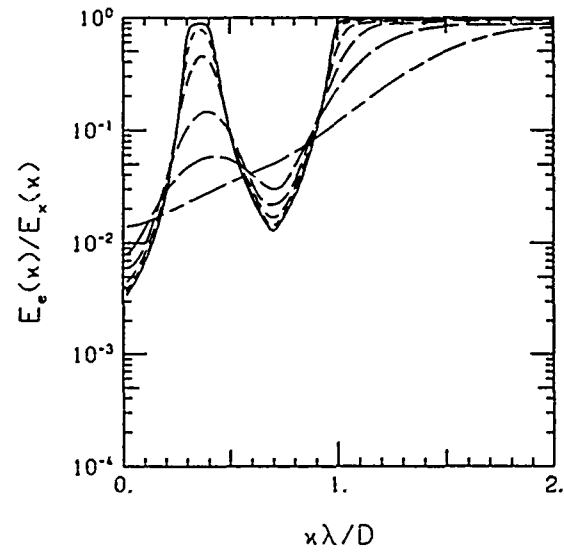


(c)

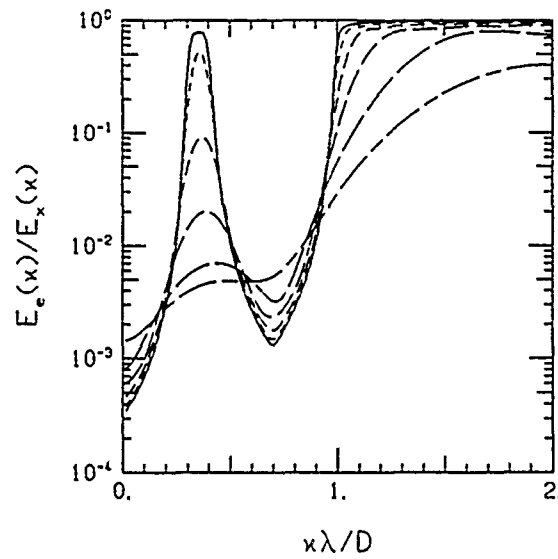
Figure 3.9. Energy spectra ratios using the PSF of Eq. (3.1) with window number three, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and $d/D = 0.2$. Figures (a)-(c) correspond to SNR_{REF} of 20 dB, 30 dB, and 40 dB, respectively. The six curves in each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$.



(a)

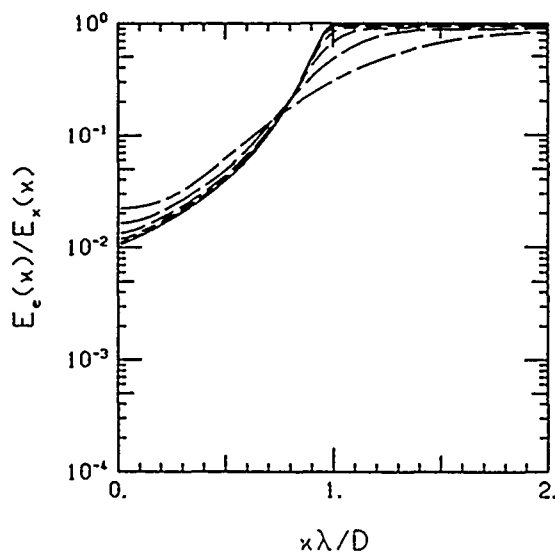


(b)

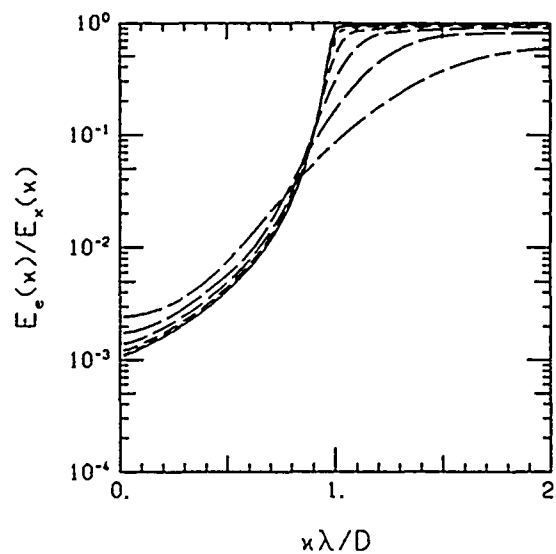


(c)

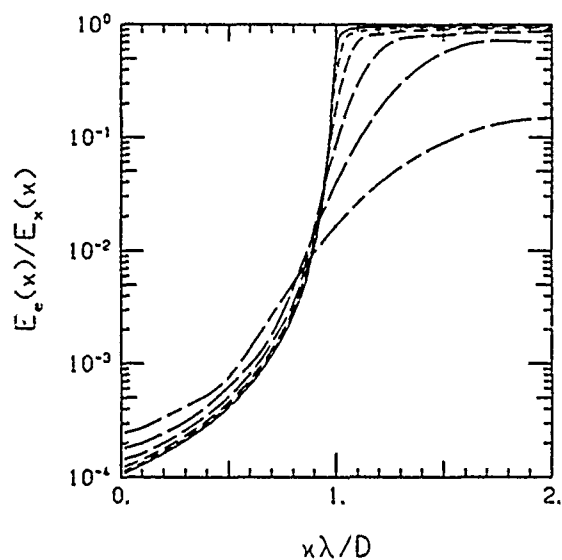
Figure 3.10. Energy spectra ratios using the PSF of Eq. (3.1) with window number three, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and $d/D = 0.3$. Figures (a)–(c) correspond to SNR_{REF} of 20 dB, 30 dB, and 40 dB, respectively. The six curves in each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$.



(a)

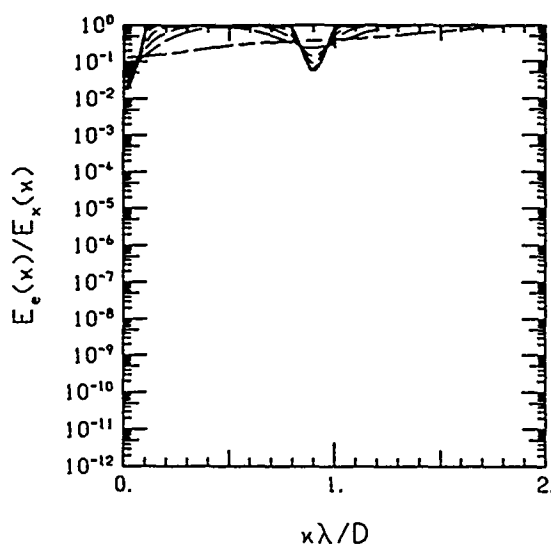


(b)

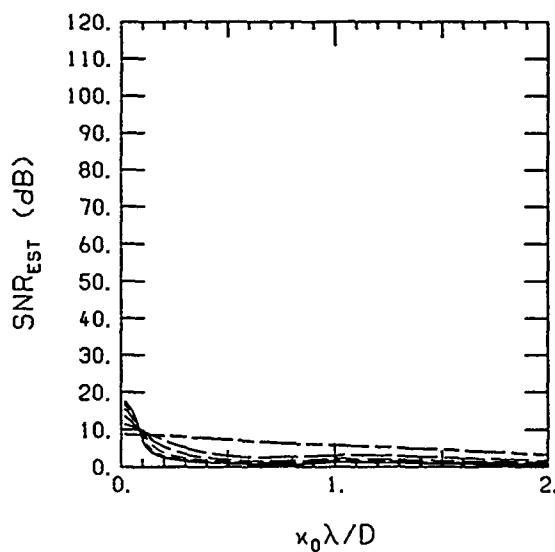


(c)

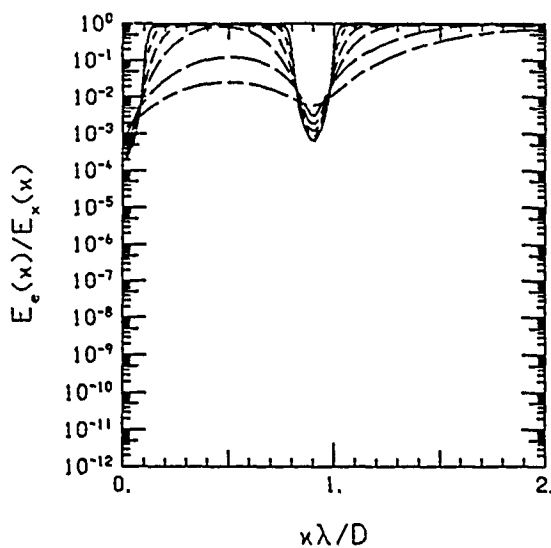
Figure 3.11. Energy spectra ratios using the PSF of Eq. (3.1), with window number three, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and $d/D = 0.5$. Figures (a)–(c) correspond to SNR_{REF} of 20 dB, 30 dB, and 40 dB, respectively. The six curves in each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$.



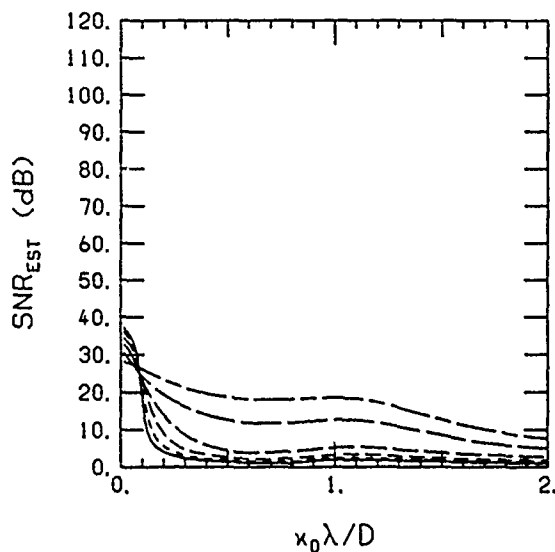
(a)



(b)

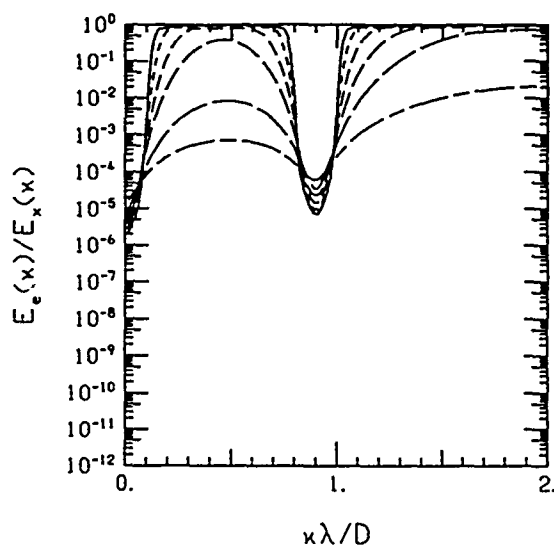


(c)

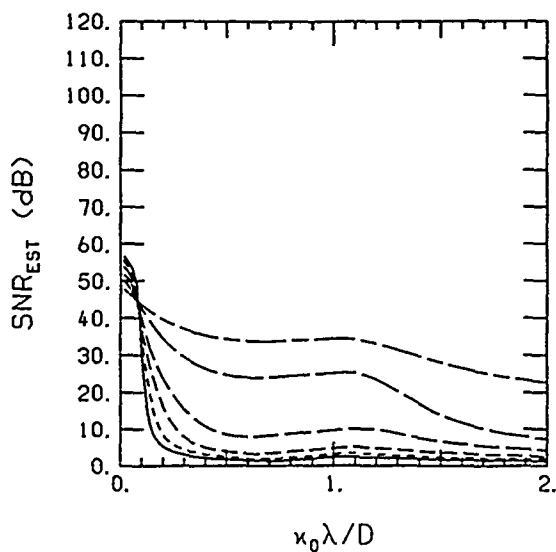


(d)

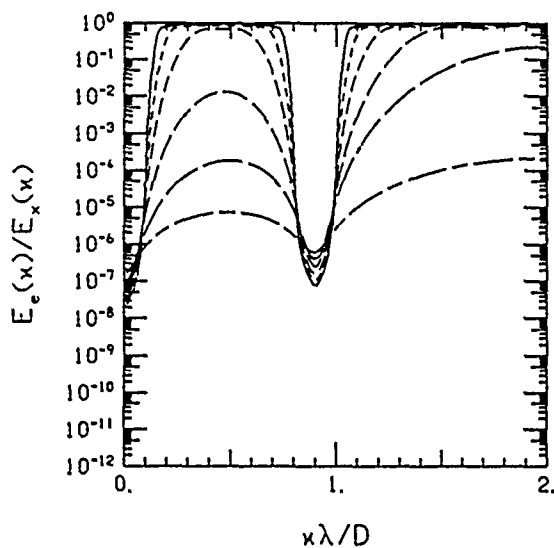
Figure 3.12. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.1$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 20$ dB and figures (c) and (d) correspond to $SNR_{REF} = 40$ dB.



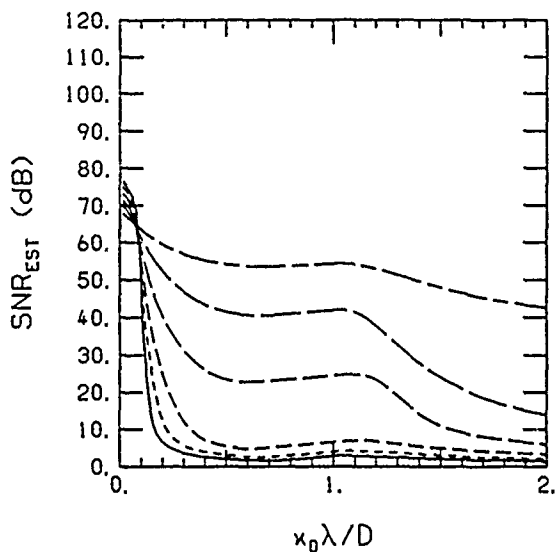
(a)



(b)

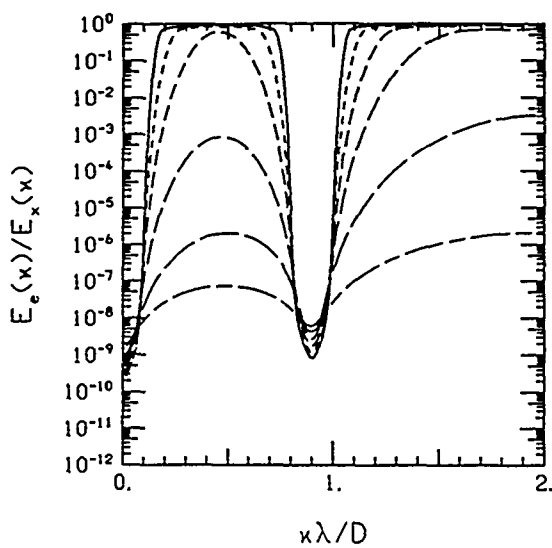


(c)

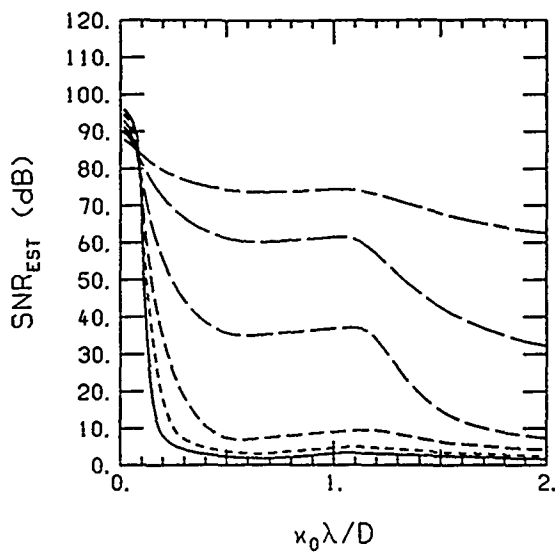


(d)

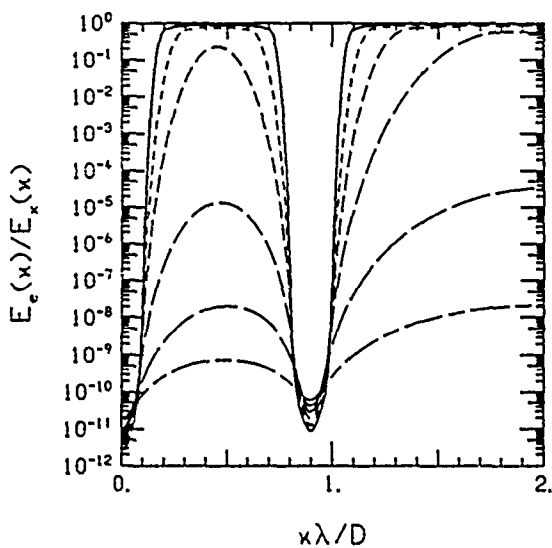
Figure 3.13. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.1$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}}=60$ dB and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}}=80$ dB.



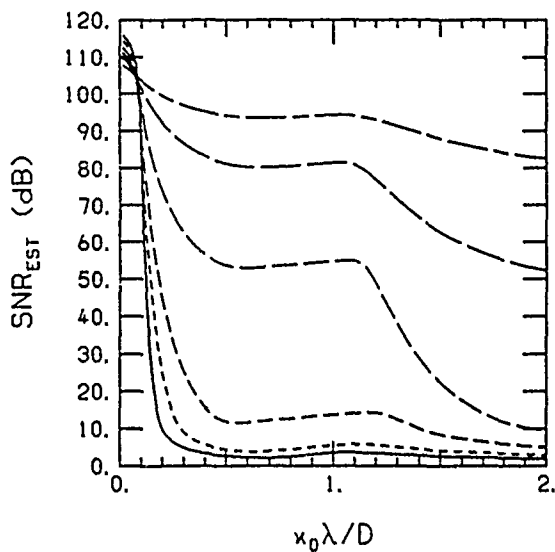
(a)



(b)

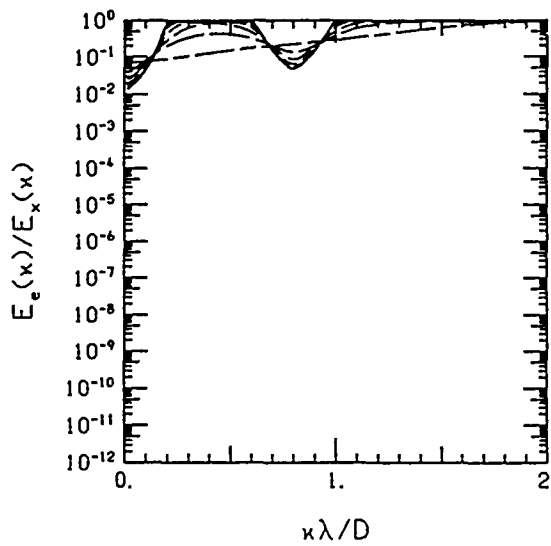


(c)

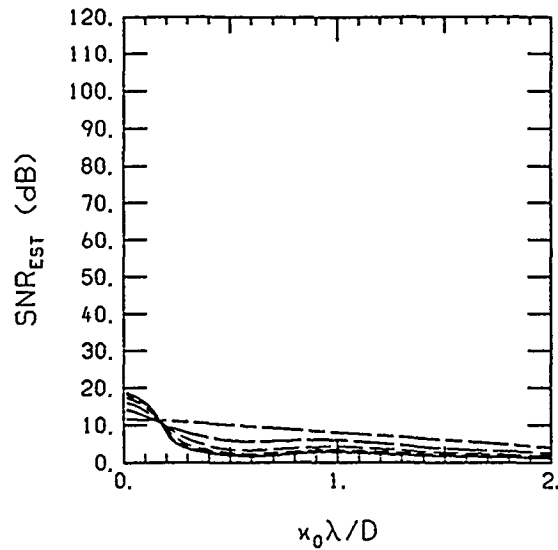


(d)

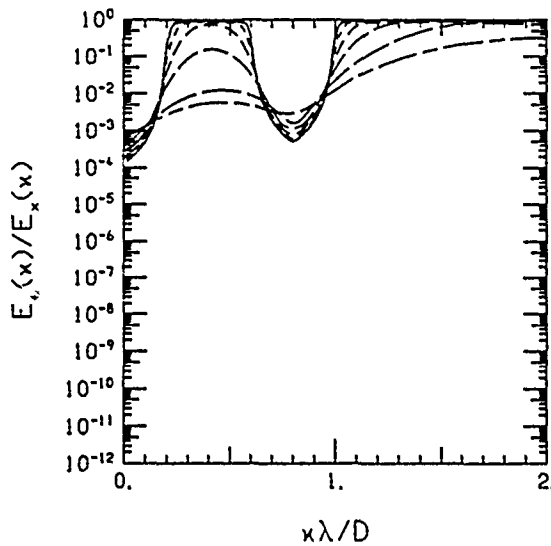
Figure 3.14. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.1$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 100\text{dB}$ and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 120\text{ dB}$.



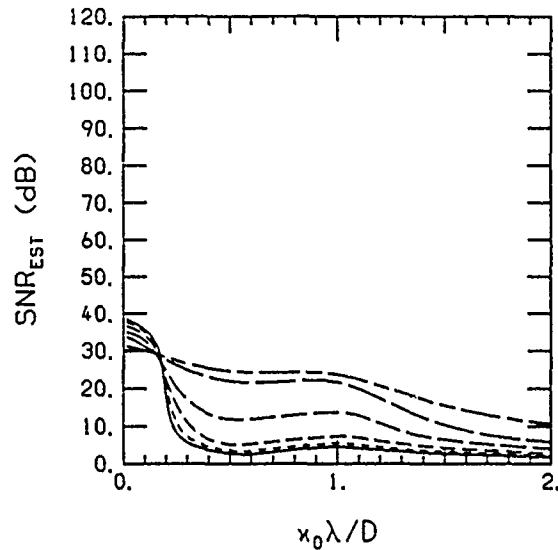
(a)



(b)

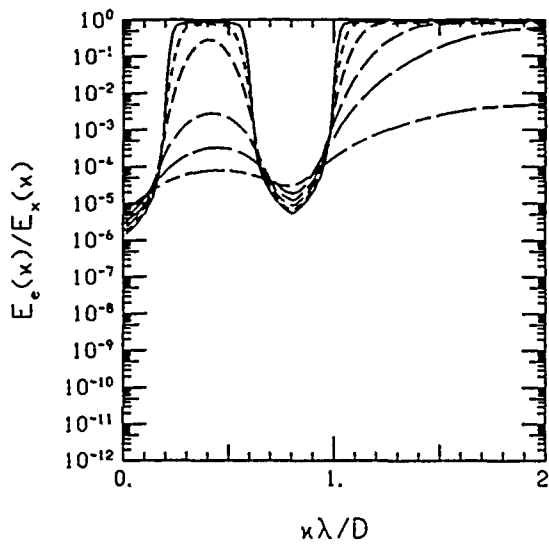


(c)

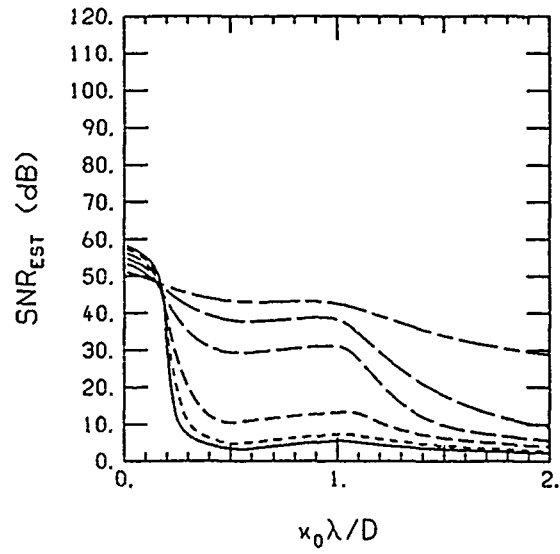


(d)

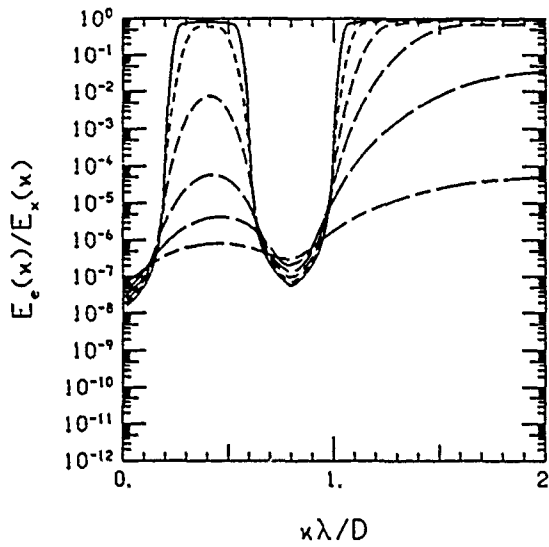
Figure 3.15. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.2$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 20\text{dB}$ and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 40\text{dB}$.



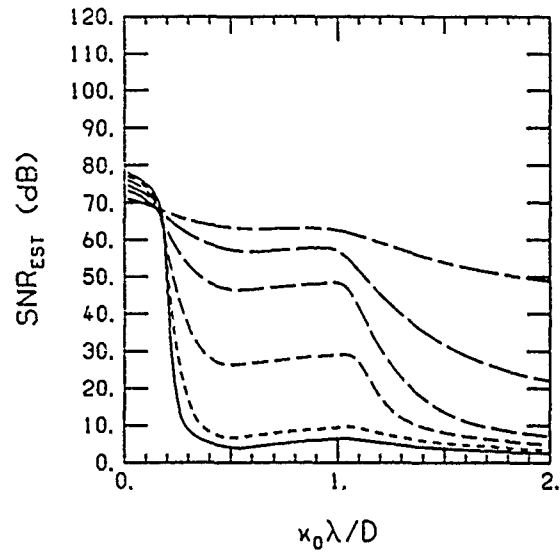
(a)



(b)

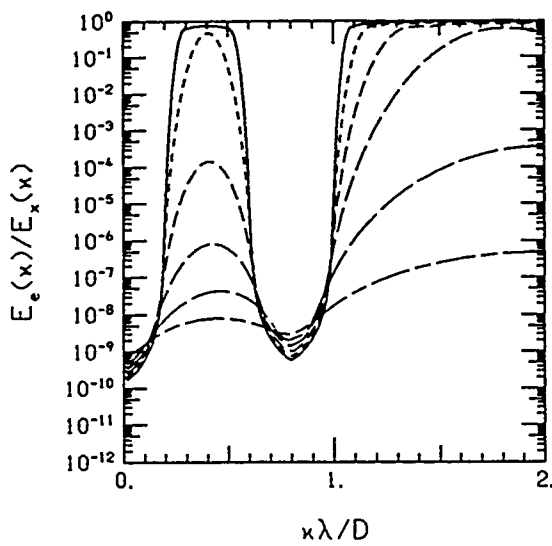


(c)

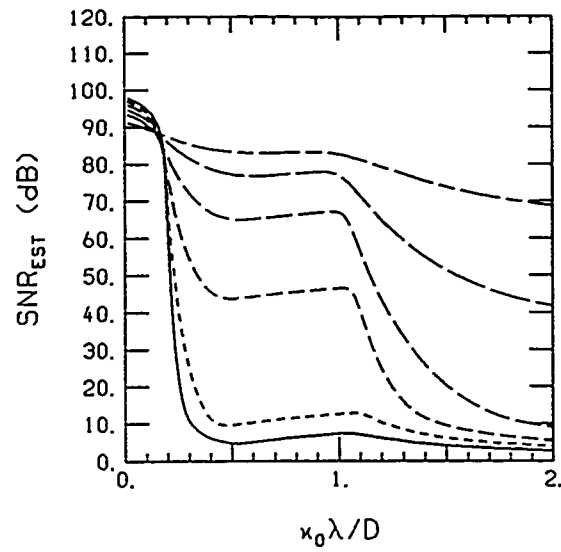


(d)

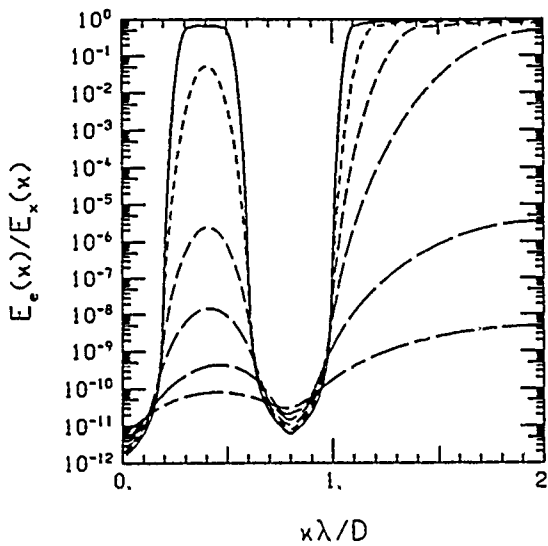
Figure 3.16. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.2$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 60\text{dB}$ and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 80\text{ dB}$.



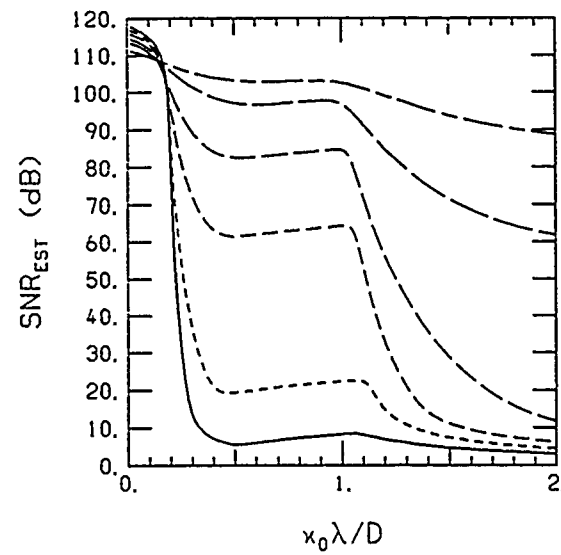
(a)



(b)

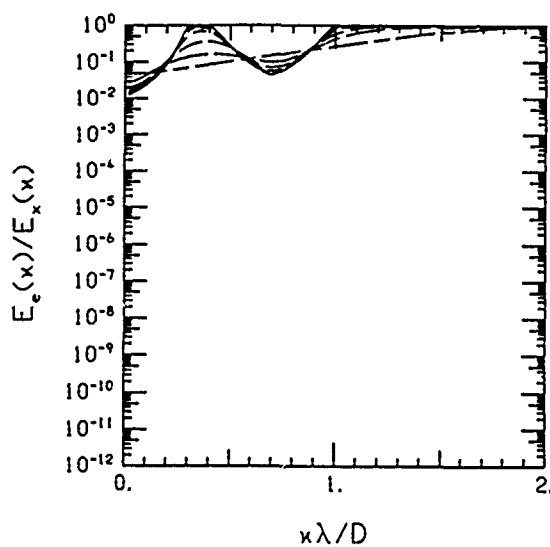


(c)

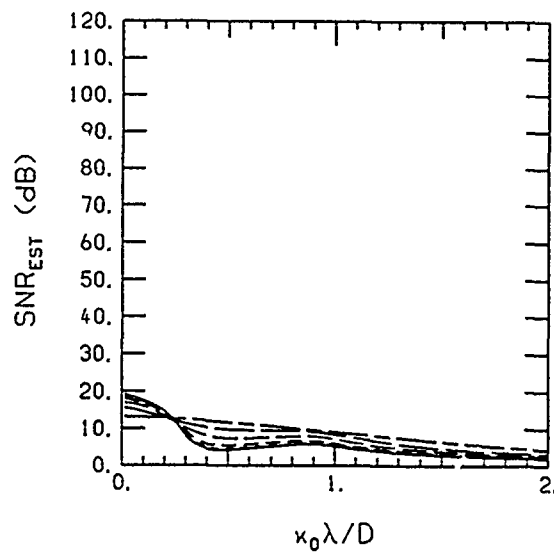


(d)

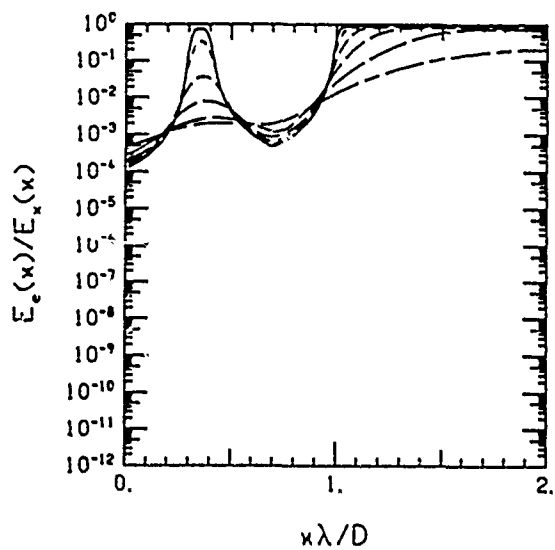
Figure 3.17. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.2$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 100\text{dB}$ and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 120\text{ dB}$.



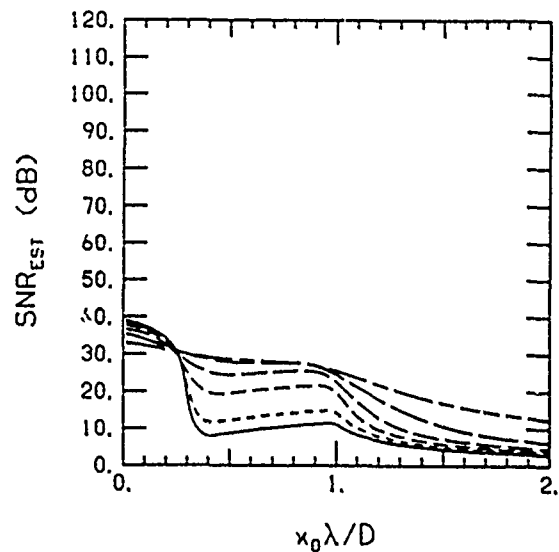
(a)



(b)

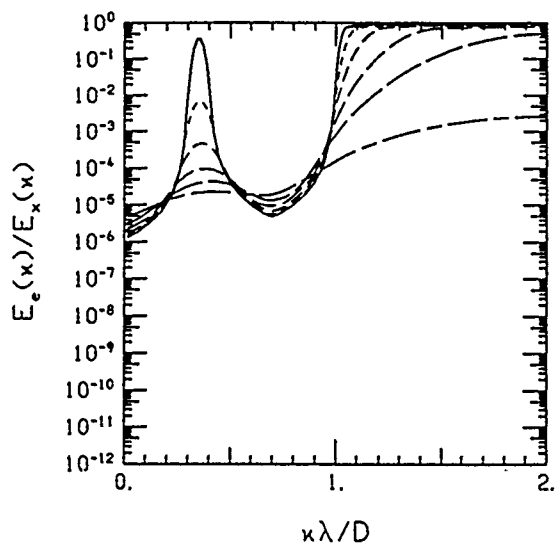


(c)

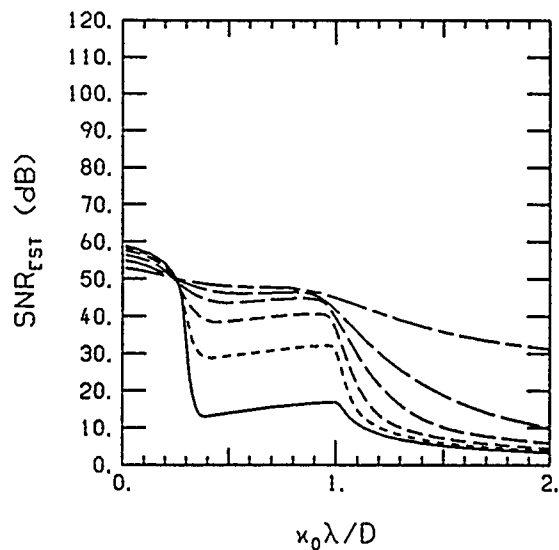


(d)

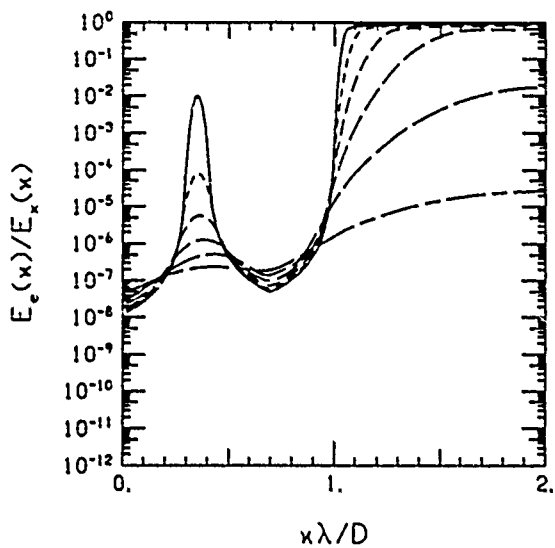
Figure 3.18. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.3$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 20\text{ dB}$ and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 40\text{ dB}$.



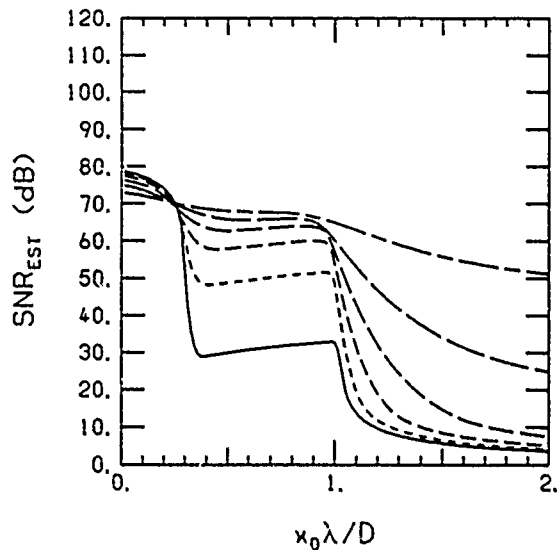
(a)



(b)

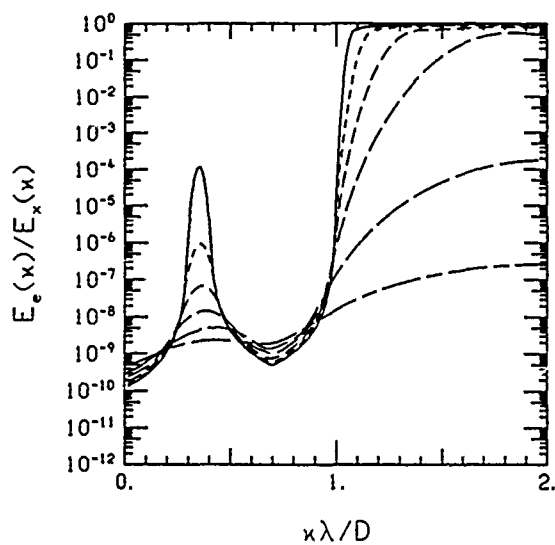


(c)

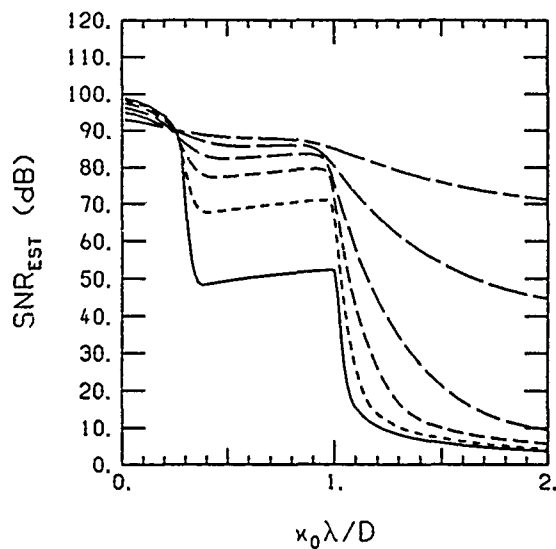


(d)

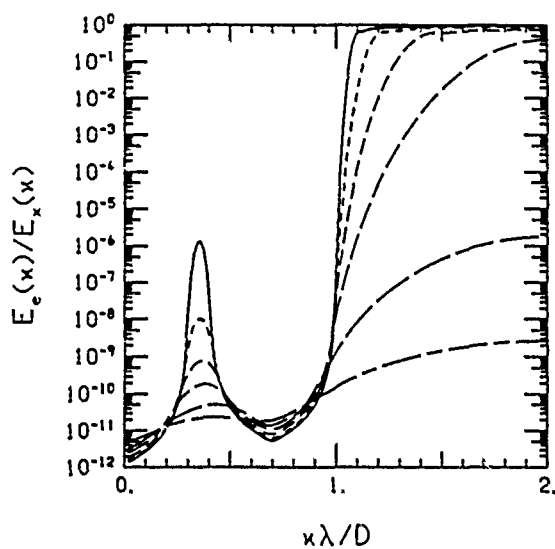
Figure 3.19. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.3$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 60\text{ dB}$ and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 80\text{ dB}$.



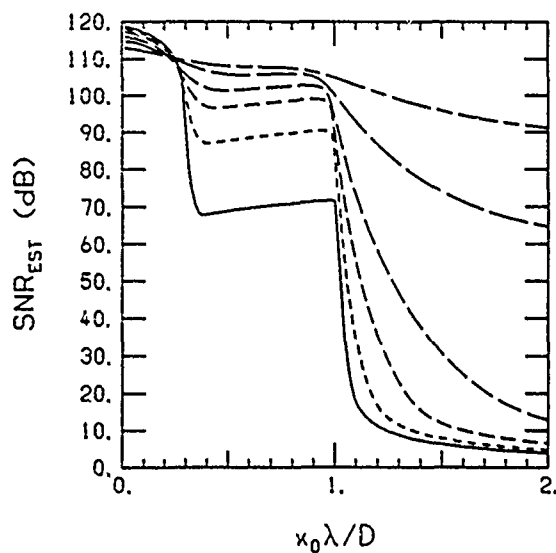
(a)



(b)

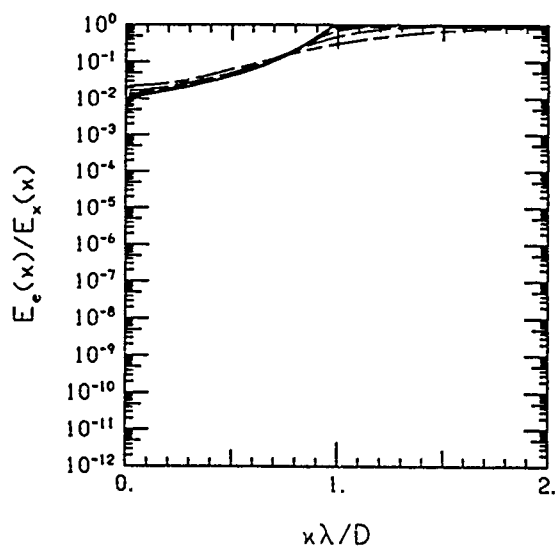


(c)

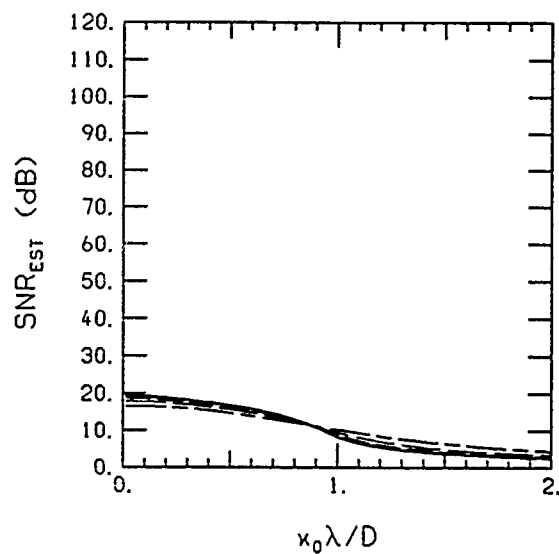


(d)

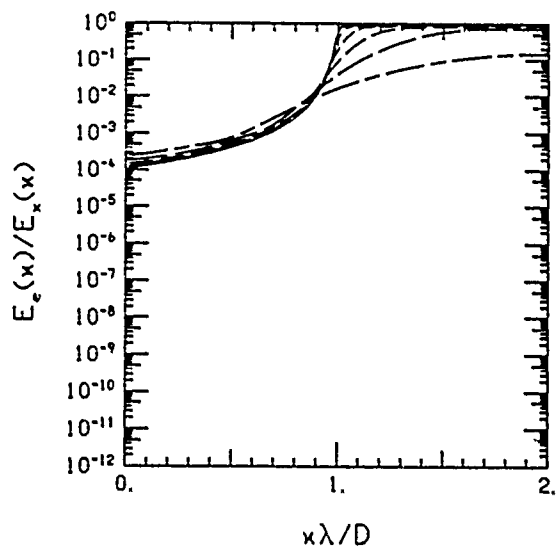
Figure 3.20. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.3$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 100\text{dB}$ and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 120\text{ dB}$.



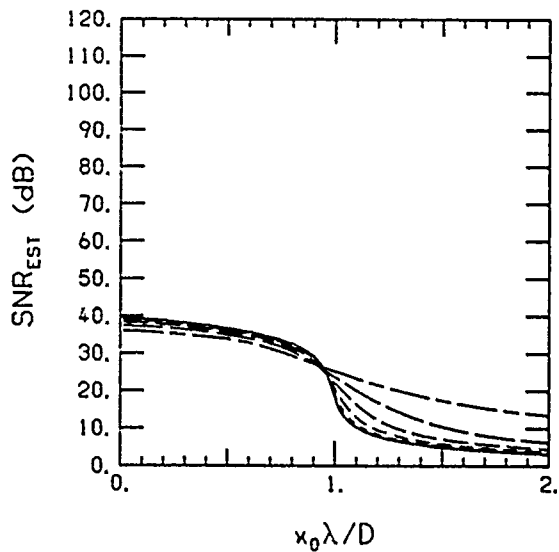
(a)



(b)

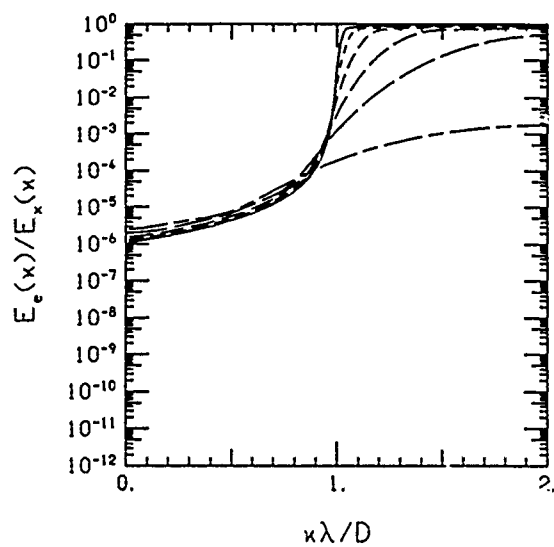


(c)

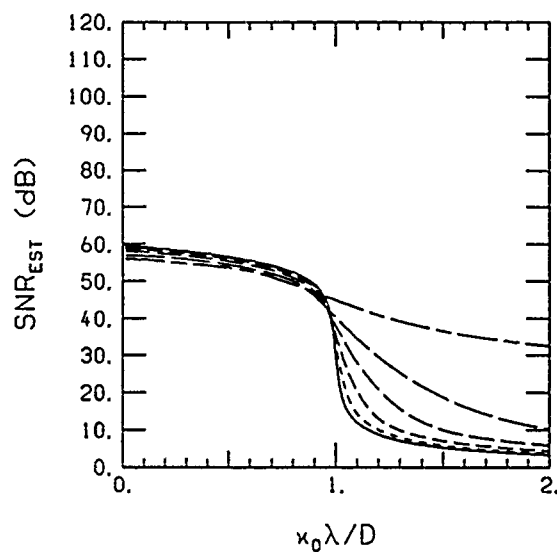


(d)

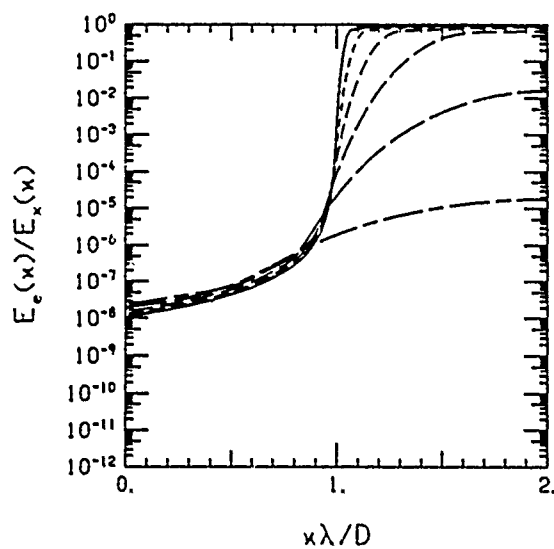
Figure 3.21. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.5$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 20\text{dB}$ and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 40\text{ dB}$.



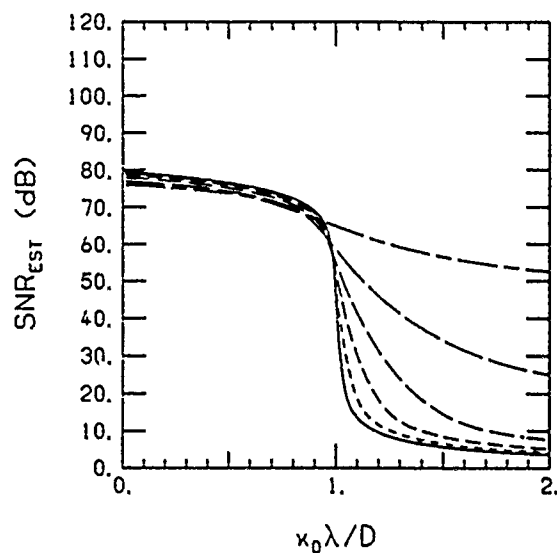
(a)



(b)

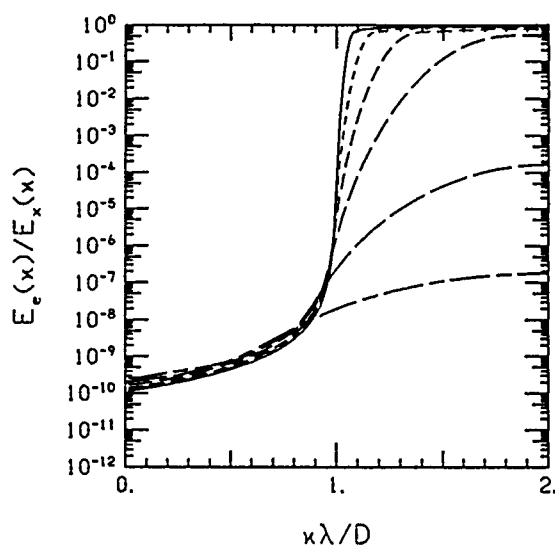


(c)

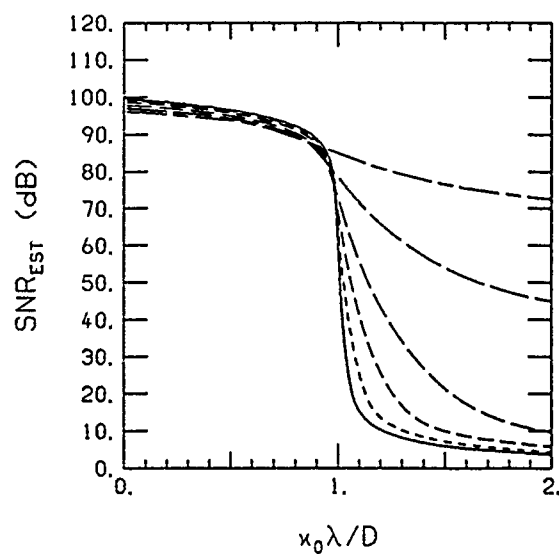


(d)

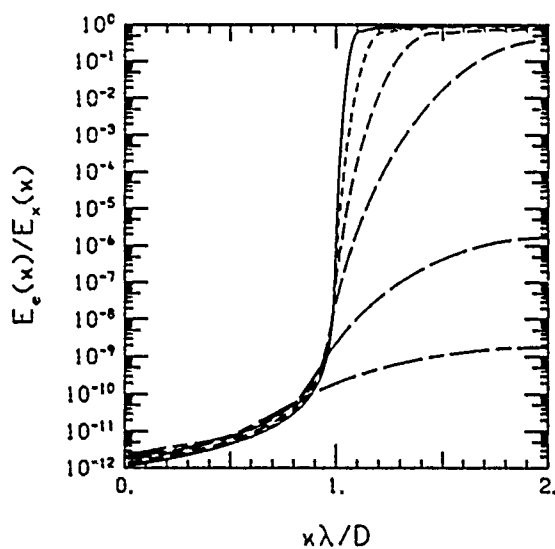
Figure 3.22. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.5$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 60\text{dB}$ and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 80\text{dB}$.



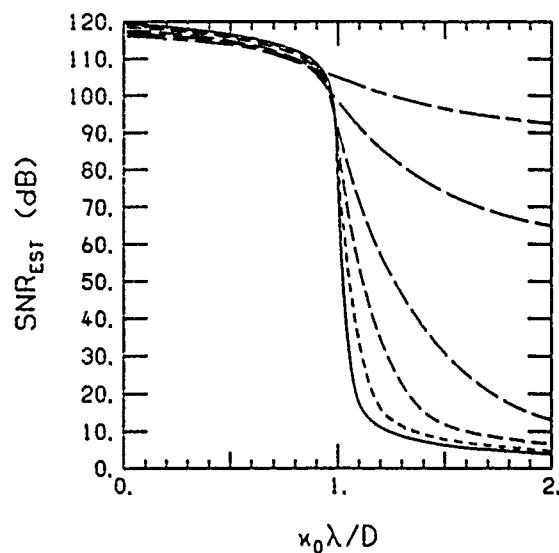
(a)



(b)



(c)



(d)

Figure 3.23. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.5$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 100$ dB and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 120$ dB.

3.4. Results Using the Least Squares Method

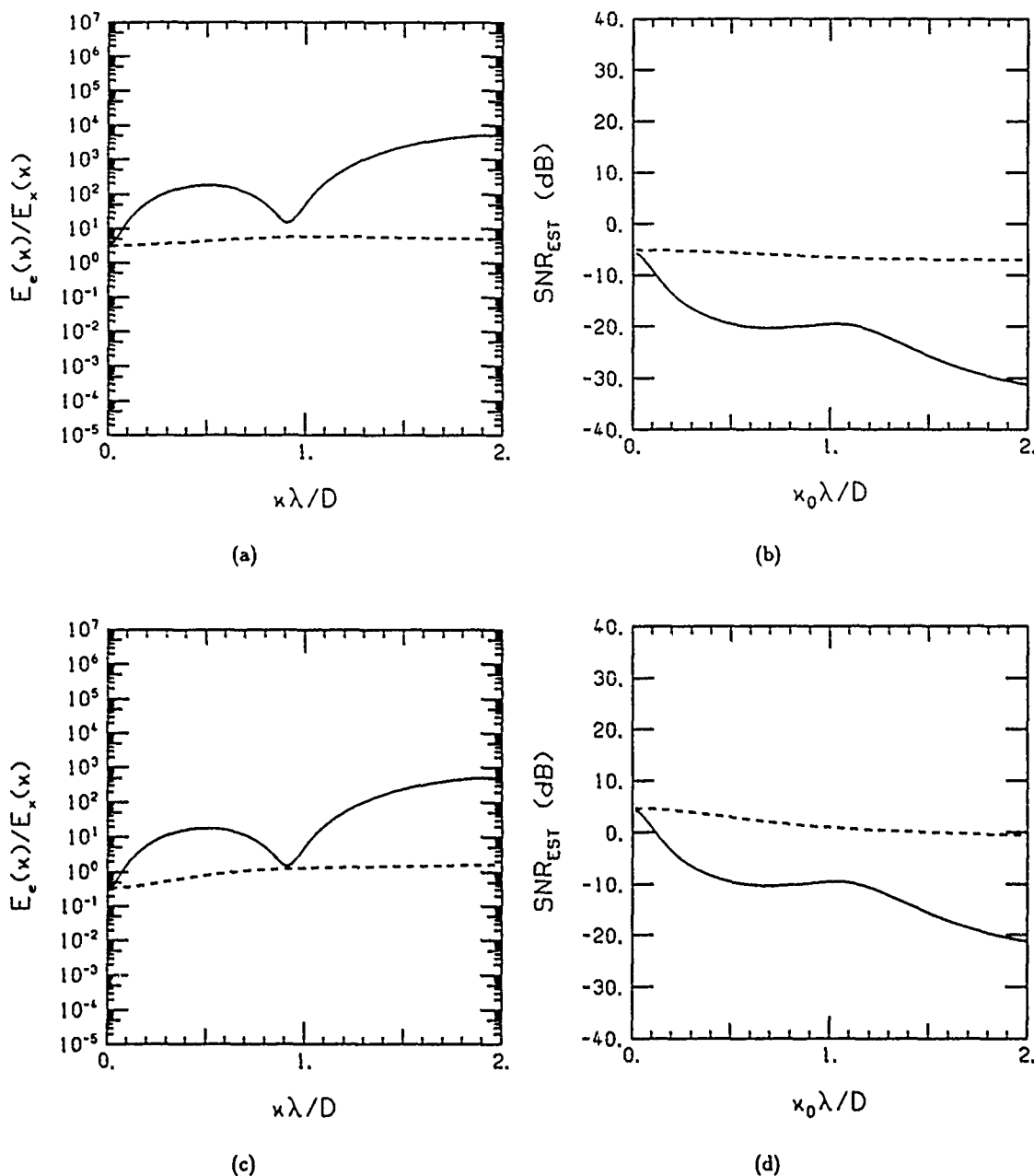
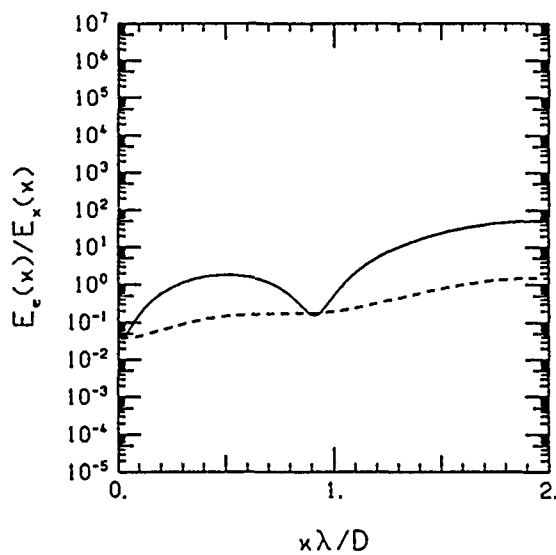
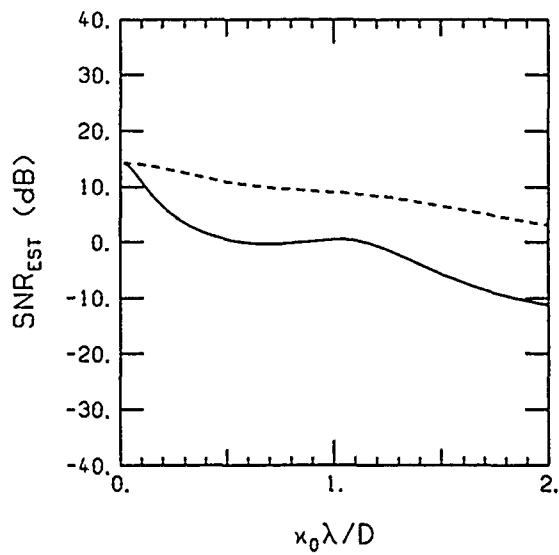


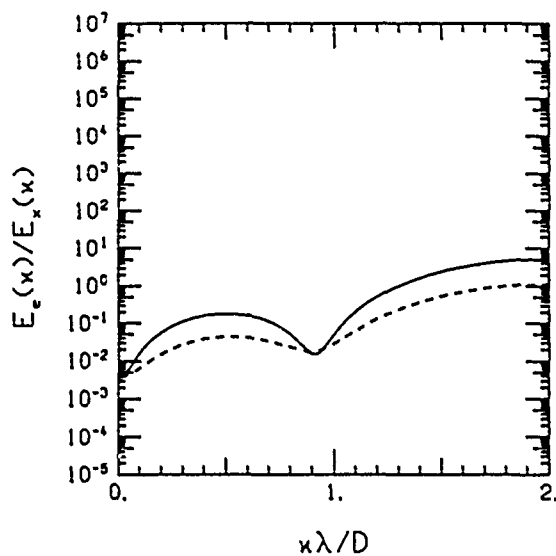
Figure 3.24. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1$, $k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 20$ dB, and figures (c) and (d) correspond to $SNR_{REF} = 30$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



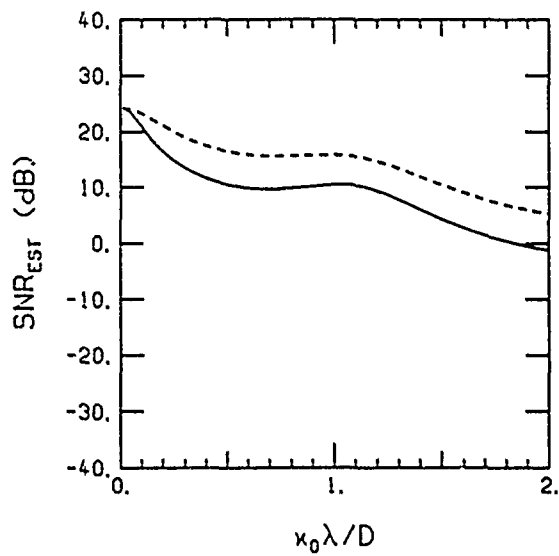
(a)



(b)



(c)



(d)

Figure 3.25. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1$, $k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of $1\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 40$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 50$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

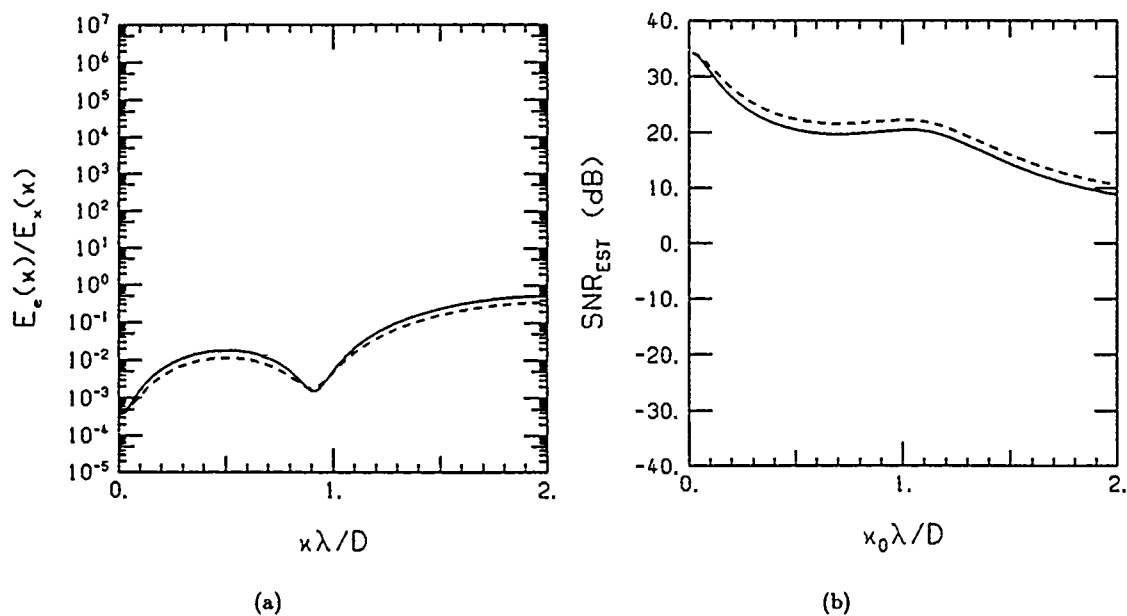
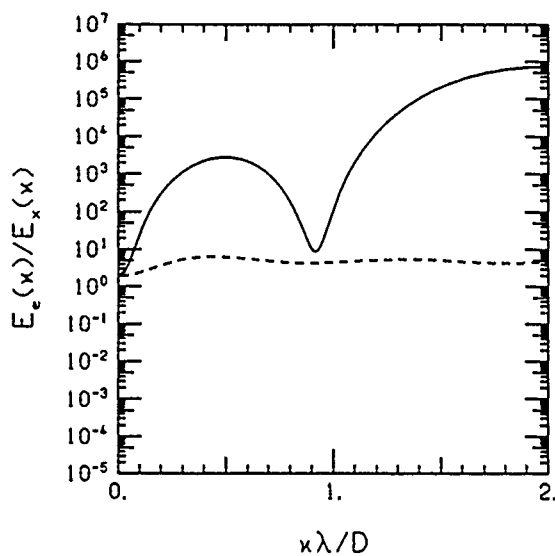
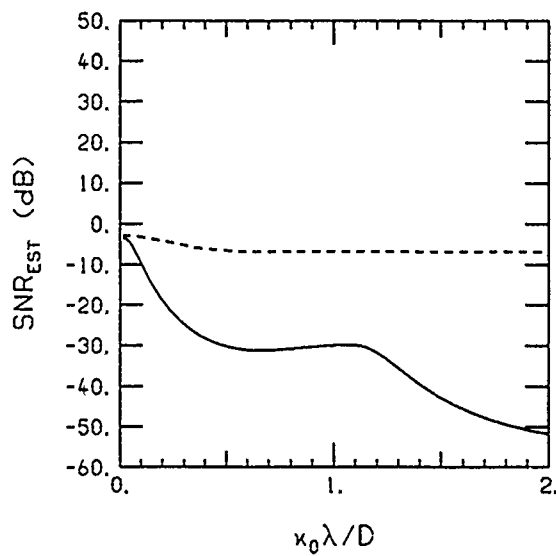


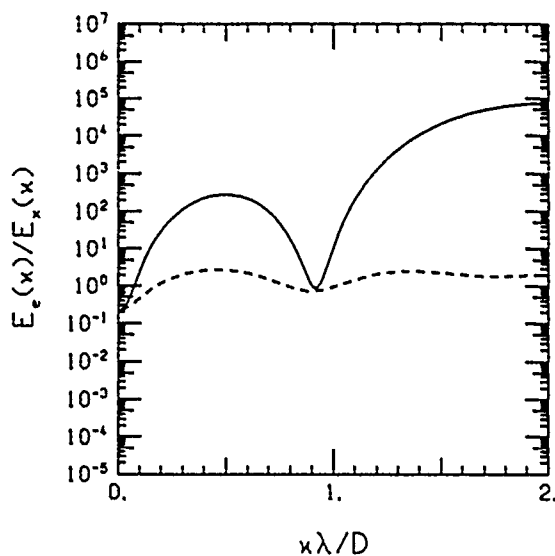
Figure 3.26. Energy Spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1$, $k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), an object support of $1\lambda/D$, and $\text{SNR}_{\text{REF}} = 60$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



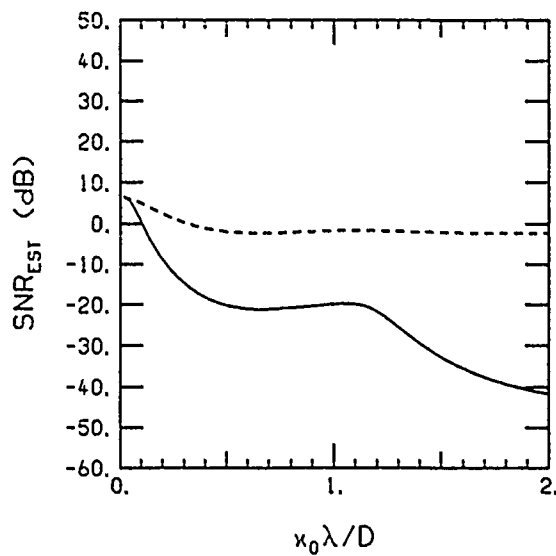
(a)



(b)

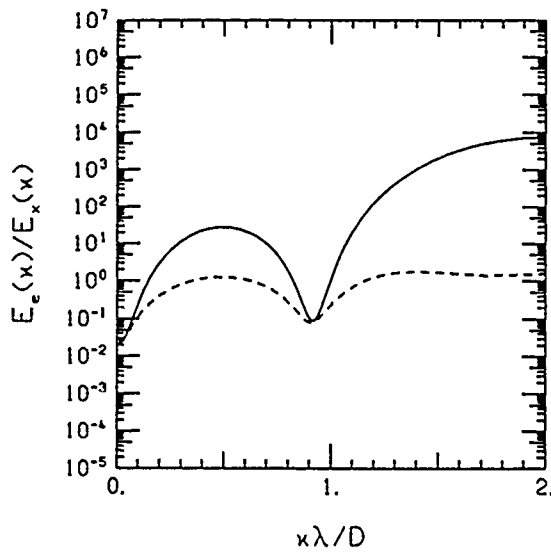


(c)

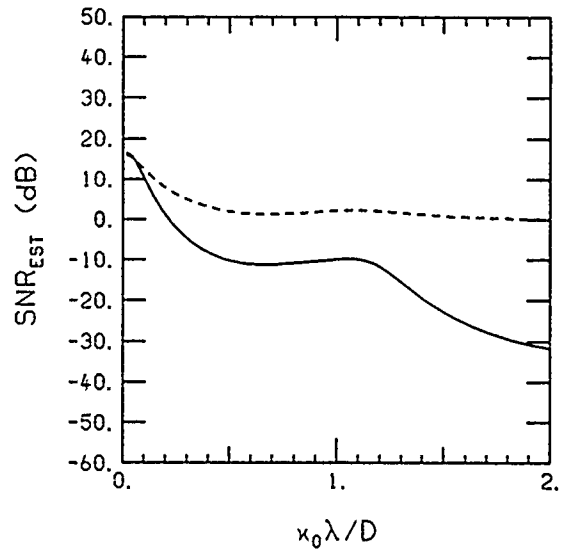


(d)

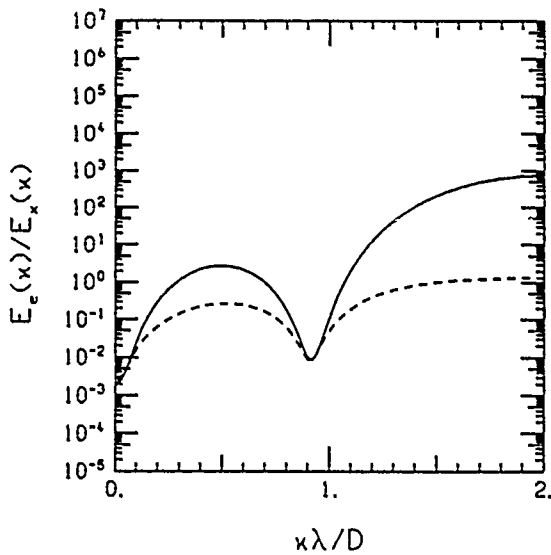
Figure 3.27. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1$, $k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of $2\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 20$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 30$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



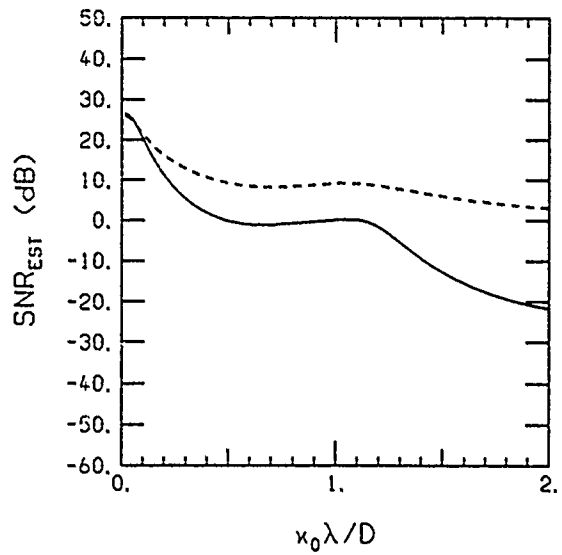
(a)



(b)

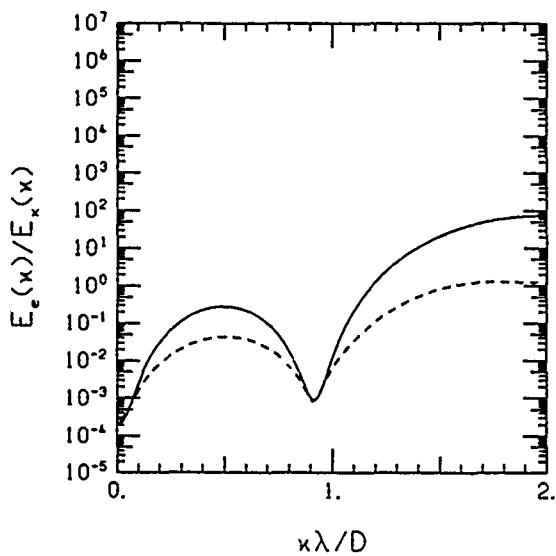


(c)

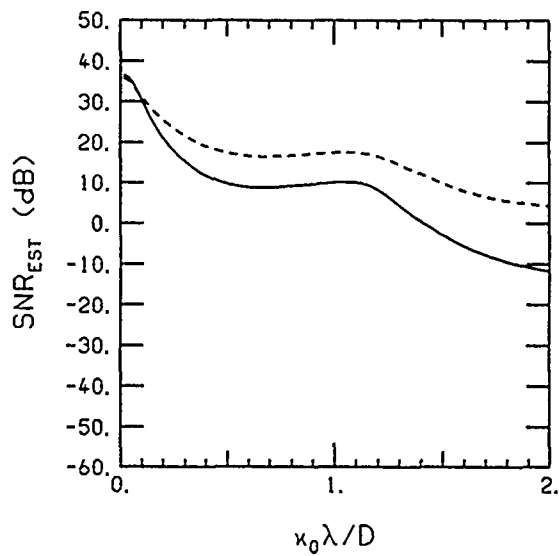


(d)

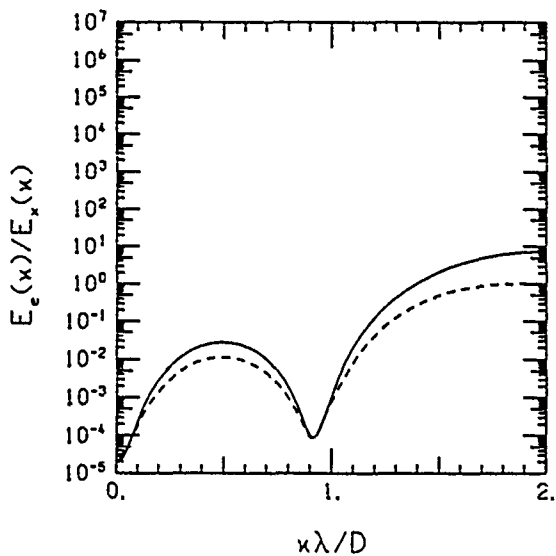
Figure 3.28. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1$, $k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of $2\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 40$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 50$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



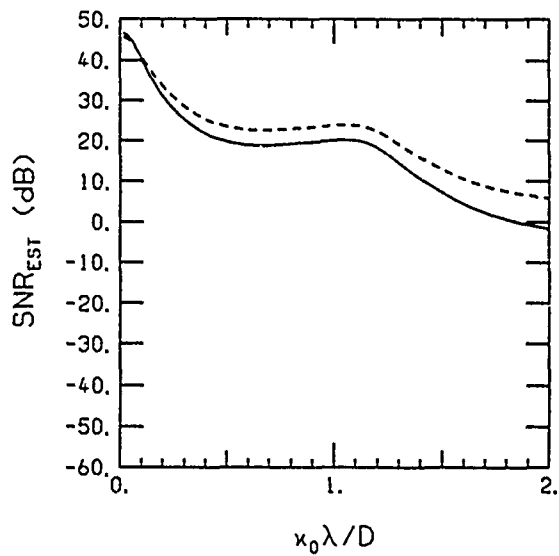
(a)



(b)

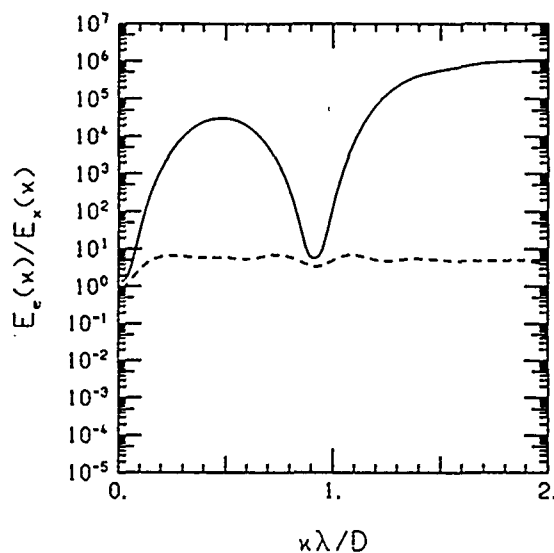


(c)

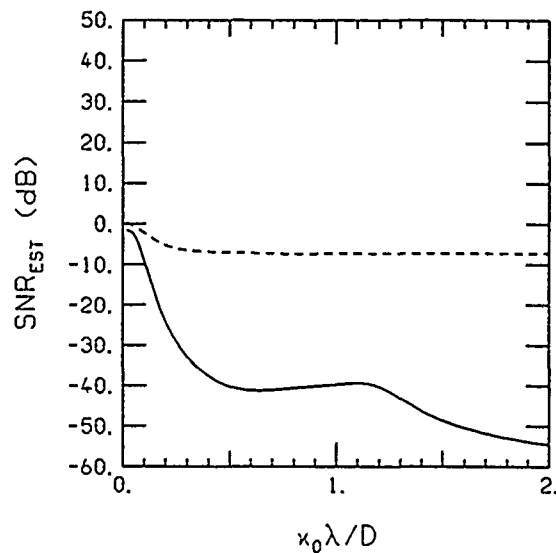


(d)

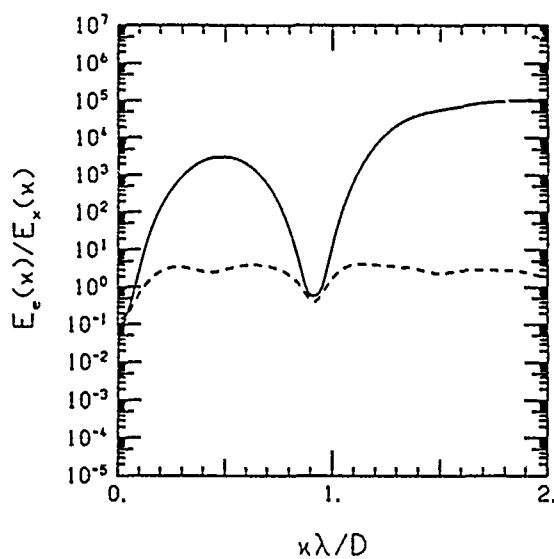
Figure 3.29. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of $2\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}}=60$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}}=70$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



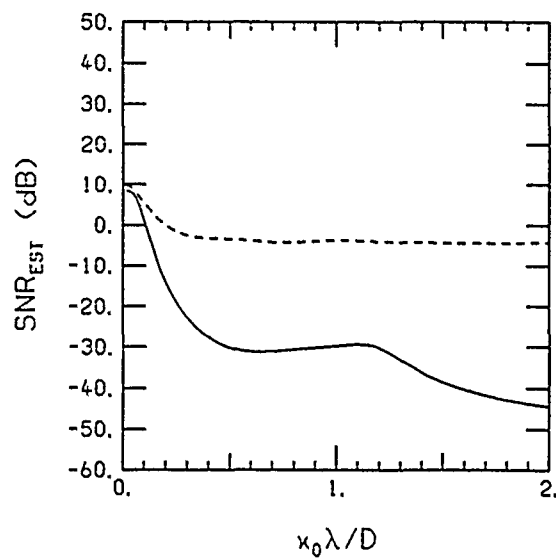
(a)



(b)

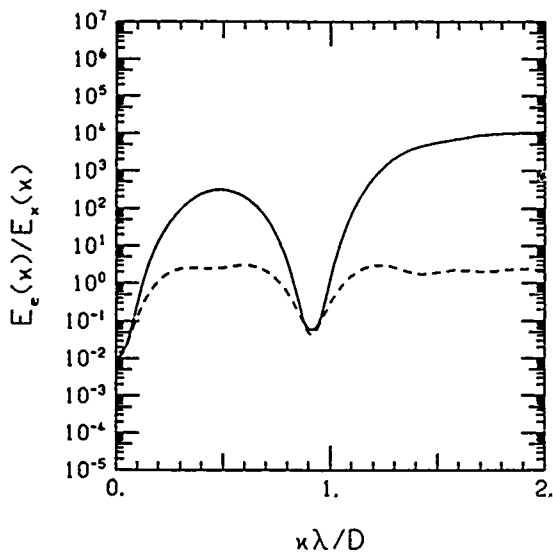


(c)

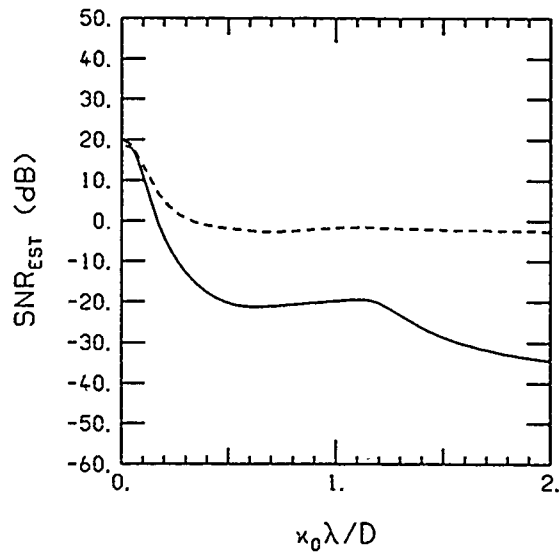


(d)

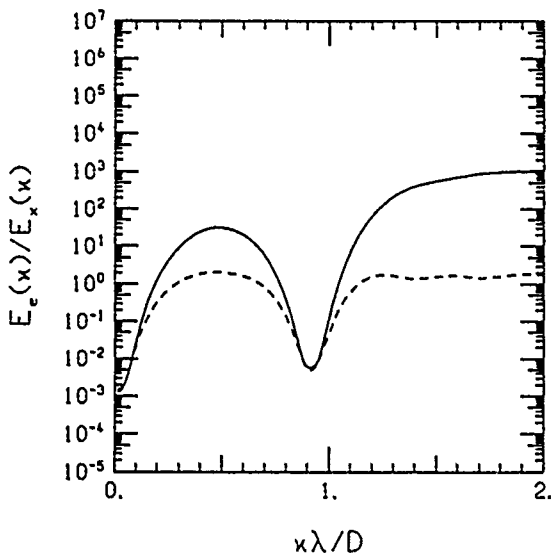
Figure 3.30. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1$, $k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of $4\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 20$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 30$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



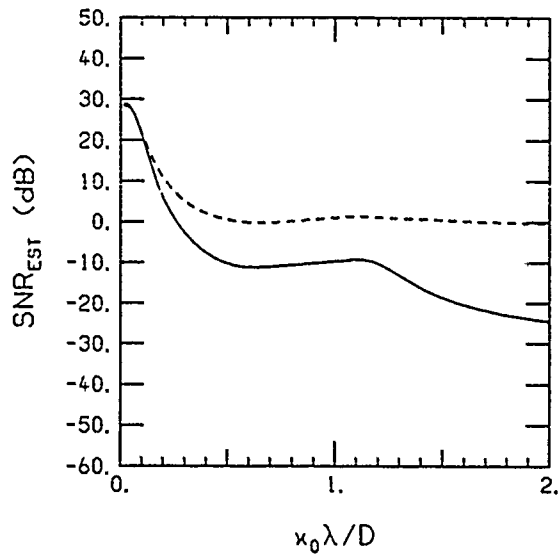
(a)



(b)



(c)



(d)

Figure 3.31. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1$, $k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of $4\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 40$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 50$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

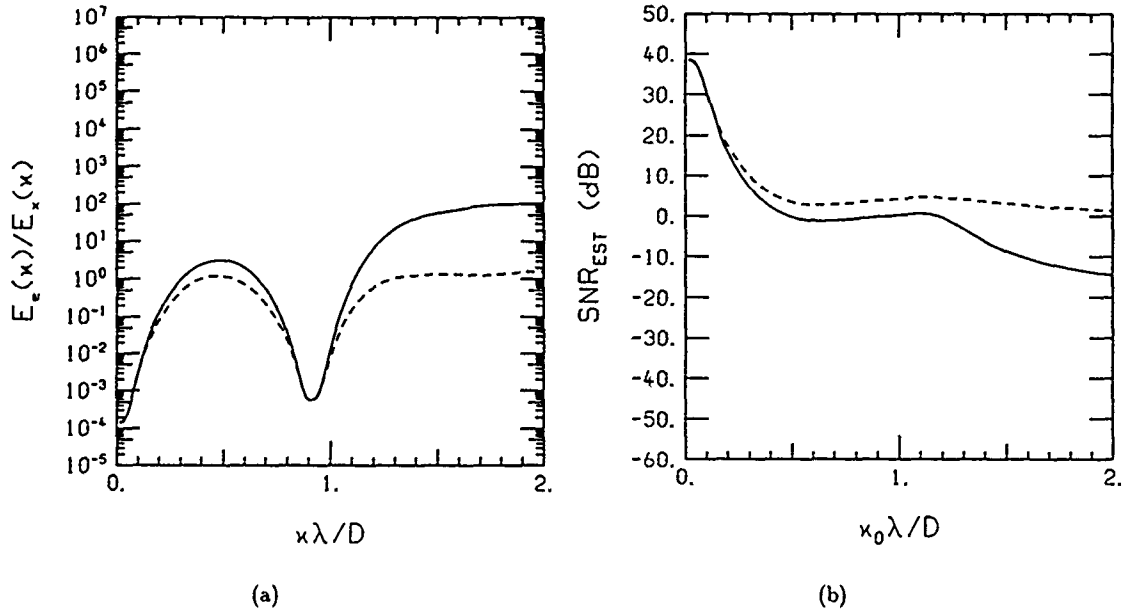
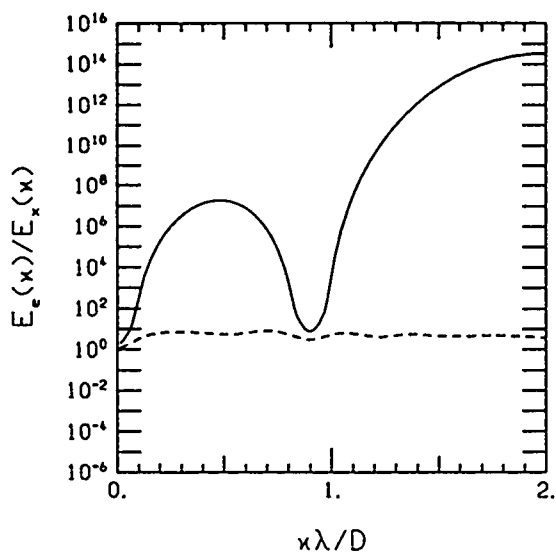
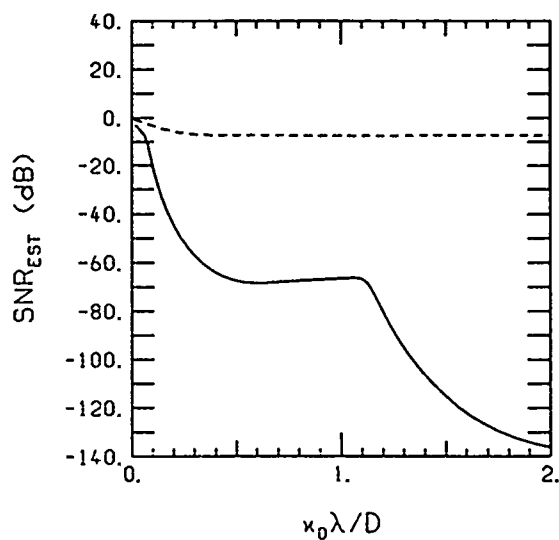


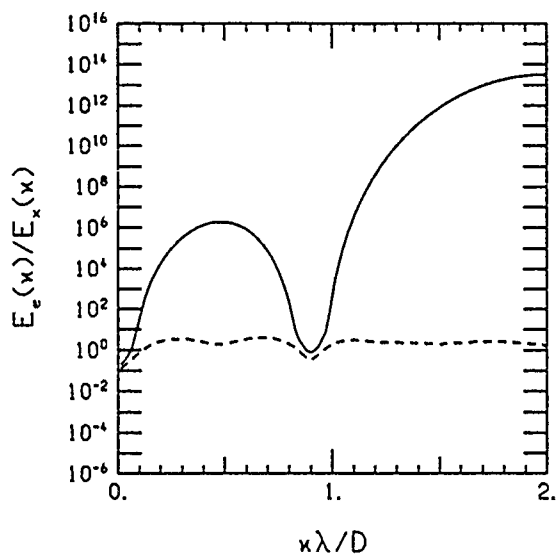
Figure 3.32. Energy Spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1$, $k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), an object support of $4\lambda/D$, and $SNR_{REF} = 60$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



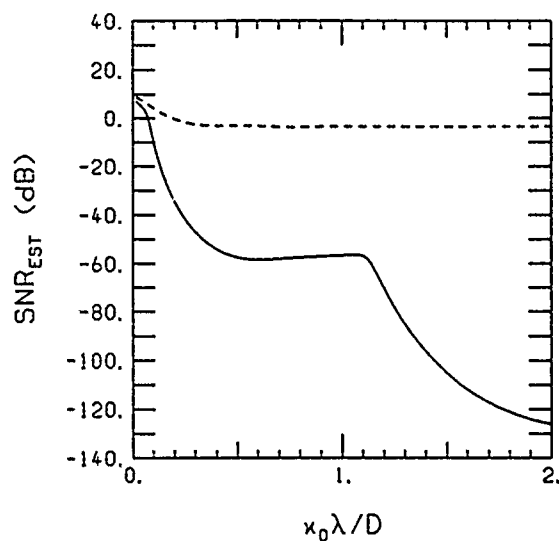
(a)



(b)

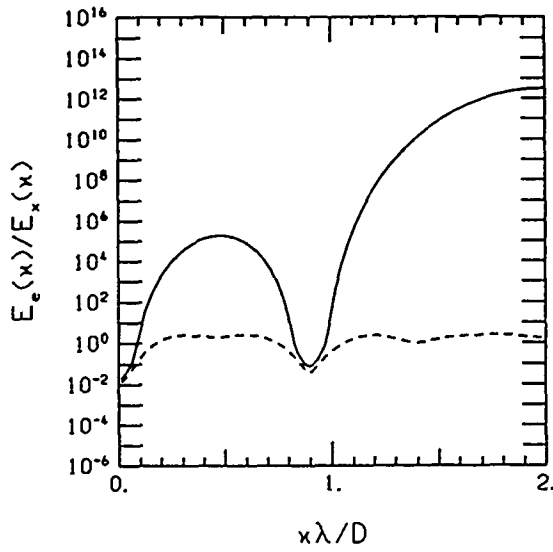


(c)

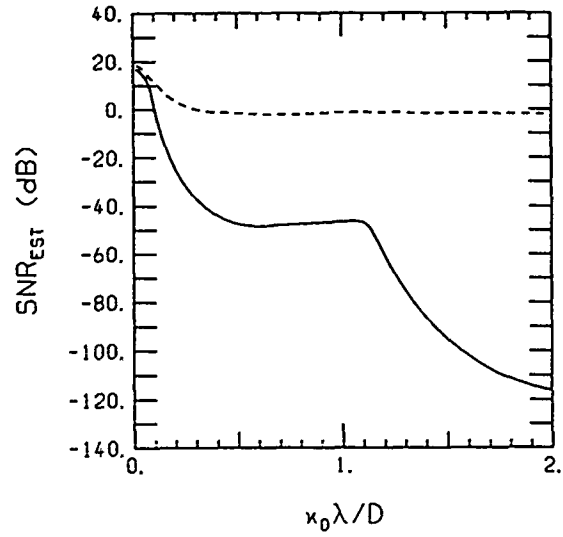


(d)

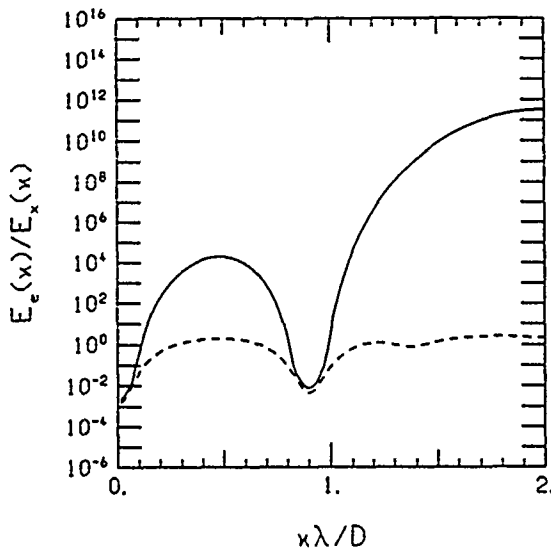
Figure 3.33. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number five and $d/D = 0.1, k = 511$ (corresponding to an image line length of 1150 pixels or $287.5\lambda/D$), and an object support of $4\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 20$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 30$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



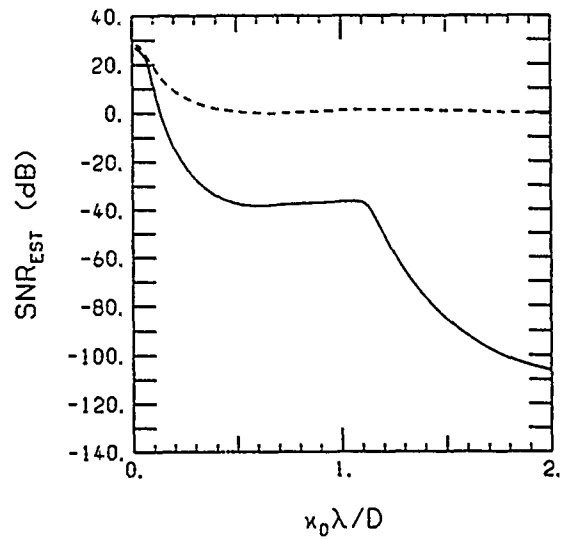
(a)



(b)

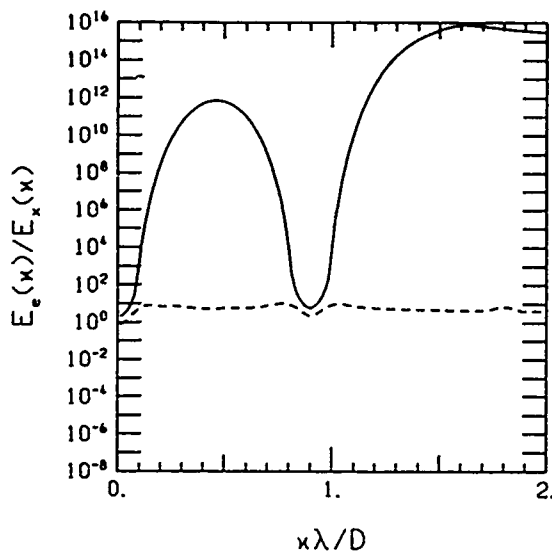


(c)

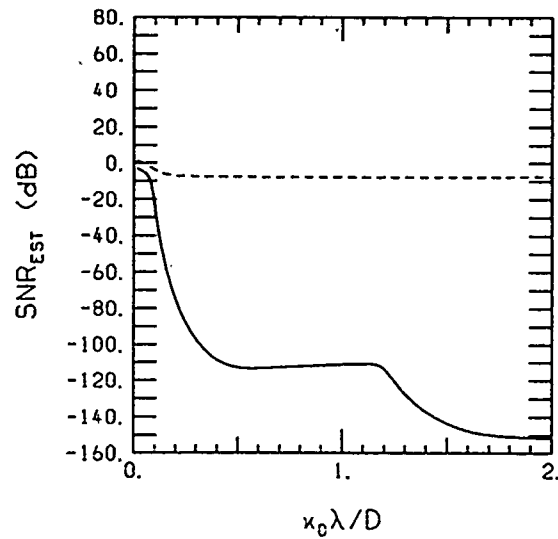


(d)

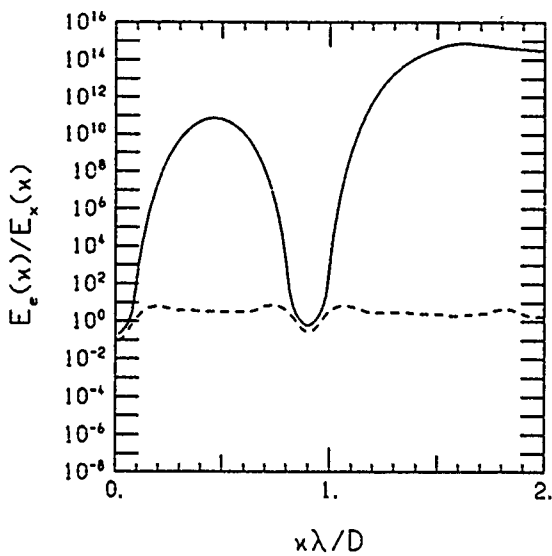
Figure 3.34. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number five and $d/D = 0.1$, $k = 511$ (corresponding to an image line length of 1150 pixels or $287.5\lambda/D$), and an object support of $4\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 40$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 50$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



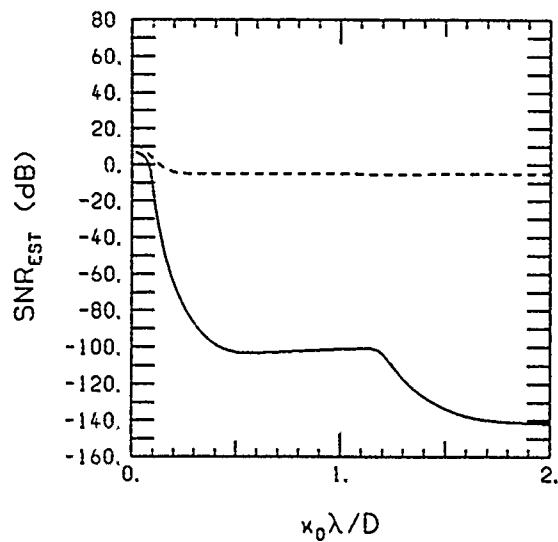
(a)



(b)

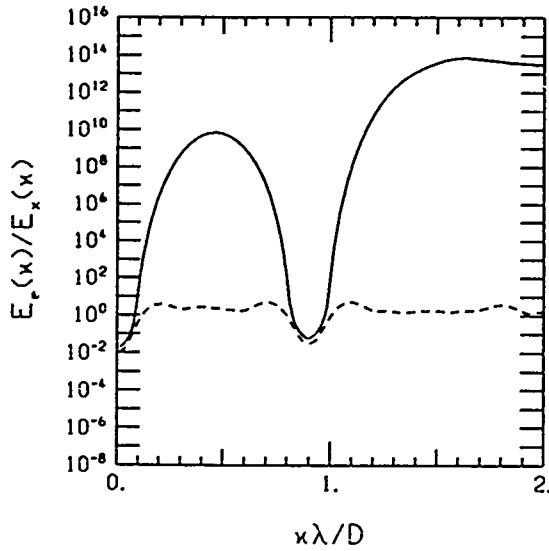


(c)

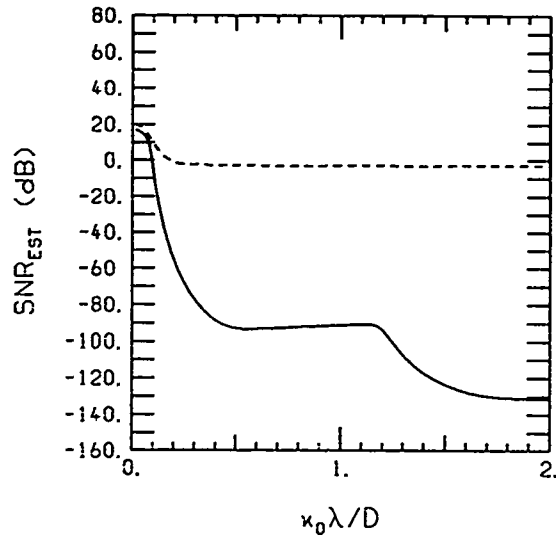


(d)

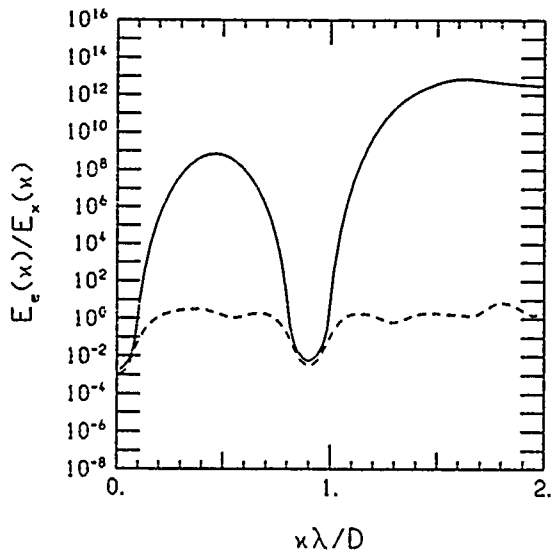
Figure 3.35. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number three and $d/D = 0.1$, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and an object support of $8\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 20$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 30$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



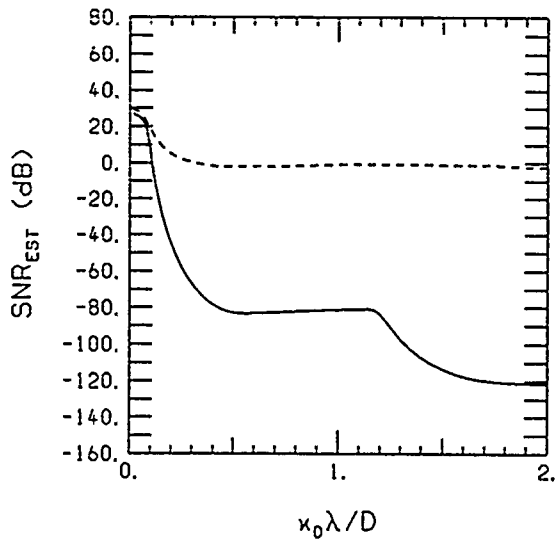
(a)



(b)

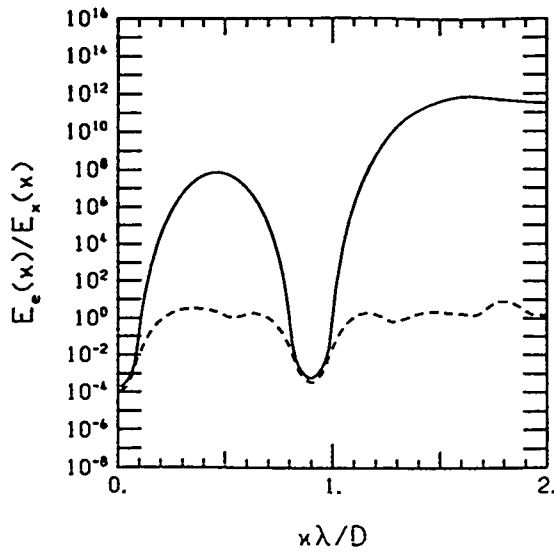


(c)

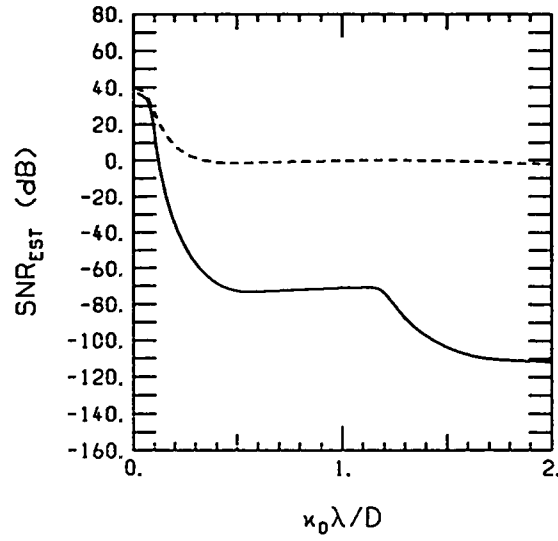


(d)

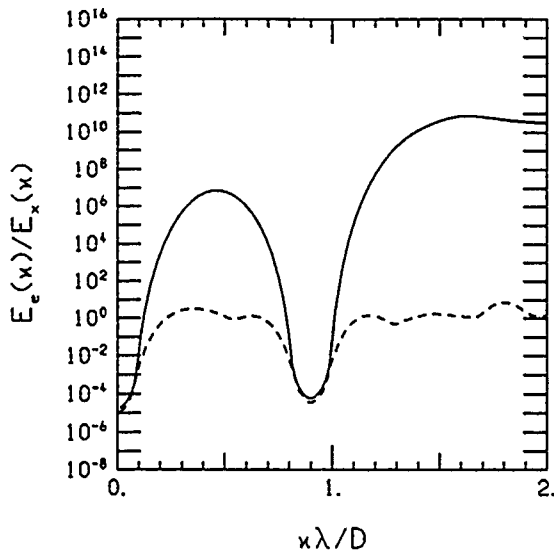
Figure 3.36. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number three and $d/D = 0.1$, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and an object support of $8\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}}=40$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}}=50$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.



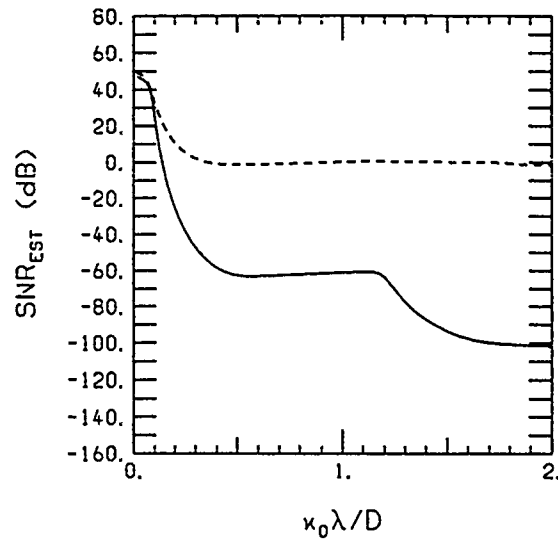
(a)



(b)



(c)



(d)

Figure 3.37. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number three and $d/D = 0.1$, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and an object support of $8\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}} = 60$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}} = 70$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

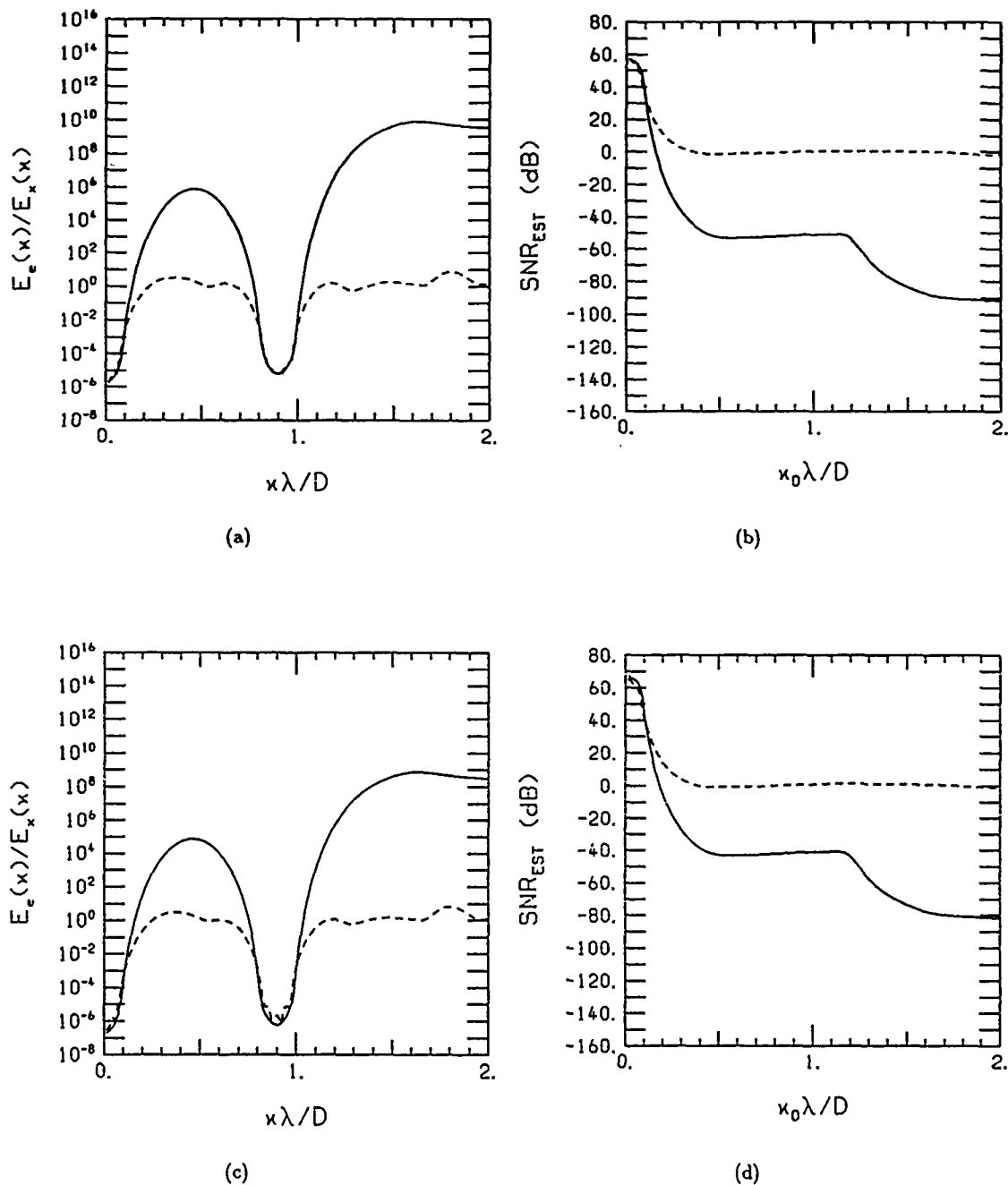


Figure 3.38. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number three and $d/D = 0.1$, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and an object support of $8\lambda/D$. Figures (a) and (b) correspond to $\text{SNR}_{\text{REF}}=80$ dB, and figures (c) and (d) correspond to $\text{SNR}_{\text{REF}}=90$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

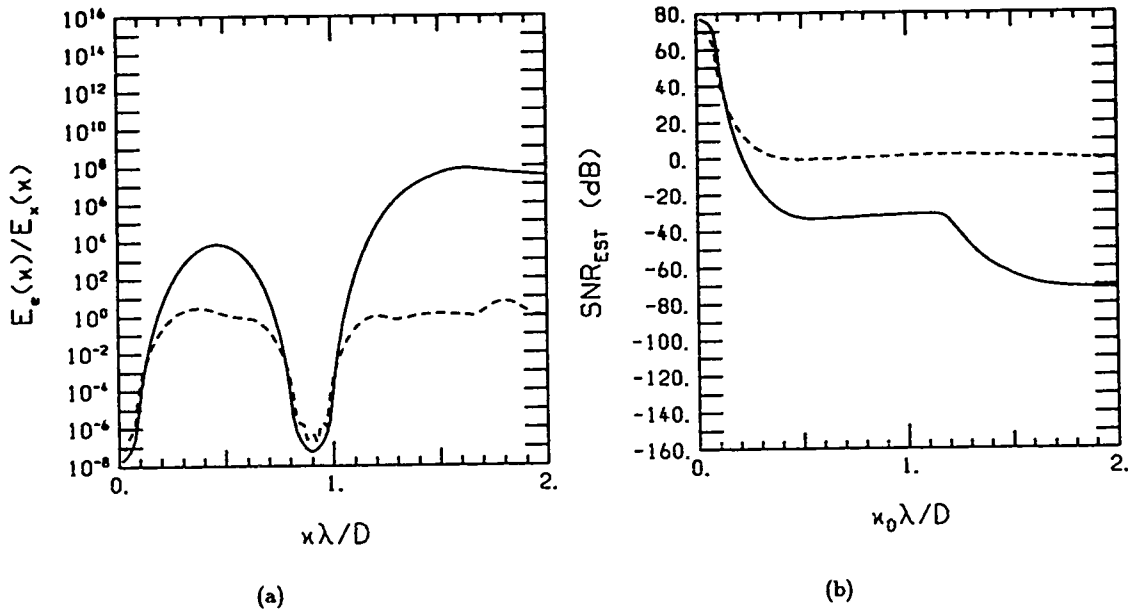


Figure 3.39. Energy Spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number three and $d/D = 0.1, k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), an object support of $8\lambda/D$, and $SNR_{REF}=100$ dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

Chapter 4

Object Reconstruction with Sparse Arrays of Optical Apertures.

Part I: Linear Methods

(originally issued as TR-1070)

4.1. Introduction

In this report we investigate the feasibility of recovering useful images of objects from noise-corrupted optical images formed by a very sparse array of apertures. By "useful image" we mean an image with a resolution commensurate with the overall dimension of the array and with a noise content low enough that the image would be considered useful. By "very sparse array" we mean an array of apertures with aperture size very small relative to aperture spacing so that the frequency response of the array comprises small "islands" of nonzero response surrounded by a "sea" of zero response.

At first, we questioned the feasibility of such a task based upon the supposition that superresolution is a dead issue, where here we are using the term "superresolution" rather loosely to mean filling in missing spatial frequency information with the use of some prior knowledge of the object. Had it not been shown that modest gains in resolution require enormous sacrifice in a signal-to-noise (SNR) ratio? However, it was brought to our attention that radio astronomers form apparently useful images from interferometric data using very sparse arrays and an algorithm called CLEAN.^{2,27} Does CLEAN have magical powers, do radio astronomers have gobs of SNR at their disposal, or is there something about superresolution that we don't understand?

The possibility of resolving power beyond the classical limit of an idealized optical system has long been recognized.^{19,22,18,20,21} The earliest mention of the subject appears to be by Coleman¹⁹ in 1947. In 1952, Toraldo di Francia²² applies the concept of super-gain in antennas to optical systems and concludes that the classical limit of $1.22 \lambda/D$ is only a practical limit and the actual resolution is limited only by noise. He discussed a procedure for designing what he calls a "super-resolving pupil" in which improved resolution can be obtained over a limited field by modifying the pupil of a diffraction-limited imaging system. In 1955, the same author¹⁸ approached the concept of resolving power from the point of view of information theory. He reasoned that information is lost when an object is transformed into an image and therefore several different objects may produce the same image. If two different objects produce identical images then they cannot be "resolved", and thus this object ambiguity must have something to do with the definition of resolution. He gives examples for the coherent light case. He suggests that prior knowledge could be used to reduce the ambiguity in the object-image mapping. J. Harris⁹ (1964) removes this difficulty by showing that no two distinctly different objects of finite angular size can have identical images. To establish this result, he uses two theorems from analytic function theory. The first theorem states that the Fourier transform of a square-integrable function of finite support is analytic throughout the entire domain of the spatial frequency plane. Harris then invokes analytic continuation (the second theorem) to demonstrate that, in the absence of noise, and starting with an arbitrarily small (but finite) piece of the Fourier transform of an object, one can find the entire Fourier transform (and thus the object itself). He next uses sampling theory to develop an algorithm to extrapolate from a piece of the Fourier transform of an object to the whole transform. The method requires solving a system of linear equations. Unfortunately, infinite precision requires an infinite number of equations. He applies his method to measuring the angular separation of two hypothetical point sources when the angle is less than the reciprocal of the spatial bandwidth of the system.

Following Harris, several researchers exploit the concept of analytic continuation to develop algorithms for object recovery. Barnes⁷ (1966) uses prolate spheroidal wave functions to reconstruct objects of finite support in a one-dimensional coherent imaging system. Frieden⁸ (1967) extends the use of prolate spheroidal wave functions to the reconstruction of partially coherently or incoherently illuminated 2-D objects of finite support. Both the methods of Barnes and Frieden require infinite series expansions in order to achieve infinite resolution. Brown¹¹ (1969) carries the development to the logical next step by examining the effect of series truncation on the amount of super-resolution achieved.

It was recognized early on that noise gums up the superresolution works. Rushforth and R. Harris¹⁰ (1968) published one of the earliest papers in optics to seriously examine the effects of noise on superresolution performance. They looked at three types of noise in the context of a one-dimensional

coherent system and finite object support: background, measurement, and computer round off. Using the object recovery method of Barnes⁷ and an extension of the method using Wiener filter theory, they computed mean-square-error as a function of the degree of extension of resolution beyond the classical limit with object support as a parameter. The paper demonstrates that even a modest extension of resolution beyond the classical limit is very costly in terms of the noisiness of the final image.

Lukosz^{5,6} (1966, 1967) approach the subject of superresolution from a somewhat more global viewpoint. He proposed an invariance theorem to explain the concepts underlying all super-resolution techniques. The theorem states that it is not the spatial bandwidth of a system that is fixed, but the number of degrees of freedom, specified by the two space-bandwidth products and the time-bandwidth product of the optical system. Thus, one can extend any parameter of the system, e.g., spatial bandwidth, beyond the classical limit by proportionately reducing some other parameter, provided some *a priori* information concerning the object, e.g., independence of time, is known.

A few researchers approached object restoration from an information-theoretical standpoint.^{17,18} As early as 1955, Fellgett and Linfoot²³ derived the information capacity of a two-dimensional optical system in terms of SNR, field-of-view, and spatial bandwidth. A recent paper by Cox and Sheppard¹ (1987) extended the results of Fellgett and Linfoot by including exposure time and temporal bandwidth to the expression for the information capacity of an optical system. They then use the invariance of capacity to demonstrate that any attempt to increase the spatial bandwidth of the optical system through analytic continuation results in a reduction of the SNR in the final image. They derive an upper bound on the resolution improvement as a function of the ratio of SNR with and without the resolution improvement. The bound is very loose, but nonetheless demonstrates that a modest improvement in resolution can require very large increases in signal strength.

In the research described so far, the concept of superresolution was generally thought of as any process which extends spatial frequency knowledge of an object beyond the spatial cutoff frequency of the optical system. In a series of three papers, Lannes et al.^{24,25,26} (1987) analyze the problem of object restoration with missing spectral information. They view the restoration problem to be a compromise between resolution and robustness. They develop a robustness theory that includes object support and the distribution of regions of missing spectral information and conclude that super-resolution extrapolation is "harder" than super-resolution interpolation. In other words, extrapolating frequency information is "hard", but interpolating between known regions, such as CLEAN does, may be "easier". Furthermore, they state that the robustness of the interpolation process is increased whenever the frequency gaps are well distributed over the aperture to be synthesized.

To simplify computations and interpretation of results, we decided to conduct our investigation in one dimension. Furthermore, we chose a discrete model of the optical system so that convolution could be simply represented as matrix multiplication. A "typical" sparse array in one dimension is shown in Fig. 4.1. This particular array is an example of a nonredundant array, so called because no pair of subapertures contributes the same spatial frequency region to the MTF as any other pair (except the region in the vicinity of zero, of course). This is witnessed by noting that all "islands" of nonzero response are of equal height (again, except for the "island" at the origin). In our studies, we will be particularly interested in the feasibility of using the concept of finite support to "fill in" the gaps shown in Fig. 4.1, when the subaperture size (d) is very small compared to the array size (D), or, put another way, when the width of the "islands" in the MTF is small compared to the spacing between islands. The results we will present were computed using nonredundant arrays of four, five and six subapertures, shown respectively in Fig's 4.2, 4.3, and 4.4.* It is in no way important to the results we present that the arrays are nonredundant. It is merely a convenience. An important feature to note of each of the arrays is the gap between islands. Our results will show that when

* These arrays were obtained from Barakat.⁴

the angular support of the object is less than the inverse of this gap, that the price, in terms of signal-to-noise ratio, that we have to pay for using a sparse array is "moderate", and otherwise, the price rapidly becomes "severe".

We begin in Section 4.2 with a description of the mathematical model of the image formed by our one-dimensional discrete optical system. In Section 4.3 we define a performance measure. It remains to pick an object recovery algorithm. There are probably at least as many recovery algorithms as there are researchers in the field.^{3,12,13,14,15,16} Most of the algorithms are iterative, mainly because of the practical limitations of the object recovery process relative to the requirements of the two dimensional problem, but also because of the ease of incorporating prior knowledge such as positivity. Since we have no desire to develop yet another algorithm, but rather are interested in "feasibility" in terms of signal-to-noise ratio, we will go on the assumption that a linear transformation of the "measurement" will probably yield performance adequate for our purposes. We thus choose two linear object recovery methods, minimum-variance, where knowledge of the first two moments of object intensity distribution and observation noise is assumed, and unweighted least-squares, where no statistical knowledge is used. The minimum-variance results are presented in Section 4.4 and those of least-squares in Section 4.5.

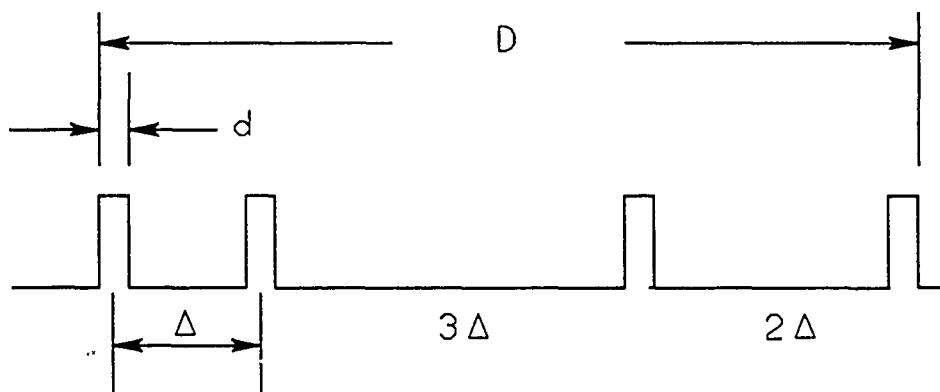
4.2. Discrete Optical Model (One-Dimensional)

Let the components of the $L \times 1$ vector $\mathbf{z}$ be the intensity pixels of the object line, let the components of the $N \times 1$ vector $\mathbf{y}$ represent the intensity pixels of the image line, and let the $N \times L$ matrix B be the transformation from object line to image line (its columns contain shifted versions of the system point-spread-function). Then an image can be represented by the matrix equation

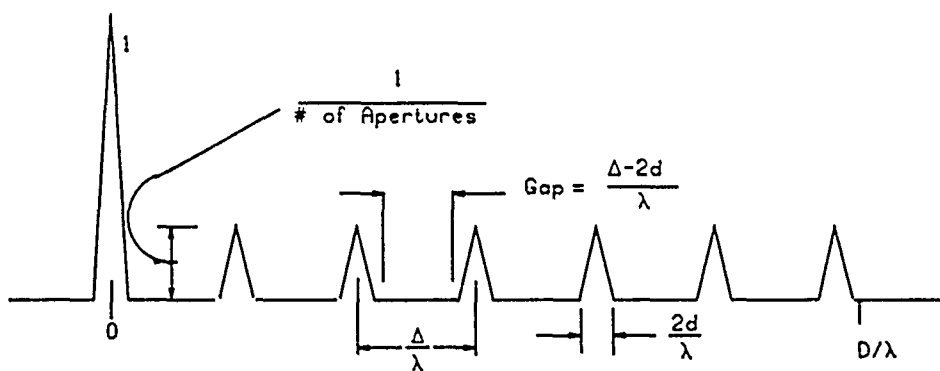
$$\mathbf{y} = B\mathbf{z} + \mathbf{n}, \quad (4.1)$$

where $\mathbf{n}$ is a $N \times 1$ noise vector. The product $B\mathbf{z}$ in Eq. (4.1) represents convolution of the object line with the point spread-function, followed by truncation of the image. Let the point spread function be $h(k)$. Then the B matrix is

$$B = \begin{bmatrix} h(-K) & h(-K-1) & \dots & h(-K-L+1) \\ h(-K+1) & h(-K) & \dots & h(-K-L+2) \\ \vdots & h(-K+1) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \dots & h(-K-2) \\ h(0) & \vdots & \dots & h(-K-1) \\ \vdots & h(0) & \dots & h(-K) \\ \vdots & \vdots & \dots & h(-K+1) \\ h(K-1) & \vdots & \dots & \vdots \\ h(K) & h(K-1) & \dots & \vdots \\ h(K+1) & h(K) & \dots & h(0) \\ h(K+2) & h(K+1) & \dots & \vdots \\ \vdots & h(K+2) & \dots & \vdots \\ \vdots & \vdots & \dots & h(K-1) \\ h(K+L-1) & h(K+L-2) & \dots & h(K) \end{bmatrix}, \quad (4.2)$$

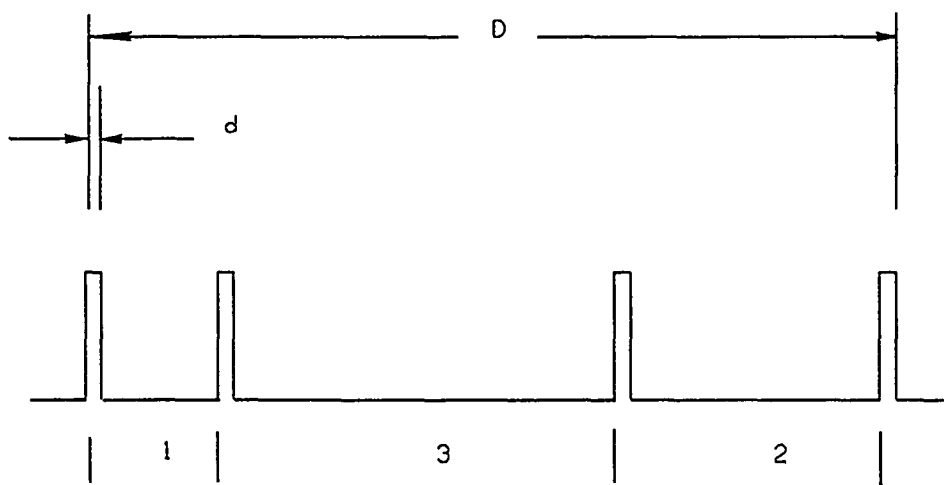


a)

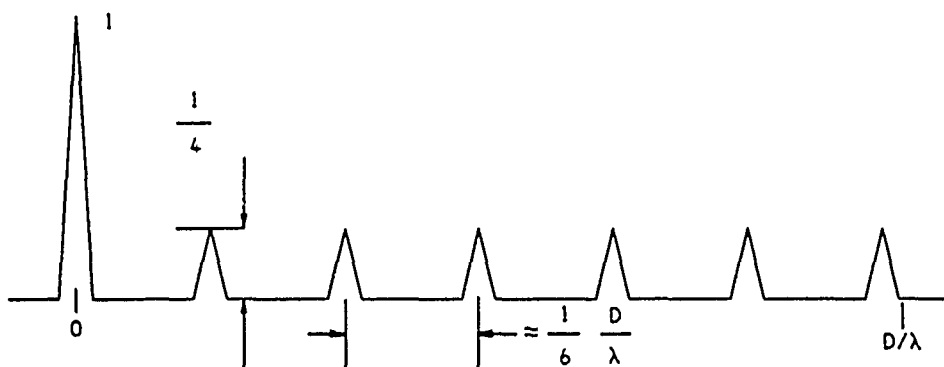


b)

Figure 4.1.
Typical nonredundant sparse array aperture function (a) and the corresponding MTF (b).

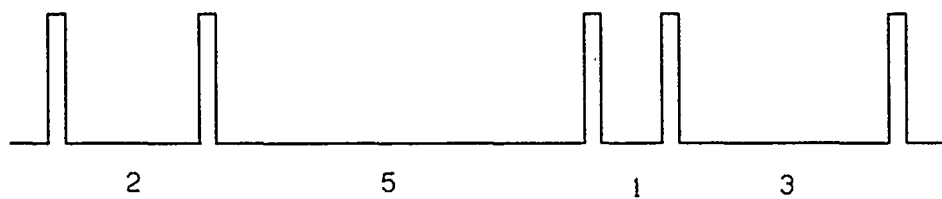


a)

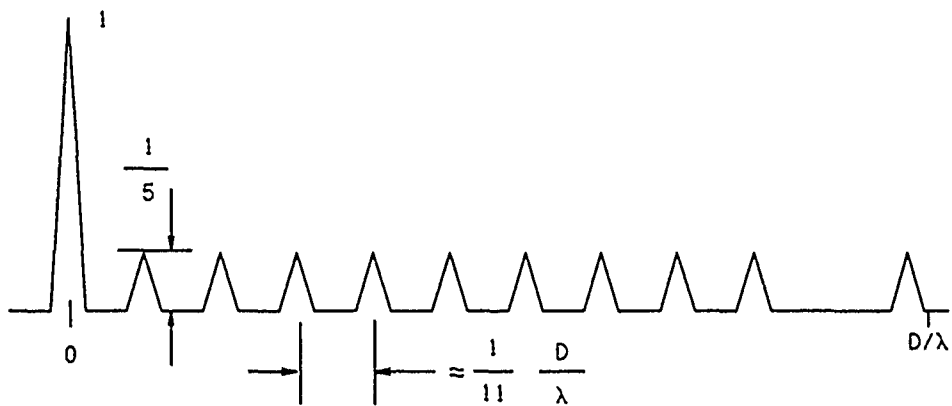


b)

Figure 4.2.
Sparse array aperture function (a) and the corresponding MTF (b) for a nonredundant four-subaperture array.

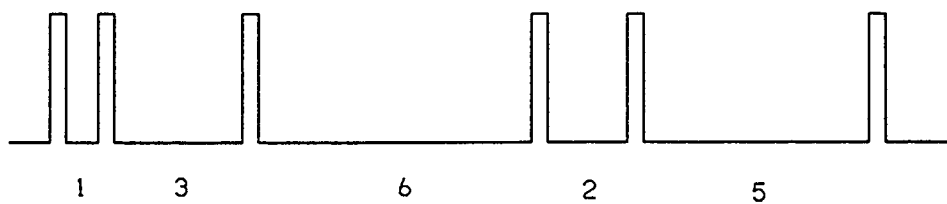


a)

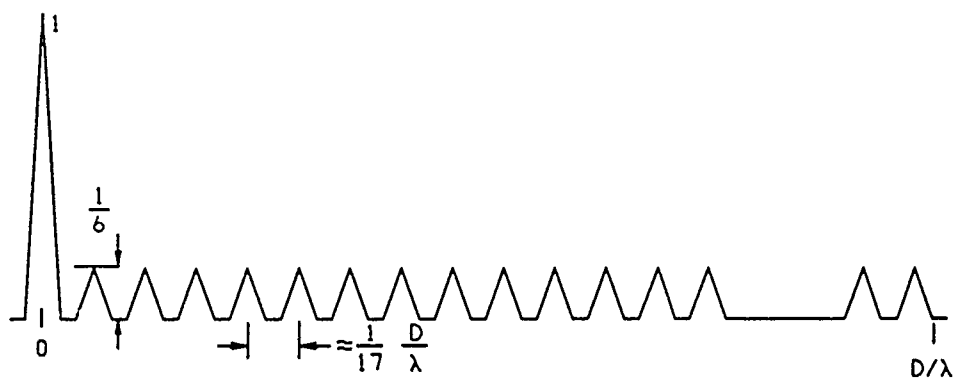


b)

Figure 4.3.
Sparse array aperture function (a) and the corresponding MTF (b) for a nonredundant five-subaperture array.



a)



b)

Figure 4.4.

Sparse array aperture function (a) and the corresponding MTF (b) for a nonredundant six-subaperture array.

where $K = (N - L)/2$.

Now let the $M \times 1$ vector $\mathbf{x}$ represent an object of finite contiguous support of length $M \leq L$ pixels located somewhere in the object line. We can write

$$\mathbf{z} = W\mathbf{x}, \quad (4.3)$$

where the $L \times M$ matrix W is of the form

$$W = \begin{bmatrix} & & & & 0 & & \\ & & & & & & \\ & 1 & & & & & 0 \\ & & 1 & & & & \\ & & & \ddots & & & \\ 0 & & & & & & 1 \\ & & & & 0 & & \\ & & & & & & \end{bmatrix}. \quad (4.4)$$

Thus, W comprises an $M \times M$ identity matrix embedded in a $L \times M$ matrix of zeros. Combining Eq.'s (4.3) and (4.1) yields

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}. \quad (4.5)$$

To complete the model, we need a point-spread function. Let $w(x)$ be an arbitrary aperture function. The corresponding modulation transfer function (MTF) is given below, where κ is the spatial frequency variable in units of cycles per radian and λ is wavelength. We write

$$MTF(\kappa) = \frac{1}{A} \int dx w(x + \frac{1}{2}\kappa\lambda)w(x - \frac{1}{2}\kappa\lambda) \quad (4.6)$$

A is chosen to be the area (in this case the length) of the aperture, so that the MTF at zero spatial frequency is unity.

The point-spread function of an optical system is the inverse Fourier transform of the MTF function, or

$$h_c(x) = \int d\kappa MTF(\kappa) \exp[2\pi i\kappa x]. \quad (4.7)$$

The subscript c denotes that this point-spread function is the continuous version. The discrete version of the point-spread function is found by sampling and scaling the continuous version. Let δ denote the pixel spacing in the image line. Then the discrete point spread function used in the B matrix given by Eq. (4.2) is

$$h(n) = \delta h_c(n\delta); \quad n = 0, \pm 1, \pm 2, \dots \quad (4.8)$$

The continuous point spread functions for the four, five, and six subaperture arrays are given in Eq.'s (9), (10), and (11) respectively. (The reader should see Fig.'s 4.2, 4.3, and 4.4 for the corresponding MTF's.) We have

$$h_c(x) = \frac{d}{\lambda} \left(\frac{\sin \pi \frac{d}{\lambda} x}{\pi \frac{d}{\lambda} x} \right)^2 \left(1 + \frac{1}{2} \sum_{i=1}^6 \cos \left[i \frac{\pi D}{3 \lambda} \left(1 - \frac{d}{D} \right) x \right] \right), \quad (4.9)$$

$$h_c(x) = \frac{d}{\lambda} \left(\frac{\sin \pi \frac{d}{\lambda} x}{\pi \frac{d}{\lambda} x} \right)^2 \left(1 + \frac{2}{5} \sum_{\substack{i=1 \\ i \neq 10}}^{11} \cos \left[i \frac{2\pi D}{11 \lambda} \left(1 - \frac{d}{D} \right) x \right] \right), \quad (4.10)$$

$$h_c(x) = \frac{d}{\lambda} \left(\frac{\sin \pi \frac{d}{\lambda} x}{\pi \frac{d}{\lambda} x} \right)^2 \left(1 + \frac{1}{3} \sum_{\substack{i=1 \\ i \neq 14 \\ i \neq 15}}^{17} \cos \left[i \frac{2\pi D}{17 \lambda} \left(1 - \frac{d}{D} \right) x \right] \right). \quad (4.11)$$

4.3 A Performance Measure

In the sections to follow we investigate two algorithms for object recovery. In order to evaluate the performance of each algorithm and to compare algorithms, we need a measure of performance that we can apply uniformly to both algorithms. Our object recovery algorithms not only attempt to fill in the gaps between measured spatial frequency components of the object (interpolative super-resolution), but also to estimate the object at spatial frequencies beyond D/λ (extrapolative super-resolution), where D is the overall length of the array. Because of array geometries, our algorithms will be performing better at recovering some spatial frequency components than others. We therefore need a frequency sensitive performance measure. To this end, let $x(n)$ be the n^{th} pixel of the object, $\hat{x}(n)$ be the n^{th} pixel of an estimate of the object, and $e(n)$ be the n^{th} pixel of the estimation error given by

$$e(n) = x(n) - \hat{x}(n), \quad (4.12a)$$

or in vector notation,

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}. \quad (4.12b)$$

One common measure of performance is mean-square-error, $\langle \mathbf{e}^T \mathbf{e} \rangle$. For our purposes this measure is not useful. It gives us no indication of how well an algorithm performs as a function of spatial frequency. Instead, the performance measure we will use is a signal-to-noise ratio (SNR) that is a function of a spatial cutoff frequency. The behavior of the SNR as a function of cutoff frequency will be an indication of the spatial resolution of the sparse array. To introduce frequency sensitivity into the performance measure, let $x_{\kappa_o}(n)$, $\hat{x}_{\kappa_o}(n)$, and $e_{\kappa_o}(n)$ be filtered versions of $x(n)$, $\hat{x}(n)$, and $e(n)$, respectively, where the filtering is ideal low-pass with cutoff frequency κ_o . To explicitly define the filtering operation, let $y(n)$ be a sequence of pixels and $\tilde{y}(\kappa)$ be its Fourier transform given by

$$\tilde{y}(\kappa) = \sum_n y(n) \exp(-2\pi i n \kappa \delta), \quad (4.13)$$

where δ is the object pixel spacing. Let $\tilde{f}(\kappa)$ be the filter transfer function given by

$$\tilde{f}(\kappa) = \begin{cases} 1, & |\kappa| \leq \kappa_o \\ 0, & |\kappa| > \kappa_o. \end{cases} \quad (4.14)$$

Then the Fourier transform of $y_{\kappa_o}(n)$, the filtered version of $y(n)$, is given by

$$\tilde{y}_{\kappa_o}(\kappa) = \tilde{y}(\kappa) \tilde{f}(\kappa). \quad (4.15)$$

Thus, we have

$$\tilde{x}_{\kappa_o}(\kappa) = \tilde{x}(\kappa) \tilde{f}(\kappa) \quad (4.16)$$

$$\tilde{\hat{x}}_{\kappa_o}(\kappa) = \tilde{\hat{x}}(\kappa) \tilde{f}(\kappa) \quad (4.17)$$

$$\begin{aligned} \tilde{e}_{\kappa_o}(\kappa) &= \tilde{e}(\kappa) \tilde{f}(\kappa) \\ &= [\tilde{x}(\kappa) - \tilde{\hat{x}}(\kappa)] \tilde{f}(\kappa) \\ &= \tilde{x}_{\kappa_o}(\kappa) - \tilde{\hat{x}}_{\kappa_o}(\kappa). \end{aligned} \quad (4.18)$$

Let us now define a signal-to-noise ratio for the filtered estimate $\hat{x}_{\kappa_o}(n)$ as the ratio of the average energy of the filtered object to the average energy of the filtered estimation error:

$$\text{SNR}_{\text{EST}}(\kappa_o) = \frac{\left\langle \sum_n x_{\kappa_o}^2(n) \right\rangle}{\left\langle \sum_n e_{\kappa_o}^2(n) \right\rangle}. \quad (4.19)$$

The angle brackets in Eq. (4.19) denote ensemble average. From Parseval's theorem we know that the quantities inside the angle brackets in Eq. (4.19) can be written in terms of their Fourier transforms, as follows:

$$\sum_n x_{\kappa_o}^2(n) = \delta \int_{-1/2\delta}^{1/2\delta} d\kappa |\tilde{x}_{\kappa_o}(\kappa)|^2 \quad (4.20)$$

$$\sum_n e_{\kappa_o}^2(n) = \delta \int_{-1/2\delta}^{1/2\delta} d\kappa |\tilde{e}_{\kappa_o}(\kappa)|^2. \quad (4.21)$$

We can use Eq.'s (4.13), (4.14), (4.16), and (4.18) to rewrite Eq.'s (4.20) and (4.21), yielding

$$\begin{aligned} \sum_n x_{\kappa_o}^2(n) &= \delta \int_{-1/2\delta}^{1/2\delta} d\kappa |\tilde{x}(\kappa)|^2 |\tilde{f}(\kappa)|^2 \\ &= 2\delta \int_0^{\kappa_o} d\kappa |\tilde{x}(\kappa)|^2 \end{aligned} \quad (4.22)$$

$$\begin{aligned} \sum_n e_{\kappa_o}^2(n) &= \delta \int_{-1/2\delta}^{1/2\delta} d\kappa |\tilde{e}(\kappa)|^2 |\tilde{f}(\kappa)|^2 \\ &= 2\delta \int_0^{\kappa_o} d\kappa |\tilde{e}(\kappa)|^2. \end{aligned} \quad (4.23)$$

Replacing the quantities inside the angle-brackets in Eq. (4.19) with their corresponding expressions given by Eq.'s (4.22) and (4.23), and bringing the angle-brackets inside the integrals, yields

$$\text{SNR}_{\text{EST}}(\kappa_o) = \frac{\int_0^{\kappa_o} d\kappa \langle |\tilde{x}(\kappa)|^2 \rangle}{\int_0^{\kappa_o} d\kappa \langle |\tilde{e}(\kappa)|^2 \rangle}. \quad (4.24)$$

Since integrating the quantities $\delta \langle |\tilde{x}(\kappa)|^2 \rangle$ and $\delta \langle |\tilde{e}(\kappa)|^2 \rangle$ over the spatial frequency band $|\kappa| \leq \kappa_o$ yields, respectively, the energy in the object and estimation error over the same band, we will call these quantities energy spectral densities or energy spectra, and use the notation

$$E_x(\kappa) = \delta \langle |\tilde{x}(\kappa)|^2 \rangle \quad (4.25)$$

$$E_e(\kappa) = \delta \langle |\tilde{e}(\kappa)|^2 \rangle. \quad (4.26)$$

So that

$$\text{SNR}_{\text{EST}}(\kappa_o) = \frac{\int_0^{\kappa_o} d\kappa E_x(\kappa)}{\int_0^{\kappa_o} d\kappa E_e(\kappa)}. \quad (4.27)$$

To compute the energy spectrum of a random sequence, we simply follow the prescription given by Eq.'s (4.25) and (4.13). Let $\mathbf{z}$ be a random vector with n^{th} component $z(n)$. Then its energy spectrum is given by

$$\begin{aligned} E_z(\kappa) &= \delta \langle |\tilde{z}(\kappa)|^2 \rangle \\ &= \delta \left\langle \left| \sum_n z(n) \exp(-2\pi i n \kappa \delta) \right|^2 \right\rangle \\ &= \delta \sum_n \sum_m \langle z(n) z(m) \rangle \exp[-2\pi i (n - m) \kappa \delta]. \end{aligned} \quad (4.28)$$

We note from Eq. (4.28) that we need *all* entries of the covariance matrix of $z(n)$ in the evaluation of its energy spectrum.

4.4. Minimum-Variance Processor

The first processor (estimator) we will investigate is the so-called minimum variance processor. We assume that we have first and second moment information on the object vector $\mathbf{x}$ and the noise vector $\mathbf{n}$ of Eq. (4.5), i.e., we know the mean vector and covariance matrices of $\mathbf{x}$ and $\mathbf{n}$. Since we know the mean values of $\mathbf{x}$ and $\mathbf{n}$, and we can compute the mean value of $\mathbf{y}$, we will assume that the mean values have been subtracted out of Eq. (4.5), and our estimate $\hat{\mathbf{x}}$ is the deviation of $\mathbf{x}$ from its mean value. In other words, we have the observation model

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}, \quad (4.29)$$

where all vectors in Eq. (4.29) have zero mean. We will use as an estimator a linear transformation of the observation vector:

$$\hat{\mathbf{x}} = H\mathbf{y}. \quad (4.30)$$

The error vector is

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - \hat{\mathbf{x}} \\ &= \mathbf{x} - H\mathbf{y}. \end{aligned} \quad (4.31)$$

We wish to choose H so as to minimize the variance of the error. Since $\mathbf{e}$ is zero-mean, this is equivalent to minimizing the mean-square-error, given by

$$\begin{aligned} \xi &= \langle \mathbf{e}^T \mathbf{e} \rangle \\ &= \text{Tr} \langle \mathbf{e} \mathbf{e}^T \rangle. \end{aligned} \quad (4.32)$$

Let e_n and x_n be the n^{th} components of the vectors $\mathbf{e}$ and $\mathbf{x}$, and let $\mathbf{h}_n$ be the n^{th} row of H . Then we can write:

$$e_n = x_n - \mathbf{y}^T \mathbf{h}_n. \quad (4.33)$$

We can see that choosing $\mathbf{h}_n$ to minimize $\langle e_n^2 \rangle$, for every n , minimizes ξ . To minimize $\langle e_n^2 \rangle$, we invoke the orthogonality principle (also known as the projection theorem). We choose $\mathbf{h}_n$ so that each component of the error is orthogonal to all components of the observation:

$$\langle e_n \mathbf{y} \rangle = 0. \quad (4.34)$$

Using Eq. (4.33) in Eq. (4.34) yields the equation

$$\langle x_n \mathbf{y} \rangle = \langle \mathbf{y} \mathbf{y}^T \rangle \mathbf{h}_n. \quad (4.35)$$

Solving for $\mathbf{h}_n$ we get

$$\mathbf{h}_n = \langle \mathbf{y} \mathbf{y}^T \rangle^{-1} \langle x_n \mathbf{y} \rangle. \quad (4.36)$$

Thus, the optimum H matrix, H_o , has a value given by the expression

$$H_o = \langle \mathbf{x} \mathbf{y}^T \rangle \langle \mathbf{y} \mathbf{y}^T \rangle^{-1}. \quad (4.37)$$

The covariance matrices in Eq. (4.37) can be computed using Eq. (4.29), yielding

$$\langle \mathbf{x} \mathbf{y}^T \rangle = \langle \mathbf{x} \mathbf{x}^T \rangle (BW)^T, \quad (4.38)$$

$$\langle \mathbf{y} \mathbf{y}^T \rangle = (BW) \langle \mathbf{x} \mathbf{x}^T \rangle (BW)^T + \langle \mathbf{n} \mathbf{n}^T \rangle. \quad (4.39)$$

In computing Eq.'s (4.38) and (4.39), it was assumed that the object and noise vectors are uncorrelated. Using Eq.'s (4.38) and (4.39) in Eq. (4.37) yields

$$\begin{aligned} H_o &= (\mathbf{x}\mathbf{x}^T)(\mathbf{B}\mathbf{W})^T [(\mathbf{B}\mathbf{W})(\mathbf{x}\mathbf{x}^T)(\mathbf{B}\mathbf{W})^T + \langle \mathbf{n}\mathbf{n}^T \rangle]^{-1} \\ &= R_{xx} G^T [G R_{xx} G^T + R_{nn}]^{-1}, \end{aligned} \quad (4.40)$$

where

$$R_{xx} = \langle \mathbf{x}\mathbf{x}^T \rangle, \quad (4.41)$$

$$R_{nn} = \langle \mathbf{n}\mathbf{n}^T \rangle, \quad (4.42)$$

$$G = \mathbf{B}\mathbf{W}. \quad (4.43)$$

One can apply a matrix inversion lemma to Eq. (4.40) to show that

$$H_o = (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1} G^T R_{nn}^{-1}. \quad (4.44)$$

In order to compute the energy spectrum of the error, we need the error covariance matrix. Using Eq. (4.31) we can write

$$\langle \mathbf{e}\mathbf{e}^T \rangle = \langle \mathbf{e}\mathbf{x}^T \rangle - \langle \mathbf{e}\mathbf{y}^T \rangle H_o^T. \quad (4.45)$$

Since we have chosen H_o so that each component of the error vector is orthogonal to the observation vector $\mathbf{y}$, the second term in Eq. (4.45) must be zero. Therefore, we have

$$\begin{aligned} \langle \mathbf{e}\mathbf{e}^T \rangle &= \langle \mathbf{e}\mathbf{x}^T \rangle \\ &= \langle \mathbf{x}\mathbf{x}^T \rangle - H_o \langle \mathbf{y}\mathbf{x}^T \rangle. \end{aligned} \quad (4.46)$$

Using Eq.'s (4.38) and (4.41) - (4.44) in Eq. (4.46) yields

$$\begin{aligned} \langle \mathbf{e}\mathbf{e}^T \rangle &= R_{xx} - (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1} G^T R_{nn}^{-1} G R_{xx} \\ &= (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1} [(G^T R_{nn}^{-1} G + R_{xx}^{-1}) R_{xx} - G^T R_{nn}^{-1} G R_{xx}] \\ &= (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1}. \end{aligned} \quad (4.47)$$

To keep things simple we will assume that both the object and noise vectors are white, i.e.,

$$R_{xx} = \sigma_x^2 I, \quad (4.48)$$

$$R_{nn} = \sigma_n^2 I, \quad (4.49)$$

where σ_x^2 and σ_n^2 are the variances of object and noise pixels, respectively. If we define a signal-to-noise ratio as

$$\text{SNR}_{\text{REF}} = \sigma_x^2 / \sigma_n^2, \quad (4.50)$$

then Eq. (4.47) can be written

$$\langle \mathbf{e}\mathbf{e}^T \rangle = (G^T G (\text{SNR}_{\text{REF}}) + I)^{-1} \sigma_x^2. \quad (4.51)$$

The numerator of the SNR_{REF} defined by Eq. (4.50) is referenced to the object plane and the denominator is referenced to the image plane. Can we relate this to a similarly-defined SNR totally referenced to the image plane? In general, the answer is no. Such an SNR would vary from pixel to pixel. Rather than attempting to develop a suitable definition for SNR in the image plane, we will instead simply make the following observation. Given a very large object of uniform intensity, the intensity of the image is also uniform and is equal to the intensity of the object scaled by the

MTF evaluated at zero spatial frequency, which is unity. Thus, in the case of a very large object of uniform intensity, the SNR of an image pixel is SNR_{REF} .

We are now in a position to make some computations using the minimum-variance processor and point-spread functions corresponding to the four-subaperture, five-subaperture, and six-subaperture arrays shown in Fig.'s 4.2 – 4.4, respectively. In all cases we will use a pixel spacing δ of

$$\delta = \frac{1}{4} \frac{\lambda}{D}. \quad (4.52)$$

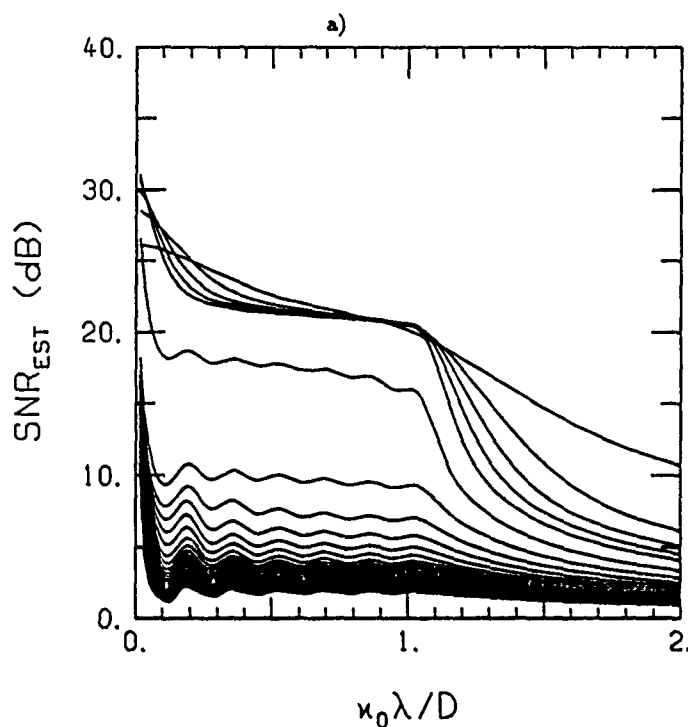
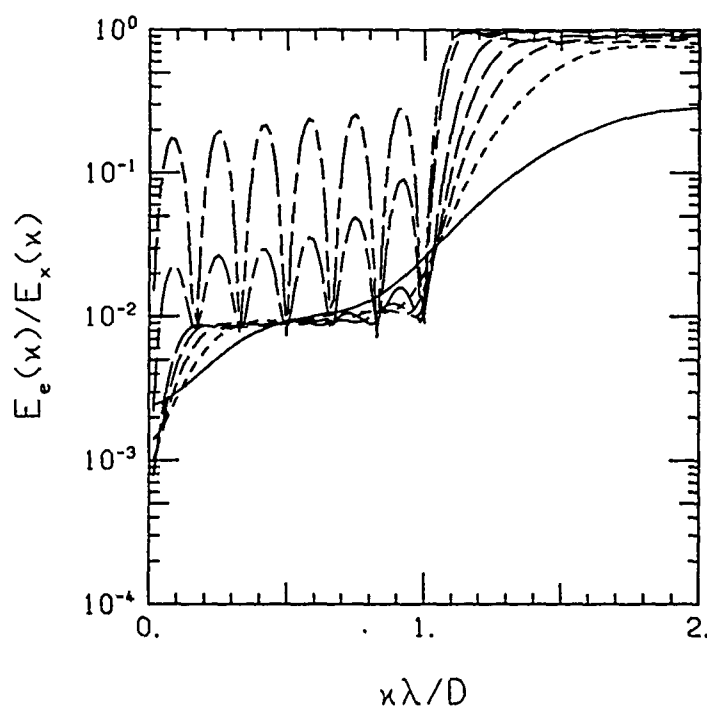
This pixel spacing corresponds to a sampling frequency of $4D/\lambda$ or a Nyquist frequency of $2D/\lambda$. This choice will allow the study of extrapolation superresolution out to a spatial frequency of $2D/\lambda$, twice the diffraction limit of an aperture of length D . The energy spectrum of the object is computed using Eq. (4.48) in Eq. (4.28).

$$E_x(\kappa) = M\delta\sigma_x^2, \quad (4.53)$$

where M is the number of pixels in the object (length of $\mathbf{x}$), δ is the pixels spacing, and σ_x^2 is the variance of an individual object pixel. Note that $M\delta$ is the support of the object. We compute the energy spectrum of the error vector using Eq. (4.51) in Eq. (4.28):

$$E_e(\kappa) = \delta\sigma_x^2 \sum_m \sum_n \left[(G^T G (\text{SNR}_{\text{REF}}) + I)^{-1} \right]_{mn}. \quad (4.54)$$

Fig. 4.5a is a plot of the ratio $E_e(\kappa)/E_x(\kappa)$ versus spatial frequency and Fig. 4.5b is a plot of SNR_{EST} as a function of filter cutoff frequency κ_o using the four-subaperture array of Fig. 4.2 with $d/D = 0.005$, and $\text{SNR}_{\text{REF}} = 50\text{dB}$ ($10 \log_{10}(\sigma_x^2/\sigma_n^2)$). There are seven curves in Fig. 4.5a corresponding to object supports of $1\lambda/D$, $2\lambda/D$, $\dots$, $7\lambda/D$, with larger support corresponding to larger values of the ratio $E_e(\kappa)/E_x(\kappa)$. Fig. 4.5b contains 32 curves for object supports of $1\lambda/D$, $2\lambda/D$, $\dots$, $32\lambda/D$. There are several interesting observations that can be made concerning these figures. First, $E_e(\kappa) \leq E_x(\kappa)$ for all spatial frequencies κ . The reason for this is that when the signal at a particular spatial frequency is very small compared to the noise, the minimum-variance estimator "turns off", i.e., $\hat{\mathbf{x}}$ goes to zero, and the error $\mathbf{e}$ converges to the object $\mathbf{x}$. Thus, a value of the ratio close to unity is "bad", and a value very much less than unity is "good". The second point to note is the rapid increase in $E_e(\kappa)/E_x(\kappa)$ and rapid fall-off of SNR_{EST} for $\kappa > D/\lambda$, i.e., extrapolative super-resolution doesn't work (a not unexpected result). The third point to note, and the most important, is the near uniformity of interpolative super-resolution for object support $\leq 5\lambda/D$. Referring to Fig. 4.2, we see that the minimum-variance algorithm is receiving direct measurements of the object spectrum at only very narrow regions centered about 0 , $\pm 1/6D/\lambda$, $\pm 2/6D/\lambda$, $\dots$, D/λ , but Fig. 4.5a tells us that the object estimates for support $\leq 5\lambda/D$ contain approximately the same quality of spectral information throughout the band below D/λ as the direct measurements. Thus, the processor is "filling in the gaps" with apparent ease. This behavior manifests itself in Fig. 4.5b also, where we see that the first five curves group together at a near uniform SNR_{EST} of about 22dB out to $\kappa_o = D/\lambda$. When object support exceeds $5\lambda/D$, we note from Fig. 4.5a the processors inability to "fill in the gaps" between measurements and, from Fig. 4.5b, the rapid drop in SNR_{EST} over the range $\kappa_o \leq D/\lambda$ with increasing object support. We can illustrate this behavior rather dramatically by taking a vertical slice through Fig. 4.5b at $\kappa_o = D/\lambda$, and plotting SNR_{EST} versus object support. This result is shown in Fig. 4.6. The curve speaks for itself. We note only that the "cliff" of Fig. 4.6 is located at a support level approximately equal to the reciprocal of the spacing between "islands" in the MTF of Fig. 4.2. The question arises: have we established a rule? That is, interpolative super-resolution "works" as long as object support is less than the reciprocal of the spacing between nonzero "islands" in the array MTF. By examining other cases, we will see that this is indeed the rule. Furthermore, we will see that even if the rule is violated by only a small amount, severe performance penalties are the result.



b)

Figure 4.5.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_0 (b) using the minimum-variance processor and the four-subaperture array with $\text{SNR}_{\text{REF}} = 50\text{dB}$ and $d/D = 0.005$. The seven curves of (a) correspond to object supports of $1\lambda/D, 2\lambda/D, \dots, 7\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D, 2\lambda/D, \dots, 32\lambda/D$, with larger SNR_{EST} for smaller support values.

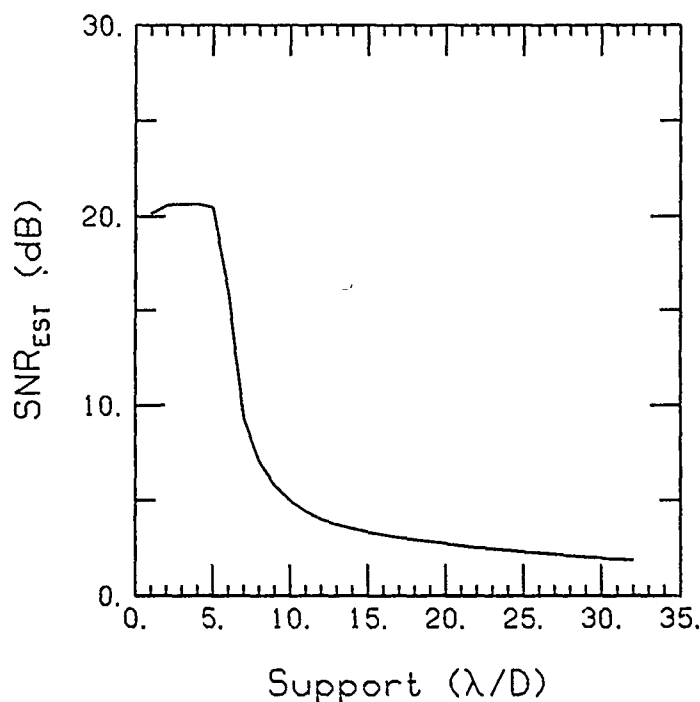


Figure 4.6.

Signal-to-noise ratio of the estimate versus object support using the minimum-variance processor and the four-subaperture array, with $\text{SNR}_{\text{REF}} = 50\text{dB}$, $d/D = 0.005$, and $\kappa_o = 1D/\lambda$.

Performance plots for the minimum-variance processor and the five-subaperture array of Fig. 4.3 are shown in Fig. 4.7. Again $d/D = 0.005$ and $\text{SNR}_{\text{REF}} = 50\text{dB}$. There are again seven curves in Fig. 4.7(a), this time corresponding to object supports of $2\lambda/D$, $4\lambda/D$, ..., $14\lambda/D$, with larger support corresponding to larger values of $E_e(\kappa)/E_x(\kappa)$. Fig. 4.7(b) again contains 32 curves corresponding to object supports of $1\lambda/D$, $2\lambda/D$, ..., $32\lambda/D$. Again, note the near uniformity of interpolative super-resolution for the first five curves of Fig. 4.7(a), this time over a spatial frequency range $|\kappa| \leq 0.81D/\lambda$, the "useful" range of the five-subaperture array (see Fig. 4.3(b)). In this case, the first five curves correspond to object support $\leq 10\lambda/D$.

Performance plots for the six-subaperture array of Fig. 4.4 are shown in Fig. 4.8. The results are so similar to those of the four and five subaperture cases that we will not further elaborate, other than to point out that the "useful" range of spatial frequencies for good interpolative super-resolution performance for objects of support $\leq 15\lambda/D$ is $|\kappa| \leq 0.76D/\lambda$, the same as the "useful" range for the six-subaperture array.

If we take vertical slices through the plots of Fig.'s 4.7(b) and 4.8(b) at the upper limit of the "useful" frequency range of their corresponding arrays, $\kappa_o = 0.81D/\lambda$ and $\kappa_o = 0.76D/\lambda$, respectively, we have the curves of Fig.'s 4.9(b) and 4.9(c). For convenience, Fig. 4.6, the corresponding curve for the four-subaperture case, is reproduced in Fig. 4.9(a). In each case, the precipitous drop in performance occurs at the point where object support exceeds the reciprocal of the spacing between "islands" in the MTF for the corresponding array. One might argue that this result is "obvious" from the sampling theorem. That is, the sparse array gives us uniformly spaced samples of the Fourier transform of the object intensity distribution throughout the "useful" range of the array. Thus, it should be possible to reconstruct the object intensity distribution from these samples out to a resolution limit equal to the highest spatial frequency of the "useful" range, so long as the support of the object is less than the reciprocal of the sample spacing. However, this argument, although a useful viewpoint in developing an intuitive understanding of the results, gives no indication of how

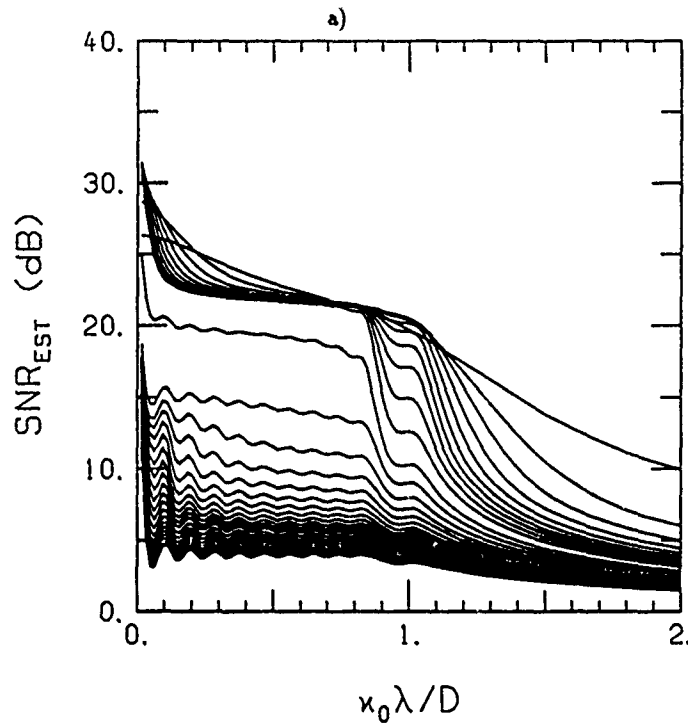
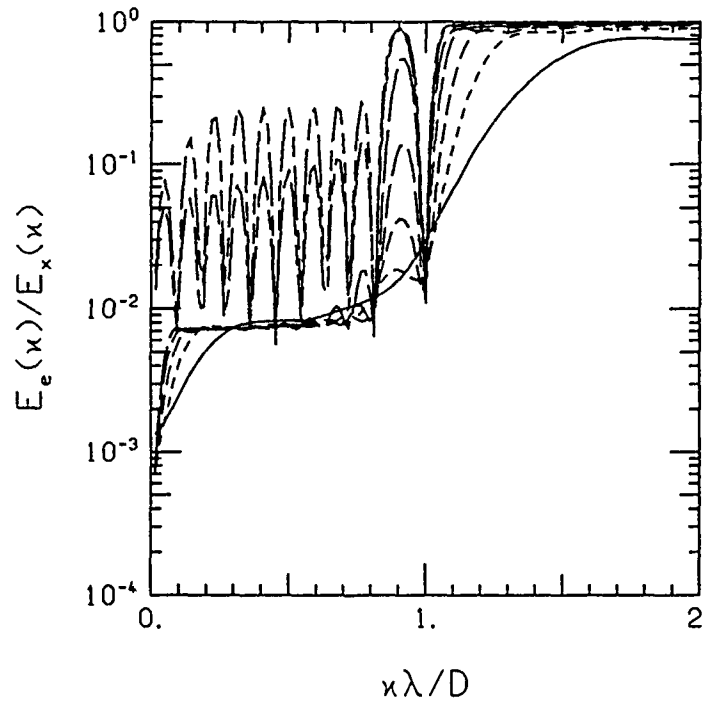


Figure 4.7.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_0 (b) using the minimum-variance processor and the five-subaperture array with $\text{SNR}_{\text{REF}} = 50\text{dB}$ and $d/D = 0.005$. The seven curves of (a) correspond to object supports of $2\lambda/D, 4\lambda/D, \dots, 14\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D, 2\lambda/D, \dots, 32\lambda/D$, with larger SNR_{EST} for smaller support values.

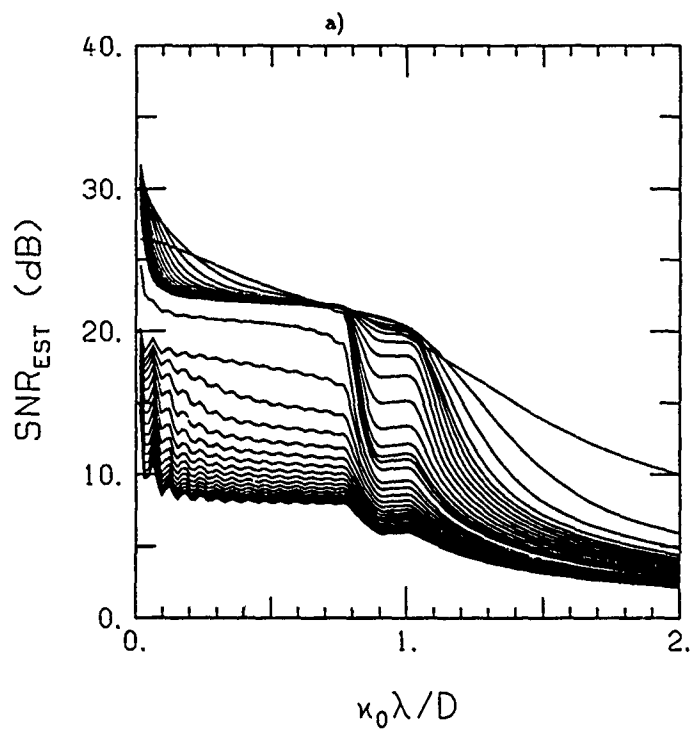
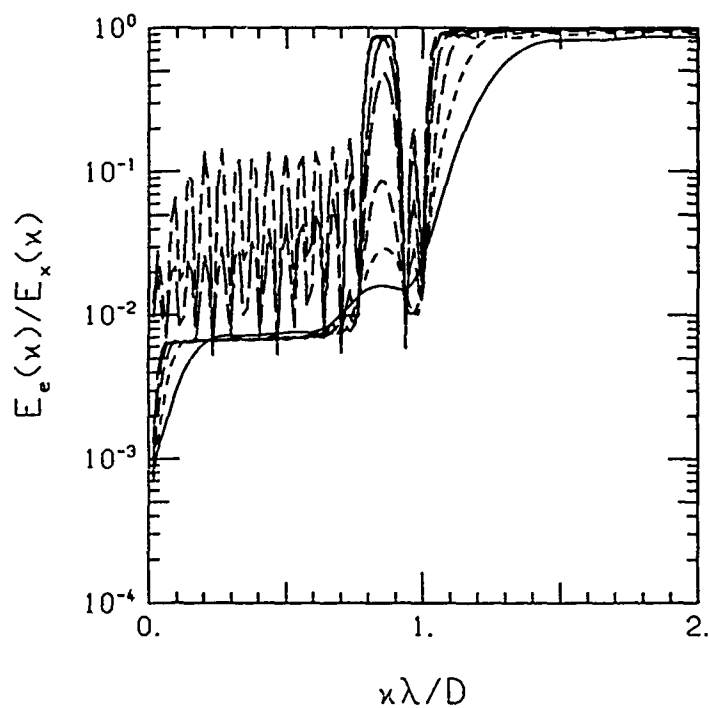
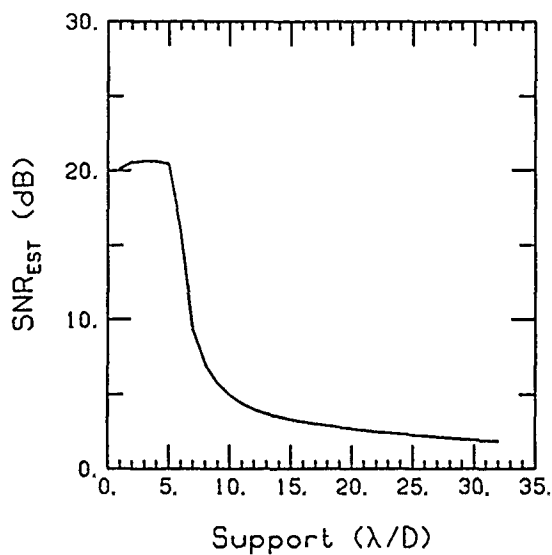
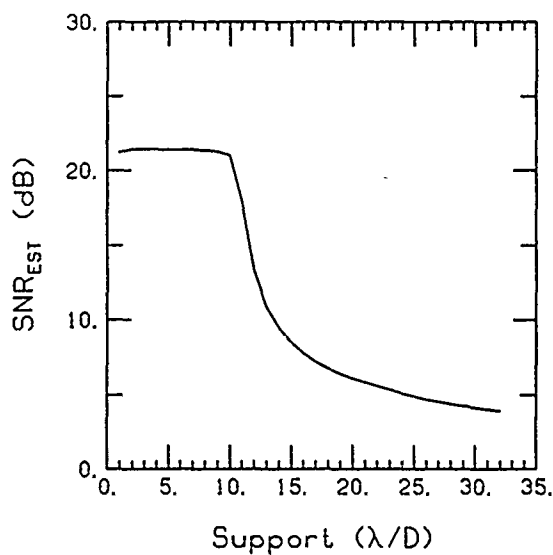


Figure 4.8.

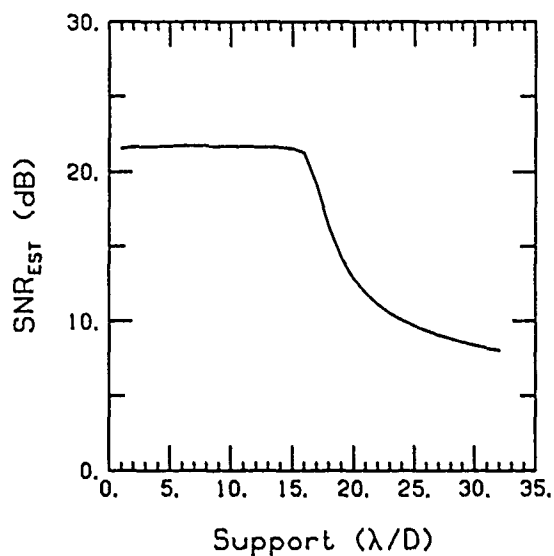
Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_0 (b) using the minimum-variance processor and the six-subaperture array with $\text{SNR}_{\text{REF}} = 50\text{dB}$ and $d/D = 0.005$. The seven curves of (a) correspond to object supports of $3\lambda/D, 6\lambda/D, \dots, 21\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D, 2\lambda/D, \dots, 32\lambda/D$, with larger SNR_{EST} for smaller support values.



a)



b)



c)

Figure 4.9.

Signal-to-noise ratio of the estimate versus object support using the minimum-variance processor, $\text{SNR}_{\text{REF}} = 50\text{dB}$, and $d/D = 0.005$. Fig.'s (a), (b), and (c) correspond respectively, to the four-subaperture array with $\kappa_o = 1D/\lambda$, the five-subaperture array with $\kappa_o = 0.81D/\lambda$, and the six-subaperture array with $\kappa_o = 0.76D/\lambda$.

severe the penalty is for violating the support constraint nor does it give any indication of how much SNR loss will occur within the "useful" spatial frequency range of the array.

4.5. Unweighted Least-Squares Processor

In computing the results of Section 4.4, we assumed knowledge of the mean and covariance of the object and noise vectors and a finite-support constraint on the object. A probably more realistic situation is one where we are presented with an image and given the MTF of the optical system, and we are required to find the object which best explains the image in some sense, given no statistical knowledge. In this section we will use a least-squares criterion for deciding which object "best" explains the image, i.e., we pick the object vector estimate $\hat{\mathbf{x}}$ which minimizes

$$\epsilon = \|\mathbf{y} - G\hat{\mathbf{x}}\|^2. \quad (4.55)$$

Here we use the same one-dimensional optical model as that described in Section 4.2, with $\mathbf{y}$ the vector of image pixels, $\mathbf{x}$ the vector of object pixels, and $G = BW$ a matrix containing the system point-spread function and the object support information. Carrying out the operation indicated in Eq. (4.55) yields

$$\begin{aligned} \epsilon &= (\mathbf{y} - G\hat{\mathbf{x}})^T (\mathbf{y} - G\hat{\mathbf{x}}) \\ &= \mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T G\hat{\mathbf{x}} + \hat{\mathbf{x}}^T G^T G\hat{\mathbf{x}}. \end{aligned} \quad (4.56)$$

The gradient of ϵ with respect to $\hat{\mathbf{x}}$ is

$$\nabla \epsilon = -2G^T \mathbf{y} + 2G^T G\hat{\mathbf{x}}, \quad (4.57)$$

where here the gradient is taken to be a column vector. Setting $\Delta \epsilon$ equal to zero and solving for $\hat{\mathbf{x}}$ yields

$$\hat{\mathbf{x}} = (G^T G)^{-1} G^T \mathbf{y}. \quad (4.58)$$

Since G has full column rank, $G^T G$ is nonsingular and its inverse is well defined. The error vector with this estimate is

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - \hat{\mathbf{x}} \\ &= \mathbf{x} - (G^T G)^{-1} G^T \mathbf{y}. \end{aligned} \quad (4.59)$$

Using $\mathbf{y}$ from Eq. (4.5) in Eq. (4.59), with $G = BW$, yields

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - (G^T G)^{-1} G^T (G\mathbf{x} + \mathbf{n}) \\ &= (G^T G)^{-1} G^T \mathbf{n}. \end{aligned} \quad (4.60)$$

The covariance matrix of the error vector is

$$\begin{aligned} \langle \mathbf{e} \mathbf{e}^T \rangle &= (G^T G)^{-1} G^T \langle \mathbf{n} \mathbf{n}^T \rangle G (G^T G)^{-1} \\ &= (G^T G)^{-1} G^T R_{nn} G (G^T G)^{-1}. \end{aligned} \quad (4.61)$$

As in the minimum-variance case, we will assume that the object and noise vectors are white, i.e.,

$$R_{xx} = \sigma_x^2 I, \quad (4.62)$$

$$R_{nn} = \sigma_n^2 I, \quad (4.63)$$

and

$$\text{SNR}_{\text{REF}} = \sigma_x^2 / \sigma_n^2. \quad (4.64)$$

Then

$$\begin{aligned} \langle ee^T \rangle &= (G^T G)^{-1} \sigma_n^2 \\ &= (G^T G (\text{SNR}_{\text{REF}}))^{-1} \sigma_x^2 \end{aligned} \quad (4.65)$$

We write the error covariance matrix in this way to emphasize its similarity to the error covariance matrix for the minimum-variance processor (see Eq. (4.51)). In fact, we see that as SNR_{REF} increases the two matrices become one. In other words, for large enough SNR_{REF} , the performance of unweighted least-squares approaches that of minimum-variance.

We compute the energy spectrum of the error vector using Eq. (4.65) in Eq. (4.28):

$$E_e(\kappa) = \delta \sigma_x^2 \sum_m \sum_n [(G^T G (\text{SNR}_{\text{REF}}))^{-1}]_{mn}. \quad (4.66)$$

The energy spectrum of the object is given by Eq. (4.53). The pixel spacing we will use is

$$\delta = \frac{1}{2} \lambda / D, \quad (4.67)$$

yielding a Nyquist rate of D/λ . There are two reasons for changing the pixel spacing from $\frac{1}{4} \lambda / D$ used in the minimum-variance processor. First, we are no longer interested in studying extrapolative super-resolution performance and thus a Nyquist rate of D/λ is adequate. Second, the inclusion of an extrapolative super-resolution region creates a very large eigenvalue spread in $G^T G$, making it very difficult to numerically evaluate $E_e(\kappa)$ with the numerical precision at our disposal. This was not a problem with the minimum-variance processor because of the stabilizing influence of the identity matrix in Eq. (4.54).

Except for pixel spacing, all of the results to follow were computed using the same parameters as with the minimum-variance processor, i.e., $d/D = 0.005$ and $\text{SNR}_{\text{REF}} = 50 \text{ dB}$. $E_e(x)/E_x(\kappa)$ and $\text{SNR}_{\text{EST}}(\kappa_o)$ are shown in Fig.'s 4.10, 4.11, and 4.12 using the four, five, and six-subaperture array, respectively. Corresponding plots of SNR_{EST} versus object support at the appropriate value of κ_o are shown in Fig. 4.13. Here we see similar features to that of the minimum-variance processor: near uniform interpolative super-resolution with object support less than the reciprocal of the "island" spacing in the MTF and a very rapid drop in performance when object support exceeds this value. Note that the least-squares processor is not "smart" enough to "turn-off" when the object estimate gets to noisy, thus allowing SNR_{EST} to drop below zero dB. For purposes of comparison, Fig. 4.14 shows the SNR_{EST} versus support curves for minimum-variance and least-squares superimposed.

4.6. Discussion

In presenting the minimum-variance and least-squares results, we have ignored certain subtleties for the sake of clarity. For example, the number of columns in the $G = BW$ matrix is determined by the support of the object. How was the number of rows determined? Since the MTF in our optical model is of finite extent, the point-spread function must be infinite, and therefore so is the image of the object. In any practical imaging system, one obviously must truncate the image. In our model, the number of pixels in the truncated image is determined by the number of rows of G . In general, the larger the number of rows of G , the better is the object reconstruction. However, a point of diminishing returns is reached whereby increasing the number of rows of G does not significantly increase performance. At present, we have no theory to predict when this point is reached. However, we determined empirically that an image line of length $256 \lambda / D$ was essentially equivalent to an infinite image line for all the cases we have presented in this report, and that is the length that was used for all cases.

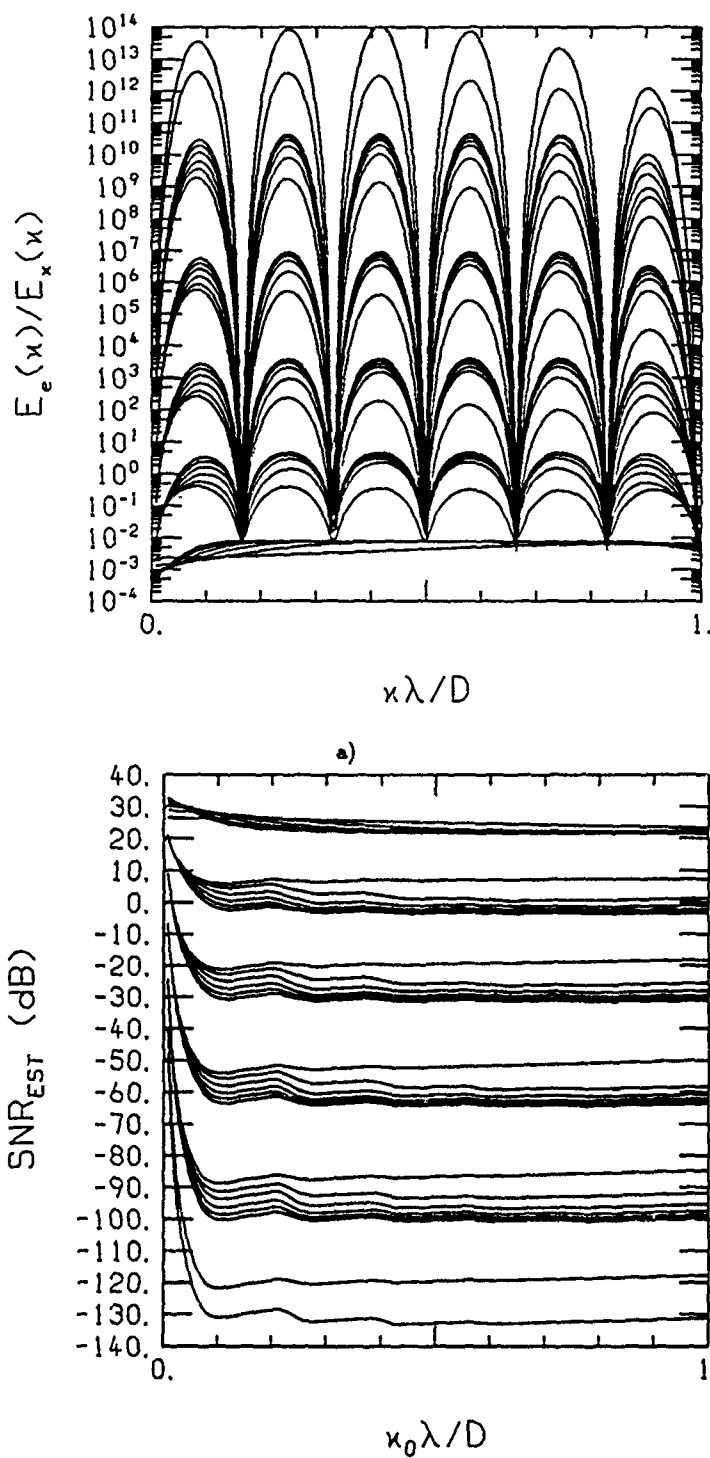


Figure 4.10.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_0 (b) using the unweighted least-squares processor and the four-subaperture array with $\text{SNR}_{\text{REF}} = 50\text{dB}$ and $d/D = 0.005$. The 32 curves of (a) correspond to object supports of $1\lambda/D, 2\lambda/D, \dots, 32\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D, 2\lambda/D, \dots, 32\lambda/D$, with larger SNR_{EST} for smaller support values.

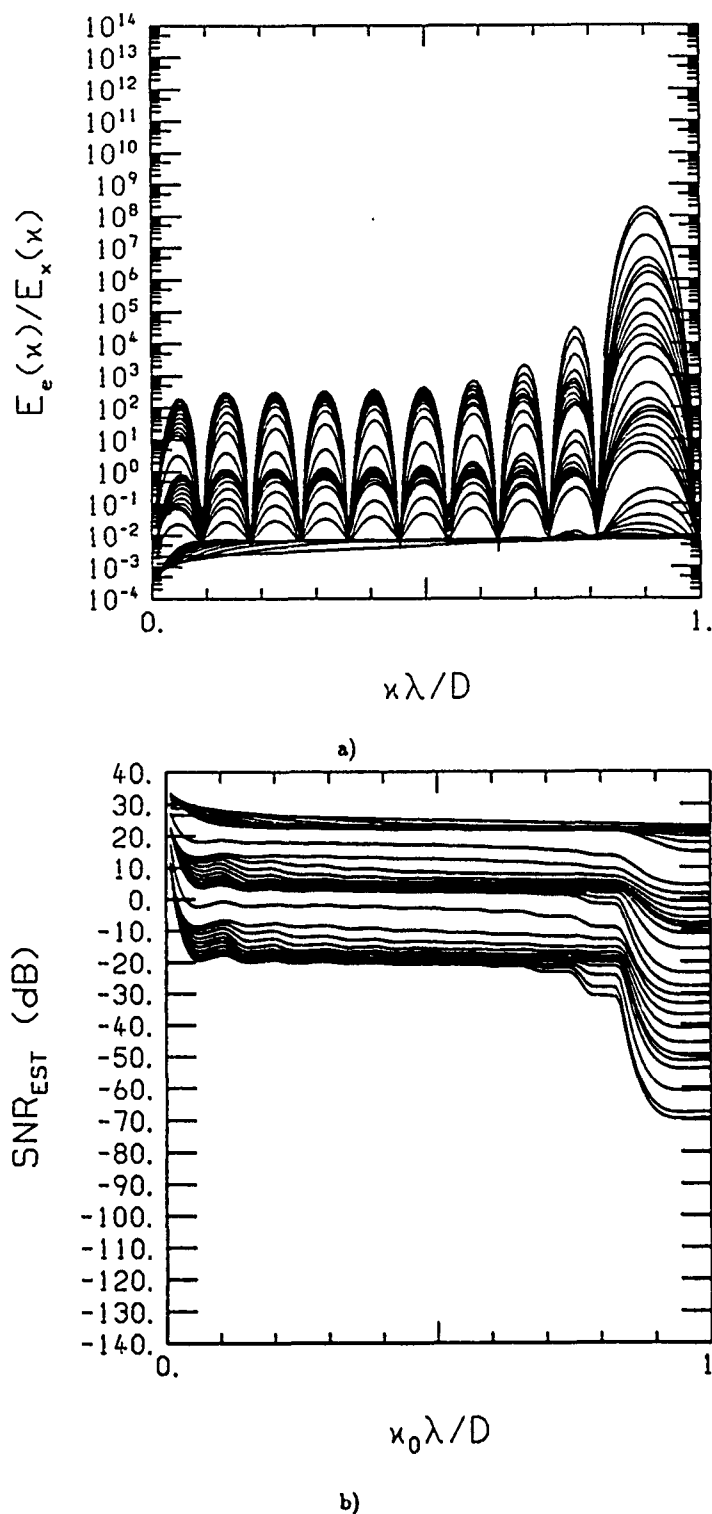
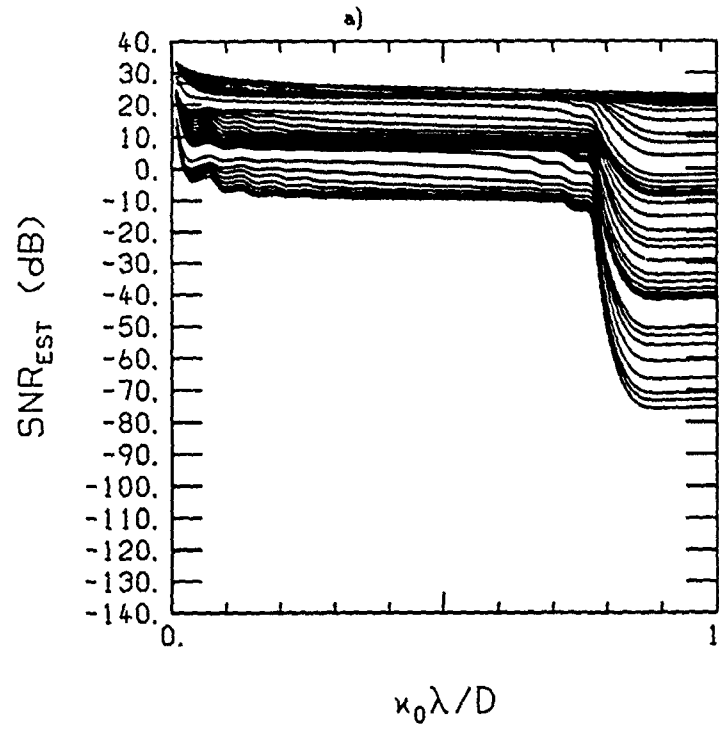
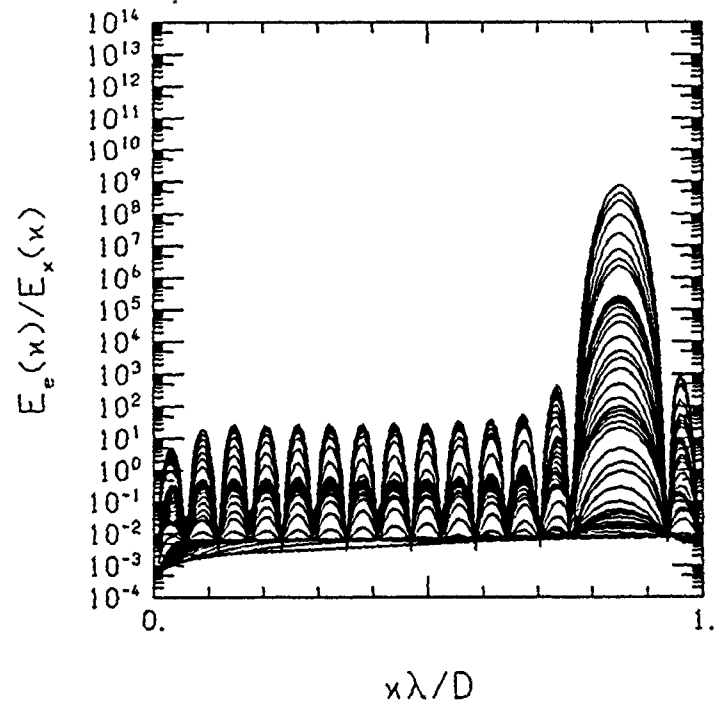


Figure 4.11.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_0 (b) using the unweighted least-squares processor and the five-subaperture array with $\text{SNR}_{\text{REF}} = 50\text{dB}$ and $d/D = 0.005$. The 32 curves of (a) correspond to object supports of $1\lambda/D$, $2\lambda/D$, ..., $32\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D$, $2\lambda/D$, ..., $32\lambda/D$, with larger SNR_{EST} for smaller support values.



b)

Figure 4.12.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_0 (b) using the unweighted least-squares processor and the six-subaperture array with $\text{SNR}_{\text{REF}} = 50\text{dB}$ and $d/D = 0.005$. The 32 curves of (a) correspond to object supports of $1\lambda/D, 2\lambda/D, \dots, 32\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D, 2\lambda/D, \dots, 32\lambda/D$, with larger SNR_{EST} for smaller support values.

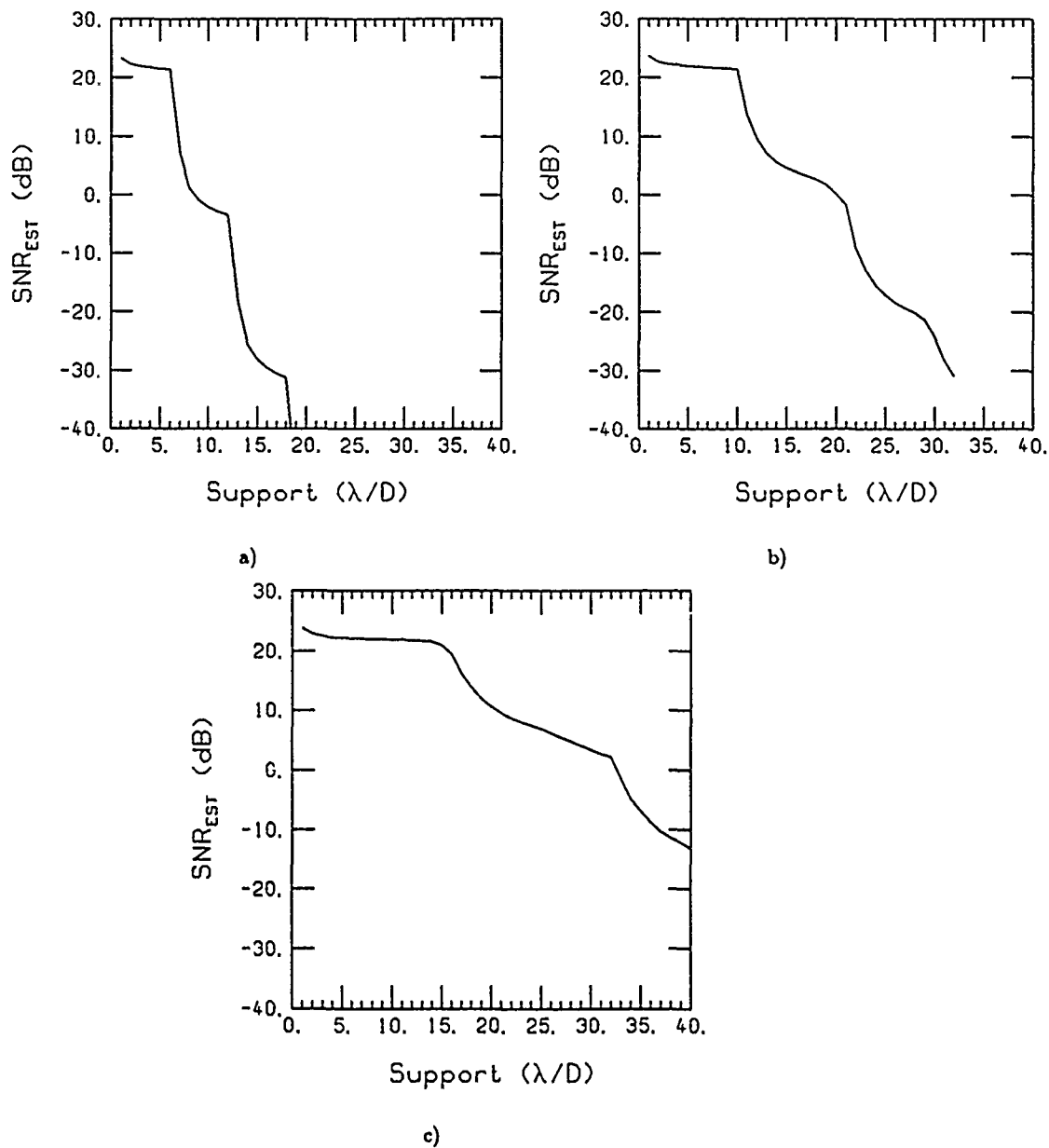


Figure 4.13.

Signal-to-noise ratio of the estimate versus object support using the unweighted least-squares processor, $\text{SNR}_{\text{REF}} = 50\text{dB}$, and $d/D = 0.005$. Fig.'s (a), (b), and (c) correspond respectively, to the four-subaperture array with $\kappa_o = 1D/\lambda$, the five-subaperture array with $\kappa_o = 0.81D/\lambda$, and the six-subaperture array with $\kappa_o = 0.76D/\lambda$.

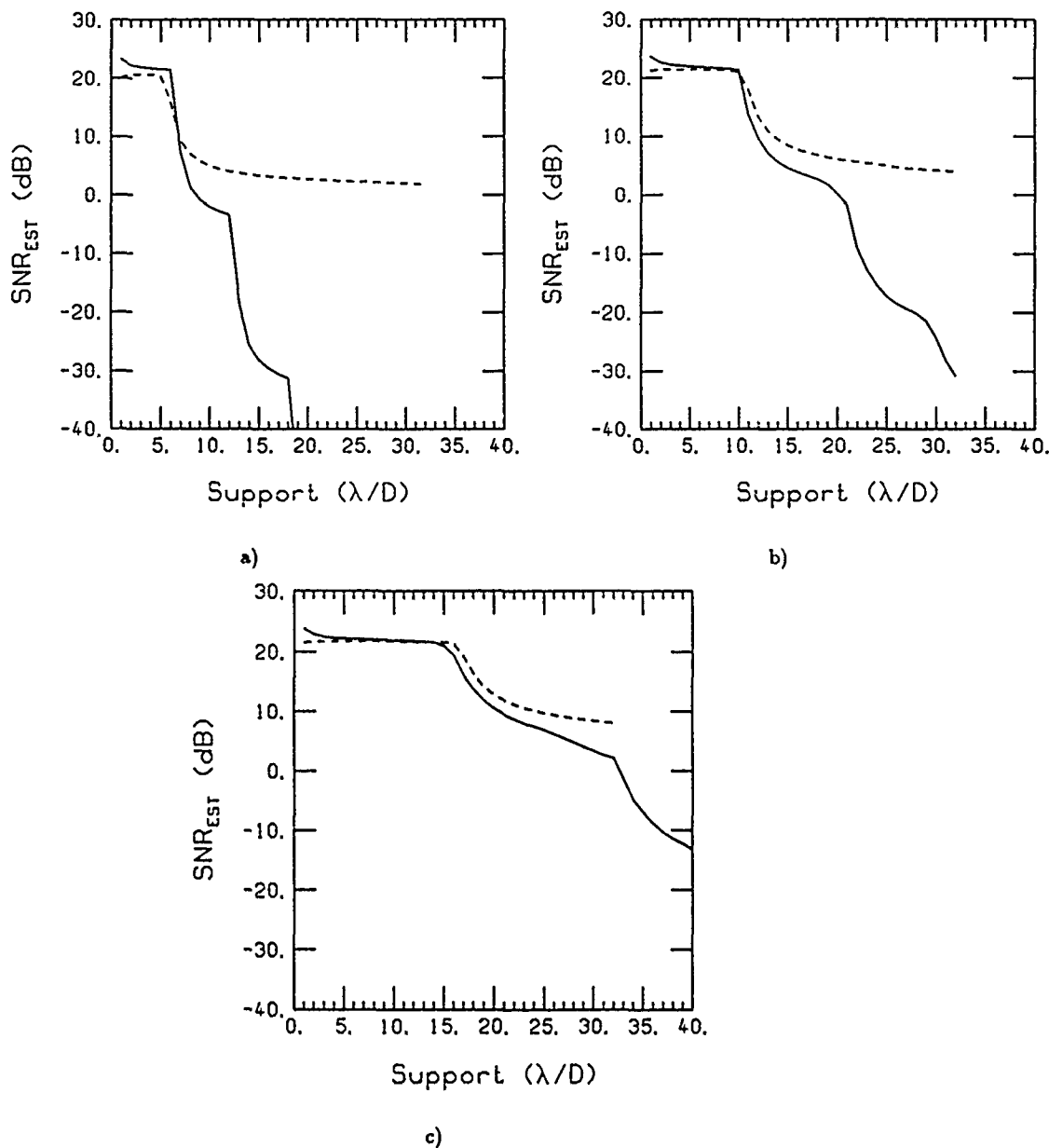


Figure 4.14.

Signal-to-noise ratio of the estimate versus object support using the unweighted least-squares processor (solid line) or the minimum-variance processor (dashed line), $\text{SNR}_{\text{REF}} = 50\text{dB}$, and $d/D = 0.005$. Fig's (a), (b), and (c) correspond respectively, to the four-subaperture array with $\kappa_o = 1D/\lambda$, the five-subaperture array with $\kappa_o = 0.81D/\lambda$, and the six-subaperture array with $\kappa_o = 0.76D/\lambda$.

Experiments were conducted using values of d/D other than 0.005, which is the value used throughout this report. We found, for small values of d/D , that SNR_{EST} to the left of the "cliff" was proportional to d/D , as long as the size of the image line was adjusted in each case so that the image line was effectively "infinity" from a performance standpoint.

All the results presented in this report used $\text{SNR}_{\text{REF}} = 50 \text{ dB}$. However, many experiments were conducted using other values of this parameter. As one would expect, SNR_{EST} tracked SNR_{REF} . In fact, we can see from Eq. (4.66) that SNR_{EST} with the least-squares processor is directly proportional to SNR_{REF} . Because of the identity matrix in Eq. (4.54), we can see that this is not the case using the minimum-variance processor. However, at $\text{SNR}_{\text{REF}} = 50 \text{ dB}$ and above, the effect of the identity matrix in Eq. (4.54) is negligible for object support sizes to the left of the "cliff" edge. We can see this from Fig. 4.14. At SNR_{REF} below 50 dB, the identity matrix in Eq. (4.54) begins to dominate, and minimum-variance performance is better than that of least-squares. However, at this level of SNR_{REF} , SNR_{EST} to the left of the "cliff" drops below 20 dB (a voltage ratio less than 10), so that the usefulness of the object reconstruction is open to question. The conclusion is, at least for the cases that we considered, that minimum-variance offers no apparent advantage over least-squares.

Chapter 5

A Random Process with Finite Support: Autocorrelation Function of Its Fourier Transform and Energy and Power Spectral Densities

(originally issued as TR-1042)

5.1. Introduction

It is a well-known property of a wide-sense stationary random process that its Fourier transform is delta-correlated. This is not the case for the Fourier transform of a random processes with finite support. In this report we compute the autocorrelation function of the Fourier transform of a particular random process with finite support. Our particular process is modeled as a windowed wide-sense-stationary random process. We use this autocorrelation function to derive the energy spectral density and power spectral density of the process. We present the results for both continuous (Section 5.2) and discrete (Section 5.3) random processes.

5.2. Continuous Case

5.2.1 Autocorrelation Function of Fourier Transform

Let $z(r)$ be a wide-sense-stationary random process with zero-mean, autocorrelation function $R_z(r)$, and power spectral density $\Phi_z(\kappa)$, defined by the equations

$$R_z(r) = \langle z(r' + r)z(r') \rangle, \quad (5.1)$$

$$\Phi_z(\kappa) = \int dr \exp(-2\pi i \kappa r) R_z(r). \quad (5.2)$$

We shall let

$$x(r) = w(r)z(r), \quad (5.3)$$

where $w(r)$ is a rectangular window function defining the support of $x(r)$, given by

$$w(r) = \begin{cases} 1, & |r| \leq L/2 \\ 0, & \text{else.} \end{cases} \quad (5.4)$$

The Fourier transform of $x(r)$ is

$$\begin{aligned} \tilde{x}(\kappa) &= \int dr x(r) \exp[-2\pi i r \kappa] \\ &= \int dr w(r) z(r) \exp[-2\pi i r \kappa]. \end{aligned} \quad (5.5)$$

Using Eq. (5.5), we form the correlation between two Fourier coefficients of $x(r)$:

$$\begin{aligned} \langle \tilde{x}(\kappa + \Delta\kappa) \tilde{x}^*(\kappa) \rangle &= \iint dr dr' w(r) w(r') \langle z(r) z^*(r') \rangle \\ &\quad \times \exp[-2\pi i(r - r')\kappa - 2\pi i r \Delta\kappa] \\ &\doteq \iint dr dr' w(r) w(r') R_z(r - r') \\ &\quad \times \exp[-2\pi i(r - r')\kappa - 2\pi i r \Delta\kappa]. \end{aligned} \quad (5.6)$$

Throughout this report, angle brackets denote ensemble average. We now make the following change of variables in Eq. (5.6):

$$u = r - r', \quad (5.7)$$

$$v = (r + r')/2, \quad (5.8)$$

$$r = v + u/2, \quad (5.9)$$

$$r' = v - u/2, \quad (5.10)$$

$$dr dr' = du dv. \quad (5.11)$$

Then we have

$$\begin{aligned}\langle \tilde{x}(\kappa + \Delta\kappa) \tilde{x}^*(\kappa) \rangle &= \iint du dv w(v + u/2) w(v - u/2) R_z(u) \\ &\quad \times \exp[-2\pi i \kappa u] \exp[-2\pi i \Delta\kappa (v + u/2)] \\ &= \int du R_z(u) F(u, \Delta\kappa) \exp[-2\pi i u (\kappa + \Delta\kappa/2)],\end{aligned}\quad (5.12)$$

where

$$\begin{aligned}F(u, \Delta\kappa) &= \int dv w(v + u/2) w(v - u/2) \exp[-2\pi i \Delta\kappa v] \\ &= \begin{cases} (L - |u|) \text{sinc}[(L - |u|)\Delta\kappa], & |u| \leq L \\ 0, & |u| > L, \end{cases}\end{aligned}\quad (5.13)$$

with

$$\text{sinc}(x) = \sin(\pi x) / \pi x. \quad (5.14)$$

Substituting Eq. (5.13) into Eq. (5.12) yields

$$\langle \tilde{x}(\kappa + \Delta\kappa) \tilde{x}^*(\kappa) \rangle = \int_{-L}^L du R_z(u) (L - |u|) \text{sinc}[(L - |u|)\Delta\kappa] \exp[-2\pi i u (\kappa + \Delta\kappa/2)]. \quad (5.15)$$

Given any $R_z(u)$, Eq. (5.15) can be used to numerically evaluate the correlation of Fourier coefficients of $x(r)$. However, if we assume that $R_z(u)$ is a "narrow" function compared to the interval $|u| \leq L$, i.e.,

$$R_z(u) \simeq 0, \quad |u| > \epsilon, \quad \epsilon \ll L, \quad (5.16)$$

then Eq. (5.15) can be approximated by

$$\begin{aligned}\langle \tilde{x}(\kappa + \Delta\kappa) \tilde{x}^*(\kappa) \rangle &\simeq L \text{sinc}(L\Delta\kappa) \int du R_z(u) \exp[-2\pi i u (\kappa + \Delta\kappa/2)] \\ &\simeq L \text{sinc}(L\Delta\kappa) \Phi_z(\kappa + \Delta\kappa/2).\end{aligned}\quad (5.17)$$

If $z(r)$ is white, i.e.,

$$R_z(r) = \Phi_z \delta(r), \quad (5.18)$$

then we have Eq. (5.17) with equality,

$$\langle \tilde{x}(\kappa + \Delta\kappa) \tilde{x}^*(\kappa) \rangle = L \text{sinc}(L\Delta\kappa) \Phi_z. \quad (5.19)$$

Note from Eq. (5.17) and (5.19) that the Fourier coefficients of $x(r)$ are uncorrelated at $\Delta\kappa$ equal to an integer multiple of $1/L$, the reciprocal of the support of $x(r)$, if $R_z(r)$ satisfies the assumption given by Eq. (5.16).

5.2.2 Energy Spectral Density and Power Spectral Density of $x(r)$

The two-sided energy spectral density of any random process $x(r)$ of finite support is given by

$$E_x(\kappa) = \langle |\tilde{x}(\kappa)|^2 \rangle. \quad (5.20)$$

If $x(r)$ is given by Eq. (5.3), then, from Eq. (5.15), we have

$$E_x(\kappa) = \int_{-L}^L du R_z(u) (L - |u|) \exp[-2\pi i u \kappa]. \quad (5.21)$$

We recognize Eq. (5.21) as the Fourier transform of the product of $R_z(u)$ with a triangle function. We can therefore write Eq. (5.21) as the convolution of the Fourier transforms of these two functions:

$$E_x(\kappa) = L^2 \int d\kappa' \Phi_z(\kappa - \kappa') \text{sinc}^2(L\kappa'). \quad (5.22)$$

The power spectral density $\Phi_x(\kappa)$ of $x(r)$ can be defined as

$$\Phi_x(\kappa) = E_x(\kappa)/L. \quad (5.23)$$

From Eq. (5.21) and (5.22) we then have

$$\Phi_x(\kappa) = \frac{1}{L} \int_{-L}^L du R_z(u)(L - |u|) \exp[-2\pi i u \kappa], \quad (5.24)$$

or

$$\Phi_x(\kappa) = L \int d\kappa' \Phi_z(\kappa - \kappa') \text{sinc}^2(L\kappa'). \quad (5.25)$$

If $R_z(r)$ satisfies the assumption given by Eq. (5.16), then the power spectral density of $x(r)$ is approximated by

$$\Phi_x(\kappa) \simeq \Phi_z(\kappa). \quad (5.26)$$

We see from Eq. (5.26) that when the assumption given by Eq. (5.16) is satisfied, the power spectral density of $x(r)$ is simply the power spectral density of $z(r)$. If $z(r)$ is white, then we have Eq. (5.26) with equality, i.e.,

$$\Phi_x(\kappa) = \Phi_z. \quad (5.27)$$

The variance of $x(r)$ is the total power of the $x(r)$ process given by

$$\begin{aligned} \sigma_x^2(r) &= \langle x(r)x^*(r) \rangle \\ &= \begin{cases} \int_{-\infty}^{\infty} d\kappa \Phi_x(\kappa), & |r| \leq L/2 \\ 0, & \text{else.} \end{cases} \end{aligned} \quad (5.28)$$

Using Eq. (5.24) in Eq. (5.28) yields

$$\begin{aligned} \sigma_x^2(r) &= w(r) R_z(0) \\ &= w(r) \int_{-\infty}^{\infty} d\kappa \Phi_z(\kappa) \\ &= w(r) \sigma_z^2. \end{aligned} \quad (5.29)$$

This result follows directly from Eq. (5.3) and the fact the $z(r)$ is wide-sense-stationary.

5.3. Discrete Case

5.3.1 Autocorrelation Function of Fourier Transform

Let $z(n)$ be a wide-sense-stationary random sequence with zero-mean, autocorrelation sequence $R_z(n)$, and power spectral density $\Phi_z(\nu)$, defined by the equations

$$R_z(n) = \langle z(n' + n)z(n') \rangle, \quad (5.30)$$

$$\Phi_z(\nu) = \sum_n R_z(n) \exp(-2\pi i n \nu). \quad (5.31)$$

We shall let

$$x(n) = w(n)z(n) \quad (5.32)$$

where $w(n)$ is a rectangular window sequence defining the support of $x(n)$.

$$w(n) = \begin{cases} 1, & |n| \leq L/2 \\ 0, & \text{else} \end{cases} \quad (5.33)$$

The variable ν in $\Phi_z(\nu)$ above is a normalized frequency variable given by

$$\nu = \frac{\kappa}{\kappa_s} \quad (5.34)$$

where κ_s is the sampling frequency. The Fourier transform of $x(n)$ is

$$\begin{aligned} \tilde{x}(\nu) &= \sum_{n=-\infty}^{\infty} x(n) \exp[-2\pi i n \nu] \\ &= \sum_{n=-\infty}^{\infty} w(n) z(n) \exp[-2\pi i n \nu]. \end{aligned} \quad (5.35)$$

Using Eq. (5.35), we form the correlation between two fourier coefficients of $x(n)$:

$$\begin{aligned} \langle \tilde{x}(\nu + \Delta\nu) \tilde{x}^*(\nu) \rangle &= \sum_m \sum_n w(m) w(n) \langle z(m) z^*(n) \rangle \\ &\quad \times \exp[-2\pi i(m-n)\nu - 2\pi i m \Delta\nu] \\ &= \sum_m \sum_n w(m) w(n) R_z(m-n) \exp[-2\pi i(m-n)\nu] \\ &\quad \times \exp[-2\pi i m \Delta\nu]. \end{aligned} \quad (5.36)$$

Now make the following change of variables to Eq. (5.36):

$$k = m - n, \quad (5.37)$$

$$l = m, \quad (5.38)$$

$$n = l - k, \quad (5.39)$$

$$\begin{aligned} \langle \tilde{x}(\nu + \Delta\nu) \tilde{x}^*(\nu) \rangle &= \sum_k \sum_l w(l) w(l-k) R_z(k) \exp[-2\pi i k \nu] \exp[-2\pi i l \Delta\nu] \\ &= \sum_k R_z(k) G(k, \Delta\nu) \exp[-2\pi i k \nu], \end{aligned} \quad (5.40)$$

where

$$\begin{aligned} G(k, \Delta\nu) &= \sum_l w(l) w(l-k) \exp[-2\pi i l \Delta\nu] \\ &= \begin{cases} \exp[-\pi i k \Delta\nu] \frac{\sin[\pi \Delta\nu (L+1-|k|)]}{\sin(\pi \Delta\nu)}, & |k| \leq L+1 \\ 0, & |k| > L+1. \end{cases} \end{aligned} \quad (5.41)$$

Substituting Eq. (5.41) into Eq. (5.40), we have

$$\langle \tilde{x}(\nu + \Delta\nu) \tilde{x}^*(\nu) \rangle = \sum_{k=-L-1}^{L+1} R_z(k) \frac{\sin[\pi \Delta\nu (L+1-|k|)]}{\sin(\pi \Delta\nu)} \exp[-2\pi i k (\nu + \Delta\nu/2)]. \quad (5.42)$$

Given any $R_z(k)$, Eq. (5.42) can be used to numerically evaluate the correlation of Fourier coefficients of $x(n)$. However, if we assume that $R_z(k)$ is a "narrow" function compared to the interval $|k| \leq L$, i.e.,

$$R_z(k) \simeq 0, \quad |k| > \ell, \quad \ell \ll L, \quad (5.43)$$

then Eq. (5.42) can be approximated by

$$\langle \tilde{x}(\nu + \Delta\nu) \tilde{x}^*(\nu) \rangle \simeq \frac{\sin[\pi\Delta\nu(L+1)]}{\sin(\pi\Delta\nu)} \Phi_z(\nu + \Delta\nu/2). \quad (5.44)$$

If $z(n)$ is a white sequence, i.e.,

$$R_z(k) = \Phi_z \delta(k), \quad (5.45)$$

where $\delta(k)$ is the Kronecker delta, then we have Eq. (5.44) with equality,

$$\langle \tilde{x}(\nu + \Delta\nu) \tilde{x}^*(\nu) \rangle = \frac{\sin[\pi\Delta\nu(L+1)]}{\sin(\pi\Delta\nu)} \Phi_z(\nu + \Delta\nu/2). \quad (5.46)$$

We note from consideration of Eq.'s (5.44) and (5.46) that if $R_z(k)$ satisfies the assumption given by Eq. (5.43) the Fourier coefficients of $x(n)$ are uncorrelated for pairs of coefficients associated with frequency differences equal to $\Delta\nu$, when $\Delta\nu$ equals an integer multiple of $1/(L+1)$ —a quantity equal to the reciprocal of the support of $x(n)$.

5.3.2 Energy Spectral Density and Power Spectral Density of $x(n)$

The two-sided energy spectral density of any random sequence of finite support is given by

$$E_x(\nu) = \langle |\tilde{x}(\nu)|^2 \rangle. \quad (5.47)$$

If $x(n)$ is given by Eq. (5.32), then, from Eq. (5.42),

$$E_x(\nu) = \sum_{k=-L-1}^{L+1} R_z(k)(L+1-|k|) \exp[-2\pi i k \nu]. \quad (5.48)$$

The power spectral density $\Phi_x(\nu)$ of $x(k)$ can be defined as

$$\Phi_x(\nu) = E_x(\nu)/L. \quad (5.49)$$

From Eq. (5.48) we have

$$\Phi_x(\nu) = \frac{1}{L+1} \sum_{k=-L-1}^{L+1} R_z(k)(L+1-|k|) \exp[-2\pi i k \nu]. \quad (5.50)$$

If $R_z(k)$ satisfies the assumption given by Eq. (5.43), then the power spectral density of $x(n)$ is approximated by

$$\Phi_x(\nu) \simeq \Phi_z(\nu). \quad (5.51)$$

If $z(n)$ is white, then we have Eq. (5.51) with equality, i.e.,

$$\Phi_x(\nu) = \Phi_z(\nu). \quad (5.52)$$

The variance of $x(n)$ is the total power of the $x(n)$ process given by

$$\begin{aligned} \sigma_x^2(n) &= \langle |x(n)|^2 \rangle \\ &= \begin{cases} \int_{-1/2}^{1/2} d\nu \Phi_x(\nu), & |n| \leq L/2 \\ 0, & \text{else.} \end{cases} \end{aligned} \quad (5.53)$$

Using Eq. (5.50) in Eq. (5.53) yields

$$\begin{aligned} \sigma_x^2(n) &= w(n) R_z(0) \\ &= w(n) \int_{-1/2}^{1/2} d\nu \Phi_z(\nu) \\ &= w(n) \sigma_z^2. \end{aligned} \quad (5.54)$$

This result follows directly from Eq. (5.32) and the fact that $z(n)$ is wide-sense-stationary.

Chapter 6

Object Reconstruction with Sparse Arrays of
Optical Apertures.

Part II:
Nonlinear Methods

(originally issued as TR-1072)

6.1 Introduction

This is the second part of Chapter 4 on the feasibility of using sparse arrays of optical apertures to form useful images of objects. We stress "feasibility," since we have no interest in developing algorithms. In part I²⁸, we created a discrete one-dimensional model of an optical imaging system, defined a performance measure, and produced results using two linear object reconstruction algorithms, minimum-variance and unweighted least-squares. Using three sparse arrays we concluded that object reconstruction with sparse arrays was indeed feasible so long as object angular support did not exceed the inverse of the size of the gaps between "islands" in the MTF of the array.

Because the algorithms we used were linear, they were capable of producing object reconstructions with negative intensity pixels. In this chapter we explore the question as to whether adding a positivity constraint to an object estimate has the potential of significantly improving performance. We examine two algorithms, unweighted least-squares with a positivity constraint and an algorithm used in radio interferometry called CLEAN. In both cases, we conclude that a positivity constraint has marginal benefit.

Throughout this chapter we assume familiarity with Ref. 28. However, for convenience we have included in their entirety Section 4.2 (optical model) and Section 4.3 (performance measure) of Chapter 4 as Section 6.2 and 6.3 of this report. We have also duplicated Fig.'s 4.1 through 4.4 of Chapter 4 showing three sparse arrays and their MTFs as Fig.'s 6.1 through 6.4 of this chapter.

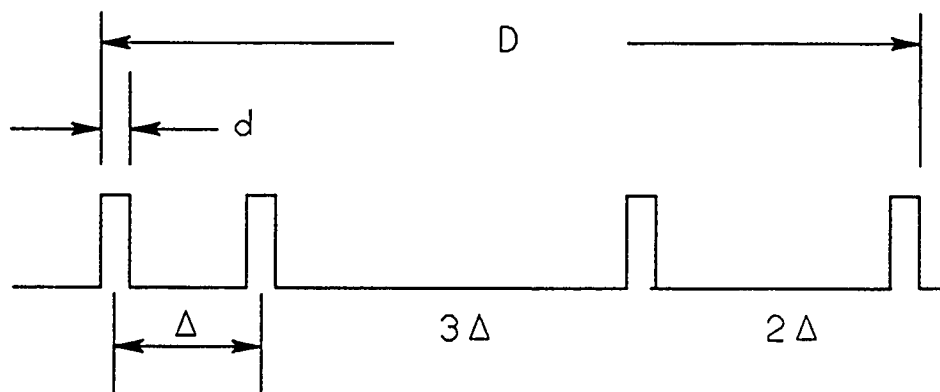
The results for nonlinear algorithms presented in this work were all obtained using Monte Carlo methods, i.e., simulation of a random process, taking averages over many random results. In almost all cases, $N = 40$ random trials were used to provide an average—though in a few cases $N < 40$ random trials were used (if the computational process ran too slowly). In those cases where $N < 40$ was used, we formed an estimate of the variance of the average quantity of interest and used a number of trials, N , large enough so that the accuracy of our estimate of the average quantity of interest would not be seriously compromised.

6.2. Discrete Optical Model (One-Dimensional)

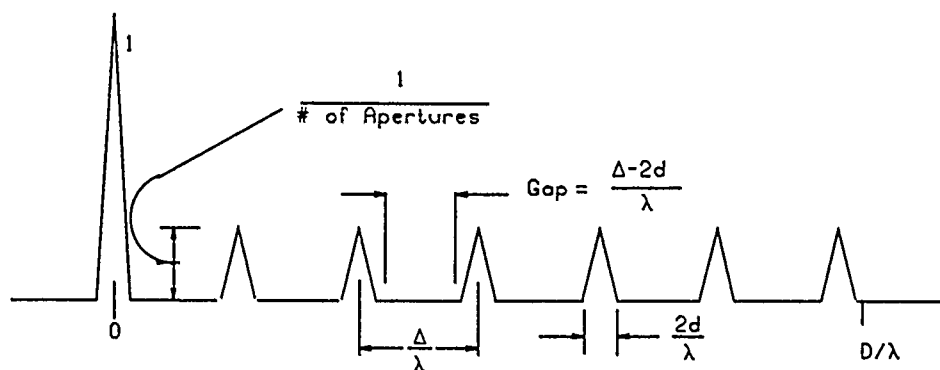
Let the components of the $L \times 1$ vector $\mathbf{z}$ be the intensity pixels of the object line, let the components of the $N \times 1$ vector $\mathbf{y}$ represent the intensity pixels of the image line, and let the $N \times L$ matrix B be the transformation from object line to image line (its columns contain shifted versions of the system point-spread-function). Then an image can be represented by the matrix equation

$$\mathbf{y} = B\mathbf{z} + \mathbf{n}, \quad (6.1)$$

where $\mathbf{n}$ is a $N \times 1$ noise vector. The product $B\mathbf{z}$ in Eq. (6.1) represents convolution of the object line with the point spread-function, followed by truncation of the image. Let the point spread function

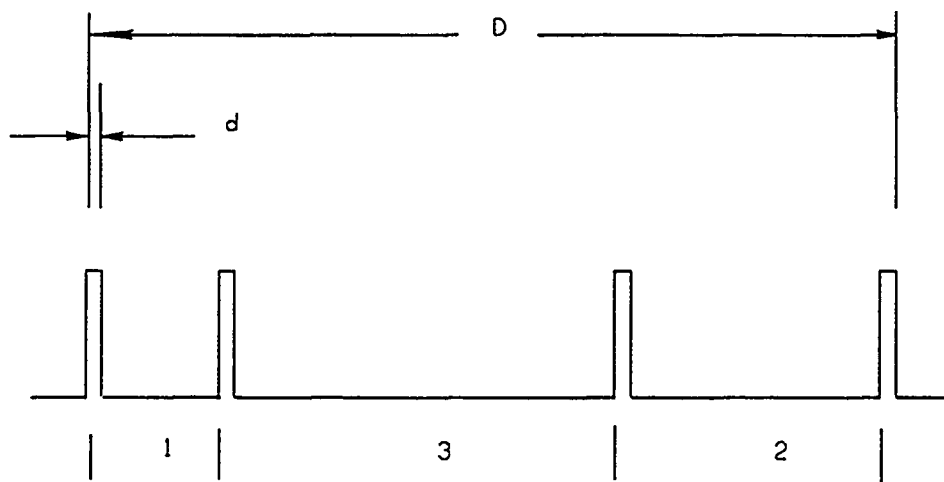


a)

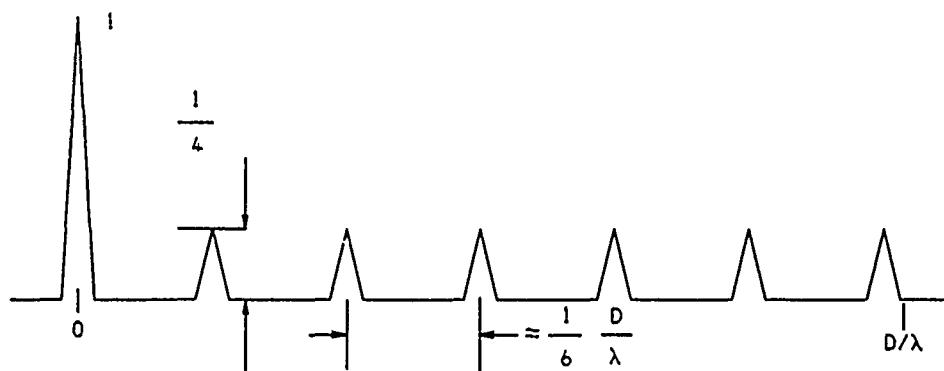


b)

Figure 6.1.
Typical sparse array aperture function (a) and the corresponding MTF (b).

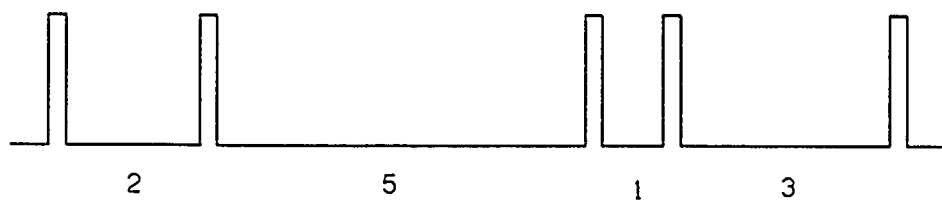


a)

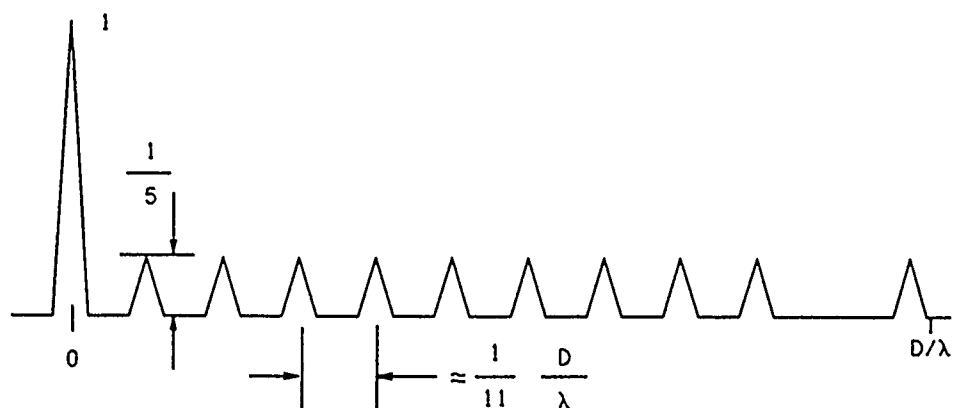


b)

Figure 6.2.
Sparse array aperture function (a) and the corresponding MTF (b) for an "efficient" four-subaperture array.

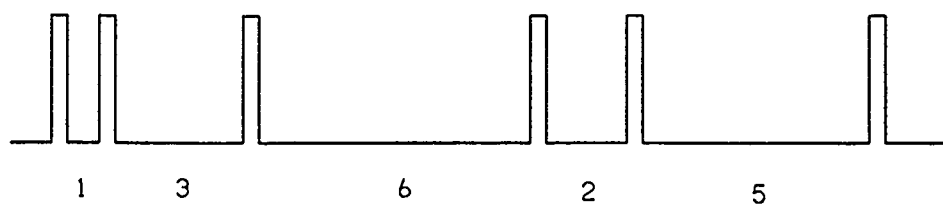


a)

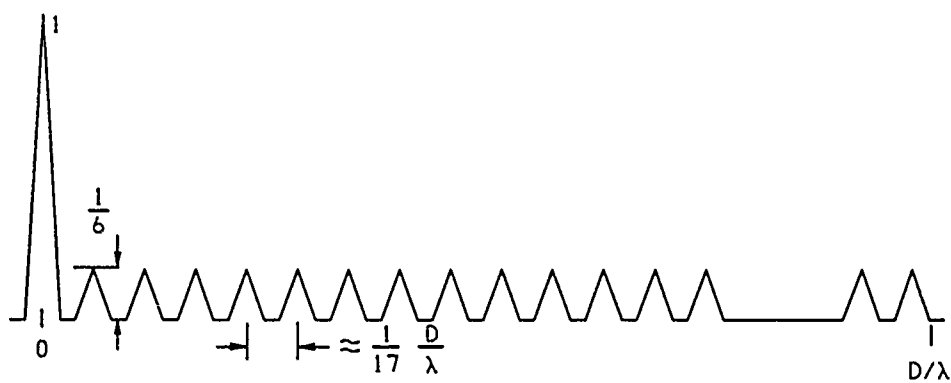


b)

Figure 6.3.
Sparse array aperture function (a) and the corresponding MTF (b) for an "efficient" five-subaperture array.



a)



b)

Figure 6.4.

Sparse array aperture function (a) and the corresponding MTF (b) for an "efficient" six-subaperture array.

be $h(k)$. Then the B matrix is

$$B = \begin{bmatrix} h(-K) & h(-K-1) & \dots & h(-K-L+1) \\ h(-K+1) & h(-K) & \dots & h(-K-L+2) \\ \vdots & h(-K+1) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \dots & h(-K-2) \\ h(0) & \vdots & \dots & h(-K-1) \\ \vdots & h(0) & \dots & h(-K) \\ \vdots & \vdots & \dots & h(-K+1) \\ h(K-1) & \vdots & \dots & \vdots \\ h(K) & h(K-1) & \dots & \vdots \\ h(K+1) & h(K) & \dots & h(0) \\ h(K+2) & h(K+1) & \dots & \vdots \\ \vdots & h(K+2) & \dots & \vdots \\ \vdots & \vdots & \dots & h(K-1) \\ h(K+L-1) & h(K+L-2) & \dots & h(K) \end{bmatrix}, \quad (6.2)$$

where $K = (N - L)/2$.

Now let the $M \times 1$ vector $\mathbf{x}$ represent an object of finite contiguous support of length $M \leq L$ pixels located somewhere in the object line. We can write

$$\mathbf{z} = W\mathbf{x}, \quad (6.3)$$

where the $L \times M$ matrix W is of the form

$$W = \begin{bmatrix} & & & 0 & & \\ \text{---} & & & \text{---} & & \\ & 1 & & & & 0 \\ & & 1 & & & \\ & & & \ddots & & \\ & 0 & & & & 1 \\ \text{---} & & & \text{---} & & \text{---} \\ & & & & 0 & \end{bmatrix}. \quad (6.4)$$

Thus, W comprises an $M \times M$ identity matrix embedded in a $L \times M$ matrix of zeros. Combining Eq.'s (6.3) and (6.1) yields

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}. \quad (6.5)$$

To complete the model, we need a point-spread function. Let $w(x)$ be an arbitrary aperture function. The corresponding modulation transfer function (MTF) is given below, where κ is the spatial frequency variable in units of cycles per radian and λ is wavelength. We write

$$MTF(\kappa) = \frac{1}{A} \int dx w(x + \frac{1}{2}\kappa\lambda) w(x - \frac{1}{2}\kappa\lambda) \quad (6.6)$$

A is chosen to be the area (in this case the length) of the aperture, so that the MTF at zero spatial frequency is unity.

The point-spread function of an optical system is the inverse Fourier transform of the MTF function, or

$$h_c(x) = \int d\kappa \text{MTF}(\kappa) \exp[2\pi i \kappa x]. \quad (6.7)$$

The subscript c denotes that this point-spread function is the continuous version. The discrete version of the point-spread function is found by sampling and scaling the continuous version. Let δ denote the pixel spacing in the image line. Then the discrete point spread function used in the B matrix given by Eq. (6.2) is

$$h(n) = \delta h_c(n\delta); \quad n = 0, \pm 1, \pm 2, \dots \quad (6.8)$$

The continuous point spread functions for the four, five, and six subaperture arrays are given in Eq.'s (9), (10), and (11) respectively. (The reader should see Figs 6.2, 6.3, and 6.4 for the corresponding MTF's.) We have

$$h_c(x) = \frac{d}{\lambda} \left(\frac{\sin \pi \frac{d}{\lambda} x}{\pi \frac{d}{\lambda} x} \right)^2 \left(1 + \frac{1}{2} \sum_{i=1}^6 \cos \left[i \frac{\pi}{3} \frac{D}{\lambda} \left(1 - \frac{d}{D} \right) x \right] \right), \quad (6.9)$$

$$h_c(x) = \frac{d}{\lambda} \left(\frac{\sin \pi \frac{d}{\lambda} x}{\pi \frac{d}{\lambda} x} \right)^2 \left(1 + \frac{2}{5} \sum_{\substack{i=1 \\ i \neq 10}}^{11} \cos \left[i \frac{2\pi}{11} \frac{D}{\lambda} \left(1 - \frac{d}{D} \right) x \right] \right), \quad (6.10)$$

$$h_c(x) = \frac{d}{\lambda} \left(\frac{\sin \pi \frac{d}{\lambda} x}{\pi \frac{d}{\lambda} x} \right)^2 \left(1 + \frac{1}{3} \sum_{\substack{i=1 \\ i \neq 14 \\ i \neq 15}}^{17} \cos \left[i \frac{2\pi}{17} \frac{D}{\lambda} \left(1 - \frac{d}{D} \right) x \right] \right). \quad (6.11)$$

6.3. A Performance Measure

In the sections to follow we investigate two algorithms for object recovery. In order to evaluate the performance of each algorithm and to compare algorithms, we need a measure of performance that we can apply uniformly to both algorithms. Our object recovery algorithms not only attempt to fill in the gaps between measured spatial frequency components of the object (interpolative super-resolution), but also to estimate the object at spatial frequencies beyond D/λ (extrapolative super-resolution), where D is the over all length of the array. Because of array geometries, our algorithms will be performing better at recovering some spatial frequency components than others. We therefore need a frequency sensitive performance measure. To this end, let $x(n)$ be the n^{th} pixel of the object, $\hat{x}(n)$ be the n^{th} pixel of an estimate of the object, and $e(n)$ be the n^{th} pixel of the estimation error given by

$$e(n) = x(n) - \hat{x}(n), \quad (6.12a)$$

or in vector notation,

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}. \quad (6.12b)$$

One common measure of performance is mean-square-error, $\langle \mathbf{e}^T \mathbf{e} \rangle$. For our purposes this measure is not useful. It gives us no indication of how well an algorithm performs as a function of spatial frequency. Instead, the performance measure we will use is a signal-to-noise ratio (SNR) that is a function of a spatial cutoff frequency. The behavior of the SNR as a function of cutoff frequency will be an indication of the spatial resolution of the sparse array. To introduce frequency sensitivity into the performance measure, let $x_{\kappa_0}(n)$, $\hat{x}_{\kappa_0}(n)$, and $e_{\kappa_0}(n)$ be filtered versions of $x(n)$, $\hat{x}(n)$, and

$e(n)$, respectively, where the filtering is ideal low-pass with cutoff frequency κ_o . To explicitly define the filtering operation, let $y(n)$ be a sequence of pixels and $\tilde{y}(\kappa)$ be its Fourier transform given by

$$\tilde{y}(\kappa) = \sum_n y(n) \exp(-2\pi i n \kappa \delta), \quad (6.13)$$

where δ is the object pixel spacing. Let $\tilde{f}(\kappa)$ be the filter transfer function given by

$$\tilde{f}(\kappa) = \begin{cases} 1, & |\kappa| \leq \kappa_o \\ 0, & |\kappa| > \kappa_o. \end{cases} \quad (6.14)$$

Then the Fourier transform of $y_{\kappa_o}(n)$, the filtered version of $y(n)$, is given by

$$\tilde{y}_{\kappa_o}(\kappa) = \tilde{y}(\kappa) \tilde{f}(\kappa). \quad (6.15)$$

Thus, we have

$$\tilde{x}_{\kappa_o}(\kappa) = \tilde{x}(\kappa) \tilde{f}(\kappa) \quad (6.16)$$

$$\tilde{\hat{x}}_{\kappa_o}(\kappa) = \tilde{\hat{x}}(\kappa) \tilde{f}(\kappa) \quad (6.17)$$

$$\begin{aligned} \tilde{e}_{\kappa_o}(\kappa) &= \tilde{e}(\kappa) \tilde{f}(\kappa) \\ &= [\tilde{x}(\kappa) - \tilde{\hat{x}}(\kappa)] \tilde{f}(\kappa) \\ &= \tilde{x}_{\kappa_o}(\kappa) - \tilde{\hat{x}}_{\kappa_o}(\kappa). \end{aligned} \quad (6.18)$$

Let us now define a signal-to-noise ratio for the filtered estimate $\hat{x}_{\kappa_o}(n)$ as the ratio of the average energy of the filtered object to the average energy of the filtered estimation error:

$$\text{SNR}_{\text{EST}}(\kappa_o) = \frac{\left\langle \sum_n x_{\kappa_o}^2(n) \right\rangle}{\left\langle \sum_n e_{\kappa_o}^2(n) \right\rangle}. \quad (6.19)$$

The angle brackets in Eq. (6.19) denote ensemble average. From Parseval's theorem we know that the quantities inside the angle brackets in Eq. (6.19) can be written in terms of their Fourier transforms, as follows:

$$\sum_n x_{\kappa_o}^2(n) = \delta \int_{-1/2\delta}^{1/2\delta} d\kappa |\tilde{x}_{\kappa_o}(\kappa)|^2 \quad (6.20)$$

$$\sum_n e_{\kappa_o}^2(n) = \delta \int_{-1/2\delta}^{1/2\delta} d\kappa |\tilde{e}_{\kappa_o}(\kappa)|^2. \quad (6.21)$$

We can use Eq.'s (6.13), (6.14), (6.16), and (6.18) to rewrite Eq.'s (6.20) and (6.21), yielding

$$\begin{aligned} \sum_n x_{\kappa_o}^2(n) &= \delta \int_{-1/2\delta}^{1/2\delta} d\kappa |\tilde{x}(\kappa)|^2 |\tilde{f}(\kappa)|^2 \\ &= 2\delta \int_0^{\kappa_o} d\kappa |\tilde{x}(\kappa)|^2 \end{aligned} \quad (6.22)$$

$$\begin{aligned} \sum_n e_{\kappa_o}^2(n) &= \delta \int_{-1/2\delta}^{1/2\delta} d\kappa |\tilde{e}(\kappa)|^2 |\tilde{f}(\kappa)|^2 \\ &= 2\delta \int_0^{\kappa_o} d\kappa |\tilde{e}(\kappa)|^2. \end{aligned} \quad (6.23)$$

Replacing the quantities inside the angle-brackets in Eq. (6.19) with their corresponding expressions given by Eq.'s (6.22) and (6.23), and bringing the angle-brackets inside the integrals, yields

$$\text{SNR}_{\text{EST}}(\kappa_o) = \frac{\int_0^{\kappa_o} d\kappa \langle |\tilde{x}(\kappa)|^2 \rangle}{\int_0^{\kappa_o} d\kappa \langle |\tilde{e}(\kappa)|^2 \rangle}. \quad (6.24)$$

Since integrating the quantities $\delta \langle |\tilde{x}(\kappa)|^2 \rangle$ and $\delta \langle |\tilde{e}(\kappa)|^2 \rangle$ over the spatial frequency band $|\kappa| \leq \kappa_o$ yields, respectively, the energy in the object and estimation error over the same band, we will call these quantities energy spectral densities or energy spectra, and use the notation

$$E_x(\kappa) = \delta \langle |\tilde{x}(\kappa)|^2 \rangle \quad (6.25)$$

$$E_e(\kappa) = \delta \langle |\tilde{e}(\kappa)|^2 \rangle. \quad (6.26)$$

So that

$$\text{SNR}_{\text{EST}}(\kappa_o) = \frac{\int_0^{\kappa_o} d\kappa E_x(\kappa)}{\int_0^{\kappa_o} d\kappa E_e(\kappa)}. \quad (6.27)$$

To compute the energy spectrum of a random sequence, we simply follow the prescription given by Eq.'s (6.25) and (6.13). Let $\mathbf{z}$ be a random vector with n^{th} component $z(n)$. Then its energy spectrum is given by

$$\begin{aligned} E_z(\kappa) &= \delta \langle |\tilde{z}(\kappa)|^2 \rangle \\ &= \delta \left\langle \left| \sum_n z(n) \exp(-2\pi i n \kappa \delta) \right|^2 \right\rangle \\ &= \delta \sum_n \sum_m \langle z(n) z(m) \rangle \exp[-2\pi i (n - m) \kappa \delta]. \end{aligned} \quad (6.28)$$

We note from Eq. (6.28) that we need *all* entries of the covariance matrix of $z(n)$ in the evaluation of its energy spectrum.

6.4. Unweighted Least-Squares, Finite Support and Positivity Constraints

In Section 6.5. of Ref. 28 we used unweighted least-squares for object reconstruction, i.e., given the observation

$$\mathbf{y} = G\mathbf{x} + \mathbf{n}, \quad (6.29)$$

pick the object vector estimate $\hat{\mathbf{x}}$ which minimizes

$$\epsilon = \|\mathbf{y} - G\hat{\mathbf{x}}\|^2, \quad (6.30)$$

where $\mathbf{x}$ is the object vector, $\mathbf{n}$ is observation noise, and the columns of G are the system point-spread-function shifted and truncated. The finite support constraint to the solution is implicitly contained in the dimension of the object vector $\mathbf{x}$ and its estimate $\hat{\mathbf{x}}$. To add the positivity constraint we simply add the statement

$$\hat{\mathbf{x}} \geq 0. \quad (6.31)$$

The object vector estimate that minimizes ϵ of Eq. (6.30) can, of course, be obtained in closed form. Unfortunately, this is not the case when the positivity constraint given by Eq. (6.31) is included. One must use some numerical iteration method for each case of an object vector and a noise vector, and perform some averaging of the results. Therefore, to evaluate the error vector covariance matrix when using positivity as a constraint requires a simulation. To generate object vectors, we generated independent Rayleigh random variables for each pixel. Thus, object intensity

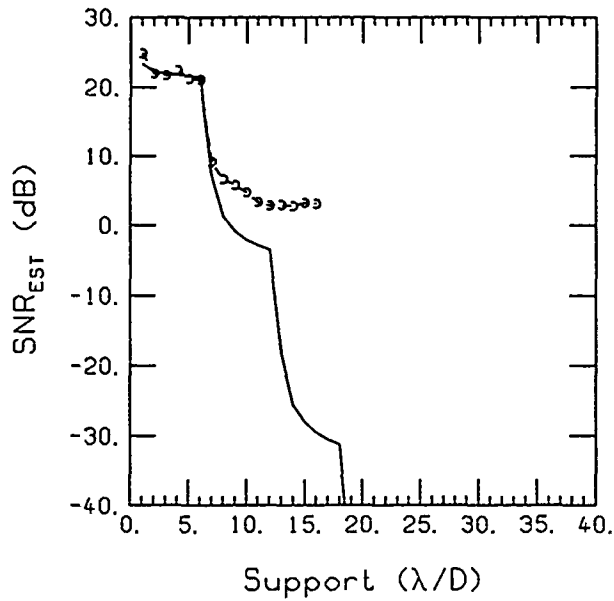


Figure 6.5.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using the unweighted least-square processor without and with a positivity constraint on the object estimate (solid line and circles, respectively). The four-subaperture array of Fig. 2 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

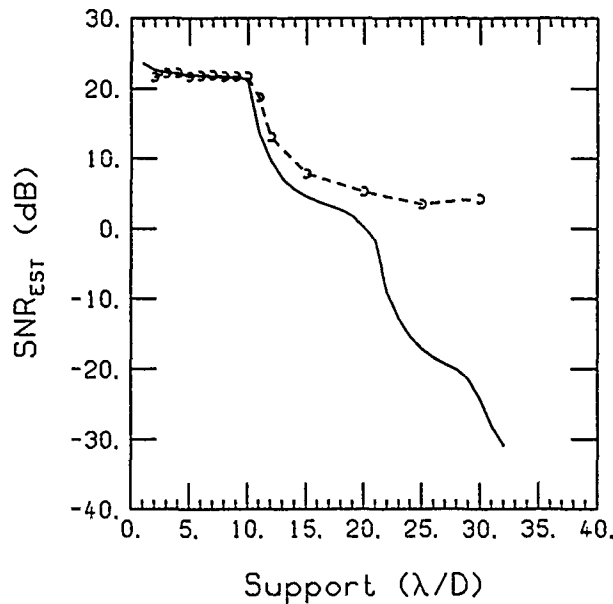


Figure 6.6

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using the unweighted least-squares processor without and with a positivity constraint on the object estimate (solid line and circles, respectively). The five-subaperture array of Fig. 3 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

pixels were always positive. Noise vectors were generated using independent, zero-mean, Gaussian random variables. An image vector, $\mathbf{y}$, was generated using Eq. (6.29) for each sample object and noise vector generated. The object vector estimate, $\hat{\mathbf{x}}$, for each case was found using an algorithm called gradient projection². It is essentially the steepest descent algorithm modified to accommodate the positivity constraint on the estimate. The algorithm is described in detail in Appendix A. If we let $\mathbf{x}(n)$ be the object vector and $\hat{\mathbf{x}}(n)$ be the object vector estimate, each for the n^{th} random trial, then the error covariance matrix was estimated using

$$\langle \mathbf{e}\mathbf{e}^T \rangle = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}(n) - \hat{\mathbf{x}}(n)) (\hat{\mathbf{x}}(n) - \hat{\mathbf{x}}(n))^T, \quad (6.32)$$

where N denotes the total number of random trials. The energy spectrum of the error vector was computed using Eq. (6.32) in Eq. (6.28). The results were then integrated to compute SNR_{EST} , as was done in Ref. 28. Plots of SNR_{EST} versus object support are shown in Fig.'s 6.5 through 6.7 for the four, five, and six subaperture cases, respectively. These calculations were performed with $N = 40$. Note that the use of a positivity constraint tends to moderate the estimate in those cases where the SNR is low, just as the use of statistical knowledge in the minimum-variance processor did. However, just as in the minimum-variance case, positivity appears to be of little help in performance improvement at SNR levels that are probably the minimum useful levels.* We also performed simulations at SNR_{REF} levels higher and lower than the 50 dB used to generate the results of Fig.'s 6.5 through 6.7. However, there were no surprises in the results. At $\text{SNR}_{\text{REF}} > 50$ dB the performance in the region where support is less than the reciprocal of the gap in the MTF "island" was the same with and without the positivity constraint. At $\text{SNR}_{\text{REF}} < 50$ dB the performance with positivity was marginally better than the performance without positivity in the same support region, with the performance gap increasing as SNR_{REF} is lowered. However, it is questionable whether the performance improvement with positivity is useful at such low levels of SNR_{EST} .

The above results were obtained with assumed and actual object support the same. The question arose as to the result if assumed object support were larger than actual object support, i.e., the object estimate the processor is allowed to formulate has a larger support than the actual size of the object. Until we introduced the positivity constraint, this question had no meaning. The minimum-variance processor requires second moment information about the object, and thus its size. Therefore, assumed and actual object size cannot be different. For unweighted least-squares without positivity, it can be shown that performance depends only on assumed object support, and as we have shown in Ref. 1, this must be less than the reciprocal of the gap in the MTF "islands" for good performance. However, for the case of unweighted least-squares with positivity, the results were somewhat surprising. We found that the critical size is neither the assumed nor actual object support, but their average, i.e., if the average of the assumed and actual object supports is smaller than the "gap", then SNR_{EST} will not fall off the "cliff". This result is illustrated in Fig.'s 6.8 through 6.10, corresponding to the four, five, and six subaperture arrays respectively. The same results are shown in Fig.'s 6.11 through 6.13 with an expanded vertical scale to show more detail. (The simulation results shown in Fig.'s 6.8, 6.9, 6.11, and 6.12 used $N = 40$ random trials, and the simulation results shown in Fig.'s 6.10 and 6.13 used $N = 20$ random trials.) Although this feature of positivity is interesting, it is hard to imagine a situation in which it would be useful.

6.5 CLEAN

The computational complexity of unweighted least-squares with a positivity constraint compels one to find a less burdensome algorithm that also incorporates a positivity constraint on the object estimate. One such algorithm is CLEAN², used in radio interferometry to reconstruct objects from

* Note that an SNR_{EST} of 20 dB corresponds to a "voltage" ratio of 10.

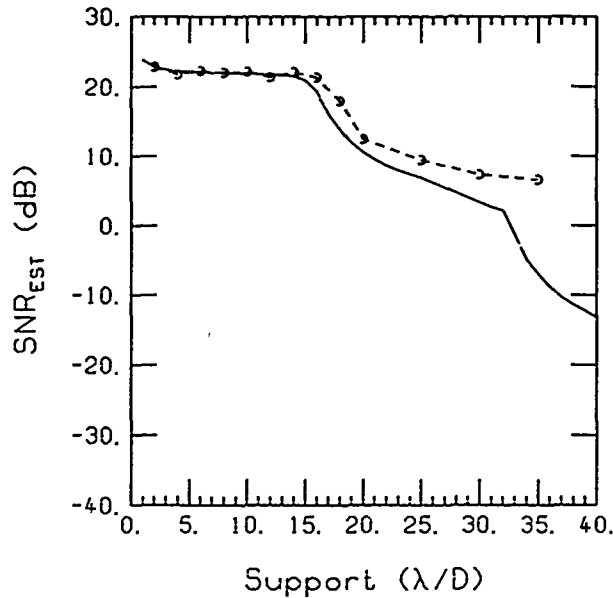


Figure 6.7.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using the unweighted least-square processor without and with a positivity constraint on the object estimate (solid line and circles, respectively). The six-subaperture array of Fig. 6.4 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

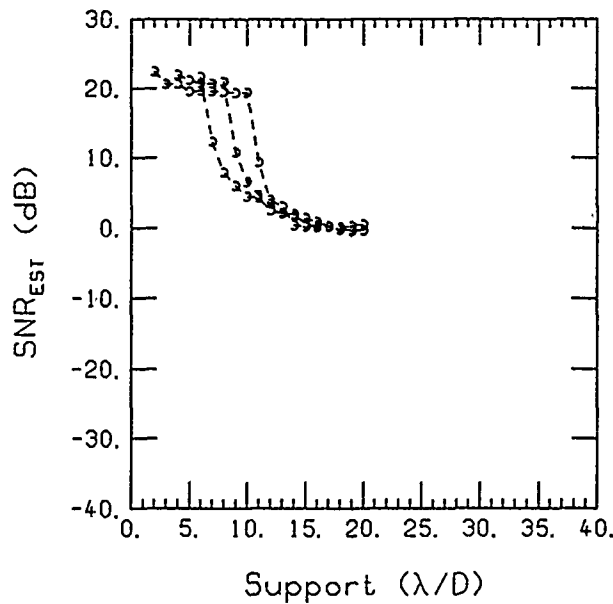


Figure 6.8.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The four-subaperture array of Fig. 6.2 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. From right to left, the three curves correspond to actual object sizes of 2, 4, and $6 \lambda/D$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

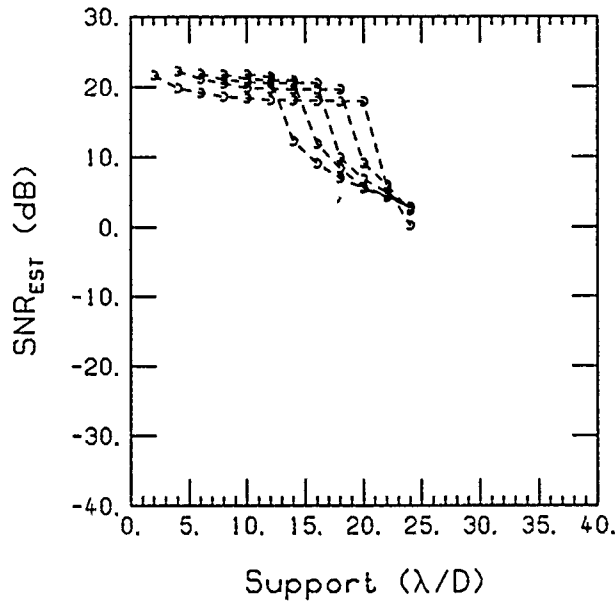


Figure 6.9.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The five-subaperture array of Fig. 6.3 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. From right to left, the five curves correspond to actual object sizes of 2, 4, 6, 8, and $10 \lambda/D$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

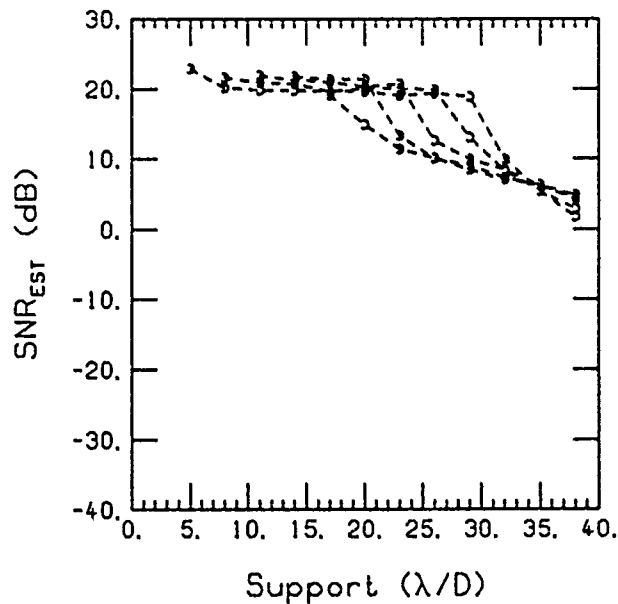


Figure 6.10.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The six-subaperture array of Fig. 6.4 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. From right to left, the five curves correspond to actual object sizes of 5, 8, 11, 14, and $17 \lambda/D$. There were $N = 20$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

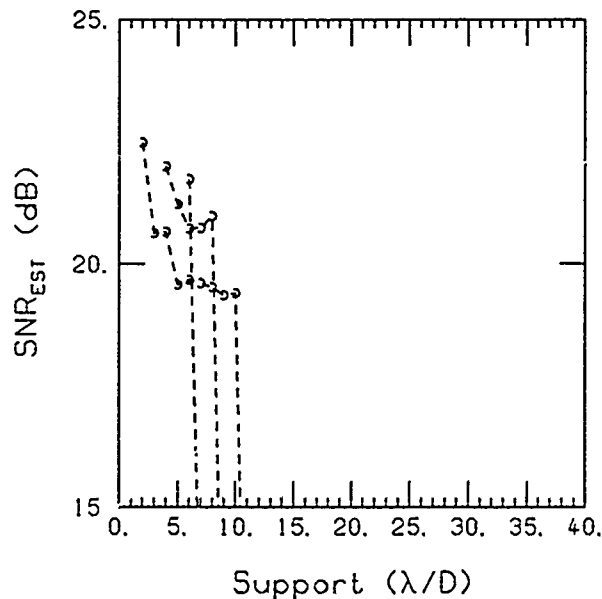


Figure 6.11.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The four-subaperture array of Fig. 6.2 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. From right to left, the three curves correspond to actual object sizes of 2, 4, and $6 \lambda/D$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

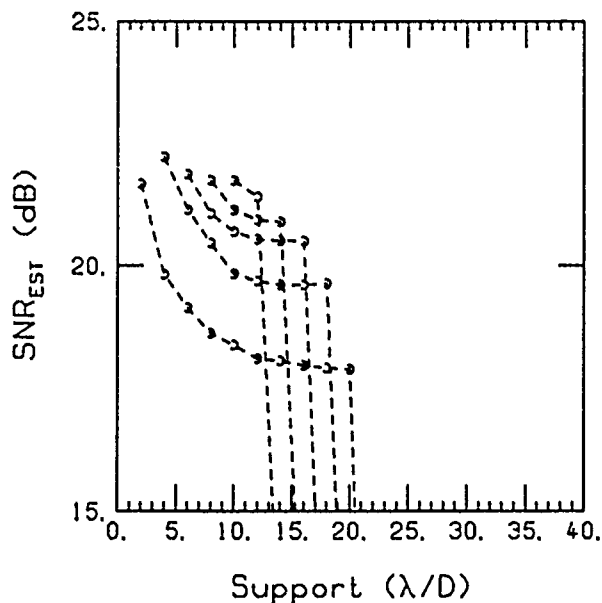


Figure 6.12.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The five-subaperture array of Fig. 6.3 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. From right to left, the five curves correspond to actual object sizes of 2, 4, 6, 8, and $10 \lambda/D$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

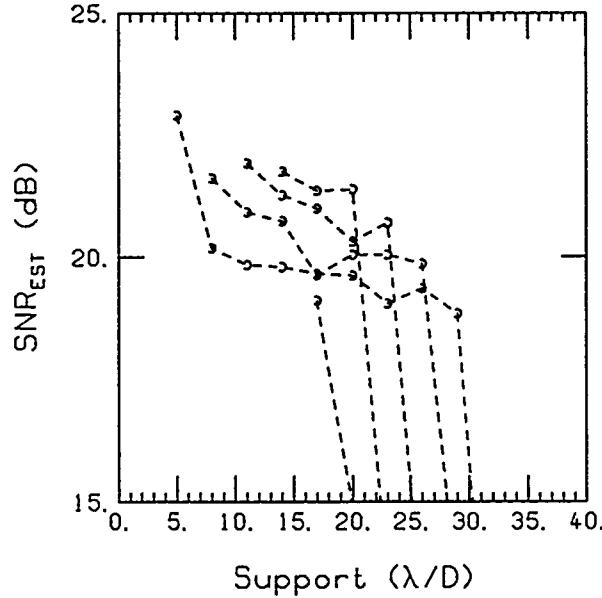


Figure 6.13.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The six-subaperture array of Fig. 6.4 was used with $SNR_{REF} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. From right to left, the five curves correspond to actual object sizes of 5, 8, 11, 14, and 17 λ/D . There were $N = 20$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

interferometric data. Since radio interferometers certainly employ sparse arrays, we were curious as to whether CLEAN was subject to the same limitations vis-a-vis object support and MTF gaps as the other algorithms that we have tried. CLEAN begins with the same observation model that we have used throughout our investigation:

$$y = Gx + n, \quad (6.33)$$

where x is the object vector, n is observation noise, and the columns of G are the system point-spread-function shifted and truncated. CLEAN is an iterative attempt to solve the system of equations

$$y = G\hat{x}, \quad (6.34)$$

with the positivity constraint

$$\hat{x} \geq 0, \quad (6.35)$$

where y and G are given, and $\hat{x}$ is unknown. If the solution exists (in our case that is equivalent to no noise), and under the appropriate circumstances, CLEAN is capable of converging to the exact solution. However, if there is observation noise, then CLEAN will not in general converge to the best solution in the constrained least-squares sense, i.e., CLEAN will not minimize ϵ of Eq. (6.30) with the constraint given by Eq. (6.31). Furthermore, as we shall see, CLEAN does not in general perform as well in terms of SNR_{EST} as least-squares. Details of our implementation of CLEAN are contained in Appendix E.

Simulations using CLEAN were carried out in the same way as for unweighted least-squares with positivity. Independent Rayleigh and Gaussian random variables were generated to create object and noise vectors, the image vector y was computed, and CLEAN was applied to form an estimate of the object vector. After many trials (in all cases presented in this report, we used $N = 40$ random trials), a sample error covariance matrix was computed using Eq. (6.32).

SNR_{EST} versus object support is plotted in Fig.'s 6.14, 6.15, and 6.16, for $\text{SNR}_{\text{REF}} = 50$ dB, 60 dB, and 70 dB, respectively. The circles represent the results of simulations using CLEAN and the solid curves are the results using unweighted least-squares. In all cases, the four-subaperture nonredundant array of Fig. 6.2 was used. The first characteristic to note in all three figures is that CLEAN exhibits the same "cliff" behavior as the other algorithms we have tried; object support must be smaller than the reciprocal of the gaps between the "islands" in the MTF or SNR_{EST} falls off the "cliff". Also, the positivity constraint intrinsic in CLEAN moderates the object estimate at low SNR, just as does positivity applied to unweighted least-squares or statistical knowledge in the minimum-variance processor. Finally, we note that, to the left of the "cliff", that CLEAN tracks least-squares with about a 14 dB performance loss. We will explore the reason for this loss shortly. However, we will now look at the CLEAN results using the other two nonredundant arrays shown in Fig.'s 6.3 and 6.4 and discover a limitation of CLEAN not suffered by the other algorithms we have tried. We begin with the 5-subaperture array of Fig. 6.3 and the corresponding CLEAN results of Fig. 6.17 with $\text{SNR}_{\text{REF}} = 50$ dB. There are no surprises in this figure. The results are what we would expect in the light of previous results. Now consider the results using $\text{SNR}_{\text{REF}} = 60$ dB shown in Fig. 6.18. We now see a considerable falloff in CLEAN performance beyond an object support of $5\lambda/D$. The result is more exaggerated in Fig. 6.19 where $\text{SNR}_{\text{REF}} = 70$ dB. The "cliff" appears to be located at $5\lambda/D$ instead of the expected location of 10 to 11 λ/D . For the present, we simply point out that the MTF of the five-subaperture array has a missing "island", so that one of the gaps is $2/11D/\lambda$, the reciprocal of which is $5.5 \lambda/D$. Now consider the results shown in Fig.'s 6.20 through 6.22 using the six-subaperture array of Fig. 6.4. Again we have a severe loss in performance for object support exceeding $5 \lambda/D$ when we would expect level performance out to a support of 16 to 17 λ/D . And again, we note that the six-subaperture array has two adjacent missing "islands" in its MTF with a corresponding gap of $3/17D/\lambda$, the reciprocal of which is $5.67 \lambda/D$. Apparently, CLEAN is confused by irregularly spaced "islands" in the MTF of the array. In other words, CLEAN appears to require that object support be less than the reciprocal of the largest gap in the MTF of the array, even though that gap is outside the cutoff frequency used in computing SNR_{EST} . We gain additional insight by examining the energy spectrum of the error vector when using CLEAN. Fig. 6.23 contains curves of energy spectrum versus spatial frequency using the five-subaperture array. Note that CLEAN is successfully interpolating between "islands" in the MTF for objects with support less than $12 \lambda/D$, but the performance gradually degrades beyond a support of $5 \lambda/D$. Fig. 6.24 reveals the same behavior using the six-subaperture array. The obvious solution is to spatially filter the observation vector $\mathbf{y}$ with a low-pass filter with a cutoff frequency equal to the upper bound of the "useful range" of the array. The point-spread function used in the CLEAN algorithm would be adjusted to reflect the "new" MTF. To demonstrate the concept while avoiding the filtering operation, we instead created two redundant arrays with the same "island" spacing as the nonredundant five and six subaperture arrays. These arrays and their corresponding MTFs are shown in Fig.'s 6.25 and 6.26, respectively, and the results of the simulations are shown in Fig.'s 6.27 through 6.32. Note that in every case we now have the behavior that we originally expected with the nonredundant arrays.

We now refer back to one of the early observations, that the performance of CLEAN is worse than least-squares in the region where object support is less than the reciprocal of the "gap" size in the MTF. In an optical system using a completely filled (nonsparse) aperture, there is very little information in the image plane about the object being imaged outside the support region of the object. This not the case with a very sparse array. As the "sparseness" is increased, information about the object is spread further out into the image plane, considerable beyond the region of object support. All of this information is potentially useful in a signal-to-noise ratio sense, and the algorithm that can use it will perform better than one that cannot. However, there is usually a point of diminishing returns, where increasing the size of the observation region will no longer significantly improve performance because of the increased amount of observation noise. In all the results that we have presented, both in this report and in Ref. 28, using minimum-variance and least-squares

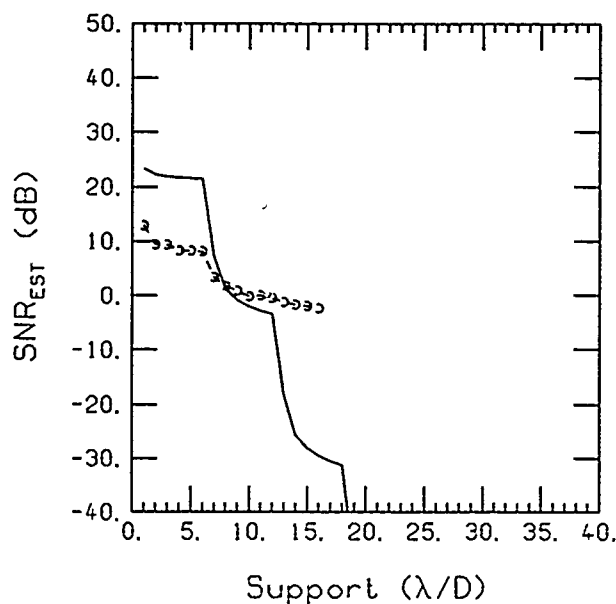


Figure 6.14.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The four-subaperture array of Fig. 6.2 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

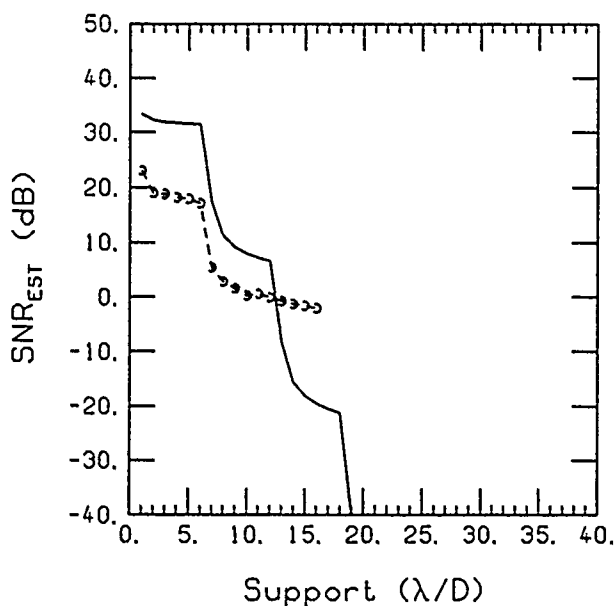


Figure 6.15.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The four-subaperture array of Fig. 6.2 was used with $\text{SNR}_{\text{REF}} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

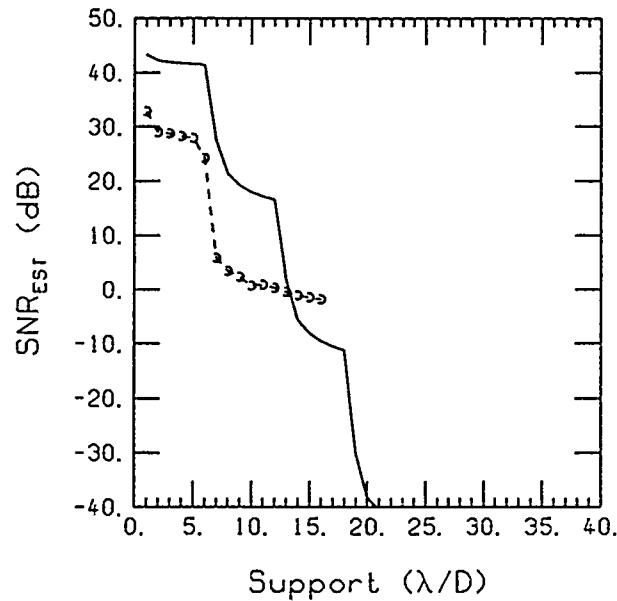


Figure 6.16.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The four-subaperture array of Fig. 6.2 was used with $\text{SNR}_{\text{REF}} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

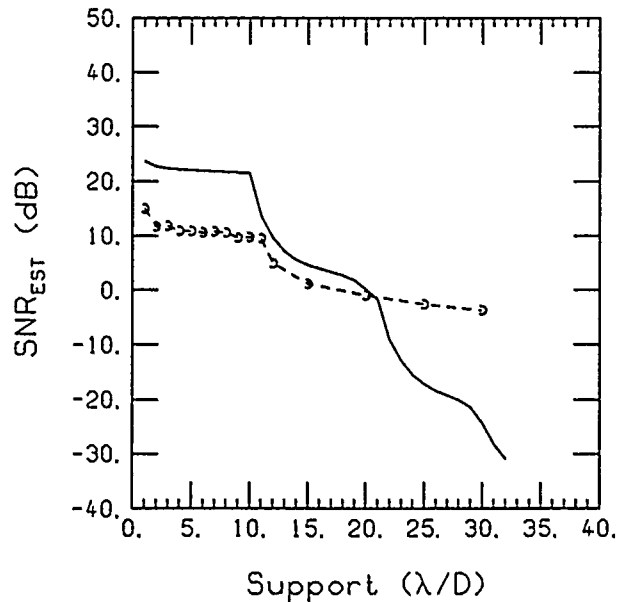


Figure 6.17.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The five-subaperture array of Fig. 6.3 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

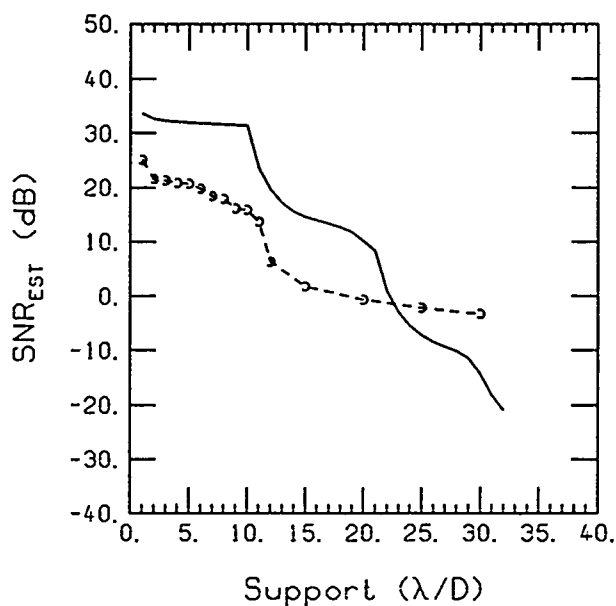


Figure 6.18.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The five-subaperture array of Fig. 6.3 was used with $\text{SNR}_{\text{REF}} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

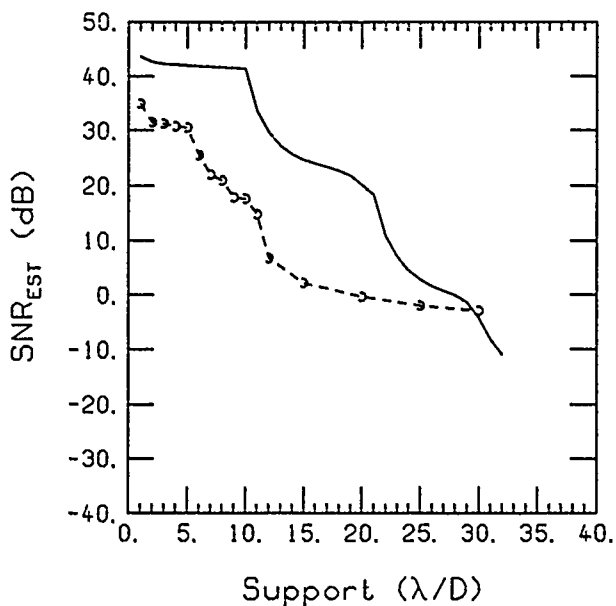


Figure 6.19.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The five-subaperture array of Fig. 6.3 was used with $\text{SNR}_{\text{REF}} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

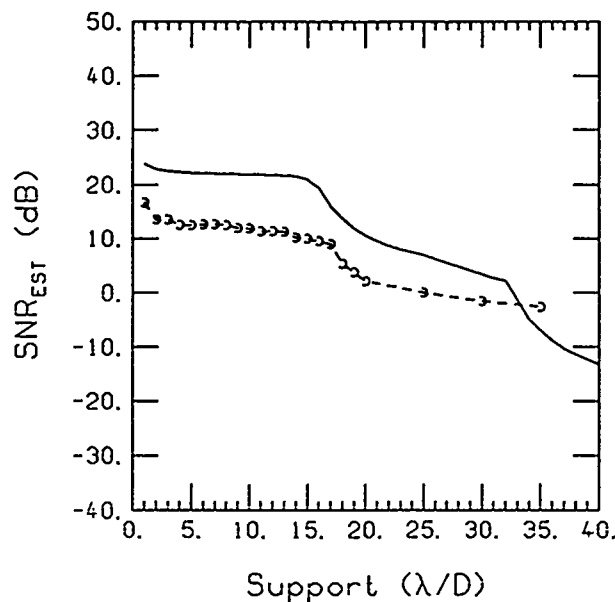


Figure 6.20.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The six-subaperture array of Fig. 6.4 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

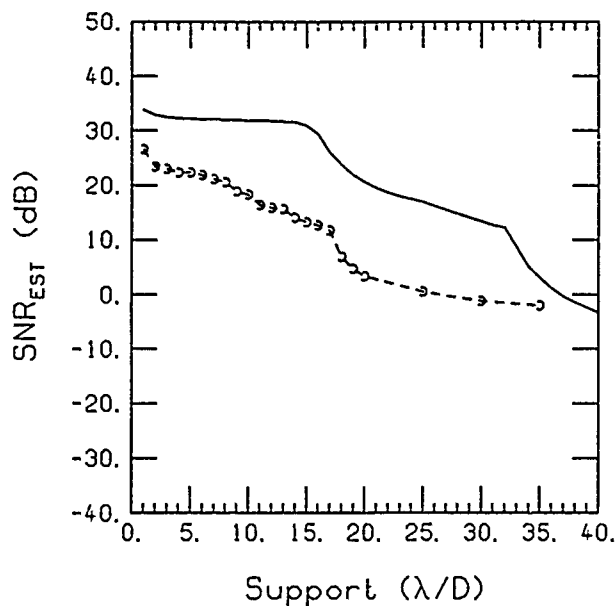


Figure 6.21.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The six-subaperture array of Fig. 6.4 was used with $\text{SNR}_{\text{REF}} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

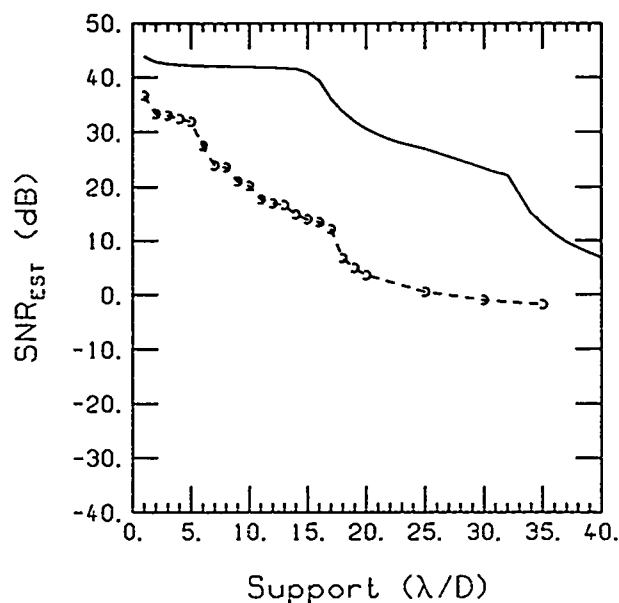


Figure 6.22.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The six-subaperture array of Fig. 6.4 was used with $\text{SNR}_{\text{REF}} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

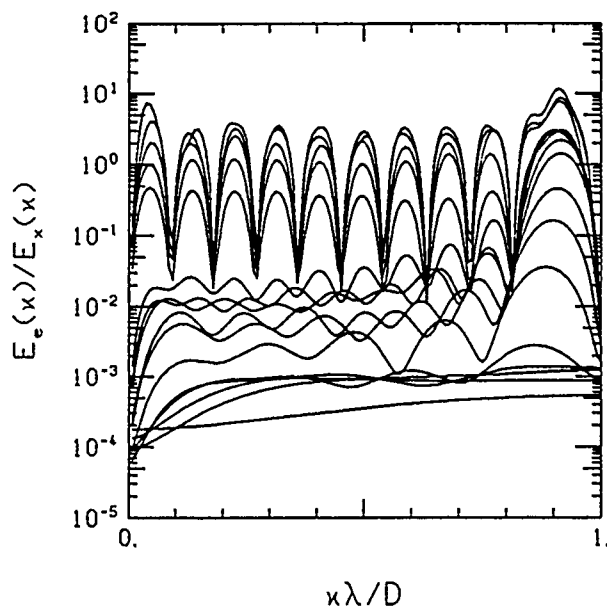


Figure 6.23.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ using the CLEAN algorithm and the five-subaperture array of Fig. 6.3 with $\text{SNR}_{\text{REF}} = 70$ dB and $d/D = 0.005$. From bottom to top, the sixteen curves correspond to object supports of $1\lambda/D$, $2\lambda/D$, ..., $12\lambda/D$, $15\lambda/D$, $20\lambda/D$, $25\lambda/D$, and $30\lambda/D$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

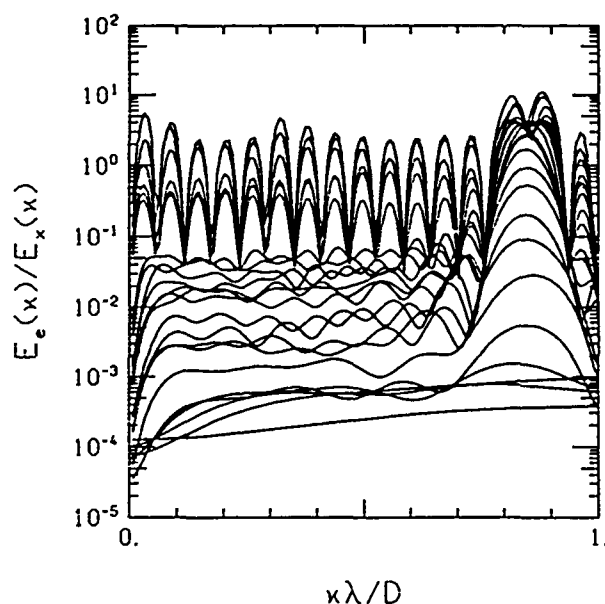
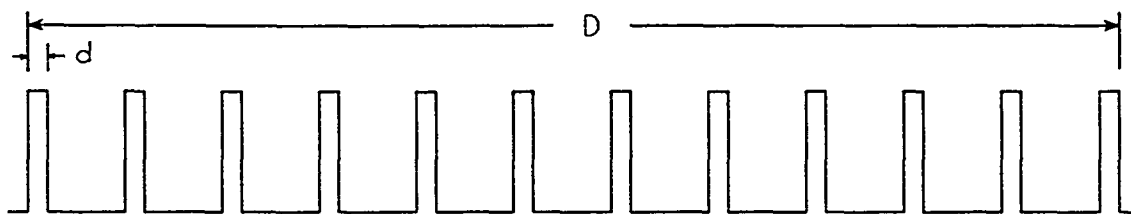


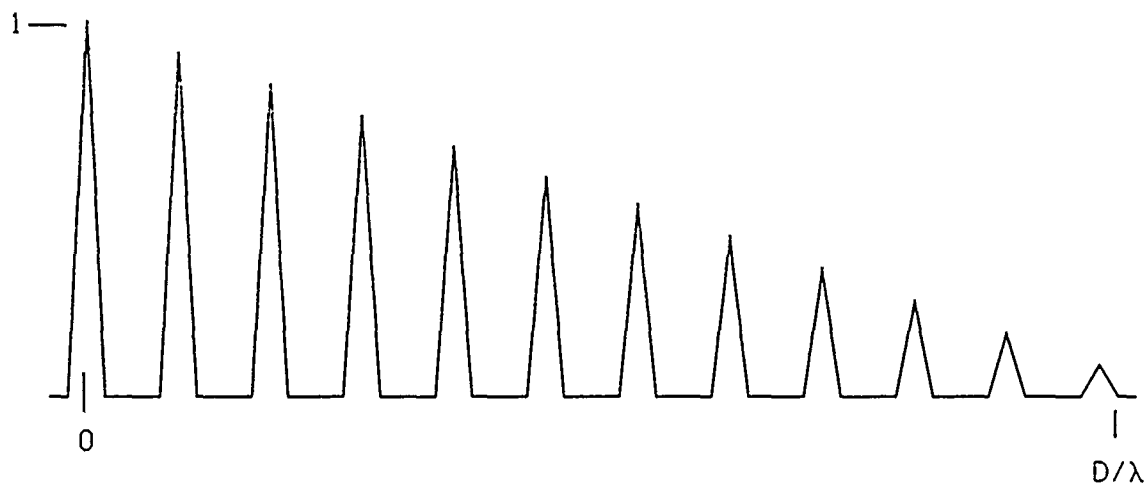
Figure 6.24.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ using the CLEAN algorithm and the six-subaperture array of Fig. 6.4 with $\text{SNR}_{\text{REF}} = 70$ dB and $d/D = 0.005$. From bottom to top, the twenty three curves correspond to object supports of $1\lambda/D$, $2\lambda/D$, ..., $20\lambda/D$, $25\lambda/D$, $30\lambda/D$, and $35\lambda/D$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$.

with and without positivity, we have "allowed" the algorithm to use as much of the image line as was necessary in order to maximize performance. In other words, from a standpoint of performance, one can consider the dimension of the observation vector y to be infinite. In experiments with CLEAN, we found that the performance did not monotonically improve as the size of the image line was increased beyond the size of the object. In fact, performance was erratic beyond that point, changing by as much as ± 4 dB in the region to the left of the "cliff" in the SNR_{EST} curve. It appears that CLEAN can not effectively utilize object information contained in the outlying region of the image line. Accordingly, our results are approximately equivalent to what we would have gotten if we had formulated the CLEAN algorithm to only use image measurement data from the set of pixels at the center of the image plane and having an extent just matching the actual size of the target object. It is as though we had limited the size of the observation vector used by CLEAN to match the object support. This leads to the question as to how least-squares would perform compared to CLEAN if the size of the observation vector was limited to object support for both algorithms. The answer is contained in Fig. 6.33. There is no difference in performance. We conclude from this that the difference in performance, for the smaller object supports, seen in Fig.'s 6.14 to 6.22 and 6.27 to 6.32 is due to the different "effective" size of the observation vectors for the two algorithms. It is CLEAN's inability to make use of the multiplicity of copies of the image that the array produces that causes the difference. It is not clear that this inability would manifest itself in processing radio astronomy data.



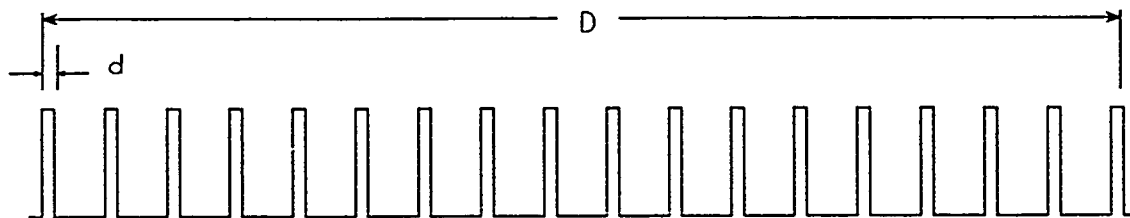
(a)



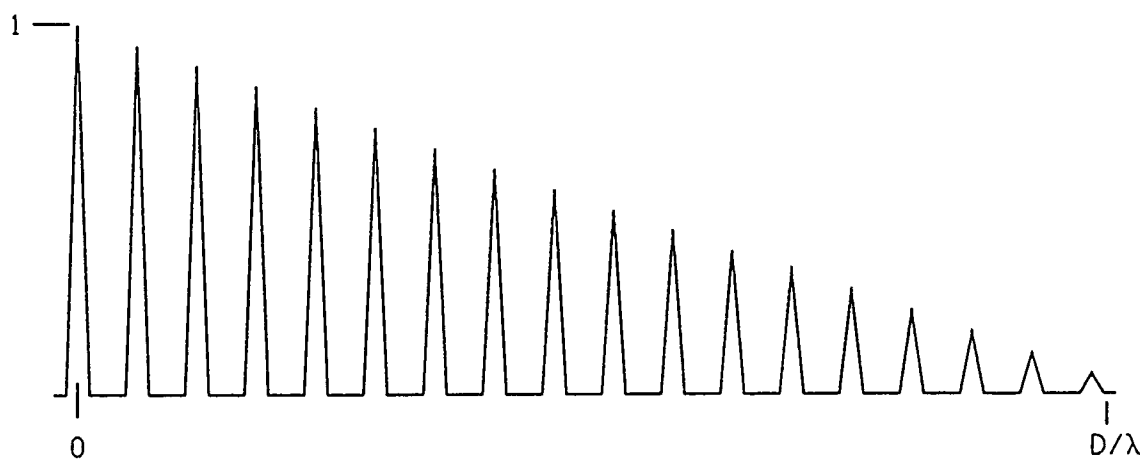
(b)

Figure 6.25.

Sparse array aperture function (a) and the corresponding MTF (b) for a redundant twelve- subaperture array.



(a)



(b)

Figure 6.26.

Sparse Array aperture function (a) and the corresponding MTF (b) for a redundant eighteen-subaperture array.

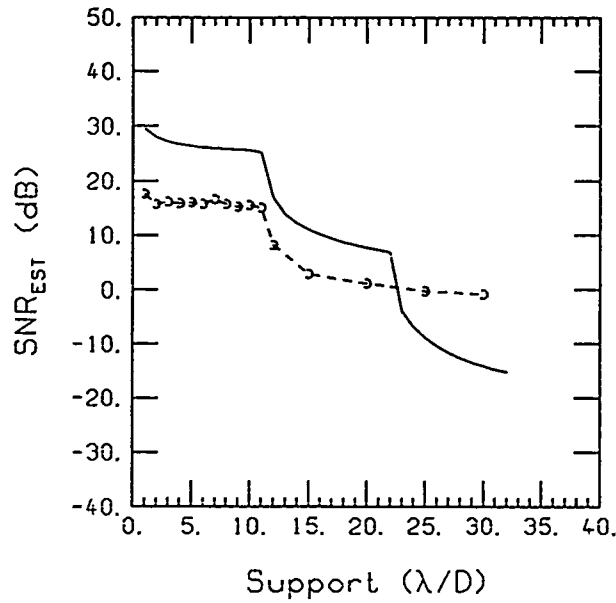


Figure 6.27.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The twelve-subaperture array of Fig. 6.25 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$ for the least-squares results and a length equal to object support for the CLEAN results.

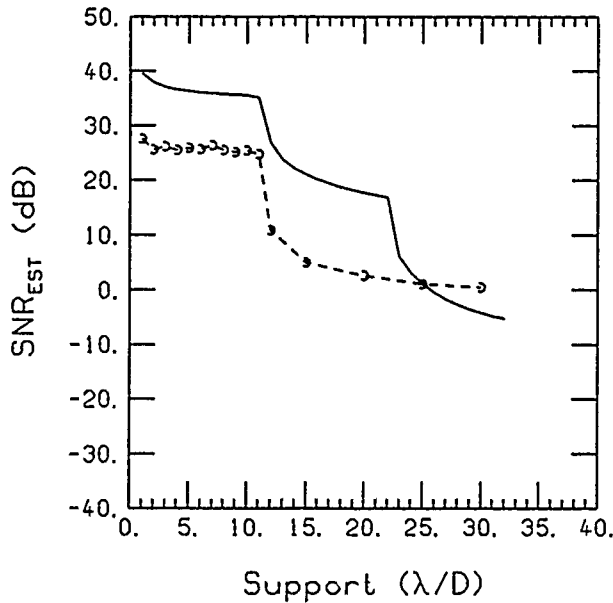


Figure 6.28.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The twelve-subaperture array of Fig. 6.25 was used with $\text{SNR}_{\text{REF}} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$ for the least-squares results and a length equal to object support for the CLEAN results.

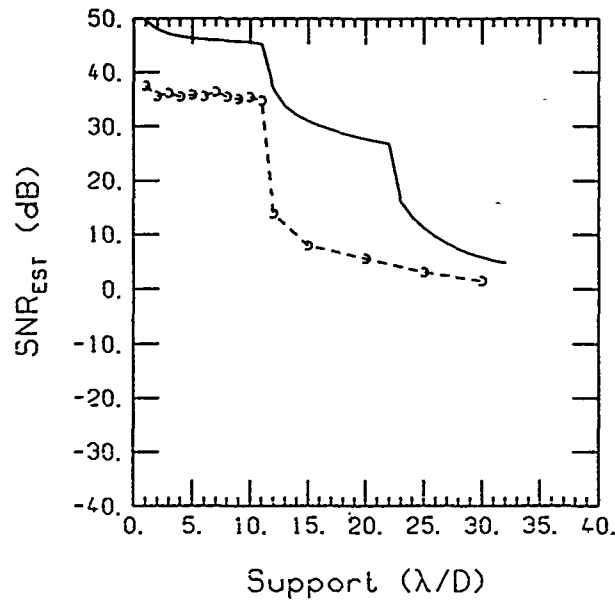


Figure 6.29.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The twelve-subaperture array of Fig. 6.25 was used with $\text{SNR}_{\text{REF}} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$ for the least-squares results and a length equal to object support for the CLEAN results.

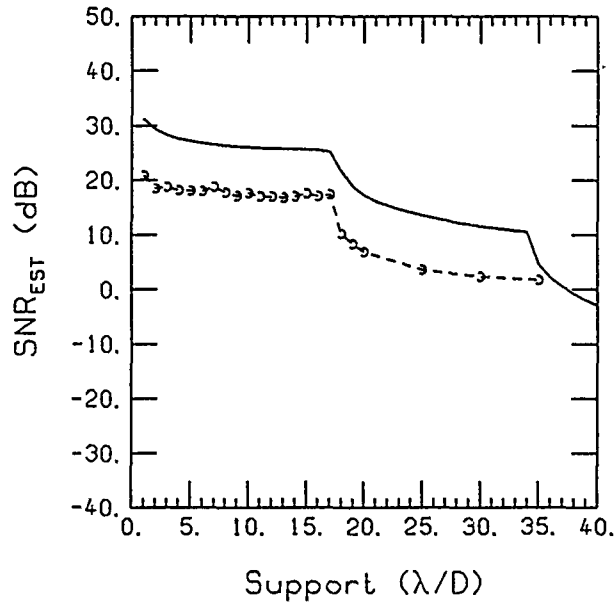


Figure 6.30.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The eighteen subaperture array of Fig. 6.26 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$ for the least-squares results and a length equal to object support for the CLEAN results.

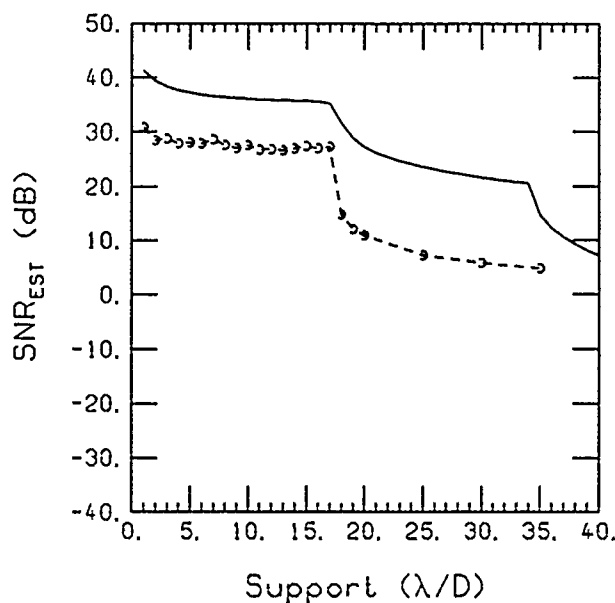


Figure 6.31.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The eighteen-subaperture array of Fig. 6.31 was used with $\text{SNR}_{\text{REF}} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$ for the least-squares results and a length equal to object support for the CLEAN results.

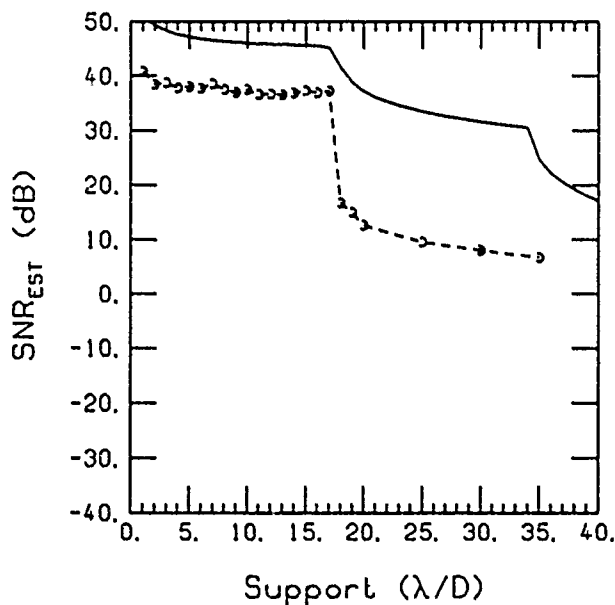


Figure 6.32.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The eighteen-subaperture array of Fig. 6.26 was used with $\text{SNR}_{\text{REF}} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, y , used here had a length of $256 \lambda/D$ for the least-squares results and a length equal to object support for the CLEAN results.

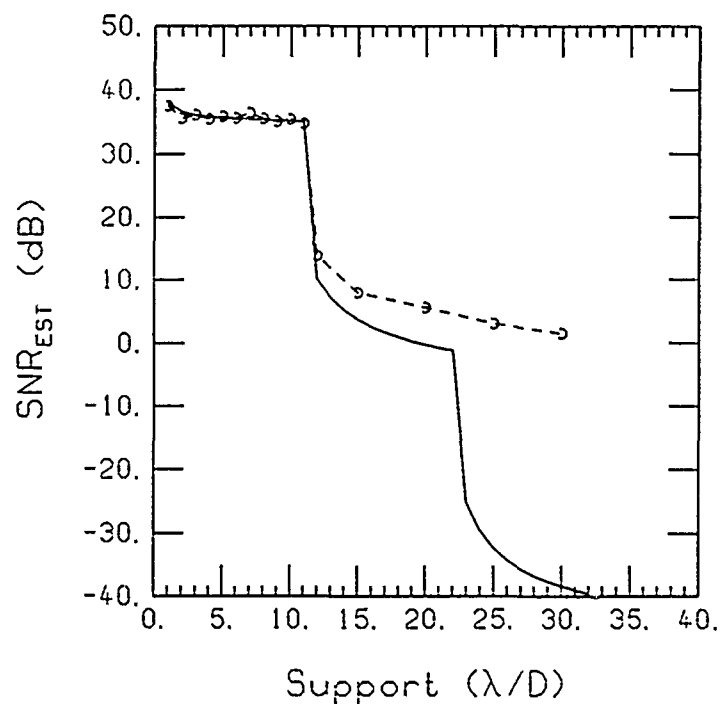


Figure 6.33.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The twelve-subaperture array of Fig. 25 was used with $\text{SNR}_{\text{REF}} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. In all cases, i.e., for both CLEAN and least-squares results, the length of the image line was the same as the length of the object support. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here.

REFERENCES

1. I. J. Cox and C. J. R. Sheppard, "Information Capacity and Resolution in an Optical System," *J. Opt. Soc. Am. A* 3, 1152-1158 (1987).
2. A. R. Thompson, J. M. Moran, and G. W. Swenson, Jr., *Interferometry and Synthesis in Radio Astronomy*, John Wiley and Sons, New York (1986).
3. R. W. Schafer, R. M. Mersereau, and M. A. Richards, "Constrained Iterative Restoration Algorithms," *Proc. IEEE* 69, 432-450 (1981).
4. R. Barakat, "Dilute Aperture Diffraction Imagery and Object Reconstruction," *Opt. Eng.* 29, 131-139 (1990).
5. W. Lukosz, "Optical Systems with Resolving Power Exceeding the Classical Limit," *J. Opt. Soc. Am.* 56, 1463-1472 (1966).
6. W. Lukosz, "Optical Systems with Resolving Power Exceeding the Classical Limit II," *J. Opt. Soc. Am.* 57, 932-941 (1967).
7. C. W. Barnes, "Object Restoration in a Diffraction-Limited Imaging System," *J. Opt. Soc. Am.* 56, 575-578 (1966).
8. B. R. Frieden, "Band-Unlimited Reconstruction of Optical Objects and Spectra," *J. Opt. Soc. Am.* 57, 1013-1019 (1967).
9. J. L. Harris, "Diffraction and Resolving Power," *J. Opt. Soc. Am.* 54, 931-936 (1964).
10. C. K. Rushforth and R. W. Harris, "Restoration, Resolution, and Noise," *J. Opt. Soc. Am.* 58, 539-545 (1968).
11. H. A. Brown, "Effect of Truncation on Image Enhancement by Prolate Spheroidal Functions," *J. Opt. Soc. Am.* 59, 228-229 (1969).
12. G. T. Herman and D. Ro, "Image Recovery Using Iterative Data Refinement with Relaxation," *Opt. Eng.* 29, 513-523 (1990).
13. J. R. Fienup, "Reconstruction of an Object From the Modulus of Its Fourier Transform," *Opt. Lett.* 3, 27-29 (1978).
14. J. R. Fienup, "Phase Retrieval Algorithms: A Comparison," *Appl. Opt.* 21, 2758-2769 (1982).
15. S. J. Reeves and R. M. Mersereau, "Optimal Estimation of the Regularization Parameter and Stabilizing Functional for Regularized Image Restoration," *Opt. Eng.* 29, 446-454 (1990).
16. R. Barakat, "Phase Retrieval in Two Dimensions," *RADC-TR-88-251*, 26-67 (1988).
17. F. T. S. Yu, *Optics and Information Theory*, John Wiley and Sons, New York, 141-168 (1976).
18. G. Toraldo di Francia, "Resolving Power and Information," *J. Opt. Soc. Am.* 45, 497 (1955).
19. H. S. Coleman and M. F. Coleman, "Theoretical Resolution Angles for Point and Line Test Objects in the Presence of a Luminous Background," *J. Opt. Soc. Am.* 37, 572 (1947).
20. V. Ranchi, *Optics, The Science of Vision*, New York University Press, New York, (1957).
21. V. Ranchi, "Resolving Power of Calculated and Detected Images," *J. Opt. Soc. Am.* 51, 458 (1961).
22. G. Toraldo di Francia, *Nuovo Cimento Suppl.* 3 IX, 426-438 (1952).
23. P. B. Fellgett and E. H. Linfoot, *Proc. R. Soc. London Ser. A* 247, 369-407 (1955).
24. A. Lannes, S. Roques, and M. J. Casanove, "Stabilized Reconstruction in Signal and Image Processing; Part I: Partial Deconvolution and Spectral Extrapolation with Limited Field," *J. Mod. Opt.* 34, 161-226 (1987).
25. A. Lannes, M. J. Casanove, and S. Roques, "Stabilized Reconstruction in Signal and Image Processing; Part II: Iterative Reconstruction With and Without Constraint. Interactive Implementation," *J. Mod. Opt.* 34, 321-370 (1987).
26. A. Lannes, S. Roques, and M. J. Casanove, "Resolution and Robustness in Image Processing: A New Regularization Principle," *J. Opt. Soc. Am.* 4, 189-199 (1987).
27. J. A. Roberts, Editor, *Indirect Imaging*, Proceedings of an International Symposium, Sydney Australia, 30 August to 2 September 1983, Cambridge University Press, London (1984).
28. D. T. Sherwood and D. L. Fried, "Object Reconstruction with Sparse Arrays of Optical Apertures. Part I, Linear Methods," (tOSC Report No. TR-1070, May 1990).

29. R. L. Fox, *Optimization Methods for Engineering Design*, Addison-Wesley, Reading Massachusetts, 1971, p. 196.

APPENDIX A for Chapter 2

Computer Listings

This appendix contains the Fortran source code of the computer programs used to generate the data for this chapter.

```

1  c
2  c
3  c      program 2dmtfmr.f
4  c
5  c
6      real*4 z1(128,128)
7      d=1.0
8      call eight(d,z1)
9      stop
10     end
11  c
12  c
13  c
14      subroutine eight(d,z1)
15      real*8 ha,a,b,c
16      real*4 z(64,64),z1(128,128),dx,dy
17      complex*8 berr(128,128)
18      a=0
19      b=2/3.*3.14159265
20      c=(4/3.)*3.14159265
21      do 150 jj=0,63
22          tk=-(jj-32)/32.
23          do 200 j=0,63
24              th=(j-32)/32.
25              call adder(a,a,d,tk,th,hh1)
26              call adder(b,a,d,tk,th,hh2)
27              call adder(a,b,d,tk,th,hh7)
28              call adder(a,c,d,tk,th,hh3)
29              call adder(b,c,d,tk,th,hh5)
30              call adder(c,a,d,tk,th,hh8)
31              call adder(c,b,d,tk,th,hh9)
32              if(d.eq.1)then
33                  z(j+1,jj+1)=hh1
34              else
35                  z(j+1,jj+1)=3*hh1+hh2+hh3+hh5+hh7+
36                      c      hh8+hh9
37              endif
38          200 continue
39      150 continue
40      call solidplt('2dmtfd1.0',64,64,z)
41      do 205 ii=1,128
42          do 205 iii=1,128
43      205      berr(ii,iii)=0
44      do 210 ii=1,64
45          do 220 iii=1,64
46              berr(32+ii,32+iii)=z(ii,iii)
47      220 continue
48      210 continue
49      dx=1
50      dy=1
51      call fft2d(berr,128,dx,dy)
52      do 300 ii=1,128
53          do 310 iii=1,128
54              z1(ii,iii)=real(berr(ii,iii))
55      310 continue
56      300 continue
57      call solidplt('2dpsfd1.0',128,128,z1)
58      return
59      end
60  c

```

```

61      function ha(d,de)
62      real*8 ha
63      ha=.5*(d**2)*(1.57079632-asin(de/d)-
64      c      (de/d)*sqrt(1-(de/d)**2))/ .78539816
65      return
66      end
67  c
68      subroutine adder(p1,p2,d,tk,th,hh)
69      real*8 ha,p1,p2
70      r1=(1-d)/2*(cos(p1)-cos(p2))+th
71      r2=(1-d)/2*(sin(p1)-sin(p2))+tk
72      de=sqrt(r1**2+r2**2)
73      if(de.le.d)then
74          hh=ha(d,de)
75      else
76          hh=0
77      endif
78      return
79      end

```

```

1  c
2  c
3  c      program first.f
4  c
5  c
6      parameter(nn=157, kk=68)
7      real*8 gx(kk), gy(kk)
8      integer idummy(1)
9  c
10     n=157
11     m=4
12     m2=(m**2)*(m**2+1)/4
13     l=31
14     k=63
15     d=.1
16     call bufasg(idummy, 128*n*m, ia)
17     call bufasg(idummy, 2*128*128, ib)
18     call zero(idummy(ia), n, m, idummy(ib), d, gx, gy, m2)
19     call bufrel(idummy(ib))
20     call bufrel(idummy(ia))
21     call plotfl('g14', m2, 'd', gx, 'd', gy)
22     stop
23     end
24  c
25  c
26     subroutine zero(bw, n, m, z1, d, gx, gy, m2)
27     real*8 bw(64, n, m), z1(128, 128), gx(m2), gy(m2)
28     call eight(d, z1)
29     call one(bw, n, m, z1, d)
30     call two(gx, gy, bw, n, m, m2)
31     return
32     end
33  c
34  c
35     subroutine one(bw, n, m, z1, d)
36     real*8 z1(128, 128), bw(64, n, m)
37     do 500 i=1, 64
38         do 501 ii=1, n
39             do 502 iii=1, m
40                 bw(i, ii, iii)=0
41 502         continue
42 501     continue
43 500     continue
44         do 510 i=1, 64
45             do 520 ii=1, 127
46                 bw(i, ii, 1)=z1(64+i, ii+1)
47 520         continue
48             do 530 iii=2, m
49                 do 540 iv=1, 127
50                     bw(i, iii+iv-1, iii)=bw(i, iii+iv-2, iii-1)
51 540             continue
52 530         continue
53 510     continue
54         return
55     end
56  c
57  c
58  c
59     subroutine two(gx, gy, bw, n, m, m2)
60     parameter(mm=4)
61     real*8 gl(mm, mm), gb(mm, mm), gx(m2), gy(m2), bw(64, n, m)
62     iv=1
63     do 600 i=1, m
64         do 605 iii=1, m
65             do 605 iv=1, m
66 605         gb(iii, iv)=0
67         j=1
68         ia=i

```

```

69      do 610 ii=1,64
70      call four(bw,ii,abs(ia-ii)+1,gl,m,n)
71      do 620 iii=1,m
72      do 630 iv=1,m
73      gb(iii,iv)=gb(iii,iv)+gl(iii,iv)
74 630      continue
75 620      continue
76 610      continue
77      jg=m*(j-1)
78      ig=m*(ia-1)
79      if(ia.ne.j)then
80      do 640 k1=1,m
81      do 650 k2=1,m
82      if(iw.gt.0)then
83      gx(iw)=gb(k1,k2)
84      iw=-iw
85      else
86      gy(abs(iw))=gb(k1,k2)
87      iw=-iw+1
88      endif
89 650      continue
90 640      continue
91      else
92      do 645 k1=1,m
93      do 655 k2=k1,m
94      if(iw.gt.0)then
95      gx(iw)=gb(k1,k2)
96      iw=-iw
97      else
98      gy(abs(iw))=gb(k1,k2)
99      iw=-iw+1
100      endif
101 655      continue
102 645      continue
103      endif
104      do 680 ix=2,m-i+1
105      j=j+1
106      jg=m*(j-1)
107      ia=ia+1
108      ig=m*(ia-1)
109      call four(bw,j,ia,gl,m,n)
110      if(j.ne.ia)then
111      do 660 k3=1,m
112      do 670 k4=1,m
113      gb(k3,k4)=gb(k3,k4)+gl(k3,k4)
114      if(iw.gt.0)then
115      gx(iw)=gb(k3,k4)
116      iw=-iw
117      else
118      gy(abs(iw))=gb(k3,k4)
119      iw=-iw+1
120      endif
121 670      continue
122 660      continue
123      else
124      do 665 k3=1,m
125      do 675 k4=k3,m
126      gb(k3,k4)=gb(k3,k4)+gl(k3,k4)
127      if(iw.gt.0)then
128      gx(iw)=gb(k3,k4)
129      iw=-iw
130      else
131      gy(abs(iw))=gb(k3,k4)
132      iw=-iw+1
133      endif
134 675      continue
135 665      continue
136      endif

```

```

137 680      continue
138 600      continue
139          return
140          end
141 c
142 c
143 c
144 c
145      subroutine four(bw,j,i,gl,m,n)
146      real*8 gl(m,m),bw(64,n,m)
147      do 400 k1=1,m
148          do 410 k2=1,m
149              gl(k1,k2)=0
150              do 420 k3=1,n
151                  gl(k1,k2)=gl(k1,k2)+bw(j,k3,k1)*bw(i,k3,k2)
152 420          continue
153 410      continue
154 400      continue
155          return
156          end
157 c
158 c
159 c
160 c
161 c
162 c
163      subroutine eight(d,z1)
164      real*8 ha,a,b,c,z1(128,128)
165      real z(64,64),dx,dy
166      complex*8 berr(128,128)
167      a=0
168      b=2/3.*3.14159265
169      c=(4/3.)*3.14159265
170      do 150 jj=0,63
171          tk=-(jj-32)/32.
172          do 200 j=0,63
173              th=(j-32)/32.
174              call adder(a,a,d,tk,th,hh1)
175              call adder(b,a,d,tk,th,hh2)
176              call adder(a,b,d,tk,th,hh7)
177              call adder(a,c,d,tk,th,hh3)
178              call adder(b,c,d,tk,th,hh5)
179              call adder(c,a,d,tk,th,hh8)
180              call adder(c,b,d,tk,th,hh9)
181              if(d.eq.1)then
182                  z(j+1,jj+1)=hh1
183              else
184                  z(j+1,jj+1)=3*hh1+hh2+hh3+hh5+hh7+
185  c          hh8+hh9
186              endif
187 200          continue
188 150      continue
189      call solidplt('solidfile',64,64,z)
190      do 205 ii=1,128
191          do 205 iii=1,128
192 205      berr(ii,iii)=0
193      do 210 ii=1,64
194          do 220 iii=1,64
195              berr(32+ii,32+iii)=z(ii,iii)
196 220          continue
197 210      continue
198      dx=1
199      dy=1
200      call fft2d(berr,128,dx,dy)
201      do 300 ii=1,128
202          do 310 iii=1,128
203              z1(ii,iii)=real(berr(ii,iii))
204 310          continue

```

```

205 300      continue
206          return
207          end
208 c
209          function ha(d,de)
210              real*8 ha
211              ha=.5*(d**2)*(1.57079632-asin(de/d)-
212                  c (de/d)*sqrt(1-(de/d)**2))/ .78539816
213              return
214              end
215 c
216          subroutine adder(p1,p2,d,tk,th,hh)
217              real*8 ha,p1,p2
218              r1=(1-d)/2*(cos(p1)-cos(p2))+th
219              r2=(1-d)/2*(sin(p1)-sin(p2))+tk
220              de=sqrt(r1**2+r2**2)
221              if(de.le.d)then
222                  hh=ha(d,de)
223              else
224                  hh=0
225              endif
226              return
227              end
228 c
229 c

```

```

1  c
2  c
3  c      program matrix4.f
4  c
5  c
6      parameter(nn=157,mm=8,kk=64)
7      real*8 z1(128,128),g(kk,kk),bw(64,nn,mm)
8      integer lwork(kk),mwork(kk)
9  c
10     n=157
11     m=8
12     m2=m**2
13     l=31
14     k=63
15     d=1
16     call eight(d,z1)
17     call one(bw,k,n,m,z1,d)
18     call two(g,bw,n,m,m2)
19  c      open(1,file='out1')
20     snr=10000
21     do 120 i=1,m2
22         g(i,i)=g(i,i)+1/snr
23 120     continue
24     call dminvw(g,m2,ier,lwork,mwork)
25     do 130 iii=1,m2
26         do 140 iv=1,m2
27             g(iii,iv)=g(iii,iv)/snr
28 140     continue
29 130     continue
30  c      write(1,*)g
31  c      close(1)
32     call ten(g,m,m2)
33     stop
34     end
35  c
36  c
37  c
38  c
39     subroutine one(bw,k,n,m,z1,d)
40     real*8 z1(128,128),bw(64,n,m)
41     do 500 i=1,64
42         do 501 ii=1,n
43             do 502 iii=1,m
44                 bw(i,ii,iii)=0
45 502         continue
46 501     continue
47 500     continue
48     do 510 i=1,64
49         do 520 ii=1,127
50             bw(i,ii,1)=z1(64+i,ii+1)
51 520         continue
52         do 530 iii=2,m
53             do 540 iv=1,127
54                 bw(i,iii+iv-1,iii)=bw(i,iii+iv-2,iii-1)
55 540         continue
56 530     continue
57 510     continue
58     return
59     end
60  c
61  c
62  c
63     subroutine two(g,bw,n,m,m2)
64     parameter(mm=8)
65     real*8 gl(mm,mm),gb(mm,mm),g(m2,m2),bw(64,n,m)
66     do 600 i=1,m
67         do 605 iii=1,m
68             do 605 iv=1,m

```

```

69 605      gb(iii,iv)=0
70          j=1
71          ia=i
72          do 610 ii=1,64
73              call four(bw,ii,abs(ia-ii)+1,gl,m,n)
74              do 620 iii=1,m
75                  do 630 iv=1,m
76                      gb(iii,iv)=gb(iii,iv)+gl(iii,iv)
77 630          continue
78 620      continue
79 610      continue
80          jg=m*(j-1)
81          ig=m*(ia-1)
82          do 640 k1=1,m
83              do 650 k2=1,m
84                  g(jg+k1,ig+k2)=gb(k1,k2)
85                  g(ig+k2,jg+k1)=gb(k1,k2)
86 650      continue
87 640      continue
88          do 680 ix=2,m-i+1
89              j=j+1
90              jg=m*(j-1)
91              ia=ia+1
92              ig=m*(ia-1)
93              call four(bw,j,ia,gl,m,n)
94              do 660 k3=1,m
95                  do 670 k4=1,m
96                      gb(k3,k4)=gb(k3,k4)+gl(k3,k4)
97                      g(jg+k3,ig+k4)=gb(k3,k4)
98                      g(ig+k4,jg+k3)=gb(k3,k4)
99 670      continue
100 660      continue
101 680      continue
102 600      continue
103          return
104          end
105      c
106      c
107      c
108      c
109      subroutine four(bw,j,i,gl,m,n)
110      real*8 gl(m,m),bw(64,n,m)
111      do 400 k1=1,m
112          do 410 k2=1,m
113              gl(k1,k2)=0
114              do 420 k3=1,n
115                  gl(k1,k2)=gl(k1,k2)+bw(j,k3,k1)*bw(i,k3,k2)
116 420      continue
117 410      continue
118 400      continue
119      return
120      end
121      c
122      c
123      c
124      c
125      c
126      c
127      subroutine eight(d,z1)
128      real*8 ha,a,b,c,z1(128,128)
129      real z(64,64),dx,dy
130      complex*8 berr(128,128)
131      a=0
132      b=2/3.*3.14159265
133      c=(4/3.)*3.14159265
134      do 150 jj=0,63
135          tk=-(jj-32)/32.
136          do 200 j=0,63

```

```

137      th=(j-32)/32.
138      call adder(a,a,d,tk,th,hh1)
139      call adder(b,a,d,tk,th,hh2)
140      call adder(a,b,d,tk,th,hh7)
141      call adder(a,c,d,tk,th,hh3)
142      call adder(b,c,d,tk,th,hh5)
143      call adder(c,a,d,tk,th,hh8)
144      call adder(c,b,d,tk,th,hh9)
145      if(d.eq.1)then
146        z(j+1,jj+1)=hh1
147      else
148        z(j+1,jj+1)=3*hh1+hh2+hh3+hh5+hh7+
149      c      hh8+hh9
150      endif
151 200      continue
152 150      continue
153 c      call solidplt('solidfile',64,64,z)
154      do 205 ii=1,128
155        do 205 iii=1,128
156 205        berr(ii,iii)=0
157      do 210 ii=1,64
158        do 220 iii=1,64
159          berr(32+ii,32+iii)=z(ii,iii)
160 220        continue
161 210      continue
162      dx=1
163      dy=1
164      call fft2d(berr,128,dx,dy)
165      do 300 ii=1,128
166        do 310 iii=1,128
167          z1(ii,iii)=real(berr(ii,iii))
168 310        continue
169 300      continue
170      return
171      end
172 c
173      function ha(d,de)
174      real*8 ha
175      ha=.5*(d**2)*(1.57079632-asin(de/d)-
176      c      (de/d)*sqrt(1-(de/d)**2))/ .78539816
177      return
178      end
179 c
180      subroutine adder(p1,p2,d,tk,th,hh)
181      real*8 ha,p1,p2
182      r1=(1-d)/2*(cos(p1)-cos(p2))+th
183      r2=(1-d)/2*(sin(p1)-sin(p2))+tk
184      de=sqrt(r1**2+r2**2)
185      if(de.le.d)then
186        hh=ha(d,de)
187      else
188        hh=0
189      endif
190      return
191      end
192 c
193 c
194 c      fft program
195      subroutine ten(g,m,m3)
196      parameter(mm=8)
197      real*4 diag(15),bdiag(128),wr(128),wi(128),dd,br(mm,mm,127)
198      real*4 bi(mm,mm,127),y(128),sur(127,127),sli(16129),x(16129)
199      real*8 gl(mm,mm),brt(mm,mm),bit(mm,mm),g(m3,m3)
200      real*4 amsur(63,63),x2(3969),sli2(3969)
201 c
202      n=128
203      nl=7
204      m2=2*m-1

```

```

205      call lookup(n,wr,wi)
206      do 10 i=1,m
207          do 20 ii=1,m
208              do 30 iii=1,m
209                  do 40 iv=1,m
210                      gl(iii,iv)=g(m*i-m+iii,m*ii-m+iv)
211 40      continue
212 30      continue
213      call adddiag(gl,m2,diag,m)
214      call pad(diag,m2,bdiag,n)
215      do 50 ix=1,n
216          y(ix)=0
217 50      continue
218      dd=1
219      call fft2(bdiag,y,n,nl,wr,wi,dd)
220      do 60 iii=1,n-1
221          br(i,ii,iii)=bdiag(iii+1)
222          br(ii,i,iii)=bdiag(iii+1)
223          bi(i,ii,iii)=y(iii+1)
224          bi(ii,i,iii)=-y(iii+1)
225 60      continue
226 20      continue
227 10      continue
228      do 70 i=1,n-1
229          do 80 ii=1,m
230              do 90 iii=1,m
231                  brt(ii,iii)=br(ii,iii,i)
232                  bit(ii,iii)=bi(ii,iii,i)
233 90      continue
234 80      continue
235      call adddiag(brt,m2,diag,m)
236      call pad(diag,m2,bdiag,n)
237      call adddiag(bit,m2,diag,m)
238      call pad(diag,m2,y,n)
239      dd=1
240      call fft2(bdiag,y,n,nl,wr,wi,dd)
241      do 100 ii=1,n-1
242          sur(i,ii)=bdiag(ii+1)/m3
243          sli((n-1)*i-(n-1)+ii)=bdiag(ii+1)/m3
244          x((n-1)*(i-1)+ii)=(n-1)*(i-1)+ii
245 100      continue
246 70      continue
247      iii=0
248      do 110 i=1,63
249          do 120 ii=1,63
250              iii=iii+1
251              smsur(i,ii)=sur(32+i,32+ii)
252              sli2(iii)=smsur(i,ii)
253              x2(iii)=x(iii)
254 120      continue
255 110      continue
256      call solidplt('s24',127,127,sur)
257      call plotfl('long24',(n-1)**2,'f',x,'f',sli)
258      call solidplt('hi',63,63,smsur)
259      call plotfl('lo',3969,'f',x2,'f',sli2)
260      stop
261      end
262 c
263 c
264 c
265      subroutine pad(diag,m2,bdiag,n)
266      real*4 diag(m2),bdiag(n)
267      do 404 i=1,n
268          bdiag(i)=0
269 404      continue
270      k=(n-m2+3)/2
271      do 406 i=1,m2
272          bdiag(k+i-1)=diag(i)

```

```

273 406      continue
274          return
275          end
276 c
277 c
278 c
279          subroutine adddiag(a,m2,diag,m)
280          real*4 diag(m2)
281          real*8 a(m,m)
282          do 601 i=1,m2
283              diag(i)=0
284 601      continue
285          do 603 i=1,m
286              do 605 ii=1,m
287                  iii=i-ii+m
288                  diag(iii)=diag(iii)+a(i,ii)
289 605      continue
290 603      continue
291          return
292          end

```

```

1 c
2 c
3 c      program matrix5
4 c
5 c
6      parameter(nn=254,mm=128)
7      real*8 bw(nn,mm),retil(128),f(128)
8      real*8 g(mm,mm)
9      real*4 dx,dy,d
10 c
11      nwt=1
12      open(nwt,file='out')
13      n=254
14      l=128
15      k=(n-1)/2.
16      nl=256
17      nl2=nl/2
18      do 90 im=0,5
19          m=2*(im+2)
20      do 100 i=1,5
21          d=-.1*i+.6
22          call one(bw,k,n,m,d)
23          do 110 ii=2,4
24              snr=10**ii
25              call two(bw,n,m,g,snr)
26              call six(g,m,nl,retil,f,snr)
27 c          call five(retil,f,n,m,d,nwt,snr,nl2)
28              if(im.eq.0)call plotfl('sparse4',nl2,'d',f,'d',retil)
29              if(im.eq.1)call plotfl('sparse8',nl2,'d',f,'d',retil)
30              if(im.eq.2)call plotfl('sparse16',nl2,'d',f,'d',retil)
31              if(im.eq.3)call plotfl('sparse32',nl2,'d',f,'d',retil)
32              if(im.eq.4)call plotfl('sparse64',nl2,'d',f,'d',retil)
33              if(im.eq.5)call plotfl('sparse128',nl2,'d',f,'d',retil)
34 110      continue
35 100      continue
36 90      continue
37      close(nwt)
38      stop
39      end
40 c
41 c
42 c
43 c
44      subroutine one(bw,k,n,m,d)
45 c          subroutine to calculate bw
46      real*8 bw(n,m),hh
47      do 500 i=1,n
48          do 500 ii=1,m
49 500      bw(i,ii)=0
50      j=-k
51      do 510 i=1,1+2*k
52          bw(i,1)=hh(j,d)
53          j=j+1
54 510      continue
55      do 520 i=1,m-1
56          do 530 ii=0,2*k
57              bw(1+ii+i,i+1)=bw(ii+i,i)
58 530      continue
59 520      continue
60      return
61      end
62 c
63 c
64      function hh(j,d)
65      real*8 hh
66      if(j.eq.0)then
67          hh=d**2
68      else

```

```

69      hh=(cos(.78539816*j*(1-d))*sin(.78539816*d*j)/
70      c      (.78539816*j))*2
71      endif
72      return
73      end
74      c
75      c
76      subroutine two(bw,n,m,g,snr)
77      c      subroutine to calculate h=(gtg)*snr + i inv
78      parameter(mm=128)
79      real*8 bw(n,m),g(m,m)
80      integer lwork(mm),mwork(mm),ier
81      do 600 i=1,m
82          do 610 ii=1,m
83              g(i,ii)=0
84              do 620 iii=1,n
85                  g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)*snr
86      620          continue
87              if(i.eq.ii)g(i,ii)=g(i,ii)+1
88      610          continue
89      600      continue
90      call dminv(g,m,ier,lwork,mwork)
91      return
92      end
93      c
94      c
95      c
96      subroutine five(retil,f,n,m,d,nwt,snr,nl2)
97      real*8 retil(nl2),f(nl2)
98      write(nwt,890)n,m,d,snr
99      890      format(' n = ',i4,' m = ',i4,' d = ',f10.5,' snr = ',
100      c      f12.5)
101      do 900 i=1,nl2/2
102          write(nwt,895)f(i),retil(i)
103      895      format(2f12.5)
104      900      continue
105      return
106      end
107      c
108      c
109      subroutine six(g,m,nl,retil,f,snr)
110      complex*8 berr(256,256)
111      real*8 g(m,m),retil(128),f(128)
112      do 200 i=1,nl
113          do 200 ii=1,nl
114      200          berr(i,ii)=0
115      nl=(nl-m)/2.
116      do 210 i=1,m
117          do 220 ii=1,m
118              berr(nl+i,nl+ii)=g(i,ii)
119      220          continue
120      210      continue
121      dx=1
122      dy=1
123      call fft2d(berr,nl,dx,dy)
124      nl2=nl/2
125      nl4=nl/4
126      do 230 i=1,nl2
127          retil(i)=real(berr(nl2+i,nl2-i+2))/m
128          x=i
129          f(i)=x/nl4
130      230      continue
131      return
132      end
133      c
134      c

```

```

1 c
2 c
3 c      program matrix6
4 c
5      parameter(nn=254,mm=128)
6      real*8 bw(nn,mm),h(mm,nn),retilu(128),f(128),x(mm),no(nn)
7      real*8 g(mm,mm),erru(mm,mm),retil2(128),err2(mm,mm)
8      real*8 retilc2(128),errc2(mm,mm)
9      real*8 y(nn),gg(mm,mm)
10     real*4 d,eps
11     integer*4 random
12     character dfile*16
13 c
14     nwt=2
15     open(nwt,file='out')
16     n=254
17     l=128
18     k=(n-1)/2.
19     print *, 'nt?'
20     read *, nt
21     print *, 'ni?'
22     read *, ni
23     print *, 'eps?'
24     read *, eps
25     print *, 'mb?'
26     read *, mb
27     print *, 'm?'
28     read *, m
29     print *, 'snr?'
30     read *, snr
31     print *, 'd/D?'
32     read *, d
33     print *, 'Destination File?'
34     read *, dfile
35     nl=256
36     nl2=nl/2
37     call srandom(29)
38     call one(bw,k,n,m,d)
39     call two(bw,h,n,m,g,gg)
40     call two5(h,n,m,err2)
41     call six(nwt,err2,m,nl,retil2,f,snr,mb)
42     call five(retil2,f2,n,m,d,nwt,snr,nl2)
43     call plotfl(dfile,nl2,'d',f,'d',retil2)
44     do 103 j=1,m
45         do 101 jj=1,m
46             erru(j,jj)=0
47             errc2(j,jj)=0
48 101     continue
49 103     continue
50     do 105 iv=1,nt
51         print *,iv
52         if (float(iv)/10.0-int(float(iv)/10.0) .lt. 1e-10) then
53             print *, iv
54         endif
55         call three(x,no,m,n,snr,bw,y,mb)
56         call four(x,h,no,erru,n,m,nt,errc2,y,bw,gg,eps,ni)
57 105     continue
58     call six(nwt,erru,m,nl,retilu,f,snr,mb)
59     call five(retilu,f,n,m,d,nwt,snr,nl2)
60     call six(nwt,errc2,m,nl,retilc2,f,snr,mb)
61     call five(retilc2,f,n,m,d,nwt,snr,nl2)
62     call plotfl(dfile,nl2,'d',f,'d',retilu)
63     call plotfl(dfile,nl2,'d',f,'d',retilc2)
64     close(nwt)
65     stop
66     end
67 c
68 c

```

```

69 c
70 c
71      subroutine one(bw,k,n,m,d)
72      real*8 bw(n,m),hh
73      do 500 i=1,n
74          do 500 ii=1,m
75      500      bw(i,ii)=0
76      j=-k
77      do 510 i=1,1+2*k
78          bw(i,i)=hh(j,d)
79          j=j+1
80      510      continue
81      do 520 i=1,m-1
82          do 530 ii=0,2*k
83              bw(1+ii+i,i+1)=bw(ii+i,i)
84      530      continue
85      520      continue
86      return
87      end
88 c
89 c
90      function hh(j,d)
91      real*8 hh
92      if(j.eq.0)then
93          hh=d**2
94      else
95          hh=(cos(.78539816*j*(1-d))*sin(.78539816*d*j)/
96      c      (.78539816*j))**2
97      endif
98      return
99      end
100 c
101 c
102      subroutine two(bw,h,n,m,g,gg)
103      parameter(mm=128)
104      real*8 bw(n,m),g(m,m),h(m,n),gg(m,m)
105      integer lwork(mm),mwork(mm),ier
106      do 600 i=1,m
107          do 610 ii=1,m
108              g(i,ii)=0
109              do 620 iii=1,n
110                  g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)
111                  gg(i,ii)=g(i,ii)
112      620          continue
113      610      continue
114      600      continue
115      call dminvw(g,m,ier,lwork,mwork)
116      do 630 i=1,m
117          do 640 ii=1,n
118              h(i,ii)=0
119              do 650 iii=1,m
120                  h(i,ii)=h(i,ii)+g(i,iii)*bw(ii,iii)
121      650          continue
122      640      continue
123      630      continue
124      return
125      end
126 c
127 c
128      subroutine two5(h,n,m,err2)
129      real*8 h(m,n),err2(m,m)
130      do 400 i=1,m
131          do 410 ii=1,m
132              err2(i,ii)=0
133              do 420 iii=1,n
134                  err2(i,ii)=err2(i,ii)+h(i,iii)*h(ii,iii)
135      420          continue
136      410      continue

```

```

137 400      continue
138          return
139          end
140  c
141          subroutine three(x,no,m,n,snr,bw,y,mb)
142          real*8 u1,u2,u3,s
143          real*8 x(m),no(n),bw(n,m),y(n)
144          integer*4 MAXINTV,random
145          parameter(MAXINTV=2147483647)
146          n2=n/2
147          do 500 i=1,n2
148  3          u1=real(random())/MAXINTV
149              if(u1.gt.1.or.u1.eq.0)goto 3
150  2          u2=real(random())/MAXINTV
151              if(u2.gt.1.or.u2.eq.0)goto 2
152              no(2*i-1)=sqrt(-2*log(u1))*cos(6.2831853*u2)
153              no(2*i)=sqrt(-2*log(u1))*sin(6.2831853*u2)
154  500      continue
155          s=snr/2
156          do 505 i=1,m
157  4          u3=real(random())/MAXINTV
158              if(u3.gt.1.or.u3.eq.0)goto 4
159              x(i)=sqrt(-2*log(u3))*sqrt(s)
160  505      continue
161          do 507 i=mb+1,m
162              x(i)=0
163  507      continue
164          do 510 i=1,n
165              y(i)=0
166              do 520 ii=1,m
167                  y(i)=y(i)+bw(i,ii)*x(ii)
168  520          continue
169                  y(i)=y(i)+no(i)
170  510      continue
171          return
172          end
173  c
174  c
175          subroutine four(x,h,no,erru,n,m,nt,errc2,y,bw,gg,eps,ni)
176          parameter(mm=128,nn=254)
177          real*8 x(m),no(n),erru(m,m),h(m,n),eu(mm),sum
178          real*8 ec2(mm),errc2(m,m),gg(m,m),gg2(mm,mm),v2(nn)
179          real*8 tmp(mm),y(n),bw(n,m),z(mm),xg(mm),tem2(mm)
180          real*4 eps
181          do 700 i=1,m
182              tmp(i)=0
183              z(i)=0
184              do 710 ii=1,n
185                  tmp(i)=tmp(i)+h(i,ii)*y(ii)
186                  z(i)=z(i)+bw(ii,i)*y(ii)*eps
187  710          continue
188  700      continue
189          do 712 i=1,m
190              do 713 ii=1,m
191                  if(i.eq.ii)then
192                      gg2(i,ii)=1-eps*gg(i,ii)
193                  else
194                      gg2(i,ii)=-eps*gg(i,ii)
195                  endif
196  713          continue
197  712      continue
198          do 715 i=1,m
199              eu(i)=tmp(i)-x(i)
200  c          if(tmp(i).gt.0.and.z(i).gt.0)then
201  c              xg(i)=tmp(i)
202  c          else
203  c              xg(i)=0
204  c          endif

```

```

205      xg(i)=x(i)
206      ec2(i)=xg(i)-x(i)
207 715  continue
208      jk=0
209      do 770 iv=1,ni
210          jk=jk+1
211          do 740 i=1,m
212              tem=0
213              do 750 ii=1,m
214                  tem=tem+gg2(i,ii)*xg(ii)
215 750  continue
216              tem2(i)=tem+z(i)
217              if(tem2(i).lt.0)tem2(i)=0
218 740  continue
219              do 760 i=1,m
220                  xg(i)=tem2(i)
221                  ec2(i)=xg(i)-x(i)
222 760  continue
223  c      if(jk.eq.100)then
224  c          jk=0
225  c          sum=0
226  c          do 745 i=1,n
227  c              v2(i)=0
228  c              do 746 ii=1,m
229  c                  v2(i)=v2(i)+bw(i,ii)*xg(ii)
230  c46  continue
231  c              v2(i)=v2(i)-y(i)
232  c              sum=sum+v2(i)**2
233  c45  continue
234  c          do 775 i=1,m
235  c              print *,xg(i)
236  c75  continue
237  c          print *,iv,sum
238  c          print *
239  c      endif
240 770  continue
241  c      call plotfl('test',m,'d',xg,'d',xg)
242      do 720 i=1,m
243          do 730 ii=1,m
244              erru(i,ii)=erru(i,ii)+eu(i)*eu(ii)/nt
245              errc2(i,ii)=errc2(i,ii)+ec2(i)*ec2(ii)/nt
246 730  continue
247 720  continue
248      return
249      end
250  c
251  c
252  c
253  c
254  c
255      subroutine five(retil,f,n,m,d,nwt,snr,nl2)
256      real*8 retil(nl2),f(nl2)
257      write(nwt,890)n,m,d,snr
258 890  format(' n = ',i4,' m = ',i4,' d = ',f10.5,' snr = ',
259  c      f16.4)
260      do 900 i=1,nl2/2
261          write(nwt,895)f(i),retil(i)
262 895  format(2f12.5)
263 900  continue
264      return
265      end
266  c
267  c
268      subroutine six(nwt,err,m,nl,retil,f,snr,mb)
269      complex*8 berr(256,256)
270      real*8 err(m,m),retil(128),f(128)
271      real*4 dx,dy
272      parameter (pi=3.14159265)

```

```

273      sum=0
274      do 190 i=1,m
275          sum=sum+err(i,i)
276 190      continue
277      write(nwt,*)'trace = ',sum
278      print *, sum
279      do 200 i=1,nl
280          do 200 ii=1,nl
281 200          berr(i,ii)=0
282      nl=(nl-m)/2.
283      do 210 i=1,m
284          do 220 ii=1,m
285              berr(nl+i,nl+ii)=err(i,ii)
286 220          continue
287 210      continue
288      dx=1
289      dy=1
290      call fft2d(berr,nl,dx,dy)
291      nl2=nl/2
292      nl4=nl/4
293      do 230 i=1,nl2
294          f(i)=real(i)/nl4
295          retil(i)=real(berr(nl2+i,nl2-i+2))/(snr*mb)
296 230      continue
297      return
298      end
299 c
300 c

```

```

1  c
2  c
3  c      program matrix7
4  c
5      parameter(nn=254,mm=128)
6      real*8 bw(nn,mm),h(mm,nn),retilu(128),f(128),x(mm),no(nn)
7      real*8 g(mm,mm),erru(mm,mm),retil2(128),err2(mm,mm)
8      real*8 retilc2(128),errc2(mm,mm)
9      real*8 y(nn),gg(mm,mm)
10     real*4 d,eps
11     integer*4 random
12     character dfile*16
13  c
14     n=254
15     l=128
16     k=(n-1)/2.
17     print *, 'snr?'
18     read *, snr
19     print *, 'd/D?'
20     read *, d
21     print *, 'Destination File?'
22     read *, dfile
23     nl=256
24     nl2=nl/2
25     do 5 i=1,6
26         m=2**(i+1)
27         mb=m
28         do 10 j=1,m
29             do 11 jj=1,m
30                 erru(j,jj)=0
31                 errc2(j,jj)=0
32 11         continue
33 10     continue
34         call one(bw,k,n,m,d)
35         call two(bw,h,n,m,g,gg)
36         call two5(h,n,m,err2)
37         call six(nwt,err2,m,nl,retil2,f,snr,mb)
38         call plotfl(dfile,nl2,'d',f,'d',retil2)
39 5     continue
40     stop
41     end
42  c
43  c
44  c
45  c
46     subroutine one(bw,k,n,m,d)
47     real*8 bw(n,m),hh
48     do 500 i=1,n
49         do 500 ii=1,m
50 500     bw(i,ii)=0
51         j=-k
52         do 510 i=1,1+2*k
53             bw(i,1)=hh(j,d)
54             j=j+1
55 510     continue
56         do 520 i=1,m-1
57             do 530 ii=0,2*k
58                 bw(1+ii+i,i+1)=bw(ii+i,i)
59 530     continue
60 520     continue
61     return
62     end
63  c
64  c
65     function hh(j,d)
66     real*8 hh
67     if(j.eq.0)then
68         hh=d**2

```

```

69      else
70      hh=(cos(.78539816*j*(1-d))*sin(.78539816*d*j)/
71      c      (.78539816*j))*2
72      endif
73      return
74      end
75  c
76  c
77      subroutine two(bw,h,n,m,g,gg)
78      parameter(mm=128)
79      real*8 bw(n,m),g(m,m),h(m,n),gg(m,m)
80      integer lwork(mm),mwork(mm),ier
81      do 600 i=1,m
82      do 610 ii=1,m
83      g(i,ii)=0
84      do 620 iii=1,n
85      g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)
86      gg(i,ii)=g(i,ii)
87  620      continue
88  610      continue
89  600      continue
90      call dminvw(g,m,ier,lwork,mwork)
91      do 630 i=1,m
92      do 640 ii=1,n
93      h(i,ii)=0
94      do 650 iii=1,m
95      h(i,ii)=h(i,ii)+g(i,iii)*bw(ii,iii)
96  650      continue
97  640      continue
98  630      continue
99      return
100     end
101  c
102  c
103      subroutine two5(h,n,m,err2)
104      real*8 h(m,n),err2(m,m)
105      do 400 i=1,m
106      do 410 ii=1,m
107      err2(i,ii)=0
108      do 420 iii=1,n
109      err2(i,ii)=err2(i,ii)+h(i,iii)*h(ii,iii)
110  420      continue
111  410      continue
112  400      continue
113      return
114      end
115  c
116      subroutine six(nwt,err,m,nl,retil,f,snr,mb)
117      complex*8 berr(256,256)
118      real*8 err(m,m),retil(128),f(128)
119      real*4 dx,dy
120      parameter (pi=3.14159265)
121      sum=0
122      do 190 i=1,m
123      sum=sum+err(i,i)
124  190      continue
125      print *, sum
126      do 200 i=1,nl
127      do 200 ii=1,nl
128  200      berr(i,ii)=0
129      ni=(nl-m)/2.
130      do 210 i=1,m
131      do 220 ii=1,m
132      berr(ni+i,ni+ii)=err(i,ii)
133  220      continue
134  210      continue
135      dx=1
136      dy=1

```

```

137      call fft2d(berr,nl,dx,dy)
138      nl2=nl/2
139      nl4=nl/4
140      do 230 i=1,nl2
141          f(i)=real(i)/nl4
142          retil(i)=real(berr(nl2+i,nl2-i+2))/(snr*mb)
143 230    continue
144      return
145      end
146 c
147 c

```

```

1 c
2 c
3 c      program   mtfmkr.f
4 c
5 c
6       real*4 xi,xj,xn,f,d,mtf,fr(200),o(200)
7       integer n
8       n = 200
9       xn = n
10      do 10 j=1, 5
11          xj = j
12          d = xj/10
13          do 20 i=0, n-1
14              xi = i
15              f = xi/xn
16              fr(i+1) = mtf(d,f) + 1e-30
17              o(i+1) = f
18 20      continue
19      call plotfl('mtfdta',200,'f',o,'f',fr)
20 10      continue
21      end
22 c-----
23      FUNCTION mtf(d,f)
24          real*4 d,f,dinv,mtf,f1,f2
25          dinv = 1/d
26          if (f .lt. d) then
27              f1 = 1.0 - dinv*f
28          else
29              f1 = 0
30          endif
31          if ((f .gt. 1-2*d) .and. (f .le. 1-d)) then
32              f2 = 0.5*dinv*(f-1) + 1
33          elseif ((f .gt. 1-d) .and. (f .le. 1)) then
34              f2 = 0.5*dinv*(1-f)
35          else
36              f2 = 0
37          endif
38          mtf = f1 + f2
39          mtf = 2.0*d*mtf
40      end

```

```

1  c
2  c
3  c      program second.f
4  c
5  c
6      parameter(kk=784)
7      real*8 g(kk,kk)
8      integer lwork(kk),mwork(kk),idummy(1)
9  c
10     m=28
11     m2=m+2
12     m3=(m2*(m2+1))/4
13     call bufasg(idummy,2*m3,ia)
14     call bufasg(idummy,2*m3,ib)
15     call three(g,m,m2,m3,idummy(ia),idummy(ib))
16     call bufrel(idummy(ib))
17     call bufrel(idummy(ia))
18     snr=100
19     do 120 i=1,m2
20         g(i,i)=g(i,i)+1/snr
21 120    continue
22     call dminvw(g,m2,ier,lwork,mwork)
23     do 130 iii=1,m2
24         do 140 iv=1,m2
25             g(iii,iv)=g(iii,iv)/snr
26 140    continue
27 130    continue
28     call ten(g,m,m2)
29     stop
30     end
31  c
32     subroutine three(g,m,m2,m3,gx,gy)
33     real*8 g(m2,m2),gx(m3),gy(m3)
34     call readfl('g328',n,'d',gx,'d',gy,1,1,m3)
35     iw=1
36     do 600 i=1,m
37         j=1
38         ia=i
39         jg=m*(j-1)
40         ig=m*(ia-1)
41         do 680 ix=1,m-i+1
42             if(ia.ne.j)then
43                 do 640 ki=1,m
44                     do 650 k2 = 1,m
45                         if(iw.gt.0)then
46                             g(jg+ki,ig+k2)=gx(iw)
47                             g(ig+k2,jg+ki)=gx(iw)
48                             iw=-iw
49                         else
50                             g(jg+ki,ig+k2)=gy(-iw)
51                             g(ig+k2,jg+ki)=gy(-iw)
52                             iw=-iw+1
53                         endif
54 650                    continue
55 640                    continue
56                 else
57                     do 645 ki=1,m
58                         do 655 k2=k1,m
59                             if(iw.gt.0)then
60                                 g(jg+ki,ig+k2)=gx(iw)
61                                 g(ig+k2,ig+k1)=gx(iw)
62                                 iw=-iw
63                             else
64                                 g(jg+ki,ig+k2)=gy(-iw)
65                                 g(ig+k2,jg+ki)=gy(-iw)
66                                 iw=-iw+1
67                             endif
68 655                    continue

```

```

69 645      continue
70      endif
71      j=j+1
72      jg=*(j-1)
73      ia=ia+1
74      ig=*(ia-1)
75 680      continue
76 600      continue
77      return
78      end
79  c
80  c
81      subroutine ten(g,m,m3)
82      parameter(mm=28)
83      real*4 diag(55),bdiag(128),wr(128),wi(128),dd,br(mm,mm,127)
84      real*4 bi(mm,mm,127),y(128),sur(127,127),sli(16129),x(16129)
85      real*8 gl(mm,mm),brt(mm,mm),bit(mm,mm),g(m3,m3)
86      real*4 smsur(63,63)
87  c      real*4 x2(3969),sli2(3969)
88  c
89      n=128
90      nl=7
91      m2=2*m-1
92      call lookup(n,wr,wi)
93      do 10 i=1,m
94          do 20 ii=1,m
95              do 30 iii=1,m
96                  do 40 iv=1,m
97                      gl(iii,iv)=g(m*i-m+iii,m*ii-m+iv)
98 40      continue
99 30      continue
100     call adddiag(gl,m2,diag,m)
101     call pad(diag,m2,bdiag,n)
102     do 50 ix=1,n
103         y(ix)=0
104 50      continue
105         dd=1
106         call fft2(bdiag,y,n,nl,wr,wi,dd)
107         do 60 iii=1,n-1
108             br(i,ii,iii)=bdiag(iii+1)
109  c      br(ii,i,iii)=bdiag(iii+1)
110             bi(i,ii,iii)=y(iii+1)
111  c      bi(ii,i,iii)=-y(iii+1)
112 60      continue
113 20      continue
114 10      continue
115         do 70 i=1,n-1
116             do 80 ii=1,m
117                 do 90 iii=1,m
118                     brt(ii,iii)=br(ii,iii,i)
119                     bit(ii,iii)=bi(ii,iii,i)
120 90      continue
121 80      continue
122         call adddiag(brt,m2,diag,m)
123         call pad(diag,m2,bdiag,n)
124         call adddiag(bit,m2,diag,m)
125         call pad(diag,m2,y,n)
126         dd=1
127         call fft2(bdiag,y,n,nl,wr,wi,dd)
128         do 100 ii=1,n-1
129             sur(i,ii)=bdiag(ii+1)/m3
130             sli((n-1)*i-(n-1)+ii)=bdiag(ii+1)/m3
131             x((n-1)*(i-1)+ii)=(n-1)*(i-1)+ii
132 100     continue
133 70      continue
134         iii=0
135         do 110 i=1,63
136             do 120 ii=1,63

```

```

137 c      iii=iii+1
138      smsur(i,ii)=sur(32+i,32+ii)
139 c      sli2(iii)=smsur(i,ii)
140 c      x2(iii)=x(iii)
141 120      continue
142 110      continue
143 c      call solidplt('sb28',127,127,sur)
144      call plotfl('tlongb28a',(n-1)*2,'f',x,'f',sli)
145      call solidplt('ttb28a',63,63,smsur)
146 c      call plotfl('slongb28',3969,'f',x2,'f',sli2)
147      stop
148      end
149 c
150 c
151 c
152      subroutine pad(diag,m2,bdiag,n)
153      real*4 diag(m2),bdiag(n)
154      do 404 i=1,n
155          bdiag(i)=0
156 404      continue
157          k=(n-m2+3)/2
158          do 406 i=1,m2
159              bdiag(k+i-1)=diag(i)
160 406      continue
161      return
162      end
163 c
164 c
165 c
166      subroutine adddiag(a,m2,diag,m)
167      real*4 diag(m2)
168      real*8 a(m,m)
169      do 601 i=1,m2
170          diag(i)=0
171 601      continue
172          do 603 i=1,m
173              do 605 ii=1,m
174                  iii=i-ii+m
175                  diag(iii)=diag(iii)+a(i,ii)
176 605      continue
177 603      continue
178      return
179      end

```

```

1 c
2 c
3 c      program slice.f
4 c
5 c
6 c      this program reads in the results of matrix4
7 c      in plotfl form and reassembles spectral matrix
8 c      and then takes out the center to plot
9      real*4 x(16129),surf(127,127),smsur(63,63)
10     real*4 x2(3969),sli2(3969),sli(16129)
11     character*10 file
12     print *, 'What is the name of the file?'
13     read *, file
14     call readfl(file,n,'f',x,'f',sli,1,1,16129)
15     i=0
16     do 10 ii=1,127
17         do 20 iii=1,127
18             i=i+1
19             surf(ii,iii)=sli(i)
20 20     continue
21 10     continue
22     i=0
23     do 30 ii=1,63
24         do 40 iii=1,63
25             i=i+1
26             smsurf(ii,iii)=surf(32+ii,32+iii)
27             sli2(i)=smsurf(ii,iii)
28             x2(i)=x(i)
29 c0     continue
30 c0     continue
31     call solidplt('solidfile',127,127,surf)
32 c     call plotfl('smlong2',3969,'f',x2,'f',sli2)
33     stop
34     end

```

APPENDIX B for Chapter 3

Computer Listings

This appendix contains the Fortran source code of the computer programs used to generate the data for this chapter.

```

1  c
2  c
3  c      program eigenmkr.f
4  c
5  c
6      parameter(mm=128)
7      real*8 bw(1150,mm),f(128)
8      real*8 g(mm,mm),retil1(128),retil2(128),g1(mm,mm),g2(mm,mm)
9      real*8 gg(mm,mm),evec(mm,mm),eval(mm),vec(mm,mm),val(mm),d
10     real*4 eps
11     integer*4 i,k,m,mb,n,l,ier>window
12     character dfile*16
13  c
14     l=128
15     print *, 'k?'
16     read *, k
17     print *, 'snr?'
18     read *, snr
19     print *, 'd/D?'
20     read *, d
21     print *, 'Window?'
22     read *, window
23  c     print *, 'Destination File?'
24  c     read *, dfile
25     n=2*k+128
26     do 10 i=1,6
27         m=2**(i+1)
28         mb=m
29         call bwmkcr(bw,k,n,m,d>window)
30         call ggmkcr(bw,n,m,g,gg,eps,ier)
31  c     call eigenmkr(m,gg,evec,eval,vec,val,g1,g2)
32         call eigenmkr(m,n,bw,evec,eval,vec,val,g1,g2)
33         call spectramkr(g1,m,retil1,f,snr,mb)
34         call spectramkr(g2,m,retil2,f,snr,mb)
35         do 20 ii=1,128
36  c             retil2(ii)=retil2(ii)+retil1(ii)
37 20     continue
38  c     call plotfl(dfile,128,'d',f,'d',retil2)
39         call plotfl('testw3k255d1.1',128,'d',f,'d',retil1)
40         call plotfl('testw3k255d1.2',128,'d',f,'d',retil2)
41 10     continue
42     stop
43     end
44  c
45  c
46  c
47     subroutine bwmkcr(bw,k,n,m,d>window)
48     real*8 bw(n,m),hh,d
49     integer n,m,k>window
50     do 500 i=1,n
51         do 500 ii=1,m
52 500         bw(i,ii)=0
53         j=-k
54         do 510 i=1,1+2*k
55             bw(i,1)=hh(j,d,k>window)
56             j=j+1
57 510         continue
58         do 520 i=1,m-1
59             do 530 ii=0,2*k
60                 bw(1+ii+i,i+1)=bw(ii+i,i)

```

```

61 530      continue
62 520      continue
63      return
64      end
65  c
66  c
67  c
68      subroutine ggakr(bw,n,m,g,gg,eps,ier)
69      parameter(mm=128)
70      real*8 bw(n,m),g(m,m),gg(m,m)
71      real*8 r(8256)
72      real*4 eps
73      integer ier
74      do 600 i=1,m
75          do 610 ii=1,m
76              g(i,ii)=0
77              do 620 iii=1,n
78                  g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)
79                  gg(i,ii)=g(i,ii)
80 620          continue
81 610      continue
82 600      continue
83  c      call squartri(r,m,g)
84  c      call dsinv(r,m,eps,ier)
85  c      call trisquar(g,m,r)
86      end
87  c
88  c
89  c
90      function hh(j,d,k>window)
91      real*8 hh,d,k1,pi,pi2,j1
92      integer k,j,m>window
93      m>window
94      pi=3.141592654
95      pi2=2*pi
96      j1=j
97      k1=k
98      if(j.eq.0)then
99          hh=d*d
100      else
101          hh=cos(.78539816*dble(j)*(1-d))*sin(.78539816*d*dble(j))/
102      c      (.78539816*dble(j))
103          hh=hh*hh
104      endif
105      if (m.eq.1) then
106          hh=hh
107      elseif (m.eq.2) then
108          hh=hh*(1-abs(j1)/k1)
109      elseif (m.eq.3) then
110          hh=hh*(.5+.5*cos(pi*j1/k1))
111      elseif (m.eq.4) then
112          hh=hh*(0.54+0.46*cos(pi*j1/k1))
113      else
114          hh=hh*(0.42+0.5*cos(pi*j1/k1)+0.08*cos(pi2*j1/k1))
115      endif
116 10      return
117      end
118  c
119  c
120  c
121      subroutine spectramkr(err,m,retil,f,snr,mb)
122      real*8 err(m,m),retil(128),f(128),xii,xi,pi2
123      real*8 sum(128)
124      integer p
125      pi2=6.283185307
126      do 10 i=1,m
127          sum(i)=0.0
128 10      continue

```

```

129      do 20 i=1,m
130      do 30 ii=1,m
131      p=abs(i-ii)+1
132      sum(p)=sum(p)+err(i,ii)
133 30      continue
134 20      continue
135      print *, sum(1)
136      do 50 i=1,128
137      xi=real(i)/256
138      retil(i)=0
139      do 60 ii=1,m
140      xii=ii-1
141      retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
142 60      continue
143 50      continue
144      do 80 i=1,128
145      f(i)=real(i)/64
146      retil(i)=retil(i)/(snr*mb)
147 80      continue
148      return
149      end
150 c
151 c
152 c
153 c      subroutine eigenmkr(m,gg,eval,vec,val,g1,g2)
154      subroutine eigenmkr(m,n,g,eval,vec,val,g1,g2)
155 c      real*8 gg(m,m),eval(m,m),eval(m),temp(128),x
156      real*8 g(m,m),eval(m,m),eval(m),temp(128),x
157      real*8 vec(m,m),val(m),g1(m,m),g2(m,m)
158      integer m,n,list(128),mid
159 c
160 c      call dmeigen(eval,vec,gg,m)
161      call svdcmp(g,n,m,eval,eval)
162      do 5 i=1,m
163      eval(i)=eval(i)*eval(i)
164 5      continue
165      x=-1d+308
166      do 10 i=1,m
167      temp(i)=eval(i)
168 10      continue
169 c ----- Sort indices from largest eigenvalue to smallest -----
170      do 20 i=1,m
171      do 30 ii=1,m
172      if(x.lt.temp(ii)) then
173      x=temp(ii)
174      list(i)=ii
175      endif
176 30      continue
177      temp(list(i))=0d0
178      x=-1d+308
179 20      continue
180 c ----- Order eigenvalues and eigenvectors -----
181      do 40 i=1,m
182      val(i)=eval(list(i))
183      do 50 ii=1,m
184      vec(ii,i)=eval(ii,list(i))
185 50      continue
186 40      continue
187 c ----- Find middle of dynamic range -----
188      x=(log(val(m))+log(val(1)))/2d0
189      do 60 i=1,m
190      if(log(val(i)).lt.x) then
191      mid=i
192      goto 65
193      endif
194 60      continue
195 c ----- Generate two halves of inverse matrix -----
196 65      do 70 i=1,m

```

```

197         do 80 ii=1,m
198             g1(i,ii)=0d0
199             g2(i,ii)=0d0
200             do 90 iii=1,m
201                 if(iii.lt.mid) then
202                     g1(i,ii)=g1(i,ii)+
203                     *             (1d0/val(iii))*vec(i,iii)*vec(ii,iii)
204                     else
205                         g2(i,ii)=g2(i,ii)+
206                         *             (1d0/val(iii))*vec(i,iii)*vec(ii,iii)
207                     endif
208 90         continue
209 80     continue
210 70     continue
211     return
212 end
213 c
214 c
215 c
216 c     subroutine svdcmp(a,m,n,np,np,v,v)
217     subroutine svdcmp(a,m,n,v,v)
218     implicit real*8 (a-h,o-z)
219     parameter (nmax=128)
220 c     dimension a(mp,np),w(np),v(np,np),rv1(nmax)
221     dimension a(m,n),w(n),v(n,n),rv1(nmax)
222     g=0d0
223     scale=0d0
224     anorm=0d0
225     do 25 i=1,n
226         l=i+1
227         rv1(i)=scale*g
228         g=0d0
229         s=0d0
230         scale=0d0
231         if (i.le.m) then
232             do 11 k=i,m
233                 scale=scale+abs(a(k,i))
234 11         continue
235         if (scale.ne.0d0) then
236             do 12 k=i,m
237                 a(k,i)=a(k,i)/scale
238                 s=s+a(k,i)*a(k,i)
239 12         continue
240         f=a(i,i)
241         g=-sign(sqrt(s),f)
242         h=f*g-s
243         a(i,i)=f-g
244         if (i.ne.n) then
245             do 15 j=l,n
246                 s=0d0
247                 do 13 k=i,m
248                     s=s+a(k,i)*a(k,j)
249 13         continue
250             f=s/h
251             do 14 k=i,m
252                 a(k,j)=a(k,j)+f*a(k,i)
253 14         continue
254 15         continue
255         endif
256         do 16 k=i,m
257             a(k,i)=scale*a(k,i)
258 16         continue
259         endif
260     endif
261     w(i)=scale *g
262     g=0d0
263     s=0d0
264     scale=0d0

```

```

265      if ((i.le.m).and.(i.ne.n)) then
266          do 17 k=1,n
267              scale=scale+abs(a(i,k))
268 17      continue
269          if (scale.ne.0d0) then
270              do 18 k=1,n
271                  a(i,k)=a(i,k)/scale
272                  s=s+a(i,k)*a(i,k)
273 18      continue
274          f=a(i,l)
275          g=-sign(sqrt(s),f)
276          h=f*g-s
277          a(i,l)=f-g
278          do 19 k=1,n
279              rv1(k)=a(i,k)/h
280 19      continue
281          if (i.ne.m) then
282              do 23 j=1,m
283                  s=0d0
284                  do 21 k=1,n
285                      s=s+a(j,k)*a(i,k)
286 21      continue
287                  do 22 k=1,n
288                      a(j,k)=a(j,k)+s*rv1(k)
289 22      continue
290 23      continue
291          endif
292          do 24 k=1,n
293              a(i,k)=scale*a(i,k)
294 24      continue
295          endif
296          endif
297          anorm=max(anorm,(abs(v(i))+abs(rv1(i))))
298 25      continue
299          do 32 i=n,1,-1
300              if (i.lt.n) then
301                  if (g.ne.0d0) then
302                      do 26 j=1,n
303                          v(j,i)=(a(i,j)/a(i,l))/g
304 26      continue
305                      do 29 j=1,n
306                          s=0d0
307                          do 27 k=1,n
308                              s=s+a(i,k)*v(k,j)
309 27      continue
310                          do 28 k=1,n
311                              v(k,j)=v(k,j)+s*v(k,i)
312 28      continue
313 29      continue
314                      endif
315                      do 31 j=1,n
316                          v(i,j)=0d0
317                          v(j,i)=0d0
318 31      continue
319                      endif
320                      v(i,i)=1d0
321                      g=rv1(i)
322                      l=i
323 32      continue
324                      do 39 i=n,1,-1
325                          l=i+1
326                          g=w(i)
327                          if (i.lt.n) then
328                              do 33 j=1,n
329                                  a(i,j)=0d0
330 33      continue
331                          endif
332                          if (g.ne.0d0) then

```

```

333      g=1d0/g
334      if (i.ne.m) then
335          do 36 j=1,n
336              s=0d0
337              do 34 k=1,m
338                  s=s+a(k,i)*a(k,j)
339 34          continue
340              f=(s/a(i,i))*g
341              do 35 k=i,m
342                  a(k,j)=a(k,j)+f*a(k,i)
343 35          continue
344 36          continue
345      endif
346      do 37 j=i,m
347          a(j,i)=a(j,i)*g
348 37          continue
349      else
350          do 38 j= i,m
351              a(j,i)=0d0
352 38          continue
353      endif
354      a(i,i)=a(i,i)+1d0
355 39      continue
356      do 49 k=n,1,-1
357          do 48 its=1,30
358              do 41 l=k,1,-1
359                  nm=l-1
360                  if ((abs(rv1(l))+anorm).eq.anorm) go to 2
361                  if ((abs(w(nm))+anorm).eq.anorm) go to 1
362 41          continue
363 1          c=0d0
364              s=1d0
365              do 43 i=1,k
366                  f=s*rv1(i)
367                  if ((abs(f)+anorm).ne.anorm) then
368                      g=w(i)
369                      h=sqrt(f*f+g*g)
370                      w(i)=h
371                      h=1d0/h
372                      c= (g*h)
373                      s=-(f*h)
374                      do 42 j=1,m
375                          y=a(j,nm)
376                          z=a(j,i)
377                          a(j,nm)=(y*c)+(z*s)
378                          a(j,i)=- (y*s)+(z*c)
379 42          continue
380              endif
381 43          continue
382 2          z=w(k)
383              if (l.eq.k) then
384                  if (z.lt.0d0) then
385                      w(k)=-z
386                      do 44 j=1,n
387                          v(j,k)=-v(j,k)
388 44          continue
389              endif
390              go to 3
391          endif
392          if (its.eq.30) pause 'no convergence in 30 iterations'
393          x=w(1)
394          nm=k-1
395          y=w(nm)
396          g=rv1(nm)
397          h=rv1(k)
398          f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
399          g=sqrt(f*f+1d0)
400          f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x

```

```

401      c=1d0
402      s=1d0
403      do 47 j=1,nm
404          i=j+1
405          g=rv1(i)
406          y=u(i)
407          h=s*g
408          g=c*g
409          z=sqrt(f+f+h*h)
410          rv1(j)=z
411          c=f/z
412          s=h/z
413          f= (x*c)+(g*s)
414          g=-(x*s)+(g*c)
415          h=y*s
416          y=y*c
417          do 45 nm=1,n
418              x=v(nm,j)
419              z=v(nm,i)
420              v(nm,j)= (x*c)+(z*s)
421              v(nm,i)=-(x*s)+(z*c)
422 45      continue
423          z=sqrt(f+f+h*h)
424          w(j)=z
425          if (z.ne.0d0) then
426              z=1d0/z
427              c=f*z
428              s=h*z
429          endif
430          f= (c*g)+(s*y)
431          x=-(s*g)+(c*y)
432          do 46 nm=1,m
433              y=a(nm,j)
434              z=a(nm,i)
435              a(nm,j)= (y*c)+(z*s)
436              a(nm,i)=-(y*s)+(z*c)
437 46      continue
438 47      continue
439          rv1(1)=0d0
440          rv1(k)=f
441          w(k)=x
442 48      continue
443 3      continue
444 49      continue
445          return
446          end

```

```

1  c
2  c
3  c      program grdsrch.f
4  c
5  c
6      parameter(nn=638,mm=128)
7      real*8 bw(nn,mm),h(mm,nn),retilu(128),f(128),x(mm),no(nn)
8      real*8 v(mm,mm),w(mm),erru(mm,mm),retilc(128),err2(mm,mm)
9      real*8 errc(mm,mm),u(nn,mm)
10     real*8 y(nn),gg(mm,mm),d
11     integer*4 random>window,cntr,ni,nj,nt,m,mb,n,l,k,nl,nl2
12     character dfile*16
13  c
14     window=3
15     n=638
16     l=128
17     k=(n-1)/2.
18     print *, 'nt?'
19     read *, nt
20     print *, 'ni?'
21     read *, ni
22     print *, 'nj?'
23     read *, nj
24     print *, 'mb?'
25     read *, mb
26     print *, 'm?'
27     read *, m
28     print *, 'snr?'
29     read *, snr
30     print *, 'd/D?'
31     read *, d
32     print *, 'Destination File?'
33     read *, dfile
34     nl=256
35     nl2=nl/2
36     call srandom(29)
37     call bwmkcr(bw,u,k,n,m,d>window)
38     call ggmkcr(bw,n,m,gg)
39     call svdmkr(u,n,m,w,v)
40     call hmkcr(h,u,w,v,m,n)
41     do 103 j=1,mm
42         do 101 jj=1,mm
43             erru(j,jj)=0
44             errc(j,jj)=0
45 101     continue
46 103     continue
47     cntr=0
48     do 105 iv=1,nt
49  c      print *,iv
50         if (float(iv)/10.0-int(float(iv)/10.0) .lt. 1e-10) then
51             print *, iv
52         endif
53         call randm(x,no,m,n,snr,bw,y,mb)
54         call grdsrch(x,h,no,erru,n,m,nt,errc,y,bw,gg,ni,nj,cntr)
55 105     continue
56     print *, real(cntr)*real(nj)/real(nt)
57     call spectramkr(erru,m,retilu,f,snr,mb)
58     call spectramkr(errc,m,retilc,f,snr,mb)
59     call plotfl(dfile,nl2,'d',f,'d',retilu)
60     call plotfl(dfile,nl2,'d',f,'d',retilc)
61     stop
62     end
63  c
64  c
65  c
66     subroutine bwmkcr(bw,u,k,n,m,d>window)
67     real*8 bw(n,m),u(n,m),hh,d
68     integer*4 window

```

```

69      do 500 i=1,n
70      do 500 ii=1,m
71 500      bw(i,ii)=0d0
72      u(i,ii)=0d0
73      j=-k
74      do 510 i=1,1+2*k
75      bw(i,1)=hh(j,d,k>window)
76      u(i,1)=bw(i,1)
77      j=j+1
78 510      continue
79      do 520 i=1,m-1
80      do 530 ii=0,2*k
81      bw(1+ii+i,i+1)=bw(ii+i,i)
82      u(1+ii+i,i+1)=bw(ii+i,i)
83 530      continue
84 520      continue
85      return
86      end
87 c
88 c
89 c
90      subroutine ggmkx(bw,n,m,gg)
91      parameter(mm=128)
92      real*8 bw(n,m),gg(m,m)
93      do 600 i=1,m
94      do 610 ii=1,m
95      gg(i,ii)=0
96      do 620 iii=1,n
97      gg(i,ii)=gg(i,ii)+bw(iii,i)*bw(iii,ii)
98      gg(i,ii)=g(i,ii)
99 620      continue
100 610      continue
101 600      continue
102      return
103      end
104 c
105 c
106 c
107      subroutine randm(x,no,m,n,snr,bw,y,mb)
108 c
109 c      Generates random vectors x, no, and y=BWx+no. x has mb
110 c      independent Rayleigh components with variance snr and m-mb
111 c      zero components. no has n independent, zero mean, Gaussian
112 c      components with unit variance.
113 c
114      real*8 u1,u2,u3,s
115      real*8 x(m),no(n),bw(n,m),y(n)
116      integer*4 MAXINTV,random
117      parameter(MAXINTV=2147483647)
118      n2=n/2
119      do 500 i=1,n2
120 3      u1=real(random())/MAXINTV
121      if(u1.gt.1.or.u1.eq.0)goto 3
122 2      u2=real(random())/MAXINTV
123      if(u2.gt.1.or.u2.eq.0)goto 2
124      no(2*i-1)=sqrt(-2*log(u1))*cos(6.2831853*u2)
125      no(2*i)=sqrt(-2*log(u1))*sin(6.2831853*u2)
126 500      continue
127      s=snr/2
128      do 505 i=1,m
129 4      u3=real(random())/MAXINTV
130      if(u3.gt.1.or.u3.eq.0)goto 4
131      x(i)=sqrt(-2*log(u3))*sqrt(s)
132 505      continue
133      do 507 i=mb+1,m
134      x(i)=0
135 507      continue
136      do 510 i=1,n

```

```

137         y(i)=0
138         do 520 ii=1,m
139             y(i)=y(i)+bw(i,ii)*x(ii)
140 520         continue
141             y(i)=y(i)+no(i)
142 510         continue
143         return
144     end
145 c
146 c
147 c
148         subroutine grdsrch(x,h,no,erru,n,m,nt,errc,y,bw,gg,
149             *           ni,nj,cntr)
150         include 'fpedefs.f'
151         parameter(mm=128)
152         real*8 x(m),no(n),erru(m,m),h(m,n),eu(mm),zz,msq,msqold
153         real*8 errc(m,m),gg(m,m),grad(mm),dmsq,s(mm)
154         real*8 y(n),bw(n,m),z(mm),xg(mm)
155         real*8 alpha,beta,ss,sggs,tem1,tem2
156         integer cntr,nneg,list(mm),ni,nj,m,n,nt
157         nneg=0
158         msq=0
159 c ----- Compute unconstrained solution and BWy -----
160         do 700 i=1,m
161             xg(i)=0
162             z(i)=0
163             do 710 ii=1,n
164                 xg(i)=xg(i)+h(i,ii)*y(ii)
165                 z(i)=z(i)+bw(ii,i)*y(ii)
166 710             continue
167             eu(i)=xg(i)-x(i)
168             if(xg(i).lt.0)then
169                 nneg=nneg+1
170             endif
171 700         continue
172         if(nneg.eq.0) goto 200
173 c ----- Gradient Projection Algorithm -----
174 c ----- Disable overflow and divide-by-zero ---
175 c ----- floating point exceptions.
176         nabl=fpgetxnabl=()
177         newabl=and(nabl,not(DIV0))
178         newabl=and(newabl,not(OFLOW))
179         call fpsetxnabl(newabl)
180 100         msqold=msq
181         cntr=cntr+1
182         do 770 iv=1,nj
183             alpha=1.797d+308
184             ss=0d0
185             sggs=0d0
186 c ----- Find active constraints and generate list -----
187             do 705 i=1,m
188                 list(i)=0
189                 if(xg(i).le.0d0) then
190                     xg(i)=0d0
191                     list(i)=1
192                 endif
193 705             continue
194 c ----- Compute gradient and correction vector S -----
195 c ----- and pick step size alpha
196             do 712 i=1,m
197                 grad(i)=0d0
198                 do 713 ii=1,m
199                     grad(i)=grad(i)+gg(i,ii)*xg(ii)
200 713             continue
201             grad(i)=grad(i)-z(i)
202 c ----- Project gradient onto active, -----
203 c ----- nonobstructing constraints.
204             if(list(i).eq.1.and.grad(i).gt.0d0) then

```

```

205         s(i)=0d0
206     else
207         s(i)=-grad(i)
208     endif
209 c     ----- Determine step size alpha: -----
210 c     Determine minimum step size for next constraint,
211 c     Determine step size for minimum along direction S,
212 c     ----- Pick smaller of the two -----
213         if(list(i).ne.1) then
214             beta=xg(i)/grad(i)
215             if(beta.gt.0d0.and.beta.lt.alpha) then
216                 alpha=beta
217             endif
218         endif
219 712     continue
220         do 714 i=1,m
221             ss=ss+s(i)*s(i)
222             do 716 j=1,m
223                 sggs=sggs+s(i)*gg(i,j)*s(j)
224 716             continue
225 714         continue
226         beta=ss/sggs
227         if(beta.gt.0d0.and.beta.lt.alpha) then
228             alpha=beta
229 c     ----- compute new gradient and test for orthogonality -----
230 c     with the S direction.
231 c         zz=0d0
232 c         do 900 i=1,m
233 c             grad(i)=0d0
234 c             do 901 ii=1,m
235 c                 grad(i)=grad(i)+gg(i,ii)*(xg(ii)+alpha*s(ii))
236 c901             continue
237 c             grad(i)=grad(i)-z(i)
238 c             zz=zz+grad(i)*s(i)
239 c900             continue
240 c             print *, 'gTs =', zz
241 c     ----- End of Test -----
242         endif
243 c     ----- Compute new Solution -----
244         do 740 i=1,m
245             xg(i)=xg(i)+alpha*s(i)
246 740         continue
247 770         continue
248 c     ----- Check ||y-Gx|| squared -----
249         msq=0
250         do 780 i=1,n
251             zz=0
252             do 790 ii=1,m
253                 zz=zz+bw(i,ii)*xg(ii)
254 790             continue
255             zz=y(i)-zz
256             msq=msq+zz*zz
257 780         continue
258 c             print *, msq
259             dmsq=abs((msqold-msq)*(10**ni))
260             if (dmsq.gt.1e+9) goto 100
261             if (int(dmsq).ne.0) goto 100
262 c     ----- Enable Floating Point Exceptions ---
263         call fpsetxflags(0)
264         call fpsetxnabls(nabls)
265 c     ----- Accumulate Error Vector Outer Product and Return -----
266 200         do 720 i=1,m
267             print *, xg(i), grad(i), s(i)
268             tem1=xg(i)-x(i)
269             do 730 ii=1,m
270                 tem2=xg(ii)-x(ii)
271                 erru(i,ii)=erru(i,ii)+eu(i)*eu(ii)/nt
272                 errc(i,ii)=errc(i,ii)+tem1*tem2/nt

```

```

273 730      continue
274 720      continue
275          print *, msq
276          return
277          end
278 c
279 c
280 c
281      subroutine spectramkr(err,m,retil,f,snr,mb)
282      real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
283      integer p
284      pi2=6.283185307
285      do 10 i=1,m
286          sum(i)=0.0
287 10      continue
288      do 20 i=1,m
289          do 30 ii=1,m
290              p=abs(i-ii)+1
291              sum(p)=sum(p)+err(i,ii)
292 30      continue
293 20      continue
294      print *, sum(1)
295      do 50 i=1,128
296          xi=real(i)/256
297          retil(i)=0
298          do 60 ii=1,m
299              xii=ii-1
300              retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
301 60      continue
302 50      continue
303      do 80 i=1,128
304          f(i)=real(i)/64
305          retil(i)=retil(i)/(snr*mb)
306 80      continue
307      return
308      end
309 c
310 c
311 c
312      function hh(j,d,k>window)
313      real*8 hh,d,k1,pi,pi2,j1
314      integer*4 k,j>window
315      pi=3.141592654
316      pi2=2*pi
317      j1=j
318      k1=k
319      if(j.eq.0)then
320          hh=d*d
321      else
322          hh=cos(.78539816*dble(j)*(1-d))*sin(.78539816*d*dble(j))/
323 c      (.78539816*dble(j))
324          hh=hh*hh
325      endif
326      if (window.eq.1) then
327          hh=hh
328      elseif (window.eq.2) then
329          hh=hh*(1-abs(j1)/k1)
330      elseif (window.eq.3) then
331          hh=hh*(.5+.5*cos(pi*j1/k1))
332      elseif (window.eq.4) then
333          hh=hh*(0.54+0.46*cos(pi*j1/k1))
334      else
335          hh=hh*(0.42+0.5*cos(pi*j1/k1)+0.08*cos(pi2*j1/k1))
336      endif
337 10      return
338      end
339 c
340 c

```

```

341 c
342     subroutine hmk(h,u,w,v,m,n)
343 c
344     real*8 u(m,n),w(n),v(n,n),h(n,m)
345     integer m,n
346 c
347     do 10 i=1,n
348         do 20 j=1,m
349             do 30 k=1,n
350                 h(i,j)=(1/w(k))*v(i,k)*u(j,k)
351 30         continue
352 20     continue
353 10     continue
354 c
355 c
356 c
357     subroutine svdmkr(a,m,n,w,v)
358 c
359 c         This routine generates the SVD of an mxn matrix A
360 c         where A = UVVtranspose, with U mxn, and V and W are
361 c         nxn. U is column orthogonal, V is row and column
362 c         orthogonal, and W is diagonal. U is returned in the
363 c         array a. The diagonal of W is returned as the vector w.
364 c
365     implicit real*8 (a-h,o-z)
366     parameter (nmax=128)
367     dimension a(m,n),w(n),v(n,n),rv1(nmax)
368 c
369     g=0d0
370     scale=0d0
371     anorm=0d0
372     do 25 i=1,n
373         l=i+1
374         rv1(i)=scale*g
375         g=0d0
376         s=0d0
377         scale=0d0
378         if (i.le.m) then
379             do 11 k=i,m
380                 scale=scale+abs(a(k,i))
381 11         continue
382         if (scale.ne.0d0) then
383             do 12 k=i,m
384                 a(k,i)=a(k,i)/scale
385                 s=s+a(k,i)*a(k,i)
386 12         continue
387         f=a(i,i)
388         g=-sign(sqrt(s),f)
389         h=f*g-s
390         a(i,i)=f-g
391         if (i.ne.n) then
392             do 15 j=l,n
393                 s=0d0
394                 do 13 k=i,m
395                     s=s+a(k,i)*a(k,j)
396 13         continue
397         f=s/h
398         do 14 k=i,m
399             a(k,j)=a(k,j)+f*a(k,i)
400 14         continue
401 15         continue
402         endif
403         do 16 k= i,m
404             a(k,i)=scale*a(k,i)
405 16         continue
406         endif
407         endif
408         w(i)=scale *g

```

```

409      g=0d0
410      s=0d0
411      scale=0d0
412      if ((i.le.m).and.(i.ne.n)) then
413          do 17 k=1,n
414              scale=scale+abs(a(i,k))
415 17      continue
416          if (scale.ne.0d0) then
417              do 18 k=1,n
418                  a(i,k)=a(i,k)/scale
419                  s=s+a(i,k)*a(i,k)
420 18      continue
421          f=a(i,1)
422          g=-sign(sqrt(s),f)
423          h=f*g-s
424          a(i,1)=f-g
425          do 19 k=1,n
426              rv1(k)=a(i,k)/h
427 19      continue
428          if (i.ne.m) then
429              do 23 j=1,m
430                  s=0d0
431                  do 21 k=1,n
432                      s=s+a(j,k)*a(i,k)
433 21      continue
434                  do 22 k=1,n
435                      a(j,k)=a(j,k)+s*rv1(k)
436 22      continue
437 23      continue
438          endif
439          do 24 k=1,n
440              a(i,k)=scale*a(i,k)
441 24      continue
442          endif
443          anorm=max(anorm,(abs(v(i))+abs(rv1(i))))
444 25      continue
445          do 32 i=n,1,-1
446              if (i.lt.n) then
447                  if (g.ne.0d0) then
448                      do 26 j=1,n
449                          v(j,i)=(a(i,j)/a(i,1))/g
450 26      continue
451                      do 29 j=1,n
452                          s=0d0
453                          do 27 k=1,n
454                              s=s+a(i,k)*v(k,j)
455 27      continue
456                          do 28 k=1,n
457                              v(k,j)=v(k,j)+s*v(k,i)
458 28      continue
459 29      continue
460          endif
461          do 31 j=1,n
462              v(i,j)=0d0
463              v(j,i)=0d0
464 31      continue
465          endif
466          v(i,i)=1d0
467          g=rv1(i)
468          l=i
469 32      continue
470          do 39 i=n,1,-1
471              l=i+1
472              g=v(i)
473              if (i.lt.n) then
474                  do 33 j=1,n
475                      a(i,j)=0d0

```

```

477 33      continue
478      endif
479      if (g.ne.0d0) then
480          g=1d0/g
481          if (i.ne.n) then
482              do 36 j=1,n
483                  s=0d0
484                  do 34 k=1,m
485                      s=s+a(k,i)*a(k,j)
486 34          continue
487                      f=(s/a(i,i))*g
488                      do 35 k=i,m
489                          a(k,j)=a(k,j)+f*a(k,i)
490 35          continue
491 36          continue
492      endif
493      do 37 j=i,m
494          a(j,i)=a(j,i)*g
495 37      continue
496      else
497          do 38 j= i,m
498              a(j,i)=0d0
499 38          continue
500      endif
501      a(i,i)=a(i,i)+1d0
502 39      continue
503      do 49 k=n,1,-1
504          do 48 its=1,30
505              do 41 l=k,1,-1
506                  nm=l-1
507                  if ((abs(rv1(l))+anorm).eq.anorm) go to 2
508                  if ((abs(w(nm))+anorm).eq.anorm) go to 1
509 41          continue
510 1          c=0d0
511              s=1d0
512              do 43 i=1,k
513                  f=s*rv1(i)
514                  if ((abs(f)+anorm).ne.anorm) then
515                      g=w(i)
516                      h=sqrt(f*f+g*g)
517                      w(i)=h
518                      h=1d0/h
519                      c= (g*h)
520                      s=-(f*h)
521                      do 42 j=1,m
522                          y=a(j,nm)
523                          z=a(j,i)
524                          a(j,nm)=(y*c)+(z*s)
525                          a(j,i)=- (y*s)+(z*c)
526 42          continue
527              endif
528 43          continue
529 2          z=w(k)
530              if (l.eq.k) then
531                  if (z.lt.0d0) then
532                      w(k)=-z
533                      do 44 j=1,n
534                          v(j,k)=-v(j,k)
535 44          continue
536              endif
537              go to 3
538          endif
539          if (its.eq.30) pause 'no convergence in 30 iterations'
540          x=w(l)
541          nm=k-1
542          y=w(nm)
543          g=rv1(nm)
544          h=rv1(k)

```

```

545      f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
546      g=sqrt(f*f+1d0)
547      f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x
548      c=1d0
549      s=1d0
550      do 47 j=1,nm
551          i=j+1
552          g=rvi(i)
553          y=w(i)
554          h=s*g
555          g=c*g
556          z=sqrt(f*f+h*h)
557          rvi(j)=z
558          c=f/z
559          s=h/z
560          f= (x*c)+(g*s)
561          g=-(x*s)+(g*c)
562          h=y*s
563          y=y*c
564          do 45 nm=1,n
565              x=v(nm,j)
566              z=v(nm,i)
567              v(nm,j)= (x*c)+(z*s)
568              v(nm,i)=-(x*s)+(z*c)
569 45      continue
570          z=sqrt(f*f+h*h)
571          w(j)=z
572          if (z.ne.0d0) then
573              z=1d0/z
574              c=f*z
575              s=h*z
576          endif
577          f= (c*g)+(s*y)
578          x=-(s*g)+(c*y)
579          do 46 nm=1,m
580              y=a(nm,j)
581              z=a(nm,i)
582              a(nm,j)= (y*c)+(z*s)
583              a(nm,i)=-(y*s)+(z*c)
584 46      continue
585 47      continue
586          rvi(1)=0d0
587          rvi(k)=f
588          w(k)=x
589 48      continue
590 3      continue
591 49      continue
592      return
593      end

```

```

1  c
2  c
3  c      program minvar.f
4  c
5  c
6      parameter(mm=128)
7      real*8 bw(1150,mm),retil(128),f(128)
8      real*8 g(mm,mm),d,snr
9      real*4 eps
10     integer*4 m>window,k,l,n,im,i,ii,ier
11     character*16 dfile
12  c
13     print *, 'k?'
14     read *, k
15     print *, 'Window number?'
16     read *, window
17     print *, 'Destination File?'
18     read *, dfile
19     l=128
20     n=2*k+1
21     eps=1e-6
22     do 90 im=1,6
23         m=2*(im+1)
24         do 100 i=1,5
25             d=dbl(i)/1d+1
26             call bwmkcr(bw,k,n,m,d>window)
27             do 110 ii=2,4
28                 snr=10**ii
29                 call two(bw,n,m,g,snr,ier,eps)
30                 call spectramkr(g,m,retil,f)
31                 call plotfl(dfile,128,'d',f,'d',retil)
32 110         continue
33 100     continue
34 90     continue
35     stop
36     end
37  c
38  c
39  c
40     subroutine two(bw,n,m,g,snr,ier,eps)
41  c      subroutine to calculate h=(gtg)*snr + i inv
42     parameter(mm=128)
43     real*8 bw(n,m),g(m,m),r(8256),snr
44     real*4 eps
45     integer ier
46     do 600 i=1,m
47         do 610 ii=1,m
48             g(i,ii)=0
49             do 620 iii=1,n
50                 g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)*snr
51 620         continue
52         if(i.eq.ii)g(i,ii)=g(i,ii)+1d+0
53 610     continue
54 600     continue
55     call squartri(r,m,g)
56     call dsinv(r,m,eps,ier)
57     call trisquar(g,m,r)
58     return
59     end
60  c
61  c
62  c
63     subroutine bwmkcr(bw,k,n,m,d>window)
64     real*8 bw(n,m),hh,d
65     integer n,m,k>window
66     do 500 i=1,n
67         do 500 ii=1,m
68 500         bw(i,ii)=0

```

```

69      j=-k
70      do 510 i=1,1+2*k
71          bw(i,1)=hh(j,d,k>window)
72          j=j+1
73 510      continue
74      do 520 i=1,m-1
75          do 530 ii=0,2*k
76              bw(1+ii+i,i+1)=bw(ii+i,i)
77 530      continue
78 520      continue
79      return
80      end
81  c
82  c
83  c
84      function hh(j,d,k>window)
85      real*8 hh,d,k1,pi,pi2,j1
86      integer k,j,m>window
87      m>window
88      pi=3.141592654
89      pi2=2*pi
90      j1=j
91      k1=k
92      if(j.eq.0)then
93          hh=d*d
94      else
95          hh=cos(.78539816*dble(j)*(1-d))*sin(.78539816*d*dble(j))/
96  c      (.78539816*dble(j))
97          hh=hh*hh
98      endif
99      if (m.eq.1) then
100         hh=hh
101     elseif (m.eq.2) then
102         hh=hh*(1-abs(j1)/k1)
103     elseif (m.eq.3) then
104         hh=hh*(.5+.5*cos(pi*j1/k1))
105     elseif (m.eq.4) then
106         hh=hh*(0.54+0.46*cos(pi*j1/k1))
107     else
108         hh=hh*(0.42+0.5*cos(pi*j1/k1)+0.08*cos(pi2*j1/k1))
109     endif
110 10      return
111      end
112  c
113  c
114  c
115      subroutine spectramkr(err,m,retil,f)
116      real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
117      integer p
118      pi2=6.283185307
119      do 10 i=1,m
120          sum(i)=0.0
121 10      continue
122      do 20 i=1,m
123          do 30 ii=1,m
124              p=abs(i-ii)+1
125              sum(p)=sum(p)+err(i,ii)
126 30      continue
127 20      continue
128      do 50 i=1,128
129          xi=real(i)/256
130          retil(i)=0
131          do 60 ii=1,m
132              xii=ii-1
133              retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
134 60      continue
135 50      continue
136      do 80 i=1,128

```

```
137      f(i)=real(i)/64
138      retil(i)=retil(i)/dble(m)
139 80    continue
140      return
141      end
```

```

1 c
2 c
3 c      program snrakr.f
4 c
5 c
6      real*4 sum,snr(128),f1(128),lg10
7      real*8 f(128),e(128)
8      character*12 file1
9      character*14 file2
10     file1='m16mb16d1w5.'
11     file2='snrm16mb16d1w5'
12     call readfl(file1//'2',n,'d',f,'d',e,1,1,128)
13     lg10=log(1e1)
14     sum=0
15     do 10 i=1,128
16         sum=sum+e(i)
17 c         snr(i)=real(i)/sum
18         snr(i)=10*log(real(i)/sum)/lg10
19         f1(i)=real(i)/64
20 10     continue
21     call plotfl(file2,128,'f',f1,'f',snr)
22     call readfl(file1//'2',n,'d',f,'d',e,3,1,128)
23     sum=0
24     do 20 i=1,128
25         sum=sum+e(i)
26 c         snr(i)=real(i)/sum
27         snr(i)=10*log(real(i)/sum)/lg10
28 20     continue
29     call plotfl(file2,128,'f',f1,'f',snr)
30     call readfl(file1//'3',n,'d',f,'d',e,1,1,128)
31     sum=0
32     do 30 i=1,128
33         sum=sum+e(i)
34 c         snr(i)=real(i)/sum
35         snr(i)=10*log(real(i)/sum)/lg10
36 30     continue
37     call plotfl(file2,128,'f',f1,'f',snr)
38     call readfl(file1//'3',n,'d',f,'d',e,3,1,128)
39     sum=0
40     do 40 i=1,128
41         sum=sum+e(i)
42 c         snr(i)=real(i)/sum
43         snr(i)=10*log(real(i)/sum)/lg10
44 40     continue
45     call plotfl(file2,128,'f',f1,'f',snr)
46     call readfl(file1//'4',n,'d',f,'d',e,1,1,128)
47     sum=0
48     do 50 i=1,128
49         sum=sum+e(i)
50 c         snr(i)=real(i)/sum
51         snr(i)=10*log(real(i)/sum)/lg10
52 50     continue
53     call plotfl(file2,128,'f',f1,'f',snr)
54     call readfl(file1//'4',n,'d',f,'d',e,3,1,128)
55     sum=0
56     do 60 i=1,128
57         sum=sum+e(i)
58 c         snr(i)=real(i)/sum
59         snr(i)=10*log(real(i)/sum)/lg10
60 60     continue
61     call plotfl(file2,128,'f',f1,'f',snr)
62     call readfl(file1//'5',n,'d',f,'d',e,1,1,128)
63     sum=0
64     do 70 i=1,128
65         sum=sum+e(i)
66 c         snr(i)=real(i)/sum
67         snr(i)=10*log(real(i)/sum)/lg10
68         f1(i)=real(i)/64

```

```

69 70      continue
70      call plotfl(file2,128,'f',f1,'f',snr)
71      call readfl(file1//'5',n,'d',f,'d',e,3,1,128)
72      sum=0
73      do 80 i=1,128
74          sum=sum+e(i)
75      c      snr(i)=real(i)/sum
76          snr(i)=10*log(real(i)/sum)/lg10
77 80      continue
78      call plotfl(file2,128,'f',f1,'f',snr)
79      call readfl(file1//'6',n,'d',f,'d',e,1,1,128)
80      sum=0
81      do 90 i=1,128
82          sum=sum+e(i)
83      c      snr(i)=real(i)/sum
84          snr(i)=10*log(real(i)/sum)/lg10
85 90      continue
86      call plotfl(file2,128,'f',f1,'f',snr)
87      call readfl(file1//'6',n,'d',f,'d',e,3,1,128)
88      sum=0
89      do 100 i=1,128
90          sum=sum+e(i)
91      c      snr(i)=real(i)/sum
92          snr(i)=10*log(real(i)/sum)/lg10
93 100     continue
94      call plotfl(file2,128,'f',f1,'f',snr)
95      call readfl(file1//'7',n,'d',f,'d',e,1,1,128)
96      sum=0
97      do 110 i=1,128
98          sum=sum+e(i)
99      c      snr(i)=real(i)/sum
100         snr(i)=10*log(real(i)/sum)/lg10
101 110     continue
102         call plotfl(file2,128,'f',f1,'f',snr)
103         call readfl(file1//'7',n,'d',f,'d',e,3,1,128)
104         sum=0
105         do 120 i=1,128
106             sum=sum+e(i)
107         c      snr(i)=real(i)/sum
108             snr(i)=10*log(real(i)/sum)/lg10
109 120     continue
110         call plotfl(file2,128,'f',f1,'f',snr)
111         call readfl(file1//'8',n,'d',f,'d',e,1,1,128)
112         sum=0
113         do 130 i=1,128
114             sum=sum+e(i)
115         c      snr(i)=real(i)/sum
116             snr(i)=10*log(real(i)/sum)/lg10
117 130     continue
118         call plotfl(file2,128,'f',f1,'f',snr)
119         call readfl(file1//'8',n,'d',f,'d',e,3,1,128)
120         sum=0
121         do 140 i=1,128
122             sum=sum+e(i)
123         c      snr(i)=real(i)/sum
124             snr(i)=10*log(real(i)/sum)/lg10
125 140     continue
126         call plotfl(file2,128,'f',f1,'f',snr)
127         stop
128         end

```

```

1 c
2 c
3 c      routine svd.f
4 c
5 c
6 c      This routine generates the SVD decomposition of an
7 c      mxn matrix A = UWVtranspose, where U is mxn, and V
8 c      and W are nxn. U is column orthogonal, V is row and
9 c      column orthogonal, and W is diagonal. U is returned
10 c      in the array a.
11 c
12 c
13 c
14      subroutine svdcmp(a,m,n,w,v)
15      implicit real*8 (a-h,o-z)
16      parameter (nmax=128)
17      dimension a(m,n),w(n),v(n,n),rv1(nmax)
18 c
19      g=0d0
20      scale=0d0
21      anorm=0d0
22      do 25 i=1,n
23          l=i+1
24          rv1(i)=scale*g
25          g=0d0
26          s=0d0
27          scale=0d0
28          if (i.le.m) then
29              do 11 k=i,m
30                  scale=scale+abs(a(k,i))
31 11          continue
32              if (scale.ne.0d0) then
33                  do 12 k=i,m
34                      a(k,i)=a(k,i)/scale
35                      s=s+a(k,i)*a(k,i)
36 12          continue
37              f=a(i,i)
38              g=-sign(sqrt(s),f)
39              h=f*g-s
40              a(i,i)=f-g
41              if (i.ne.n) then
42                  do 15 j=l,n
43                      s=0d0
44                      do 13 k=i,m
45                          s=s+a(k,i)*a(k,j)
46 13          continue
47                      f=s/h
48                      do 14 k=i,m
49                          a(k,j)=a(k,j)+f*a(k,i)
50 14          continue
51              continue
52              endif
53              do 16 k=i,m
54                  a(k,i)=scale*a(k,i)
55 16          continue
56          endif
57      endif
58      w(i)=scale*g
59      g=0d0
60      s=0d0
61      scale=0d0
62      if ((i.le.m).and.(i.ne.n)) then
63          do 17 k=l,n
64              scale=scale+abs(a(i,k))
65 17          continue
66              if (scale.ne.0d0) then
67                  do 18 k=l,n
68                      a(i,k)=a(i,k)/scale

```

```

69         s=s+a(i,k)*a(i,k)
70 18      continue
71         f=a(i,1)
72         g=-sign(sqrt(s),f)
73         h=f*g-s
74         a(i,1)=f-g
75         do 19 k=1,n
76             rv1(k)=a(i,k)/h
77 19      continue
78         if (i.ne.m) then
79             do 23 j=1,m
80                 s=0d0
81                 do 21 k=1,n
82                     s=s+a(j,k)*a(i,k)
83 21      continue
84                 do 22 k=1,n
85                     a(j,k)=a(j,k)+s*rv1(k)
86 22      continue
87 23      continue
88         endif
89         do 24 k=1,n
90             a(i,k)=scale*a(i,k)
91 24      continue
92         endif
93     endif
94     anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
95 25      continue
96     do 32 i=n,1,-1
97         if (i.lt.n) then
98             if (g.ne.0d0) then
99                 do 26 j=1,n
100                     v(j,i)=(a(i,j)/a(i,1))/g
101 26      continue
102                 do 29 j=1,n
103                     s=0d0
104                     do 27 k=1,n
105                         s=s+a(i,k)*v(k,j)
106 27      continue
107                     do 28 k=1,n
108                         v(k,j)=v(k,j)+s*v(k,i)
109 28      continue
110 29      continue
111         endif
112         do 31 j=1,n
113             v(i,j)=0d0
114             v(j,i)=0d0
115 31      continue
116         endif
117         v(i,i)=1d0
118         g=rv1(i)
119         l=i
120 32      continue
121     do 39 i=n,1,-1
122         l=i+1
123         g=w(i)
124         if (i.lt.n) then
125             do 33 j=1,n
126                 a(i,j)=0d0
127 33      continue
128         endif
129         if (g.ne.0d0) then
130             g=1d0/g
131             if (i.ne.n) then
132                 do 36 j=1,n
133                     s=0d0
134                     do 34 k=1,m
135                         s=s+a(k,i)*a(k,j)
136 34      continue

```

```

137         f=(s/a(i,i))*g
138         do 35 k=i,m
139             a(k,j)=a(k,j)+f*a(k,i)
140 35         continue
141 36         continue
142     endif
143     do 37 j=i,m
144         a(j,i)=a(j,i)*g
145 37     continue
146     else
147         do 38 j= i,m
148             a(j,i)=0d0
149 38         continue
150     endif
151     a(i,i)=a(i,i)+1d0
152 39     continue
153     do 49 k=n,1,-1
154         do 48 its=1,30
155             do 41 l=k,1,-1
156                 nm=l-1
157                 if ((abs(rv1(l))+anorm).eq.anorm) go to 2
158                 if ((abs(v(nm))+anorm).eq.anorm) go to 1
159 41             continue
160 1             c=0d0
161             s=1d0
162             do 43 i=l,k
163                 f=s*rv1(i)
164                 if ((abs(f)+anorm).ne.anorm) then
165                     g=w(i)
166                     h=sqrt(f*f+g*g)
167                     w(i)=h
168                     h=1d0/h
169                     c= (g*h)
170                     s=- (f*h)
171                     do 42 j=1,m
172                         y=a(j,nm)
173                         z=a(j,i)
174                         a(j,nm)=(y*c)+(z*s)
175                         a(j,i)=-(y*s)+(z*c)
176 42                     continue
177                 endif
178 43             continue
179 2             z=w(k)
180             if (l.eq.k) then
181                 if (z.lt.0d0) then
182                     w(k)=-z
183                     do 44 j=1,n
184                         v(j,k)=-v(j,k)
185 44                     continue
186                 endif
187                 go to 3
188             endif
189             if (its.eq.30) pause 'no convergence in 30 iterations'
190             x=w(l)
191             nm=k-1
192             y=w(nm)
193             g=rv1(nm)
194             h=rv1(k)
195             f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
196             g=sqrt(f*f+1d0)
197             f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x
198             c=1d0
199             s=1d0
200             do 47 j=l,nm
201                 i=j+1
202                 g=rv1(i)
203                 y=w(i)
204                 h=s*g

```

```

205      g=c*g
206      z=sqrt(f*f+h*h)
207      rv1(j)=z
208      c=f/z
209      s=h/z
210      f= (x*c)+(g*s)
211      g=-(x*s)+(g*c)
212      h=y*s
213      y=y*c
214      do 45 nm=1,n
215          x=v(nm,j)
216          z=v(nm,i)
217          v(nm,j)= (x*c)+(z*s)
218          v(nm,i)=-(x*s)+(z*c)
219 45      continue
220      z=sqrt(f*f+h*h)
221      w(j)=z
222      if (z.ne.0d0) then
223          z=1d0/z
224          c=f*z
225          s=h*z
226      endif
227      f= (c*g)+(s*y)
228      x=-(s*g)+(c*y)
229      do 46 nm=1,m
230          y=a(nm,j)
231          z=a(nm,i)
232          a(nm,j)= (y*c)+(z*s)
233          a(nm,i)=-(y*s)+(z*c)
234 46      continue
235 47      continue
236      rv1(1)=0d0
237      rv1(k)=f
238      w(k)=x
239 48      continue
240 3      continue
241 49      continue
242      return
243      end

```

```

1 c
2 c
3 c      program minvar2.f
4 c
5 c
6      parameter(mm=128)
7      real*8 bw(1150,mm),retil(128),f(128)
8      real*8 g(mm,mm),d,snr
9      real*4 eps
10     integer*4 m,n,im,i,ii,ni,ier,max,uniform,apertures
11     character*16 dfile
12 c
13     n=1024
14     print *, 'snr?'
15     read *, snr
16     print *, 'd/D?'
17     read *, d
18     print *, 'uniformity?(yes=1,no=0)'
19     read *, uniform
20     print *, 'number of apertures?'
21     read *, apertures
22     print *, 'Largest support?'
23     read *, max
24     print *, 'Destination File?'
25     read *, dfile
26     eps=1e-6
27     do 90 im=1,max
28         m=4*im
29         call bwmkcr(bw,n,m,d,uniform,apertures)
30         call two(bw,n,m,g,snr,ier,eps)
31         call spectramkr(g,m,retil,f)
32         call plotfl(dfile,128,'d',f,'d',retil)
33 90     continue
34     stop
35     end
36 c
37 c
38 c
39     subroutine two(bw,n,m,g,snr,ier,eps)
40 c      subroutine to calculate h=(gtg)*snr + i inv
41     parameter(mm=128)
42     real*8 bw(n,m),g(m,m),r(8256),snr
43     real*4 eps
44     integer ier
45     do 600 i=1,m
46         do 610 ii=1,m
47             g(i,ii)=0
48             do 620 iii=1,n
49                 g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)*snr
50 620         continue
51         if(i.eq.ii)g(i,ii)=g(i,ii)+1d+0
52 610     continue
53 600     continue
54     call squartri(r,m,g)
55     call dsinv(r,m,eps,ier)
56     call trisquar(g,m,r)
57     return
58     end
59 c
60 c
61 c
62     subroutine bwmkcr(bw,n,m,d,uniform,apertures)
63     real*8 bw(n,m),hh,d
64     integer n,m,uniform,apertures
65     do 520 i=1,n
66         do 530 ii=1,m
67             bw(i,ii)=hh*((m-n)/2-ii+i,d,uniform,apertures)
68 530     continue

```

```

69 520      continue
70          return
71          end
72 c
73 c
74 c
75      function hh(j,d,uniform,apertures)
76      real*8 hh,d,pi2,d1,j1,j2,x1,x2,xj,xk,xi
77      integer j,k,l,uniform,apertures
78 c
79      k=apertures
80      xk=k
81      if(j.eq.0)then
82          hh=xk*d/4d0
83      else
84          pi2=2d0*atan(1d0)
85          d1=1d0-d
86          xj=j
87          j1=xj*d1*pi2
88          j2=xj*d*pi2/2d0
89          x1=sin(j2)/j2
90          x1=x1*x1*d/4d0
91          x2=0d0
92          if(uniform.eq.1) then
93              do 10 i=1,k-1
94                  xi=i
95                  x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
96 10          continue
97              hh=x1*(1d0+2d0*x2)
98          elseif(uniform.eq.0.and.apertures.eq.3) then
99              do 20 i=1,3
100                 xi=i
101                 x2=x2+cos(xi*j1/3d0)
102 20          continue
103                 hh=x1*(1d0+2d0*x2/3d0)
104             elseif(uniform.eq.0.and.apertures.eq.4) then
105                 do 30 i=1,6
106                     xi=i
107                     x2=x2+cos(xi*j1/6d0)
108 30          continue
109                 hh=x1*(1d0+x2/2d0)
110             elseif(uniform.eq.0.and.apertures.eq.5) then
111                 do 40 i=1,11
112                     if(i.eq.10) goto 40
113                     xi=i
114                     x2=x2+cos(xi*j1/11d0)
115 40          continue
116                 hh=x1*(1d0+2d0*x2/5d0)
117             elseif(uniform.eq.0.and.apertures.eq.6) then
118                 do 50 i=1,17
119                     if(i.eq.14.or.i.eq.15) goto 50
120                     xi=i
121                     x2=x2+cos(xi*j1/17d0)
122 50          continue
123                 hh=x1*(1d0+x2/3d0)
124             endif
125         endif
126         return
127         end
128 c
129 c
130 c
131      subroutine spectrankr(err,m,retil,f)
132      real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
133      integer p
134      pi2=6.283185307
135      do 10 i=1,m
136          sum(i)=0.0

```

```

137 10      continue
138          do 20 i=1,m
139              do 30 ii=1,m
140                  p=abs(i-ii)+1
141                  sum(p)=sum(p)+err(i,ii)
142 30      continue
143 20      continue
144          do 50 i=1,128
145              xi=real(i)/256
146              retil(i)=0
147              do 60 ii=1,m
148                  xii=ii-1
149                  retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
150 60      continue
151 50      continue
152          do 80 i=1,128
153              f(i)=real(i)/64
154              retil(i)=retil(i)/dble(m)
155 80      continue
156          return
157          end

```

```

1 c
2 c
3 c      program snrmkr2 f
4 c
5 c
6      real*4 sum,snr(128),f1(128),lg10,slice(32),count(32)
7      real*8 f(128),e(128)
8 c
9      lg10=log(1e1)
10 c    do 5 j=1,180
11 c    do 5 j=1,8
12 c    do 5 j=1,32
13 c      call readfl('minvar2n1024',n,'d',f,'d',e,j,1,128)
14 c      call readfl('newmtfd005s6',n,'d',f,'d',e,j,1,128)
15 c      call readfl('s5d005u0a6s32',n,'d',f,'d',e,j,1,128)
16 c      call readfl('s4d005u1a11s32',n,'d',f,'d',e,j,1,128)
17 c      call readfl('s6d005u1a6s8',n,'d',f,'d',e,j,1,128)
18      sum=0
19      do 10 i=1,128
20        sum=sum+e(i)
21        snr(i)=10*log(real(i)/sum)/lg10
22        f1(i)=real(i)/64
23 10    continue
24 c      call plotfl('snrminvarn1024',128,'f',f1,'f',snr)
25 c      call plotfl('snrnwtfd005s6',128,'f',f1,'f',snr)
26 c      call plotfl('snrs5d005u0a6',128,'f',f1,'f',snr)
27 c      call plotfl('snrs4d005u1a11',128,'f',f1,'f',snr)
28 c      slice(j)=snr(64)
29 c      slice(j)=snr(52)
30 c      slice(j)=snr(49)
31      count(j)=j
32 5    continue
33      call plotfl('slcs5d005u0a6',32,'f',count,'f',slice)
34 c      call plotfl('slcs4d005u1a11',32,'f',count,'f',slice)
35 c      call plotfl('slcs6d005u1a6',8,'f',count,'f',slice)
36      stop
37      end

```

APPENDIX C for Chapter 4

Computer Listings

This appendix contains the Fortran source code of the computer programs used to generate the data for this chapter.

```

1  c
2  c
3  c      program minvar2.f
4  c
5  c
6      parameter(mm=128)
7      real*8 bw(1150,mm),retil(128),f(128)
8      real*8 g(mm,mm),d,snr
9      real*4 eps
10     integer*4 m,n,im,i,ii,ni,ier,max,uniform,apertures
11     character*16 dfile
12  c
13     n=1024
14     print *, 'snr?'
15     read *, snr
16     print *, 'd/D?'
17     read *, d
18     print *, 'uniformity?(yes=1,no=0)'
19     read *, uniform
20     print *, 'number of apertures?'
21     read *, apertures
22     print *, 'Largest support?'
23     read *, max
24     print *, 'Destination File?'
25     read *, dfile
26     eps=1e-6
27     do 90 im=1,max
28         m=4*im
29         call bwmkcr(bw,n,m,d,uniform,apertures)
30         call two(bw,n,m,g,snr,ier,eps)
31         call spectramkr(g,m,retil,f)
32         call plotfl(dfile,128,'d',f,'d',retil)
33 90     continue
34     stop
35     end
36  c
37  c
38  c
39     subroutine two(bw,n,m,g,snr,ier,eps)
40  c      subroutine to calculate h=(gtg)*snr + i inv
41     parameter(mm=128)
42     real*8 bw(n,m),g(m,m),x(8256),snr
43     real*4 eps
44     integer ier
45     do 600 i=1,m
46         do 610 ii=1,m
47             g(i,ii)=0
48             do 620 iii=1,m
49                 g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)*snr
50 620         continue
51         if(i.eq.ii)g(i,ii)=g(i,ii)+1d+0
52 610     continue
53 600     continue
54     call squartri(r,m,g)
55     call dsinv(r,m,eps,ier)
56     call trisquar(g,m,x)
57     return
58     end
59  c

```

```

60 c
61 c
62      subroutine bwmkr(bw,n,m,d,uniform,apertures)
63      real*8 bw(n,m),hh,d
64      integer n,m,uniform,apertures
65      do 520 i=1,n
66          do 530 ii=1,m
67              bw(i,ii)=hh((m-n)/2-ii+i,d,uniform,apertures)
68 530      continue
69 520      continue
70      return
71      end
72 c
73 c
74 c
75      function hh(j,d,uniform,apertures)
76      real*8 hh,d,pi2,d1,j1,j2,x1,x2,xj,xk,xi
77      integer j,k,l,uniform,apertures
78 c
79      k=apertures
80      xk=k
81      if(j.eq.0)then
82          hh=xk*d/4d0
83      else
84          pi2=2d0*atan(1d0)
85          d1=1d0-d
86          xj=j
87          j1=xj*d1*pi2
88          j2=xj*d*pi2/2d0
89          x1=sin(j2)/j2
90          x1=x1*x1*d/4d0
91          x2=0d0
92          if(uniform.eq.1) then
93              do 10 i=1,k-1
94                  xi=i
95                  x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
96 10      continue
97                  hh=x1*(1d0+2d0*x2)
98          elseif(uniform.eq.0.and.apertures.eq.3) then
99              do 20 i=1,3
100                  xi=i
101                  x2=x2+cos(xi*j1/3d0)
102 20      continue
103                  hh=x1*(1d0+2d0*x2/3d0)
104          elseif(uniform.eq.0.and.apertures.eq.4) then
105              do 30 i=1,6
106                  xi=i
107                  x2=x2+cos(xi*j1/6d0)
108 30      continue
109                  hh=x1*(1d0+x2/2d0)
110          elseif(uniform.eq.0.and.apertures.eq.5) then
111              do 40 i=1,11
112                  if(i.eq.10) goto 40
113                  xi=i
114                  x2=x2+cos(xi*j1/11d0)
115 40      continue
116                  hh=x1*(1d0+2d0*x2/5d0)
117          elseif(uniform.eq.0.and.apertures.eq.6) then
118              do 50 i=1,17
119                  if(i.eq.14.or.i.eq.15) goto 50
120                  xi=i
121                  x2=x2+cos(xi*j1/17d0)
122 50      continue
123                  hh=x1*(1d0+x2/3d0)
124      endif
125      endif
126      return
127      end

```

```

128 c
129 c
130 c
131      subroutine spectramkr(err,m,retil,f)
132      real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
133      integer p
134      pi2=6.283185307
135      do 10 i=1,m
136          sum(i)=0.0
137 10      continue
138      do 20 i=1,m
139          do 30 ii=1,m
140              p=abs(i-ii)+1
141              sum(p)=sum(p)+err(i,ii)
142 30      continue
143 20      continue
144      do 50 i=1,128
145          xi=real(i)/256
146          retil(i)=0
147          do 60 ii=1,m
148              xii=ii-1
149              retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
150 60      continue
151 50      continue
152      do 80 i=1,128
153          f(i)=real(i)/64
154          retil(i)=retil(i)/dble(m)
155 80      continue
156      return
157      end

```

```

1 c
2 c
3 c      program snrmkr2.f
4 c
5 c
6      real*4 sum,snr(128),f1(128),lg10,slice(32),count(32)
7      real*8 f(128),e(128)
8 c
9      lg10=log(1e1)
10 c      do 5 j=1,180
11 c      do 5 j=1,8
12 c      do 5 j=1,32
13 c          call readfl('minvar2n1024',n,'d',f,'d',e,j,1,128)
14 c          call readfl('newmtfd005s6',n,'d',f,'d',e,j,1,128)
15 c          call readfl('s5d005u0a6s32',n,'d',f,'d',e,j,1,128)
16 c          call readfl('s4d005u1a11s32',n,'d',f,'d',e,j,1,128)
17 c          call readfl('s6d005u1a6s8',n,'d',f,'d',e,j,1,128)
18 c          sum=0
19 c          do 10 i=1,128
20 c              sum=sum+e(i)
21 c              snr(i)=10*log(real(i)/sum)/lg10
22 c              f1(i)=real(i)/64
23 10      continue
24 c          call plotfl('snrminvarn1024',128,'f',f1,'f',snr)
25 c          call plotfl('snrnwtfd005s6',128,'f',f1,'f',snr)
26 c          call plotfl('snrs5d005u0a6',128,'f',f1,'f',snr)
27 c          call plotfl('snrs4d005u1a11',128,'f',f1,'f',snr)
28 c          slice(j)=snr(64)
29 c          slice(j)=snr(52)
30 c          slice(j)=snr(49)
31 c          count(j)=j
32 5      continue
33 c          call plotfl('slcs5d005u0a6',32,'f',count,'f',slice)
34 c          call plotfl('slcs4d005u1a11',32,'f',count,'f',slice)
35 c          call plotfl('slcs6d005u1a6',8,'f',count,'f',slice)
36      stop
37      end

```

```

1  c
2  c
3  c      program lsqmr2.f
4  c
5  c
6      parameter(mm=128)
7      real*8 bw(1150,mm),f(128),v(mm,mm),w(mm)
8      real*8 g(mm,mm),retil(128),d
9      integer*4 i,m,mb,n
10     integer*4 max,uniform,apertures
11     character dfile*16
12  c
13  c      n=1024
14  c      n=512
15      print *, 'Number of pixels in image line?'
16      read *, n
17      print *, 'snr?'
18      read *, snr
19      print *, 'd/D?'
20      read *, d
21      print *, 'uniformity?(yes=1,no=0)'
22      read *, uniform
23      print *, 'number of apertures?'
24      read *, apertures
25      print *, 'largest support?'
26      read *, max
27      print *, 'Destination File?'
28      read *, dfile
29      do 5 i=1,max
30          m=2*i
31          mb=m
32          call bwmkr(bw,n,m,d,uniform,apertures)
33          call svdcmp(bw,n,m,w,v)
34          call gmkr(w,v,m,g)
35          call spectrankr(g,m,retil,f,snr,mb)
36          call plotfl(dfile,128,'d',f,'d',retil)
37  5      continue
38      stop
39      end
40  c
41  c
42  c
43      subroutine bwmkr(bw,n,m,d,uniform,apertures)
44      real*8 bw(n,m),hh,d
45      integer n,m,uniform,apertures
46      do 520 i=1,n
47          do 530 ii=1,m
48              bw(i,ii)=hh*((m-n)/2-ii+i,d,uniform,apertures)
49  530      continue
50  520      continue
51      return
52      end
53  c
54  c
55  c
56      subroutine gmkr(w,v,m,g)
57      real*8 g(m,m),v(m),w(m,m)
58  c
59      do 600 i=1,m
60          do 610 ii=1,m
61              g(i,ii)=w(i,ii)/v(ii)
62  610      continue
63  600      continue
64      return
65      end
66  c
67  c
68  c

```

```

69      subroutine spectramkr(err,m,retil,f,snr,mb)
70      real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
71      integer p
72      pi2=8d0*atan(1d0)
73      do 10 i=1,m
74          sum(i)=0d0
75      10  continue
76      do 20 i=1,m
77          do 30 ii=1,m
78              p=abs(i-ii)+1
79              do 40 iii=1,m
80                  sum(p)=sum(p)+err(i,iii)*err(ii,iii)
81          40  continue
82      30  continue
83      20  continue
84      print *, sum(1)
85      do 50 i=1,128
86          xi=real(i)/256
87          retil(i)=0
88          do 60 ii=1,m
89              xii=ii-1
90              retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
91      60  continue
92      50  continue
93      do 80 i=1,128
94          f(i)=real(i)/128
95          retil(i)=retil(i)/(snr*mb)
96      80  continue
97      return
98      end
99  c
100 c
101 c
102 function hh(j,d,uniform,apertures)
103 real*8 hh,d,pi,d1,j1,j2,x1,x2,xj,xk,xi
104 integer j,k,l,uniform,apertures
105 c
106 k=apertures
107 xk=k
108 if(j.eq.0)then
109     hh=xk*d/2d0
110 else
111     pi=4d0*atan(1d0)
112     d1=1d0-d
113     xj=j
114     j1=xj*d1*pi
115     j2=xj*d*pi/2d0
116     x1=sin(j2)/j2
117     x1=x1*x1*d/2d0
118     x2=0d0
119     if(uniform.eq.1) then
120         do 10 i=1,k-1
121             xi=i
122             x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
123     10  continue
124     hh=x1*(1d0+2d0*x2)
125     elseif(uniform.eq.0.and.apertures.eq.3) then
126         do 20 i=1,3
127             xi=i
128             x2=x2+cos(xi*j1/3d0)
129     20  continue
130     hh=x1*(1d0+2d0*x2/3d0)
131     elseif(uniform.eq.0.and.apertures.eq.4) then
132         do 30 i=1,6
133             xi=i
134             x2=x2+cos(xi*j1/6d0)
135     30  continue
136     hh=x1*(1d0+x2/2d0)

```

```

137         elseif(uniform.eq.0.and.apertures.eq.5) then
138             do 40 i=1,11
139                 if(i.eq.10) goto 40
140                 xi=i
141                 x2=x2+cos(xi*j1/11d0)
142 40         continue
143             hh=x1*(1d0+2d0*x2/5d0)
144         elseif(uniform.eq.0.and.apertures.eq.6) then
145             do 50 i=1,17
146                 if(i.eq.14.or.i.eq.15) goto 50
147                 xi=i
148                 x2=x2+cos(xi*j1/17d0)
149 50         continue
150             hh=x1*(1d0+x2/3d0)
151         endif
152     endif
153     return
154 end
155 c
156 c
157 c
158 c     This routine generates the SVD decomposition of an
159 c     mxn matrix A = UVVtranspose, where U is mxn, and V
160 c     and W are nxn. U is column orthogonal, V is row and
161 c     column orthogonal, and W is diagonal. U is returned
162 c     in the array a.
163 c
164 c
165 c
166     subroutine svdcmp(a,m,n,v,v)
167     implicit real*8 (a-h,o-z)
168     parameter (nmax=128)
169     dimension a(m,n),w(n),v(n,n),rv1(nmax)
170 c
171     g=0d0
172     scale=0d0
173     anorm=0d0
174     do 25 i=1,n
175         l=i+1
176         rv1(i)=scale*g
177         g=0d0
178         s=0d0
179         scale=0d0
180         if (i.le.m) then
181             do 11 k=i,m
182                 scale=scale+abs(a(k,i))
183 11         continue
184             if (scale.ne.0d0) then
185                 do 12 k=i,m
186                     a(k,i)=a(k,i)/scale
187                     s=s+a(k,i)*a(k,i)
188 12         continue
189             f=a(i,i)
190             g=-sign(sqrt(s),f)
191             h=f*g-s
192             a(i,i)=f-g
193             if (i.ne.n) then
194                 do 15 j=l,n
195                     s=0d0
196                     do 13 k=i,m
197                         s=s+a(k,i)*a(k,j)
198 13         continue
199             f=s/h
200             do 14 k=i,m
201                 a(k,j)=a(k,j)+f*a(k,i)
202 14         continue
203 15         continue
204     endif

```

```

205         do 16 k= i,m
206             a(k,i)=scale*a(k,i)
207 16         continue
208         endif
209     endif
210     w(i)=scale *g
211     g=0d0
212     s=0d0
213     scale=0d0
214     if ((i.le.m).and.(i.ne.n)) then
215         do 17 k=1,n
216             scale=scale+abs(a(i,k))
217 17         continue
218         if (scale.ne.0d0) then
219             do 18 k=1,n
220                 a(i,k)=a(i,k)/scale
221                 s=s+a(i,k)*a(i,k)
222 18             continue
223             f=a(i,1)
224             g=-sign(sqrt(s),f)
225             h=f*g-s
226             a(i,1)=f-g
227             do 19 k=1,n
228                 rv1(k)=a(i,k)/h
229 19             continue
230             if (i.ne.m) then
231                 do 23 j=1,m
232                     s=0d0
233                     do 21 k=1,n
234                         s=s+a(j,k)*a(i,k)
235 21                     continue
236                     do 22 k=1,n
237                         a(j,k)=a(j,k)+s*rv1(k)
238 22                     continue
239 23                 continue
240             endif
241             do 24 k=1,n
242                 a(i,k)=scale*a(i,k)
243 24             continue
244         endif
245     endif
246     anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
247 25 continue
248     do 32 i=n,1,-1
249         if (i.lt.n) then
250             if (g.ne.0d0) then
251                 do 26 j=1,n
252                     v(j,i)=(a(i,j)/a(i,1))/g
253 26                 continue
254                 do 29 j=1,n
255                     s=0d0
256                     do 27 k=1,n
257                         s=s+a(i,k)*v(k,j)
258 27                 continue
259                 do 28 k=1,n
260                     v(k,j)=v(k,j)+s*v(k,i)
261 28                 continue
262 29                 continue
263             endif
264             do 31 j=1,n
265                 v(i,j)=0d0
266                 v(j,i)=0d0
267 31             continue
268         endif
269         v(i,i)=1d0
270         g=rv1(i)
271         l=i
272 32     continue

```

```

273      do 39 i=n,1,-1
274          l=i+1
275          g=w(i)
276          if (i.lt.n) then
277              do 33 j=1,n
278                  a(i,j)=0d0
279 33          continue
280      endif
281      if (g.ne.0d0) then
282          g=1d0/g
283          if (i.ne.n) then
284              do 36 j=1,n
285                  s=0d0
286                  do 34 k=1,m
287                      s=s+a(k,i)*a(k,j)
288 34          continue
289                      f=(s/a(i,i))*g
290                      do 35 k=i,m
291                          a(k,j)=a(k,j)+f*a(k,i)
292 35          continue
293 36          continue
294      endif
295      do 37 j=i,m
296          a(j,i)=a(j,i)*g
297 37          continue
298      else
299          do 38 j= i,m
300              a(j,i)=0d0
301 38          continue
302      endif
303      a(i,i)=a(i,i)+1d0
304 39      continue
305      do 49 k=n,1,-1
306          do 48 its=1,30
307              do 41 l=k,1,-1
308                  nm=l-1
309                  if ((abs(rv1(l))+anorm).eq.anorm) go to 2
310                  if ((abs(w(nm))+anorm).eq.anorm) go to 1
311 41          continue
312 1          c=0d0
313          s=1d0
314          do 43 i=1,k
315              f=s*rv1(i)
316              if ((abs(f)+anorm).ne.anorm) then
317                  g=w(i)
318                  h=sqrt(f*f+g*g)
319                  w(i)=h
320                  h=1d0/h
321                  c= (g*h)
322                  s=-(f*h)
323                  do 42 j=1,m
324                      y=a(j,nm)
325                      z=a(j,i)
326                      a(j,nm)=(y*c)+(z*s)
327                      a(j,i)=-(y*s)+(z*c)
328 42          continue
329      endif
330 43          continue
331 2          z=w(k)
332          if (l.eq.k) then
333              if (z.lt.0d0) then
334                  w(k)=-z
335                  do 44 j=1,n
336                      v(j,k)=-v(j,k)
337 44          continue
338      endif
339          go to 3
340      endif

```

```

341      if (its.eq.30) pause 'no convergence in 30 iterations'
342      x=w(1)
343      nm=k-1
344      y=v(nm)
345      g=rv1(nm)
346      h=rv1(k)
347      f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
348      g=sqrt(f*f+1d0)
349      f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x
350      c=1d0
351      s=1d0
352      do 47 j=1,nm
353          i=j+1
354          g=rv1(i)
355          y=v(i)
356          h=s*g
357          g=c*g
358          z=sqrt(f*f+h*h)
359          rv1(j)=z
360          c=f/z
361          s=h/z
362          f= (x*c)+(g*s)
363          g=- (x*s)+(g*c)
364          h=y*s
365          y=y*c
366          do 45 nm=1,n
367              x=v(nm,j)
368              z=v(nm,i)
369              v(nm,j)= (x*c)+(z*s)
370              v(nm,i)=- (x*s)+(z*c)
371 45      continue
372          z=sqrt(f*f+h*h)
373          w(j)=z
374          if (z.ne.0d0) then
375              z=1d0/z
376              c=f*z
377              s=h*z
378          endif
379          f= (c*g)+(s*y)
380          x=- (s*g)+(c*y)
381          do 46 nm=1,n
382              y=a(nm,j)
383              z=a(nm,i)
384              a(nm,j)= (y*c)+(z*s)
385              a(nm,i)=- (y*s)+(z*c)
386 46      continue
387 47      continue
388      rv1(1)=0d0
389      rv1(k)=f
390      w(k)=x
391 48      continue
392 3      continue
393 49      continue
394      return
395      end

```

```

1 c
2 c
3 c      program snrmkr3.f
4 c
5 c
6      real*4 sum,snr(128),f1(128),lg10,slice(50),count(50)
7      real*8 f(128),e(128)
8 c
9      lg10=log(1e1)
10 c
11      do 5 j=1,16
12      do 5 j=1,32
13      do 5 j=1,40
14          call readfl('s7d005u1a12s32',n,'d',f,'d',e,j,1,128)
15          call readfl('s7d005u1a12n=m',n,'d',f,'d',e,j,1,128)
16          call readfl('s7d005u1a18s40',n,'d',f,'d',e,j,1,128)
17          call readfl('test.ls',n,'d',f,'d',e,j,1,128)
18          call readfl('grdtstae',n,'d',f,'d',e,j,1,128)
19          sum=0
20          do 10 i=1,128
21              sum=sum+e(i)
22              snr(i)=10*log(real(i)/sum)/lg10
23              f1(i)=real(i)/128
24      10      continue
25      call plotfl('snrs7d005u1a12',128,'f',f1,'f',snr)
26      call plotfl('snr7005u112n=m',128,'f',f1,'f',snr)
27      call plotfl('snrs7d005u1a18',128,'f',f1,'f',snr)
28      call plotfl('snrtest.ls',128,'f',f1,'f',snr)
29      call plotfl('snrgrdtstae',128,'f',f1,'f',snr)
30      slice(j)=snr(128)
31      count(j)=j
32      5      continue
33      call plotfl('slcs7d005u1a12',32,'f',count,'f',slice)
34      call plotfl('slc7005u112n=m',32,'f',count,'f',slice)
35      call plotfl('slcs7d005u1a18',40,'f',count,'f',slice)
36      call plotfl('slctest.ls',16,'f',count,'f',slice)
37      call plotfl('slcgrdtstae',8,'f',count,'f',slice)
38      stop
39      end

```

APPENDIX D for Chapter 6

In this appendix we describe the gradient projection algorithm²⁹ as it was used by us to implement the positivity constraint on the object estimate obtained by unweighted least-squares. We begin with a statement of the problem. Given the observation

$$\mathbf{y} = G\mathbf{x} + \mathbf{n}, \quad (\text{D1})$$

chose $\hat{\mathbf{x}}$ to minimize

$$\epsilon = \|\mathbf{y} - G\hat{\mathbf{x}}\|^2, \quad (\text{D2})$$

with the constraining

$$\hat{\mathbf{x}} \geq 0. \quad (\text{D3})$$

We begin by computing the gradient of ϵ of Eq. (D1) with respect to the object estimate $\hat{\mathbf{x}}(n)$ at the n^{th} iteration:

$$\nabla \epsilon(n) = G^T G \hat{\mathbf{x}}(n) - G^T \mathbf{y}. \quad (\text{D4})$$

The search for a solution proceeds along the negative gradient direction unless that direction would violate the constraint. Consider the following iteration equation:

$$\hat{\mathbf{x}}(n+1) = \hat{\mathbf{x}}(n) - \alpha \nabla \epsilon(n), \quad (\text{D5})$$

with

$$\alpha > 0. \quad (\text{D6})$$

If α is chosen so that the above difference equation is stable, then it would converge to the unconstrained solution

$$\hat{\mathbf{x}} = (G^T G)^{-1} G^T \mathbf{y}, \quad (\text{D7})$$

which is the well-known solution to the unconstrained least-squares problem. Now consider how we might modify our algorithm in order to satisfy the positivity constraint. Assume for the moment that all components of $\hat{\mathbf{x}}(n)$ are nonnegative. If any component of the gradient is negative, then the corresponding component of $\hat{\mathbf{x}}(n+1)$ will move away from the constraint boundary and there is no problem. If any component of the gradient is strictly positive, and the corresponding component of $\hat{\mathbf{x}}(n)$ is strictly positive, then an appropriate upper bound must be placed on α so that the constraint boundary is not crossed (that component of $\hat{\mathbf{x}}(n+1)$ does not go negative). For the i^{th} component, that bound is obviously $(\hat{\mathbf{x}}(n))_i / (\nabla \epsilon(n))_i$. If any component of the gradient is positive, and the corresponding component of $\hat{\mathbf{x}}(n)$ is zero, then that component of the gradient must be set equal to zero. We are now in a position to define the algorithm. Let $A(n)$ be the set of all indices of $\hat{\mathbf{x}}(n)$ which correspond to zero components. The value of the object estimate at iteration $n+1$ is

$$\hat{\mathbf{x}}(n+1) = \hat{\mathbf{x}}(n) + \alpha(n) \mathbf{s}(n), \quad (\text{D8})$$

where the i^{th} component of $\mathbf{s}(n)$ is given by

$$(s(n))_i = \begin{cases} 0, & i \in A(n) \text{ and } (\nabla \epsilon(n))_i > 0 \\ -(\nabla \epsilon(n))_i, & \text{else.} \end{cases} \quad (\text{D9})$$

We pick the step size $\alpha(n)$ as follows,

$$\alpha(n) = \min [\alpha_1(n), \alpha_2(n)], \quad (\text{D10})$$

where

$$\alpha_1(n) = \frac{\mathbf{s}^T(n) \mathbf{s}(n)}{\mathbf{s}^T(n) G G^T \mathbf{s}(n)}, \quad (\text{D11})$$

and

$$\alpha_2(n) = \min_{\alpha_i > 0} \left[\alpha_i = \frac{(\hat{\mathbf{x}}(n))_i}{(\nabla \epsilon(n))_i}, i \in A(n) \text{ and } (\nabla \epsilon(n))_i > 0 \right]. \quad (\text{D12})$$

As previously discussed, $\alpha_2(n)$ assures that no constraint boundary is crossed. However, the step size may still be too large, or the set of α_i on Eq. (D12) may be empty. We need an additional bound on step size. A reasonable maximum step size (and the one used in the unconstrained steepest descent algorithm) is one in which $\mathbf{s}(n+1)$ is orthogonal to $\mathbf{s}(n)$. It can be shown that $\alpha_1(n)$ is that step size.

It remains only to choose an initial estimate $\hat{\mathbf{x}}(0)$. We chose the unconstrained solution given by Eq. (D7) with all negative components set equal to zero.

APPENDIX E for Chapter 6

In this appendix we describe the CLEAN³ algorithm as it was implemented by us to obtain the results of Section 6.5. Let $\mathbf{y}$ be the observation vector given by

$$\mathbf{y} = G\mathbf{x} + \mathbf{n}. \quad (\text{E1})$$

The columns of G are the system point-spread-function shifted and truncated, $\mathbf{x}$ is the object vector, and $\mathbf{n}$ is observation noise. Let $\mathbf{g}_i$ be the i^{th} column of G . In the noiseless case, we can write $\mathbf{y}$ as a linear combination of the $\mathbf{g}_i$ as follows:

$$\mathbf{y} = \sum_i x_i \mathbf{g}_i, \quad (\text{E2})$$

where x_i is the i^{th} component of the object vector $\mathbf{x}$. Given $\mathbf{y}$ and the $\mathbf{g}_i$, CLEAN attempts to iteratively determine the x_i .

Let $\mathbf{y}(n)$ be the n^{th} vector of a sequence of vectors where

$$\mathbf{y}(0) = \mathbf{y}, \quad (\text{E3})$$

and let $y_i(n)$ be the i^{th} component of $\mathbf{y}(n)$. Let $\hat{\mathbf{x}}(n)$ be our estimate of $\mathbf{x}$ after n iterations, with components $\hat{x}_i(n)$, with

$$\hat{x}_i(0) = 0. \quad (\text{E4})$$

Finally, let the integer k_n be the index of the maximum component of $\mathbf{y}(n)$ over the region of support of $\mathbf{x}$, so that $y_{k_n}(n)$ is the maximum value. Then the following set of iteration equations define CLEAN:

$$\mathbf{y}(n+1) = \mathbf{y}(n) - \alpha(n)\mathbf{g}_{k_n}, \quad (\text{E5})$$

$$\hat{x}_i(n+1) = \begin{cases} \hat{x}_i(n) + \alpha(n), & i = k_n \\ \hat{x}_i(n), & i \neq k_n, \end{cases} \quad (\text{E6})$$

where $\alpha(n)$ is given by

$$\alpha(n) = \epsilon \frac{y_{k_n}(n)}{g_{\max}}. \quad (\text{E7})$$

The notation $g_{\max}$ denotes the peak value of $\mathbf{g}_i$, which (presumably) is independent of i . ϵ is a gain parameter—necessarily less than unit, and with a value of $0.4 \rightarrow 0.5$ in our work. At each iteration, we also compute

$$\xi(n) = \|\mathbf{y}(n)\|^2. \quad (\text{E8})$$

If $\xi(n+1) > \xi(n)$ than the iteration is terminated and $\hat{\mathbf{x}}(n)$ is used as the object vector estimate.

APPENDIX F for Chapter 6

Computer Program Listings

This appendix contains the Fortran source code of the computer programs used to generate the data for this chapter.

```

1  c
2  c
3  c      program grdsrch6.f
4  c
5  c
6      parameter(nn=638,mm=128)
7      real*8 bw(nn,mm),h(mm,nn),retilu(128),f(128),x(mm),no(nn)
8      real*8 v(mm,mm),w(mm),erru(mm,mm),retilc(128),errc(mm,mm)
9      real*8 u(nn,mm)
10     real*8 y(nn),gg(mm,mm),d
11     integer*4 random,cntr,ni,nj,nt,m,mb,n,nneg,cntr2
12     integer*4 uniform,apertures
13     character dfile*16
14  c
15     n=512
16     print *, 'nt?'
17     read *, nt
18     print *, 'ni?'
19     read *, ni
20     print *, 'nj?'
21     read *, nj
22     print *, 'mb?'
23     read *, mb
24  c     print *, 'm?'
25  c     read *, m
26     print *, 'snr?'
27     read *, snr
28     print *, 'd/D?'
29     read *, d
30     print *, 'Uniformity? (yes=1 and no=0)'
31     read *, uniform
32     print *, 'Number of apertures?'
33     read *, apertures
34     print *, 'Destination File?'
35     read *, dfile
36     do 5 m=mb,76,6
37         call srandom(29)
38         call bwskr(bw,u,n,m,d,uniform,apertures)
39         call ggskr(bw,n,m,gg)
40         call svdskr(u,n,m,w,v)
41         call hmskr(h,u,w,v,m,n)
42         do 103 j=1,mm
43             do 101 jj=1,mm
44                 erru(j,jj)=0
45                 errc(j,jj)=0
46 101         continue
47 103     continue
48         cntr=0
49         cntr2=0
50         print *, 'grdsrch trials'
51     do 105 iv=1,nt
52  c     print *,iv
53         if (float(iv)/10.0-int(float(iv)/10.0) .lt. 1e-10) then
54             print *, iv
55         endif
56         call randm(x,no,m,n,snr,bw,y,mb)
57         call grdsrch(x,h,no,erru,n,m,nt,errc,y,bw,gg,ni,nj,cntr,
58 *             nneg)
59         if(nneg.eq.0) then
60             cntr2=cntr2+1

```

```

61         endif
62 105      continue
63         print *, real(cntr)*real(nj)/real(nt), cntr2
64 c       call spectramkr(erru,m,retilu,f,snr,mb)
65         call spectramkr(errc,m,retilc,f,snr,mb)
66 c       call plotfl(dfile,128,'d',f,'d',retilu)
67         call plotfl(dfile,128,'d',f,'d',retilc)
68 5       continue
69         stop
70         end
71 c
72 c
73 c
74         subroutine bwkr(bw,u,n,m,d,uniform,apertures)
75         real*8 bw(n,m),u(n,m),hh,d
76         integer n,m,uniform,apertures
77         do 520 i=1,n
78             do 530 ii=1,m
79                 bw(i,ii)=hh((m-n)/2-ii+i,d,uniform,apertures)
80                 u(i,ii)=bw(i,ii)
81 530         continue
82 520         continue
83         return
84         end
85 c
86 c
87 c
88         function hh(j,d,uniform,apertures)
89         real*8 hh,d,pi,d1,j1,j2,x1,x2,xj,xk,xi
90         integer j,k,l,uniform,apertures
91 c
92         k=apertures
93         xk=k
94         if(j.eq.0)then
95             hh=xk*d/2d0
96         else
97             pi=4d0*atan(1d0)
98             d1=1d0-d
99             xj=j
100            j1=xj*d1*pi
101            j2=xj*d*pi/2d0
102            x1=sin(j2)/j2
103            x1=x1*x1*d/2d0
104            x2=0d0
105            if(uniform.eq.1) then
106                do 10 i=1,k-1
107                    xi=i
108                    x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
109 10            continue
110                hh=x1*(1d0+2d0*x2)
111            elseif(uniform.eq.0.and.apertures.eq.3) then
112                do 20 i=1,3
113                    xi=i
114                    x2=x2+cos(xi*j1/3d0)
115 20            continue
116                hh=x1*(1d0+2d0*x2/3d0)
117            elseif(uniform.eq.0.and.apertures.eq.4) then
118                do 30 i=1,6
119                    xi=i
120                    x2=x2+cos(xi*j1/6d0)
121 30            continue
122                hh=x1*(1d0+x2/2d0)
123            elseif(uniform.eq.0.and.apertures.eq.5) then
124                do 40 i=1,11
125                    if(i.eq.10) goto 40
126                    xi=i
127                    x2=x2+cos(xi*j1/11d0)
128 40            continue

```

```

129      hh=x1*(1d0+2d0*x2/5d0)
130      elseif(uniform.eq.0.and.apertures.eq.6) then
131          do 50 i=1,17
132              if(i.eq.14.or.i.eq.15) goto 50
133              xi=i
134              x2=x2+cos(xi*j1/17d0)
135 50      continue
136          hh=x1*(1d0+x2/3d0)
137      endif
138  endif
139  return
140  end

141  c
142  c
143  c
144      subroutine hmkcr(h,u,v,m,n)
145  c
146      real*8 u(n,m),v(m),v(m,m),h(m,n)
147      integer m,n
148  c
149      do 10 i=1,m
150          do 20 j=1,n
151              h(i,j)=0d0
152              do 30 k=1,m
153                  h(i,j)=h(i,j)+(1/w(k))*v(i,k)*u(j,k)
154 30      continue
155 20      continue
156 10      continue
157      return
158      end

159  c
160  c
161  c
162      subroutine ggmkcr(bw,n,m,gg)
163  c
164      real*8 bw(n,m),gg(m,m)
165      do 600 i=1,m
166          do 610 ii=1,m
167              gg(i,ii)=0
168              do 620 iii=1,n
169                  gg(i,ii)=gg(i,ii)+bw(iii,i)*bw(iii,ii)
170 620      continue
171 610      continue
172 600      continue
173      return
174      end

175  c
176  c
177  c
178      subroutine randm(x,no,m,n,snr,bw,y,mb)
179  c
180  c      Generates random vectors x, no, and y=BWx+no. x has mb
181  c      independent Rayleigh components with variance snr and m-mb
182  c      zero components. no has n independent, zero mean, Gaussian
183  c      components with unit variance.
184  c
185      real*8 u1,u2,u3,s
186      real*8 x(m),no(n),bw(n,m),y(n)
187      integer*4 MAXINTV,random
188      parameter(MAXINTV=2147483647)
189      n2=n/2
190      do 10 i=1,m
191          x(i)=0d0
192 10      continue
193      do 500 i=1,n2
194          u1=real(random())/MAXINTV
195          if(u1.gt.1.or.u1.eq.0)goto 3
196          u2=real(random())/MAXINTV

```

```

197         if(u2.gt.1.or.u2.eq.0)goto 2
198         no(2*i-1)=sqrt(-2*log(u1))*cos(6.2831853*u2)
199         no(2*i)=sqrt(-2*log(u1))*sin(6.2831853*u2)
200 500      continue
201         s=snr/2
202         do 505 i=1,mb
203 4         u3=real(random())/MAXINTV
204         if(u3.gt.1.or.u3.eq.0)goto 4
205         x(i+(m-mb)/2)=sqrt(-2*log(u3))*sqrt(s)
206 505      continue
207         do 510 i=1,n
208         y(i)=0
209         do 520 ii=1,m
210         y(i)=y(i)+bw(i,ii)*x(ii)
211 520      continue
212         y(i)=y(i)+no(i)
213 510      continue
214         return
215         end
216 c
217 c
218 c
219         subroutine grdsrch(x,h,no,erru,n,m,nt,errc,y,bw,gg,
220 *         ni,nj,cntr,nneg)
221         include 'fpedefs.f'
222         parameter(mm=128)
223         real*8 x(m),no(n),erru(m,m),h(m,n),eu(mm),zz,msq,msqold
224         real*8 errc(m,m),gg(m,m),grad(mm),dmsq,s(mm)
225         real*8 y(n),bw(n,m),z(mm),xg(mm)
226         real*8 alpha,beta,ss,sggs,tem1,tem2
227         integer cntr,nneg,list(mm),ni,nj,m,n,nt
228         nneg=0
229         msq=0
230 c ----- Compute unconstrained solution and BWy -----
231         do 700 i=1,m
232         xg(i)=0
233         z(i)=0
234         do 710 ii=1,n
235         xg(i)=xg(i)+h(i,ii)*y(ii)
236         z(i)=z(i)+bw(ii,i)*y(ii)
237 710      continue
238         eu(i)=xg(i)-x(i)
239         if(xg(i).lt.0) then
240         nneg=nneg+1
241         endif
242 700      continue
243         if(nneg.eq.0) goto 200
244 c ----- Gradient Projection Algorithm -----
245 c ----- Disable overflow and divide-by-zero ---
246 c ----- floating point exceptions.
247         nabl=fpgetxnabl()
248         newabl=and(nabl,not(DIV0))
249         newabl=and(newabl,not(OFLOW))
250         call fpsetxnabl(newabl)
251 100      msqold=msq
252         cntr=cntr+1
253         do 770 iv=1,nj
254         alpha=1.797d+308
255         ss=0d0
256         sggs=0d0
257 c ----- Find active constraints and generate list -----
258         do 705 i=1,m
259         list(i)=0
260         if(xg(i).le.0d0) then
261         xg(i)=0d0
262         list(i)=1
263         endif
264 705      continue

```

```

265 c ----- Compute gradient and correction vector S -----
266 c         and pick step size alpha
267         do 712 i=1,m
268             grad(i)=0d0
269             do 713 ii=1,m
270                 grad(i)=grad(i)+gg(i,ii)*xg(ii)
271 713         continue
272             grad(i)=grad(i)-z(i)
273 c             ----- Project gradient onto active, -----
274 c                 nonobstructing constraints.
275             if(list(i).eq.1.and.grad(i).gt.0d0) then
276                 s(i)=0d0
277             else
278                 s(i)=-grad(i)
279             endif
280 c ----- Determine step size alpha: -----
281 c         Determine minimum step size for next constraint,
282 c         Determine step size for minimum along direction S,
283 c ----- Pick smaller of the two -----
284             if(list(i).ne.1) then
285                 beta=xg(i)/grad(i)
286                 if(beta.gt.0d0.and.beta.lt.alpha) then
287                     alpha=beta
288                 endif
289             endif
290 712         continue
291             do 714 i=1,m
292                 ss=ss+s(i)*s(i)
293                 do 716 j=1,m
294                     sggs=sggs+s(i)*gg(i,j)*s(j)
295 716                 continue
296 714             continue
297             if(ss.eq.0d0) goto 300
298             beta=ss/sggs
299             if(beta.gt.0d0.and.beta.lt.alpha) then
300                 alpha=beta
301 c ----- compute new gradient and test for orthogonality -----
302 c         with the S direction.
303 c             zz=0d0
304 c             do 900 i=1,m
305 c                 grad(i)=0d0
306 c                 do 901 ii=1,m
307 c                     grad(i)=grad(i)+gg(i,ii)*(xg(ii)+alpha*s(ii))
308 c901                 continue
309 c                 grad(i)=grad(i)-z(i)
310 c                 zz=zz+grad(i)*s(i)
311 c900                 continue
312 c                 print *, 'gTs =', zz
313 c ----- End of Test -----
314             endif
315 c ----- Compute new Solution -----
316             do 740 i=1,m
317                 xg(i)=xg(i)+alpha*s(i)
318 740             continue
319 770         continue
320 c ----- Check ||y-Gx|| squared -----
321             msq=0
322             do 780 i=1,n
323                 zz=0
324                 do 790 ii=1,m
325                     zz=zz+bu(i,ii)*xg(ii)
326 790                 continue
327                 zz=y(i)-zz
328                 msq=msq+zz*zz
329 780             continue
330 c                 print *, msq
331             dmsq=abs((msqold-msq)*(10**ni))
332             if (dmsq.gt.1e+9) goto 100

```

```

333         if (int(dmsq).ne.0) goto 100
334 c         ----- Enable Floating Point Exceptions ---
335 300         call fpsetxflags(0)
336         call fpsetxnabls(nabls)
337 c ----- Accumulate Error Vector Outer Product and Return -----
338 200         do 720 i=1,m
339 c             print *, xg(i), grad(i), s(i)
340             tem1=xg(i)-x(i)
341             do 730 ii=1,m
342                 tem2=xg(ii)-x(ii)
343                 erru(i,ii)=erru(i,ii)+eu(i)*eu(ii)/nt
344                 errc(i,ii)=errc(i,ii)+tem1*tem2/nt
345 730             continue
346 720         continue
347 c         print *, msq, nneg
348         return
349     end
350 c
351 c
352 c
353     subroutine spectramkr(err,m,retil,f,snr,mb)
354     real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
355     integer p
356     pi2=8d0*atan(1d0)
357     do 10 i=1,m
358         sum(i)=0d0
359 10     continue
360         do 20 i=1,m
361             do 30 ii=1,m
362                 p=abs(i-ii)+1
363                 sum(p)=sum(p)+err(i,ii)
364 30         continue
365 20     continue
366         print *, sum(1)
367         do 50 i=1,128
368             xi=real(i)/256
369             retil(i)=0
370             do 60 ii=1,m
371                 xii=ii-1
372                 retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
373 60         continue
374 50     continue
375         do 80 i=1,128
376             f(i)=real(i)/128
377             retil(i)=retil(i)/(snr*mb)
378 80     continue
379         return
380     end
381 c
382 c
383 c
384     subroutine svdmkr(a,m,n,w,v)
385 c
386 c         This routine generates the SVD of an mxn matrix A
387 c         where A = UWVtranspose, with U mxn, and V and W are
388 c         nxn. U is column orthogonal, V is row and column
389 c         orthogonal, and W is diagonal. U is returned in the
390 c         array a. The diagonal of W is returned as the vector w.
391 c
392     implicit real*8 (a-h,o-z)
393     parameter (nmax=128)
394     dimension a(m,n),w(n),v(n,n),rv1(nmax)
395 c
396     g=0d0
397     scale=0d0
398     anorm=0d0
399     do 25 i=1,n
400         l=i+1

```

```

401      rv1(i)=scale*g
402      g=0d0
403      s=0d0
404      scale=0d0
405      if (i.le.m) then
406          do 11 k=i,m
407              scale=scale+abs(a(k,i))
408 11      continue
409          if (scale.ne.0d0) then
410              do 12 k=i,m
411                  a(k,i)=a(k,i)/scale
412                  s=s+a(k,i)*a(k,i)
413 12      continue
414          f=a(i,i)
415          g=-sign(sqrt(s),f)
416          h=f*g-s
417          a(i,i)=f-g
418          if (i.ne.n) then
419              do 15 j=1,n
420                  s=0d0
421                  do 13 k=i,m
422                      s=s+a(k,i)*a(k,j)
423 13      continue
424                  f=s/h
425                  do 14 k=i,m
426                      a(k,j)=a(k,j)+f*a(k,i)
427 14      continue
428 15      continue
429          endif
430          do 16 k= i,m
431              a(k,i)=scale*a(k,i)
432 16      continue
433          endif
434      endif
435      v(i)=scale *g
436      g=0d0
437      s=0d0
438      scale=0d0
439      if ((i.le.m).and.(i.ne.n)) then
440          do 17 k=1,n
441              scale=scale+abs(a(i,k))
442 17      continue
443          if (scale.ne.0d0) then
444              do 18 k=1,n
445                  a(i,k)=a(i,k)/scale
446                  s=s+a(i,k)*a(i,k)
447 18      continue
448          f=a(i,1)
449          g=-sign(sqrt(s),f)
450          h=f*g-s
451          a(i,1)=f-g
452          do 19 k=1,n
453              rv1(k)=a(i,k)/h
454 19      continue
455          if (i.ne.m) then
456              do 23 j=1,m
457                  s=0d0
458                  do 21 k=1,n
459                      s=s+a(j,k)*a(i,k)
460 21      continue
461                  do 22 k=1,n
462                      a(j,k)=a(j,k)+s*rv1(k)
463 22      continue
464 23      continue
465          endif
466          do 24 k=1,n
467              a(i,k)=scale*a(i,k)
468 24      continue

```

```

469         endif
470     endif
471     anorm=max(anorm,(abs(v(i))+abs(rv1(i))))
472 25  continue
473     do 32 i=n,1,-1
474         if (i.lt.n) then
475             if (g.ne.0d0) then
476                 do 26 j=1,n
477                     v(j,i)=(a(i,j)/a(i,1))/g
478 26  continue
479                     do 29 j=1,n
480                         s=0d0
481                         do 27 k=1,n
482                             s=s+a(i,k)*v(k,j)
483 27  continue
484                             do 28 k=1,n
485                                 v(k,j)=v(k,j)+s*v(k,i)
486 28  continue
487 29  continue
488             endif
489             do 31 j=1,n
490                 v(i,j)=0d0
491                 v(j,i)=0d0
492 31  continue
493             endif
494             v(i,i)=1d0
495             g=rv1(i)
496             l=i
497 32  continue
498     do 39 i=n,1,-1
499         l=i+1
500         g=v(i)
501         if (i.lt.n) then
502             do 33 j=1,n
503                 a(i,j)=0d0
504 33  continue
505             endif
506             if (g.ne.0d0) then
507                 g=1d0/g
508                 if (i.ne.n) then
509                     do 36 j=1,n
510                         s=0d0
511                         do 34 k=1,m
512                             s=s+a(k,i)*a(k,j)
513 34  continue
514                             f=(s/a(i,i))*g
515                             do 35 k=i,m
516                                 a(k,j)=a(k,j)+f*a(k,i)
517 35  continue
518 36  continue
519                 endif
520                 do 37 j=i,m
521                     a(j,i)=a(j,i)*g
522 37  continue
523             else
524                 do 38 j= i,m
525                     a(j,i)=0d0
526 38  continue
527             endif
528             a(i,i)=a(i,i)+1d0
529 39  continue
530     do 49 k=n,1,-1
531         do 48 its=1,30
532             do 41 l=k,1,-1
533                 nm=l-1
534                 if ((abs(rv1(l))+anorm).eq.anorm) go to 2
535                 if ((abs(v(nm))+anorm).eq.anorm) go to 1
536 41  continue

```

```

537 1      c=0d0
538      s=1d0
539      do 43 i=1,k
540          f=s*rv1(i)
541          if ((abs(f)+anorm).ne.anorm) then
542              g=w(i)
543              h=sqrt(f*f+g*g)
544              w(i)=h
545              h=1d0/h
546              c= (g*h)
547              s=-(f*h)
548              do 42 j=1,m
549                  y=a(j,nm)
550                  z=a(j,i)
551                  a(j,nm)=(y*c)+(z*s)
552                  a(j,i)=-(y*s)+(z*c)
553 42      continue
554      endif
555 43      continue
556 2      z=w(k)
557      if (1.eq.k) then
558          if (z.lt.0d0) then
559              w(k)=-z
560              do 44 j=1,n
561                  v(j,k)=-v(j,k)
562 44      continue
563      endif
564      go to 3
565      endif
566      if (its.eq.30) pause 'no convergence in 30 iterations'
567      x=w(1)
568      nm=k-1
569      y=w(nm)
570      g=rv1(nm)
571      h=rv1(k)
572      f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
573      g=sqrt(f*f+1d0)
574      f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x
575      c=1d0
576      s=1d0
577      do 47 j=1,nm
578          i=j+1
579          g=rv1(i)
580          y=w(i)
581          h=s*g
582          g=c*g
583          z=sqrt(f*f+h*h)
584          rv1(j)=z
585          c=f/z
586          s=h/z
587          f= (x*c)+(g*s)
588          g=-(x*s)+(g*c)
589          h=y*s
590          y=y*c
591          do 45 nm=1,n
592              x=v(nm,j)
593              z=v(nm,i)
594              v(nm,j)= (x*c)+(z*s)
595              v(nm,i)=-(x*s)+(z*c)
596 45      continue
597      z=sqrt(f*f+h*h)
598      w(j)=z
599      if (z.ne.0d0) then
600          z=1d0/z
601          c=f*z
602          s=h*z
603      endif
604      f= (c*g)+(s*y)

```

```

605      x=-(s*g)+(c*y)
606      do 46 nm=1,m
607          y=a(nm,j)
608          z=a(nm,i)
609          a(nm,j)= (y*c)+(z*s)
610          a(nm,i)=-(y*s)+(z*c)
611 46      continue
612 47      continue
613          rvl(1)=0d0
614          rvl(k)=f
615          w(k)=x
616 48      continue
617 3      continue
618 49      continue
619      return
620      end

```

```

1 c
2 c
3 c      program clean4.f
4 c
5 c
6      parameter(nn=638,mm=128)
7      real*8 bw(nn,mm),f(128),x(mm),no(nn)
8      real*8 retile(128),errc(mm,mm)
9      real*8 y(nn),d,eps,psf(2*nn)
10     integer*4 random,cntr,nt,m,mb,n
11     integer*4 uniform,apertures,flag
12     character dfile*16
13 c
14 c      n=512
15     print *, 'nt?'
16     read *, nt
17     print *, 'eps?'
18     read *, eps
19     print *, 'mb?'
20 c      read *, mb
21 c      print *, 'm?'
22 c      read *, m
23     print *, 'snr?'
24     read *, snr
25     print *, 'Add noise? (yes=1 and no=0)'
26     read *, flag
27     print *, 'd/D?'
28     read *, d
29     print *, 'Uniformity? (yes=1 and no=0)'
30     read *, uniform
31     print *, 'Number of apertures?'
32     read *, apertures
33     print *, 'Destination File?'
34     read *, dfile
35     do 10 i=1,23
36         if(i.lt.21) then
37             m=2*i
38         else
39             m=(i-16)*10
40         endif
41         mb=m
42         n=m
43         call srandom(29)
44         call bwmk(bw,psf,n,m,d,uniform,apertures)
45         do 103 j=1,mm
46             do 101 jj=1,mm
47                 errc(j,jj)=0
48 101         continue
49 103     continue
50         cntr=0
51         print *, 'clean trials'
52         do 105 iv=1,nt
53 c          print *,iv
54             if (float(iv)/10.0-int(float(iv)/10.0) .lt. 1e-10) then
55                 print *, iv
56             endif
57             call randm(x,no,m,n,snr,bw,y,mb,flag)
58             call clean(x,a,m,nt,eps,errc,y,cntr,d,apertures,psf)
59 105     continue
60         print *, real(cntr)/real(nt)
61         call spectramkr(errc,m,retile,f,snr,mb)
62         call plotfl(dfile,128,'d','f','d',retile)
63 10     continue
64         stop
65     end
66 c
67 c
68 c

```

```

69      subroutine bwmk(bw,psf,n,m,d,uniform,apertures)
70      real*8 bw(n,m),psf(2*n),hh,d
71      integer k,n,m,uniform,apertures
72      do 500 i=1,2*n
73          psf(i)=hh(i-n,d,uniform,apertures)
74 500      continue
75          do 520 i=1,n
76              do 530 ii=1,m
77                  k=(m-n)/2-ii+i+n
78                  bw(i,ii)=psf(k)
79 530      continue
80 520      continue
81      return
82      end
83  c
84  c
85  c
86      function hh(j,d,uniform,apertures)
87      real*8 hh,d,pi,d1,j1,j2,x1,x2,xj,xk,xi
88      integer j,k,l,uniform,apertures
89  c
90      k=apertures
91      xk=k
92      if(j.eq.0)then
93          hh=xk*d/2d0
94      else
95          pi=4d0*atan(1d0)
96          d1=1d0-d
97          xj=j
98          j1=xj*d1*pi
99          j2=xj*d*pi/2d0
100         x1=sin(j2)/j2
101         x1=x1*x1*d/2d0
102         x2=0d0
103         if(uniform.eq.1) then
104             do 10 i=1,k-1
105                 xi=i
106                 x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
107 10         continue
108             hh=x1*(1d0+2d0*x2)
109         elseif(uniform.eq.0.and.apertures.eq.3) then
110             do 20 i=1,3
111                 xi=i
112                 x2=x2+cos(xi*j1/3d0)
113 20         continue
114             hh=x1*(1d0+2d0*x2/3d0)
115         elseif(uniform.eq.0.and.apertures.eq.4) then
116             do 30 i=1,6
117                 xi=i
118                 x2=x2+cos(xi*j1/6d0)
119 30         continue
120             hh=x1*(1d0+x2/2d0)
121         elseif(uniform.eq.0.and.apertures.eq.5) then
122             do 40 i=1,11
123                 if(i.eq.10) goto 40
124                 xi=i
125                 x2=x2+cos(xi*j1/11d0)
126 40         continue
127             hh=x1*(1d0+2d0*x2/5d0)
128         elseif(uniform.eq.0.and.apertures.eq.6) then
129             do 50 i=1,17
130                 if(i.eq.14.or.i.eq.15) goto 50
131                 xi=i
132                 x2=x2+cos(xi*j1/17d0)
133 50         continue
134             hh=x1*(1d0+x2/3d0)
135         endif
136     endif

```

```

137      return
138      end
139 c
140 c
141 c
142      subroutine randm(x,no,m,n,snr,bw,y,mb,flag)
143 c
144 c      Generates random vectors x, no, and y=BWx+no. x has mb
145 c      independent Rayleigh components with variance snr and m-mb
146 c      zero components. no has n independent, zero mean, Gaussian
147 c      components with unit variance.
148 c
149      real*8 u1,u2,u3,s
150      real*8 x(m),no(n),bw(n,m),y(n)
151      integer*4 MAXINTV,random,flag
152      parameter(MAXINTV=2147483647)
153      n2=n/2
154      do 10 i=1,m
155          x(i)=0d0
156 10      continue
157      do 500 i=1,n2
158          u1=real(random())/MAXINTV
159          if(u1.gt.1.or.u1.eq.0)goto 3
160 2          u2=real(random())/MAXINTV
161          if(u2.gt.1.or.u2.eq.0)goto 2
162          no(2*i-1)=sqrt(-2*log(u1))*cos(6.2831853*u2)
163          no(2*i)=sqrt(-2*log(u1))*sin(6.2831853*u2)
164 500      continue
165          s=snr/2
166      do 505 i=1,mb
167 4          u3=real(random())/MAXINTV
168          if(u3.gt.1.or.u3.eq.0)goto 4
169          x((i+m-mb)/2)=sqrt(-2*log(u3))*sqrt(s)
170 505      continue
171      do 510 i=1,n
172          y(i)=0
173          do 520 ii=1,m
174              y(i)=y(i)+bw(i,ii)*x(ii)
175 520      continue
176          if(flag.eq.1) then
177              y(i)=y(i)+no(i)
178          endif
179 510      continue
180      return
181      end
182 c
183 c
184 c
185      subroutine clean(x,n,m,nt,eps,errc,y,cntr,d,apertures,psf)
186 c
187      parameter(mm=128)
188      real*8 x(m),y(n),errc(m,m),xhat(mm),alpha,xa,sumsq,ssold
189      real*8 tmp,h0,eps,psf(2*n),d
190      integer*4 cntr,n,m,k,nt,apertures
191 c
192      sumsq=1d37
193      xa=apertures
194      h0=xa*d/2d0
195      do 10 i=1,mm
196          xhat(i)=0d0
197 10      continue
198 100      ssold=sumsq
199          cntr=cntr+1
200          tmp=y((n-m)/2+1)
201          k=((n-m)/2+1)
202          do 20 i=(n-m)/2+2,(n+m)/2
203              if(y(i).gt.tmp) then
204                  tmp=y(i)

```

```

205         k=i
206     endif
207 20     continue
208         alpha=eps*tmp/h0
209         do 30 i=1,n
210             y(i)=y(i)-alpha*psf(i-k+n)
211 30     continue
212         xhat(k-(n-m)/2)=xhat(k-(n-m)/2)+alpha
213         sumsq=0d0
214         do 40 i=1,n
215             sumsq=sumsq+y(i)*y(i)
216 40     continue
217         if(sumsq.lt.ssold) goto 100
218         print *, ssold, sumsq
219 c ----- Accumulate Error Vector Outer Product and Return -----
220         do 50 i=1,m
221             tem1=x(i)-xhat(i)
222             do 60 ii=1,m
223                 tem2=x(ii)-xhat(ii)
224                 errc(i,ii)=errc(i,ii)+tem1*tem2/nt
225 60     continue
226 50     continue
227         return
228         end
229
230 c
231 c
232 c
233         subroutine spectramkr(err,m,retil,f,snr,mb)
234         real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
235         integer p
236         pi2=8d0*atan(1d0)
237         do 10 i=1,m
238             sum(i)=0d0
239 10     continue
240         do 20 i=1,m
241             do 30 ii=1,m
242                 p=abs(i-ii)+1
243                 sum(p)=sum(p)+err(i,ii)
244 30     continue
245 20     continue
246         print *, sum(1)
247         do 50 i=1,128
248             xi=real(i)/256
249             retil(i)=0
250             do 60 ii=1,m
251                 xii=ii-1
252                 retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
253 60     continue
254 50     continue
255         do 80 i=1,128
256             f(i)=real(i)/128
257             retil(i)=retil(i)/(snr*mb)
258 80     continue
259         return
260         end

```

```

1 c
2 c
3 c      program snrmkr2.f
4 c
5 c
6      real*4 sum,snr(128),f1(128),lg10,slice(50),count(50)
7      real*8 f(128),e(128)
8 c
9      lg10=log(1e1)
10 c      do 5 j=1,16
11      do 5 j=1,32
12 c      do 5 j=1,40
13 c          call readfl('s7d005u1a12s32',n,'d',f,'d',e,j,1,128)
14 c          call readfl('s7d005u1a12n=m',n,'d',f,'d',e,j,1,128)
15 c          call readfl('s7d005u1a18s40',n,'d',f,'d',e,j,1,128)
16 c          call readfl('test.lsqr',n,'d',f,'d',e,j,1,128)
17 c          call readfl('grdtstae',n,'d',f,'d',e,j,1,128)
18      sum=0
19      do 10 i=1,128
20          sum=sum+e(i)
21          snr(i)=10*log(real(i)/sum)/lg10
22          f1(i)=real(i)/128
23 10      continue
24 c      call plotfl('snrs7d005u1a12',128,'f',f1,'f',snr)
25 c      call plotfl('snr7005u112n=m',128,'f',f1,'f',snr)
26 c      call plotfl('snrs7d005u1a18',128,'f',f1,'f',snr)
27 c      call plotfl('snrtest.lsqr',128,'f',f1,'f',snr)
28 c      call plotfl('snrgrdtstae',128,'f',f1,'f',snr)
29      slice(j)=snr(128)
30      count(j)=j
31 5      continue
32 c      call plotfl('slcs7d005u1a12',32,'f',count,'f',slice)
33 c      call plotfl('slc7005u112n=m',32,'f',count,'f',slice)
34 c      call plotfl('slcs7d005u1a18',40,'f',count,'f',slice)
35 c      call plotfl('slcstest.lsqr',16,'f',count,'f',slice)
36 c      call plotfl('slcgrdtstae',8,'f',count,'f',slice)
37      stop
38      end

```