November 1990

J. J. Hosker

Phase Retrieval Using The Error-Reduction Algorithms

ABSTRACT

This Document, provides a detailed analysis of the creation and implementation of a final algorithm for the phase retrieval of a two-dimensional image that is real and nonnegative using a combination of the Gerchberg-Saxton, error-reduction and hybrid input-output algorithms. The Gerchberg-Saxton algorithm is an image enhancement algorithm which is used to retrieve distorted images. In particular, this document investigates the phase retrieval of an image in which only information on the magnitude of the image in the Fourier domain is known. The document is divided into four sections and seven appendices. The first section provides background and historical information on the algorithms. The second section provides an introduction to and explanation of the Gerchberg-Saxton, error-reduction and input-output algorithms. The third section describes the software implementation of the final algorithm and includes the problems discovered during the programming of the final algorithm. The fourth section provides conclusions and recommendations on the use of the algorithms. Finally, appendix A shows images that were retrieved using the final algorithm, appendix B provides a mathematical interpretation of the hybrid input-output algorithm, and appendices C through G provide a listing of the source code.

ACKNOWLEDGEMENTS

This document was initially written to fulfill the requirements for a Master's Degree in Science in Electrical Engineering from Tufts University. This document was made possible with the assistance of two people: first, my advisor and the director of the Tufts University Electro-Optics Technology Center (EOTC), Professor Robert A. Gonsalves;

TABLE OF CONTENTS

LIST OF FIGURES

SECTION 1

BACKGROUND

R. W. Gerchberg and W. O. Saxton first presented their findings in 1972 in the journal *Optik* which was published in West Germany. They developed single and dual intensity algorithms for the rapid solution of the phase of a complete wave function whose intensity in the diffraction and imaging planes of an imaging system were known. They further presented evidence that the error between successive iterations of the algorithm must decrease and that a unique solution can always be found.[1-2] Robert A. Gonsalves separately discovered a similar algorithm in the United States in 1976. He describes the phase retrieval as an extraction of the phase from the modulus of a complex signal. He gives examples where the modulus of the Fourier transform of a signal is used to retrieve a corrupted or noisy version of the signal through a computer simulation.[3] The mathematical and theoretical premise for the work of Gerchberg, Saxton and Gonsalves in using the modulus of the Fourier transform to retrieve the phase has existed since 1963.[4-6]

It was not until James R. Fienup investigated the topic of phase retrieval using modulus information of the signal that research and information on this phase retrieval technique again progressed. In many articles from 1978 to the present, Fienup presented many different existing techniques for phase retrieval including the Gerchberg-Saxton algorithm, the steepest-descent method, and gradient search methods. In his experimental comparison of phase-retrieval algorithms, he found a combination of two algorithms that produced optimal results. In particular, Fienup presented a generalized version of the Gerchberg-Saxton algorithm which he called the error-reduction algorithm and presented a hybrid algorithm of the error-reduction algorithm which he called the input-output algorithm.[6-13] The error-reduction and input-output algorithms are the focus of this document because the combination of these two techniques provides the optimal unique solution in the fewest number of iterations.

There are many application of the error-reduction and input-output algorithms. The number of problems that can be solved by this iterative algorithm (the error-reduction algorithm) is only limited by the ingenuity in defining different combinations of information that might be available in both domains. The best way to classify each of the problems is by the type of information available. The following examples of the application of the algorithms are outlined in detail by Feinup in reference 14.

The first problem this algorithm can solve are those problems in which the modulus or magnitude of a complex function and the modulus of its Fourier transform are known and one wants to know the phase of the Fourier transform pair. This application is useful for phase retrieval in electron microscopy; for phase retrieval in wavefront sensing; for design optimization of radar signals and antenna arrays having desirable properties; and for

phase coding and spectrum shaping problems found in computer generated holograms as well as other applications. Electron microscopy was one of the earliest applications of the error-reduction algorithm and perhaps has been the most investigated application. Wavefront sensing is similar to electron microscopy, but spectrum shaping is a complex synthesis problem.

The second problem is one where the function is known to be real and nonnegative and the modulus of its Fourier transform is known or measured. This application is useful for the phase problems associated with x-ray crystallography, Fourier transform spectroscopy, pupil function retrieval, and imaging through atmospheric turbulence using data from an interferometer. This problem of phase retrieval arises in spectroscopy as a one-dimensional problem, in astronomy as a two-dimensional problem, and in x-ray crystallography as a three-dimensional problem. For the one-dimensional problem, the use of the iterative algorithm is of limited interest since the solution is not generally unique. For the problem of reconstructing a two-dimensional nonnegative function from the modulus of it Fourier transform, the use of the iterative algorithm is of great importance in astronomical reconstruction and pupil synthesis. *This application of the algorithm is the focus of this document.*

The third problem is one where a low-pass filtered function is measured and the function is known to have a finite extent. By saying the low-pass filtered function is measured, it means that the complex Fourier transform of the function is measured only over a certain interval and by saying it has a finite extent, it means that the function is zero outside of some known region. This application is useful for the spectral extrapolation, for the superresolution problem of band-limited time signals, or for the imaging of objects of finite

The fourth problem is one where the function is known to be nonnegative and of finite extent. Its complex Fourier transform is measured only over a partially filled aperture. This application is useful for the interpolation of the complex visibility function over long baseline radio interferometry and for the missing-cone problem in x-ray tomography.

The fifth problem is one where the modulus of a complex function is given and one wants to find an associated phase function that results in a Fourier transform whose complex values are in part of a prescribed set of quantized complex values. This application is useful for the reduction of quantized noise in computer-generated holograms and in coded signal transmission.

The sixth problem is one where the modulus of complex function is reconstructed from the phase of the function given that the Fourier transform of the function has finite support. Fienup uses the relaxation-parameter algorithm to solve this application.[14] Essentially, he adds an extra step to the error-reduction algorithm. This document will not deal with relaxation-parameter algorithm.

The application that has been most intriguing is the application described in the second problem. For this document, the Gerchberg-Saxton algorithm is used to retrieve a distorted image using the modulus of the Fourier transform of the original undistorted image, which is real and nonnegative. The algorithms are able to retrieve the original image from a distorted image using an exact estimate of the Fourier modulus of the original undistorted image. This document will outline an algorithm capable of retrieving this type of image and will provide an in-depth analysis on how the error in the image is reduced after each iteration.

SECTION 2

INTRODUCTION

This section is broken into three subsections. The first subsection describes the Gerchberg-Saxton algorithm. The second subsection describes the error-reduction algorithm. Finally, the third subsection describes the hybrid input-output algorithm.

2.1 GERCHBERG-SAXTON ALGORITHM

In the many applications described in the section 1 of this document, one usually wants to retrieve an object f(x) which is related to its Fourier transform F(u). The functions f(x) and F(u) are a unique Fourier transform pair.

$$\begin{aligned} F(u) &= |F(u)| \exp[i\psi(u)] = \mathcal{FT}[f(x)] \\ &= \int_{-\infty}^{\infty} f(x) \exp(-i2\pi u \cdot x)\, dx \end{aligned} \tag{1}$$

$$\begin{aligned} f(x) &= |f(x)| \exp[i\alpha(x)] = \mathcal{FT}^{-1}[F(u)] \\ &= \int_{-\infty}^{\infty} F(u) \exp(i2\pi u \cdot x)\, du \end{aligned} \tag{2}$$

where $\mathcal{FT}$ is an abbreviation for the Fourier transform and $\mathcal{FT}^{-1}$, for the inverse Fourier transform. Moreover, for the purpose of this document, an expression surrounded by two bars such as $|f(x)|$ represents the magnitude or modulus of the expression. In this case, x is a multidimensional spatial coordinate and u is a multidimensional spatial frequency coordinate. For the purpose of this document, the analysis is limited to two-dimensions. If the data is stored in a computer, then the algorithm will uses the discrete Fourier transform, where for the case of two-dimensions each u has coordinates b and c and each x has coordinates m and n. The values of b, c, m, and n = 0,1,2,3, ... (N-1); therefore, the image is square in two-dimensions.

$$F(b,c) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} f(m,n) \exp(-i2\pi b \cdot m/N) \exp(-i2\pi c \cdot n/N), \quad \text{for} \begin{cases} b = 0 \text{ to } N-1 \\ c = 0 \text{ to } N-1 \end{cases}$$

$$f(m,n) = (N)^{-2} \sum_{b=0}^{N-1} \sum_{c=0}^{N-1} F(b,c) \exp(i2\pi b \cdot m/N) \exp(i2\pi c \cdot n/N), \text{ for} \begin{cases} m = 0 \text{ to } N-1 \\ n = 0 \text{ to } N-1 \end{cases}$$

This document will represent x as a two-dimensional discrete variable with coordinates m and n and will represent u as a two-dimensional discrete variable with coordinates b and c. Therefore, the two-dimensional representation of the discrete Fourier transform is reduced to the following notation:

$$F(u) = \sum_{x=0}^{N-1} f(x) \exp(-i2\pi u \cdot x/N), \qquad \text{for } u = 0 \text{ to } N-1 \tag{3}$$

$$f(x) = (N)^{-2} \sum_{u=0}^{N-1} F(u) \exp(i2\pi u \cdot x/N), \qquad \text{for } x = 0 \text{ to } N-1 \tag{4}$$

For the analysis in this document, the data is received in a discrete sampled form with some period T where N is the number of sampled periods. The computer simulation will use the Fast Fourier Transform (FFT) to perform the analysis.[19]

For the original analysis, Gerchberg and Saxton were interested in retrieving the phase from a dual intensity measurement as in electron microscopy.

$$f(x) = |f(x)| \exp[i\alpha(x)] \tag{5}$$

$$F(u) = |F(u)| \exp[i\psi(u)] \tag{6}$$

They wanted to recover $\alpha(x)$ from $|f(x)|$ and $\psi(u)$ from $|F(u)|$. For a single intensity measurement, they were interested in recovering $\psi(u)$ or $f(x)$ given $|F(u)|$ and constraints that $f(x)$ is real and nonnegative. Therefore, each component of $f(x)$ must be greater than or equal to zero.

$$f(x) \geq 0 \tag{7}$$

The Gerchberg-Saxton algorithm is successful in solving these two problems. The error-reduction algorithm and input-output algorithms expanded on the types of problems that can be solved as described in section 1 of this document. These algorithms involve an iterative process of transforming back and forth between the object-domain and the Fourier-domain after the application of measured or known constaints in each of the two domains. The term object-domain can also be referred to as the function-domain, the spatial-domain, or the X-domain. The term Fourier-domain can also be referred to as the spatial frequency-domain or the U-domain. *In this document, the author will try to remain consistent and refer to the object-domain and Fourier-domain as the X-domain and the U-domain respectively.*

The Gerchberg-Saxton algorithm was initially created to reconstruct phase from dual intensity measurements. The algorithm has four basic steps:

1. Fourier transform an estimate of the original object.
2. Replace the modulus of the computed Fourier transform with the measured Fourier modulus to form a estimate of the Fourier transform.
3. Inverse Fourier transform the estimate of the Fourier transform.
4. Replace the modulus of the resulting computed image with the measured object modulus to form a new estimate of the image.

In the form of equations, the Gerchberg-Saxton algorithm can be reduced to the following:

$$G_k(u) = \mathcal{FT}[g_k(u)] = |G_k(u)| \exp[i\varphi_k(u)], \quad (8)$$

$$G'_k(u) = |F(u)| \exp[i\varphi_k(u)], \quad (9)$$

$$g'_k(x) = \mathcal{FT}^{-1}[G'_k(u)] = |g'_k(x)| \exp[i\theta'_k(x)], \quad (10)$$

$$g_{k+1}(x) = |f(x)| \exp[i\theta'_k(x)] = |f(x)| \exp[i\theta_{k+1}(x)], \quad (11)$$

where g_k, θ_k, G'_k, and φ_k are all estimates of f, α, F, and ψ respectively and where k represents the k^{th} iteration.

2.2 ERROR-REDUCTION ALGORITHM

As already stated in the section 1 of this document, the Gerchberg-Saxton algorithm can be used for any problem where partial constraints are known in each of the two domains, usually the X-domain and the U-domain. Transforming back and forth between the two domains while satisfying the contraints in each domain before returning to the other domain is a generalized version of the Gerchberg-Saxton algorithm. The generalized Gerchberg-Saxton algorithm can be used for any problem in which contraints are known in each of the two domains - the X-domain and U-domain. The contraints are usually from measured data or information which is obtained *a priori*.[7] Fienup refers to this generalized algorithm as the error-reduction algorithm.

The error-reduction algorithm is reduced to four basic steps:

1. Fourier transform $g_k(x)$, which is an estimate of f(x).

2. Make changes in $G_k(u)$, the computed Fourier transform; then, satisfy the Fourier-domain or U-domain contraints to form $G'_k(u)$, which is an estimate of $F(u)$.

3. Inverse Fourier transform $G'_k(u)$.

4. Make changes in $g'_k(x)$, the computed image; then satisfy the object-domain or X-domain constraints to form $g_{k+1}(x)$, which is a new estimate of the image.

The first three steps for a single intensity measurement are the same as for the Gerchberg-Saxton algorithm, but the fourth step is given by the following:

$$g_{k+1}(x) = \begin{cases} g'_k(x) & \text{for } x \notin \gamma \\ 0 & \text{for } x \in \gamma \end{cases} \qquad (12)$$

In this case, γ represents the set of points at which $g'_k(x)$ violates the X-domain constraints. For the purpose of this document, the violation occurs when $g'_k(x)$ is negative. However, other violations can occur when $g'_k(x)$ exceeds the known diameter of the object. Figure 2-1 depicts the error-reduction or generalized Gerchberg-Saxton algorithm. In figure 1, $|F|$ is the saved Fourier modulus or magnitude of the original object or estimate of the object.

The iterations of the algorithm continue until all the U-domain and X-domain constraints are satisfied. When this occurs, a solution has been found by the algorithm. The algorithm converges when the mean-squared error goes to zero. In the U-domain, the squared error is the sum of the squares of the difference between $G_k(u)$, the computed Fourier transform, and $G'_k(u)$, the estimate of $F(u)$ after the U-domain constraints have been applied.[7]

$$e^2_{Fk} = N^{-2} \sum_u |G_k(u) - G'_k(u)|^2 , \qquad \text{for generalized measurements} \qquad (13)$$

$$= N^{-2} \sum_u [|G_k(u)| - |F(u)|]^2 , \qquad \text{for dual intensity measurements} \qquad (14)$$

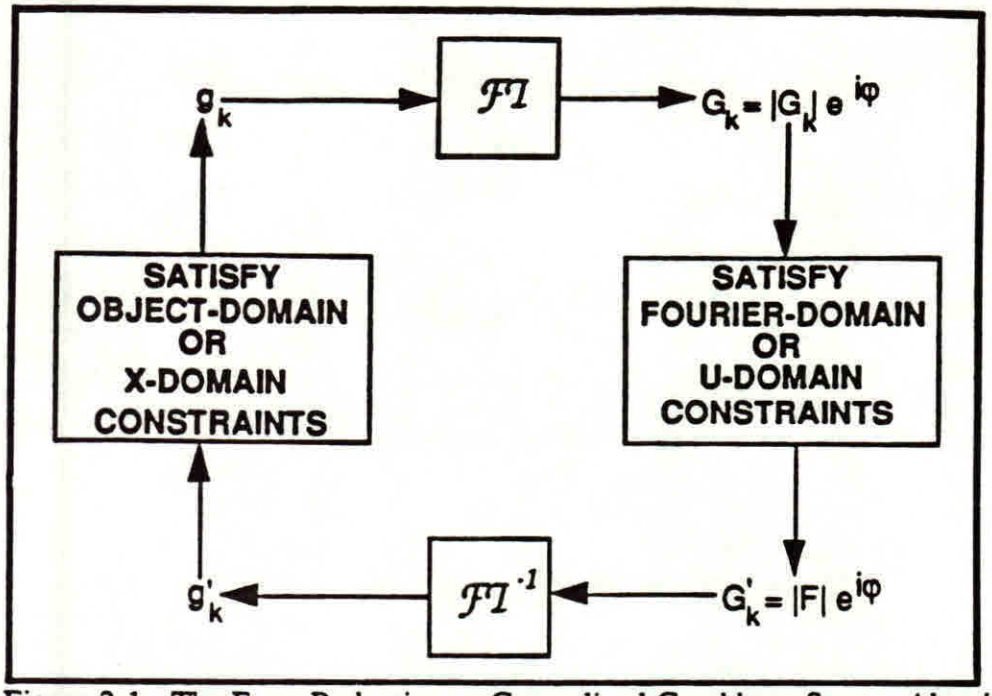

Figure 2-1. The Error-Reduction or Generalized Gerchberg-Saxton Algorithm

In this case e^2_{Fk} is the mean-squared error in the U-domain or Fourier-domain and e^2_{Ok} is the mean squared error in the X-domain or object-domain.[7,14]

$$e^2_{Ok} = \sum_x |g_{k+1}(x) - g'_k(x)|^2 , \quad \text{for generalized measurements} \quad (15)$$

$$= \sum_x [|f(x)| - |g'_k(x)|]^2 , \quad \text{for dual intensity measurements} \quad (16)$$

$$= \sum_{x \varepsilon \gamma} [g'_k(x)]^2 , \quad \text{for single intensity measurements} \quad (17)$$

For the error-reduction algorithm, the mean-squared error in the U-domain can be converted to the X-domain using Parseval's theorem.[15] The energy computed in the X-domain must equal the energy computed in the U-domain, which is given by the following derivation:

$$e_{Fk}^2 = N^{-2} \sum_u |G_k(u) - G'_k(u)|^2 \quad (18)$$

If $H_1(u) = |G_k(u) - G'_k(u)|$, then

$$e_{Fk}^2 = N^{-2} \sum_u [H_1(u)]^2$$

$$= N^{-2} \sum_u H_1(u) \cdot H_1(-u)$$

$$= N^{-2} \cdot N^2 \sum_x [h_1(x)]^2$$

where $h_1(x) = |g_k(x) - g'_k(x)|$,

$$e_{Fk}^2 = \sum_x |g_k(x) - g'_k(x)|^2 \quad (19)$$

Both $g_k(x)$ and $g_{k+1}(x)$ always satisfy the X-domain constraints and $g_{k+1}(x)$ is always the nearest value of $g'_k(x)$ that satisfies the X-domain constraints.[7,14] Therefore, the error in the $k^{th}+1$ iteration is always less than or equal to the error in the k^{th} iteration:

$$|g_{k+1}(x) - g'_k(x)| \leq |g_k(x) - g'_k(x)| \quad (20)$$

or equivalently,

$$e_{Ok}^2 \leq e_{Fk}^2 \quad (21)$$

Similarly, if Parseval's theorem is applied to the X-domain,

$$e_{Ok}^2 = \sum_x |g_{k+1}(x) - g'_k(x)|^2 \quad (22)$$

If $|g_{k+1}(x) - g'_k(x)| = h_2(x)$, then

$$e_{Ok}^2 = \sum_x [h_2(x)]^2$$

$$= N^{-2} \sum_u H_2(u) \cdot H_2(-u)$$

$$= N^{-2} \sum_u [H_2(u)]^2$$

where $H_2(u) = |G_{k+1}(u) - G'_k(u)|$,

$$e_{Ok}^2 = N^{-2} \sum_u |G_{k+1}(u) - G'_k(u)|^2 \quad (23)$$

Both $G'_k(u)$ and $G'_{k+1}(u)$ always satisfy the U-domain constraints and $G'_{k+1}(u)$ is always the nearest value of $G_{k+1}(u)$ that satisfies the U-domain constraints.[7,14] Therefore, the error in the $k^{th}+1$ iteration is always less than or equalled to the error in the k^{th} iteration:

$$|G_{k+1}(u) - G'_{k+1}(u)| \leq |G_{k+1}(u) - G'_k(u)| \quad (24)$$

or equivalently,

$$e_{F\,k+1}^2 \leq e_{Ok}^2 \quad (25)$$

Combining equations 21 and 25 for the X-domain and U-domain analysis [7,14]:

$$e_{F\,k+1}^2 \leq e_{Ok}^2 \leq e_{Fk}^2 \quad (26)$$

Therefore, the error can only decrease with each successive iteration.

For the case of the use of the error-reduction algorithm in this document, the normalized mean-squared error is monitored. The normalized mean-squared error is defined in the U-domain as:

$$E_{Fk}^2 = \frac{\int_{-\infty}^{\infty} |G_k(u) - G'_k(u)|^2 \, du}{\int_{-\infty}^{\infty} |G'_k(u)|^2 \, du} \tag{27}$$

where k is for the k^{th} iteration.[14] The normalized mean-squared error is defined in the X-domain as:

$$E_{Ok}^2 = \frac{\int_{-\infty}^{\infty} |g_{k+1}(x) - g'_k(x)|^2 \, dx}{\int_{-\infty}^{\infty} |g'_k(x)|^2 \, dx} \tag{28}$$

where k is for the k^{th} iteration.[14] In the U-domain, the integral in the numerator is the squared modulus of the amount by which the computed function violates the constraints of the U-domain. A similar explanation is true for the X-domain.

As will be discussed in section 3 of this document, the error-reduction or generalized Gerchberg-Saxton algorithm gave very disappointing results. The error initially decreases rapidly but after the first few iterations begins to stagnate. The speed of the convergence depends on the constraints used in the application of the algorithm. Convergence for dual intensity measurements and one-dimensional single intensity applications is excellent, but convergence is slow for two-dimensional single intensity applications. Fienup performs error analysis in which he discovers that in some cases more than 10^5 iterations are necessary to reduce the root mean squared (RMS) error by 10^{-2} to 10^{-3}.[7] However, as Feinup, the author and others have discovered, there are plateaus where the convergence is extremely slow. After these plateaus, the error again decreases rapidly within a few iterations and then stagnates on another plateau.[1,2,7,16] Other algorithms had to be developed to work in conjunction with the error-reduction algorithm.

2.3 INPUT-OUTPUT ALGORITHM

The input-output algorithm is similar to the error-reduction algorithm except for the X-domain operations. The first three steps of the input-output algorithm are the same as the error-reduction algorithm. The input-output algorithm, shown in figure 2-2, always has an output that satisfies the U-domain constraints; therefore, if $g'_k(x)$ also satisfies the X-domain constraints, then it is the solution.[7,14] The input $g_k(x)$ is no longer the current best estimate of the object unlike the error-reduction algorithm; rather, it is the driving function for the next output of $g'_k(x)$. Therefore, the input $g_k(x)$ does not necessarily satisfy the X-domain constraints. This allows ingenuity in devising X-domain constraints that select the next input and for devising new algorithms to accelerate the convergence process.

A small change in the input results in a small change in the output in the same direction. For a small change in the input, the expected value for the change in the output is a constant α multiplied by the change in the input.[7,14] By changing the input, the output can be steered in the direction of the correct output. If a change of $\Delta g(x)$ is needed in the output, then the change made to the input would be $\beta \Delta g(x)$, where β is a constant equalled to:

$$\beta = \frac{1}{\alpha} \tag{29}$$

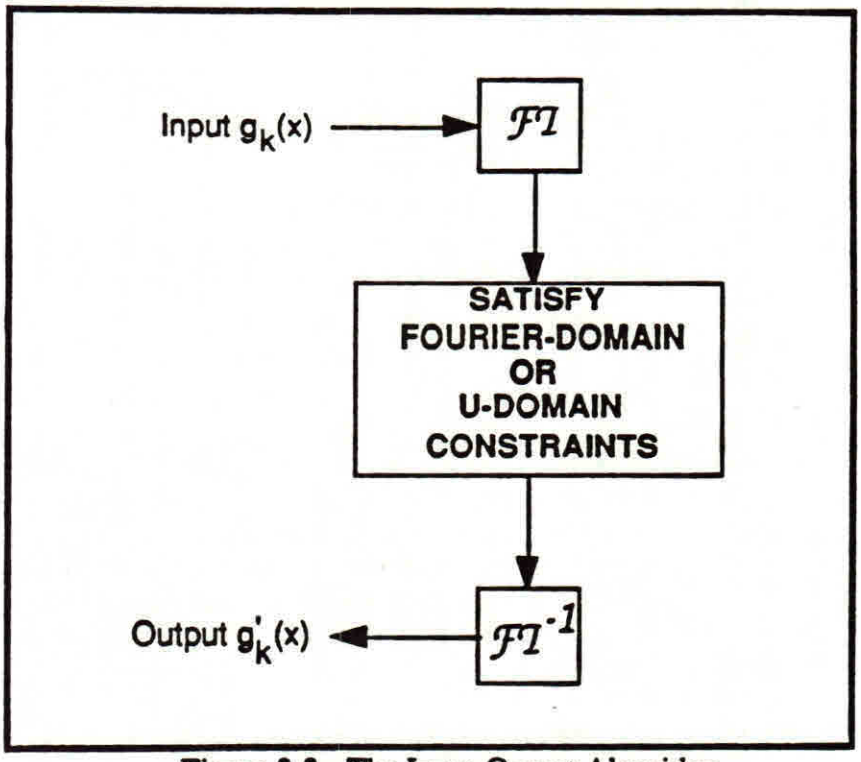

Figure 2-2. The Input-Output Algorithm

There are many classes of input-output algorithms such as the basic input-output, the output-output, and hybrid input-output algorithms which Fienup describes in detail in references 7 and 14. This document will only briefly describe the basic input-output algorithm and in particular, the hybrid input-output algorithm.

For the problem of phase retrieval in a single intensity measurement, the change in the output should be the following:

$$\Delta g_k(x) = \begin{cases} 0 & \text{for } x \notin \gamma \\ -g'_k(x) & \text{for } x \in \gamma \end{cases} \quad (30)$$

where γ is the set of points that violates the X-domain constraints. When the constraints are satisfied, there is no change required in the output; when the constraints are not satisfied, the change in the output needed to satisfy the X-domain constraints drives it to zero. The process that drives it to zero occurs when the change in the input is the negative of the output at the points that violate the X-domain constraints.[7] The next input is then the following:

$$\begin{aligned} g_{k+1}(x) &= g_k(x) + \beta \Delta g_k(x) \\ &= \begin{cases} g_k(x) & \text{for } x \notin \gamma \\ g_k(x) - \beta g'_k(x) & \text{for } x \in \gamma \end{cases} \end{aligned} \quad (31)$$

This is called the basic input-output algorithm.[7]

In the hybrid input-output algorithm the next input $g_{k+1}(x)$ is created in the following way:

$$g_{k+1}(x) = \begin{cases} g'_k(x) & \text{for } x \notin \gamma \\ g_k(x) - \beta g'_k(x) & \text{for } x \in \gamma \end{cases} \quad (32)$$

where γ is the set of points that violates the X-domain contraints. The hybrid input-output algorithm avoids the stagnation that occurs in the output-output algorithm and the error-reduction algorithm. If for a given value of x the output remains negative for more than one iteration, the corresponding point in the output grows more positive until eventually it becomes nonnegative. Notice if β is equalled to one, then the hybrid input-output algorithm reduces to the error-reduction algorithm.[7] Appendix B provides a mathematical interpretation of the hybrid input-output algorithm.[14]

These input-output algorithms can be used on their own. However, for the input-output algorithms, the error E_{Fk} is usually meaningless, since the input $g_k(x)$ is no longer an estimate of the object. Therefore, the mean-squared error in the X-domain usually increases or becomes negative. Figure 2-3 shows a graph of the root mean squared (RMS) error vs. the β of the input-output algorithm for a specific 16 by 16 input matrix after 30 iteration of the algorithm (see section 3.3.4 for more information on how E_{Ok} is calculated). With each execution of the hybrid input-output algorithm, the algorithm started with the same distorted image but each had a different value of β. Each individual β for the input test matrix or test object produced different results - some were good results others were bad results. If β was too small the RMS of E_{Ok} did not reduce enough or increased; where as if too large a β was used, the RMS of E_{Ok} increased too much. However, as β increases, the RMS of E_{Ok} may increase but this may not be a bad

result because in some cases, especially in the combination of the error-reduction and hybrid input-output algorithms, it is good for E_{Ok} to increase. It increases the energy in certain areas of the distorted image to improve and accelerate the convergence of the algorithm to the correct image. Reducing the RMS of E_{Ok} in the hybrid input-output algorithm does not necessarily mean an improvement in the quality of the image just as an increase in the RMS of E_{Ok} in the hybrid input-output algorithm does not mean an increase the amount of distortion within the image. However, when the hybrid input-output algorithm is used in conjunction with the error-reduction algorithm, it reduces the stagnation problem of the error-reduction algorithm and accelerates the convergence process of the error-reduction algorithm in significantly fewer iterations (see figure 3-8 in section 3.3.4).

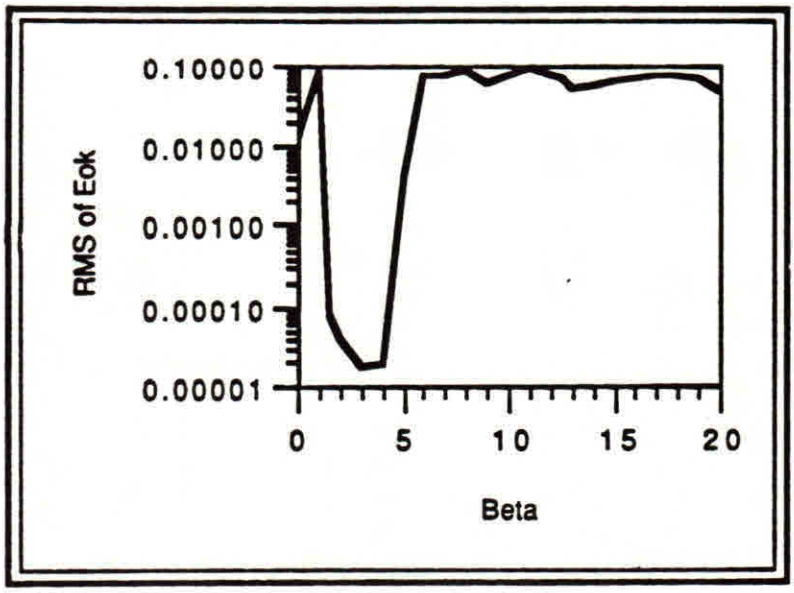

Figure 2-3. RMS Error of E_{Ok} vs. β for Hybrid Input-Output Algorithm (Log Graph)

For this document, section 3 will investigate the application of the error-reduction and hybrid input-output algorithms to problems were only information on the magnitude of an object is known in the U-domain. This is a single intensity application of the algorithm. In the X-domain, all that is known is that the object is real and nonnegative. Therefore, the stagnation encountered in single intensity applications can be overcome by alternating between the error-reduction algorithm and the hybrid input-output algorithm. The application of the error-reduction algorithm will prevent the error in hybrid input-output algorithm from increasing if it begins to diverge from the object. In section 3 of this document, it is found that 10 to 30 iterations of the input-output algorithm followed by 5 to 10 iterations of the

error-reduction algorithm is the best strategy. This combination of the error-reduction and hybrid input-output algorithms significantly reduces the iterations needed to retrieve the phase of the object from, in some cases, more than 10^5 to a few hundred iterations.

SECTION 3

SOFTWARE IMPLEMENTATION OF THE ALGORITHM

The software implementation of the algorithm was performed on an IBM* PS/2 - 80 under DOS* version 3.32 using version 5.0 of Turbo Pascal*. To use the program, a computer must be PC compatible and must have a EGA, CGA or VGA color graphics card. The simulation was designed for a specific application of the error-reduction algorithm. The application of this algorithm is similar to the second application discussed in section 1 of this document. In the test case, the error-reduction algorithm is used to retrieve the phase of a two-dimensional single intensity measurement which is real and nonnegative. This application is perhaps one of the most difficult because in most cases the algorithm is applied to dual intensity measurement or one-dimensional single intensity measurements. By single intensity, there is information on only the real components of an object in one of the domains (for this case the X-domain) and information on the real and the imaginary components in the other domain (for this case the U-domain); by dual intensity, there is information on the real and the imaginary components in both the X-domain and the U-domain. An algorithm for the reconstruction of a multidimensional sequence from the modulus or magnitude of its Fourier transform has been the objective of many research efforts as described in sections 1 and 2 of this document.

This section is broken down into three subsections. Subsection 1 will outline the initial task of the simulation and discuss the problems that were encountered in the software simulation. Subsection 2 will outline the final algorithm that was used to solve the initial problem. Subsection 3 will detail the software implementation of the algorithms with a technical overview of each of the modules that were used to create the final algorithm.

3.1 PROBLEMS IN THE APPLICATIONS OF THE ALGORITHMS

Initially, it was desired to implement this application of the algorithms and create a database of estimated Fourier object moduli that the algorithms could compare against a corrupted input object. Unfortunately, do to problems in the programming of the algorithm, this was not possible because there was just not enough time. There are some problems with this kind of application as this section will outline. In fact, there were many problems to overcome just to get a working algorithm.

---

* IBM, DOS, and PS/2 are registered trademarks of the International Business Machine Corporation. Turbo Pascal is a registered trademark of the Borland International Corporation.

First, a simple algorithm was programmed for a one-dimensional single intensity measurement. The input was a waveform which was corrupted by a random phase and then retrieve using the error-reduction algorithm. The test was a complete success. In fact, the algorithm retrieved every waveform successfully and within 10 to 200 iterations. However, the possibility still exists that the solution may not be unique for a one-dimensional application especially for complex poles that are negative. It was thought that this application was going to work just as successfully in two-dimensions but it did not.

As with the one-dimensional application, the two-dimensional application for an object had the following constraints in the X-domain and the U-domain:

X-domain: All components must be real and nonnegative, which means that all the imaginary terms were set to zero and that all negative real terms were set to zero.

U-domain: The new best estimate of $G_k'(u)$ was equalled to the saved Fourier modulus or magnitude of the initial object F(u) multiplied by the $G_k(u)$ and divided by the modulus or magnitude of $G_k(u)$:

$$G_k'(u) = |F(u)| \, [G_k(u)/|G_k(u)|] \tag{33}$$

Including time to research articles and information on the algorithms, it took approximately four to five weeks to program a version of the error-reduction algorithm with 36 grey scale levels for the graphical display of a 8 by 8, 32 by 32, or 64 by 64 object. Unfortunately, the results were terrible due to the problem of stagnation. At first, it looked as if the algorithm was working for symmetrical objects (i.e. squares and rectangles); however, a closer look at the numerical calculation found that it was not working at all. Furthermore, for non-symmetrical or strange shapes (i.e an airplane) the algorithm almost never converged. Thousands of iterations were made on many 32 by 32 objects, and little to no change was made to the corrupted image. It was at this point that alternative algorithms had to be investigated to overcome the problem of stagnation and the hybrid input-output algorithm, developed by Fienup, was chosen. This algorithm was the best choice for two-dimensional single intensity measurement applications.

The results of the stand alone application of the hybrid input-output algorithm were dismal and so was the combined application of the error-reduction and hybrid input-output algorithms to the retrieval problem. Not only was stagnation a problem but with the application of the hybrid input-output algorithm, excessive divergence became a problem. The entire program was examined carefully from the error-reduction and hybrid input-output algorithm implementation to the FFT implementation.

The algorithm was broken down into a 16 by 16 form to be applied to a 8 by 8 object. and the code was examined carefully line by line as it executed. The numerical analysis of the error-reduction algorithm performed exactly as Fienup describe in references 7 through 14. It stagnated after the first few iterations. However, the addition of the hybrid input-output algorithm did not accelerate the process at all. Several different modifications to the original error-reduction algorithm were made including a unique random number generation process that creates the random phase which corrupts the original object (see figure 3-2 in section 3.2). The random phase generation algorithm uses the same random phase in the object plane it does in the rest of the three planes (see section 3.2 for more information). Again the application of this random phase modification made very little improvement.

Finally, a different application of the algorithm was tested. The application that was implemented was the first application described in section 1 of this document. This dual intensity application had information on the real and imaginary components in the X-domain and the U-domain of the modulus of the original object, specifically |f(x)| and |F(u)|. These two moduli were used to retrieve corrupted objects successfully. It was at this point that is was evident that the code that was developed was working correctly. Research was the last option available in uncovering more information on the algorithms; otherwise, this application would have to be abandoned.

With more research, information was found on problems that others had with this specific single intensity application of the algorithm. Professor Monson H. Hayes of the Georgia Institute of Technology described similar problems in the magnitude-only reconstruction with information on the modulus of the object in only the U-domain when he used the error-reduction algorithm. His results described the same problems that occurred in the coding of the error-reduction algorithm. Hayes claims that there are two problems preventing the development of such an algorithm. First, there is the non-linear relationship between coefficients of a multidimensional sequence and the modulus or magnitude of its Fourier transform. Second, without any phase information, it is not generally possible to form an accurate estimate of the unknown sequence from just the modulus or magnitude of its Fourier transform. Unlike the phase-only iteration technique described by Hayes, he observes that the magnitude-only iteration will not generally converge to the correct solution or to any solution even if the desired object or sequence satisfies the constraints and the uniqueness criteria.[17]

Fienup wrote a letter responding to the claims of Hayes. In this letter, Fienup comments on the article by Hayes in which Fienup refutes the claim of Hayes that the magnitude of the Fourier transform does not uniquely specify a sequence or object. Fienup outlines two important constraints which must occur. First, the bounds must be known in the X-domain and second the sequence or object must be real and nonnegative. However, it may still not be possible to reconstruct the sequence or object if the region of finite support is not known. *Fienup then states that when an upper bound in the X-domain is placed on the region of support, the algorithm converges must faster in 100 or 200 iterations than when using only the nonnegativity constraint in which case several*

thousand iterations are necessary to retrieve the sequence or object.[18] Apparently, the error-reduction algorithm will converge after several thousand iterations but Fienup accelerated the process by placing a maximum bound in the X-domain based on the maximum intensity his instruments were capable of measuring. Fienup goes on to state that for some objects or sequences, this technique of image retrieval may not perform well even with the added constraint in the X-domain using the error-reduction algorithm.

Six months of endless modifications to and research on the algorithm finally produced a working algorithm. With the addition of the maximum bound in the X-domain, the combination of the error-reduction and hybrid input-output algorithms worked successfully. The number of iterations dropped from thousands to a few hundred. However, there were still more problems to address.

The second problem was discovered in the implementation of the hybrid input-output algorithm. As described in section 3.2 of this document, the error-reduction algorithm and hybrid input-output algorithm were combined into one algorithm that alternated between the two algorithms after a few iterations of each one. Thousands of iterations were still needed to retrieve some objects. Therefore, the final algorithm in section 3.2 was modified to first allow the error-reduction algorithm to iterate until it stagnates. When the error-reduction algorithm stagnated, the MSE would go to zero. The final algorithm would then save the image and then the hybrid input-output algorithm would start to iterate with a β of 2. After 20 to 30 iteration of the hybrid input-output algorithm, the hybrid input-output algorithm would stop and the error-reduction algorithm would begin until it again stagnated. At this point the image would be checked. If there was no improvement from the saved image, then the hybrid input-output algorithm would start over with the saved image but this time with a β equalled to the former β plus 2. Therefore, the next set of iterations of the hybrid input-output algorithm would have a β of 4. This process would continue until there was improvement in the new image as compared to the saved image and as compared to the save modulus of the original estimated object that the algorithm desires to retrieve. However, the results of this modification were dismal. Therefore, the final algorithm merely stopped when stagnation occurred in the error-reduction algorithm and then prompted the operator for a β and for the number of iterations to use this β in the hybrid input-output algorithm (see section 3.2).

The hybrid input-output algorithm in general was very sensitive to the β and to the number of iterations. As stated in section 2.3, the MSE analysis is useless during the application of the hybrid input-output algorithm because the error may increase before it begins to decrease or it may decrease and then increase. If the β was to high, the MSE increased too much and the image quality deteriorated. If there were too many iteration with a specific β, the results went from good to terrible as the image quality improved and then deteriorated. If the β was too small, the image did not change at all. Therefore, the best way to implement the final algorithm (described in section 3.2) was to allow the operator to decide what β to use and for how many iterations of the algorithm to use it before returning to the error-reduction algorithm. The best results were achieve when a

value of β (from 3 to 45) was used in the beginning of the algorithm but returned to a low value of β (0 to 3) when the operator saw the image quality improving. As noted with the hybrid input-output algorithm, sometimes the error or distortion had to increase significantly before the hybrid input-output and error-reduction algorithms decrease the error and improved the quality of the image. Further analysis of the β input should be performed to determine the mathematical relationship of the β to the algorithm.

The third problem was the shifting of the image, which occurred during the execution of the error-reduction and hybrid input-output algorithms. The image would sometimes move around within the 8 by 8, 32 by 32, or 64 by 64 area of pixels. To prevent this from occurring, two diagonal corner pixels were chosen in the object. These two pixels were then set to the maximum intensity possible. For the case of 36 grey scale level, they were set to 36. The two pixels became part of the retrieved image and prevented the retrieved image from moving around (see figure 3.2 in section 3.3). The addition of these two pixels merely added a constant DC term to the original object. Alternatively, a high intensity border was also placed around the object that was to be retrieved which also prevented movement.

The fourth problem was retrieving the mirrored image of the original object. This is due to the fact that the error-reduction and the hybrid input-output algorithm have an inability to tell the difference between g(x) and g(-x). Therefore, depending on the random phase that is generated, the algorithms can retrieve g(-x) from g(x). Moreover, this is also true since the algorithm uses the magnitude of the Fourier modulus to retrieve the image. This was only a problem when the retrieved object was compared against the the stored estimate of the object. It poses only a small problem because a filter can be designed to rotate the image by 180°. Optically, a 180° phase plate can be inserted into the optical system.

Finally, both algorithms exhibited extreme sensitivity to the stored Fourier modulus of the object. This stored modulus must be an accurate estimate of the object that is to be retrieved. This modulus, |F(u)|, will be used by the hybrid input-output algorithm to steer the convergence of the object in the correct direction. However, in testing the algorithms, some interesting results were discovered. If the stored modulus of the estimate is very different from the original input or object, the algorithm has a tendency to retrieve an object that resembles the stored modulus because the hybrid input-output algorithm steers the retrieval process in the direction of the stored modulus. However, both algorithms have difficulty in retrieving a complete object. If the stored modulus of the estimate is close to the original input or object, the algorithms have a tendency to retrieve an object that resembles the stored modulus because the hybrid input-output algorithm again steers the retrieval process in the direction of the stored modulus. The recognition of bounds of distorted objects and the sensitivity to the Fourier modulus will be discussed in further detail in section 4. This area of investigation also requires more research and analysis.

3.2 OUTLINE OF THE FINAL ALGORITHM

This subsection will detail the final algorithm that was implemented but specific parts of the algorithm are described in full detailed in section 3.3. The final algorithm was a combination of the error-reduction and hybrid input-output algorithms. The algorithm itself is designed to retrieve a two-dimensional object from a distorted image that is real and nonnegative using the Fourier modulus of the original object. The following are the X-domain and U-domain constraints used in the implementation of the final algorithm which is a combination of the error-reduction and hybrid input-output algorithms described in section 2:

Error-Reduction Algorithm

X-domain: All terms must be real and nonnegative, which means all the imaginary components and all the negative real components were set to zero during each iteration of the algorithm. If any real components were greater than the maximum bound (in this case 36 for 36 grey scale levels) or exceeded the maximum intensity, then that real component was reset to the value of the maximum bound (in this case 36).

U-domain: The new best estimate of $G'_k(u)$ was equalled to the saved Fourier modulus or magnitude of the initial object F(u) multiplied by the $G_k(u)$ and divided by the modulus or magnitude of $G_k(u)$:

$$G'_k(u) = |F(u)| \, [G_k(u)/|G_k(u)|] \tag{33}$$

and

$$G_{k+1}(u) = G'_k(u) \tag{34}$$

Hybrid Input-Output Algorithm

X-domain: All components must be real and nonnegative, which means all imaginary components were set to zero and all negative real components are set to the following as described in section 2.3:

$$g_k(x) = g_s(x) - \beta g'_k(x) \tag{35}$$

where $g_k(x)$ is the new object based on $g_s(x)$, which is the saved $g_k(x)$ from the previous iteration, and based on $g'_k(x)$, which is the

estimate of the object after the X-domain constraints are applied. All other components are retained:

$$g_k(x) = g'_k(x) \tag{36}$$

This description of the hybrid input-output algorithm is similar to the description found in section 2.3. If any real components were greater than the maximum bound (in this case 36 for 36 grey scale levels) or exceeded the maximum intensity, then that real component was reset to the value of the maximum bound (in this case 36).

U-domain: The new best estimate of $G'_k(u)$ was equalled to the saved Fourier modulus or magnitude of the initial object F(u) multiplied by the $G_k(u)$ and divided by the modulus or magnitude of $G_k(u)$:

$$G'_k(u) = |F(u)| \, [G_k(u)/|G_k(u)|] \tag{33}$$

and then

$$G_{k+1}(u) = G'_k(u) \tag{34}$$

To allow the final algorithm to converge more rapidly and clearly, any real component is set to zero if it is less than 0.0001 instead of zero in both the error-reduction and hybrid input-output algorithms.

The final algorithm is depicted in figure 3-1. First, a two-dimensional input or object is sampled. This input or object is called f(x), which must be real and nonnegative. The object f(x) is displayed on the screen of the computer for the operator to view. The operator can then continue by pressing return or quit the program by pressing 'q' on the keyboard. Then f(x) is Fourier transformed using the Fast Fourier Transform (FFT) to yield F(u). The modulus or magnitude of F(u) is calculated and saved. Therefore, |F(u)| becomes the saved Fourier estimate of the object or input, which will be used in the implementation of the U-domain constraints. The magnitude of F(u) is displayed on the screen of the computer for the operator to view. The operator can then continue by pressing return or quit the program by pressing 'q' on the keyboard. The Fourier modulus is then multiplied by a random phase $\exp[i\delta(u)]$ to create $G'_k(u)$. The random phase distorts and corrupts the image with noise. $G'_k(u)$ is then used as the input into the final algorithm. The magnitude of $G'_k(u)$ is displayed on the screen of the computer for the operator to view and is called $R(u) = |G'_k(u)|$. The operator can then continue by pressing return or quit the program by pressing 'q' on the keyboard.

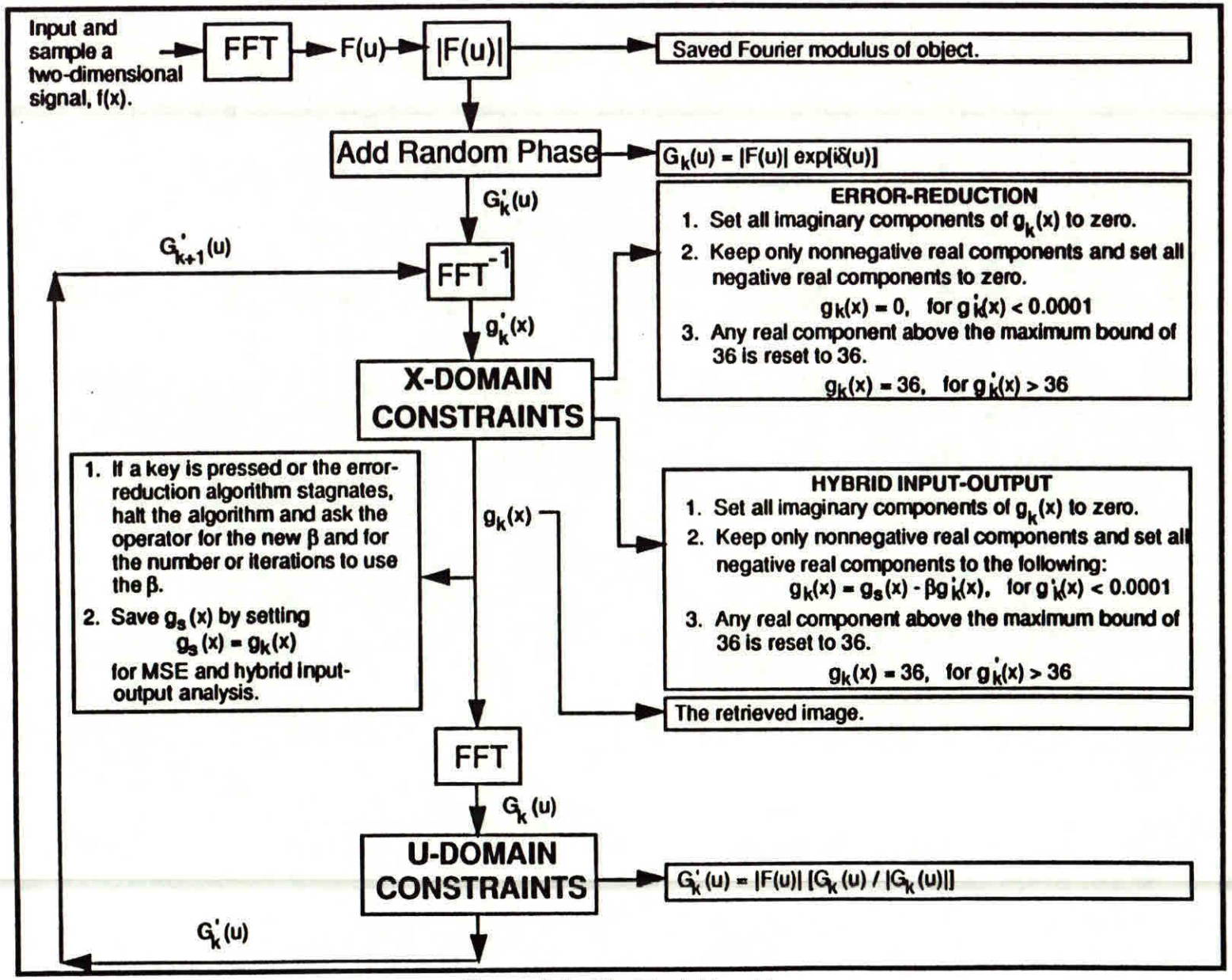

Figure 3-1. The Final Algorithm

$G'_k(u)$ is inversed Fourier transformed using the inverse FFT to create $g'_k(x)$. The X-domain constraints are then imposed on $g'_k(x)$. There are three X-domain contraints for both the error-reduction and hybrid input-output algorithms. The first X-domain constraint of the error-reduction algorithm sets all the imaginary components to zero; the second X-domain constraint keeps only nonnegative real components and sets any real negative components to zero; and the third X-domain constraint sets any real component above the maximum bound value of 36 to the maximum value of 36 for 36 grey scale levels. The first X-domain constraint of the hybrid input-output algorithm sets all the imaginary components to zero; the second X-domain constraint keeps only nonnegative or positive real components and set all the negative components to the following:

$$g_k(x) = \begin{cases} g'_k(x) & \text{if real components are positive} \\ g_s(x) - \beta g'_k(x) & \text{if real components are negative} \end{cases} \qquad (37)$$

where β is determined by the operator, $g_s(x)$ is the object $g_k(x)$ from the previous iteration of the final algorithm and $g_k(x)$ is the new value of $g'_k(x)$ after the X-domain constraints have been applied; and the third X-domain constraint sets any real component above the maximum bound value of 36 to the maximum value of 36 for 36 grey scale levels. The application of the X-domain constraints for the error-reduction and hybrid input-output algorithm yields $g_k(x)$.

The final algorithm will then evaluate $g_k(x)$ and stop if the error-reduction algorithm stagnates or the operator presses a key to halt the execution of the final algorithm. When the algorithm stops, the operator can press return to enter new data, 's' to save that iteration to the hard disk and then enter new data, or 'q' to quit the program. When the operator presses return or the 's' key, the operator will be prompted for a new β and for the number of iterations to use the β. If a β other than 1 is selected, then the final algorithm will implement the hybrid input-output algorithm with that β and for the number of iterations indicated by the operator. Upon completing the iterations of the hybrid input-output algorithm, the final algorithm will implement the error-reduction algorithm until it stagnates again or until the operator again presses a key to halt the execution of the final algorithm. If the operator selects a β of 1, then the final algorithm will implement the error-reduction algorithm for the number of iteration indicated by the operator and then will continue executing the error-reduction algorithm until it stagnates. When stagnation occurs in the error-reduction algorithm, the final algorithm will again stop and prompt the operator for the same information again. The operator is always able to stop the algorithm during any iteration by merely pressing any key on the keyboard to interrupt the final algorithm and either modify the selection of the β or force a return to the error-reduction algorithm. If the operator press a key, then the algorithm will stop after the next iteration is complete. The algorithm determines if the error-reduction algorithm stagnates using the mean squared error (MSE) analysis which is described in detail in section 3.3.

Finally, before continuing the algorithm stores $g_k(x)$ as $g_s(x)$ for the next iteration of the hybrid input-output algorithm for the MSE analysis. At this point, $g_k(x)$ is Fourier transformed by the FFT to create $G_k(u)$. The U-domain constraints are imposed on $G_k(u)$ to yield $G'_k(u)$, which is created using the modulus or magnitude of the saved Fourier transform of the original object or estimate, F(u):

$$G'_k(u) = |F(u)| \, [G_k(u)/|G_k(u)|] \tag{33}$$

Here, $|F(u)|$ is multiplied by the phase information of $G_k(u)$ divided by the modulus or magnitude of $G_k(u)$. After creating $G'_k(u)$, $G'_{k+1}(u)$ is set equal to $G'_k(u)$ and then the final algorithm begins its next iteration with $G'_{k+1}(u)$ as the next input to be inversed Fourier transformed by the FFT.

The operator has total control over the retrieval process. Besides prompting the operator for an new β and for the number of iterations to use the β in the hybrid input-output algorithm, the operator has other options. As the final algorithm starts its iterations, the operator will have a picture of the original object in the left corner of the screen of the computer with the heading 'GS=0' below the image indicating zero iterations. Along with the original object, the operator will be able to see some of the iterations of the final algorithms during the retrieval process. Every 5 iterations, an image is added to the screen of the computer. When the screen is full of images, there is an upgrade of one of the images on the screen of the computer excluding the image of the original object. In addition, there is an upgrade of one of the images on the screen of the computer when the operator interrupts the execution of the algorithm by pressing any key on the keyboard. The image displayed is the last iteration of the algorithm before it was interrupted. Below each image on the computer screen is displayed the type of iteration along with the iteration number. For an iteration of the error-reduction algorithm, 'GS=' appears below the image along with the iteration number; and for an iteration of the hybrid input-output algorithm, 'IO=' appears below the image along with the iteration number. Between each iteration, the RMS of E_{Ok} is displayed at the bottom of the screen (see section 3.3.4 for more information on the RMS of E_{Ok}). The final algorithm saves iterations 5, 10, 20, 40, 60, 80, 100, 250, 500, 1000, 1500, 2000, 3000, 4000, 5000, 7500, and 10000 in data files on the hard disk called D5.DAT for the 5^{th} iteration, D10.DAT for the 10^{th} iteration and similar names for the other iterations. The final algorithm also saves the original input matrix in a data file called ORIGINAL.DAT which also can be displayed. The maximum number of iterations is 10,000 after which the program completes its execution. Furthermore, after the final algorithm halts, the operator can save an iteration by pressing the key 's', which represents the word save. When the algorithm halts and the key 's' is pressed, the operator will be prompted for the normal information (a new β and the number of iterations to use this β). The final algorithm will automatically save that iteration on the hard disk in a data file with a name similar to the data files stored automatically. Finally, after the final algorithm halts, the operator has the capability to

end the execution of the final algorithm at any time by press the key 'q', which represents the word quit. The program SIMUL.PAS will end execution but will automatically save the last iteration to a data file called LASTIT.DAT, which like other data files can be displayed. Appendix A shows some images that were retrieved using this final algorithm.

In addition a separate program was developed to plot the saved data files on the screen of the computer called program NPLOT.PAS. It prompts the operator for the name of the file the operator wants to display and gives an error message if the operator specifies a file that does not exist or a file that is not a Gerchberg-Saxton file. If it is a Gerchberg-Saxton file, then it plots that file on the screen of the computer for the operator to view. This program is helpful upon the completion of the final algorithm.

3.3 DETAILS ON THE SOFTWARE IMPLEMENTATION OF THE ALGORITHM

This subsection details the software coding of the final algorithm. Each subsection highlights the major components that make up the final algorithm. The subsections in this section will document the memory, FFT, MSE and plotting implementations which need to be discussed in more detail than described in the source code. The final algorithm is contained in program SIMUL.PAS which has three external modules. Program SIMUL.PAS is the main control module in the execution of the final algorithm. The first module is the FFT program, called FFT_LIB.PAS and contains all the FFT procedures and functions. The second module is the operations library, called OP_LIB.PAS and contains all the different procedures for the operations performed by the final algorithm including the implementation of the X-domain and U-domain constraints. The third module is the plot library, called PLOT_LIB.PAS and contains all the procedures that plot images on the screen for the operator to view. Each procedure and function is described in the source code. The Pascal source code for SIMUL.PAS, FFT_LIB.PAS, OP_LIB.PAS, and PLOT_LIB.PAS is a total of 2146 lines (including the comments) and can be found respectively in appendices C, D, E and F. The Pascal source code for NPLOT.PAS is an additional 725 lines (including the comments) and can be found in appendix G. The source code in appendices C through G is configured to process a 64 by 64 input matrix. *Perhaps one of the most important subsections is 3.3.4.*

Before the execution of SIMUL.EXE the operator should first modify the file INPUT.DAT. The file INPUT.DAT should contain the original object that is to be retrieved and should be 16 rows by 16 columns, 64 rows by 64 columns, or 128 rows by 128 columns with a space separating each column and with a line feed or carriage return separating each row. Upon the completing the execution of program SIMUL.PAS (see appendix C), the operator can display any of the saved images by execution program NPLOT.PAS, which was described at the end of section 3.2 and in appendix G. NPLOT.EXE will not be described in any further detail in this document but the source code can be found in the main module of program nplot, NPLOT.PAS.

There are three sizes of objects used to test the final algorithm: 16 by 16, 64 by 64 and 128 by 128. In this document, X stands for the number of columns and Y stands for the number of rows in an X by Y matrix. Twice the bandwidth in the U-domain must be allowed than in the X-domain in order to retrieve the entire image. A bed of zeros twice the size of the image is necessary to retrieve the image in full detail. Using a bed of zeros less than twice the size of the image, may not completely retrieve the image. The image is embedded in either the center of a matrix of zeros or in the left hand corner of a matrix of zeros. In all the examples in the appendix A, the image is placed in a bed of zeros that it twice the size of the image. For example, a 16 by 16 matrix only uses an 8 by 8 plane for the object. The other three 8 by 8 planes are nothing more than zeros. The zeros are used as in the one-dimensional signal application to give the object an area to be retrieved from. On each successive iteration of the algorithm, these three zero planes in two-dimensions are reset to zero as part of the X-domain constraints. This allows the retrieval process to be influence by the effects of the data in the U-domain. Figure 3-2 shows how this setup works for a 16 by 16 matrix where the object plane is in the upper left corner. An alternate technique is to place the object plane in the center of the 16 by 16 matrix and surround the object plane with zeros. There are four dark pixels in diagonal corners used to prevent the image from moving around the screen during the retrieval process as described in section 3.1. There are similar setups for a 64 by 64 and 128 by 128 matrix. Therefore, the actual size of the object is contained in a 8 by 8 matrix for a 16 by 16 input, a 32 by 32 matrix for a 64 by 64 input, and 64 by 64 matrix for a 128 by 128 input. The 16 by 16, 64 by 64, or 128 by 128 input should be contained in the file INPUT.DAT.

The zero padding in two-dimensions is similar to the zero padding in one-dimensions due to the algorithm begin based on the autocorrelation function. Clearly placing the U-domain constraint of multiplying the $|F(u)|$ by $|G_k(u)|$ to get $G'_k(u)$ implies convolution of $g(x)$ with $f(x)$ in the X-domain; however, since $g(x)$ is a estimate of $f(x)$, then it is an approximation of the autocorrelation of $f(x)$. Therefore, zero padding is necessary in the X-domain to achieve the correct autocorrelation of $f(x)$ in the Fourier domain. In one-dimensions, twice the bandwidth of the signal is needed and half of that signal is all zeros. If a signal from bandwidth zero to N is used without zero padding, then folding of the signal will occur when the autocorrelation of the signal is taken. Double the bandwidth is necessary to retrieve the signal to prevent folding and the possibility of circular rather than linear convolution. The reader should refer to chapter three of reference 23 for more information on two-dimensional linear convolution using zero padding.

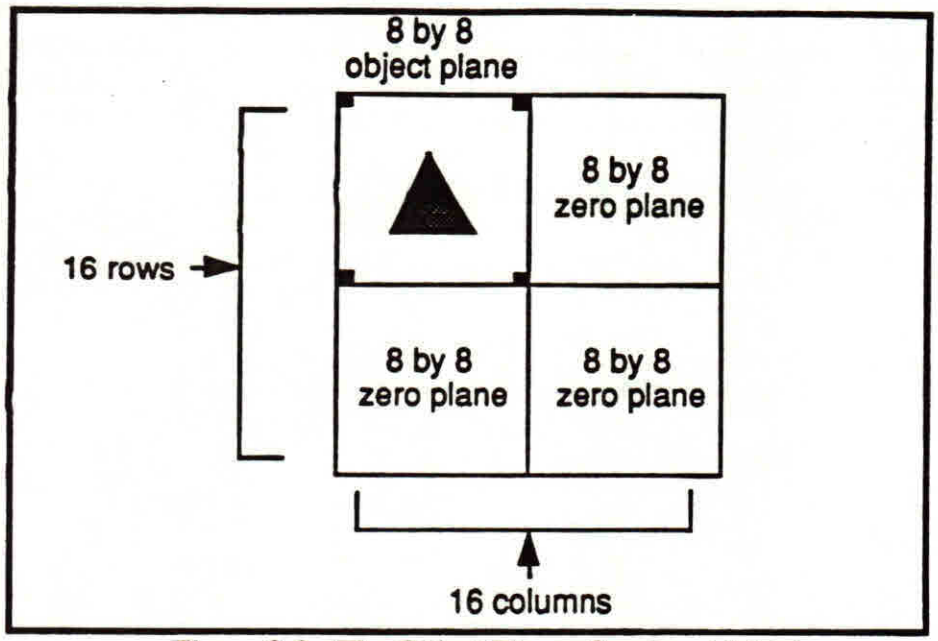

Figure 3-2. The Object Plane of an Input Matrix

3.3.1 Memory

The memory of a PC is a constraint because for DOS the Pascal source code was restricted to 640 KB. Therefore, the operator must determine three parameters before recompiling the code using version 5.0 of Turbo Pascal. First, the object must be within a square amount of pixels such as 8 by 8, 32 by 32, or 64 by 64. The operator then must decide the size of the blocks in memory and then set the following constants in the module: *max_samples*, *sample_block* and *blocks*.

1. *max_samples*: is a set to the square size of the entire input matrix. For example for a 16 by 16 matrix, it is set to 16; for 64 by 64, it is set to 64; and for a 128 by 128, it is set to 128.

2. *sample_blocks*: is the size of the sample block. The sample block is the square size of the blocks which make up the size of the entire input matrix. For a 16 by 16 matrix, the object is broken into four planes each 8 by 8.

3. *blocks*: is the number of planes that make up the object. For a 16 by 16 matrix, there are four blocks each containing an 8 by 8 matrix that makes up the input matrix.

4. *sqr_sample_block*: is automatically calculated as the square of the size of the sample block. If the sample block is 8, its square is 64, which is the total number of entries in the sample block.

5. *sqr_max_samples*: is automatically calculated as the square of the size of the maximum number of samples. If the maximum number of input samples for the object is 16, then its square is 256, which represents the total number of entries in the input matrix.

6. *rows*: is automatically calculated as the maximum samples divided by the number of blocks:

$$(\text{max\_samples} / \text{blocks}) \tag{38}$$

It represents the number of rows in a sample block. The value of sample_blocks represents the number of columns in a sample block. For a 16 by 16 matrix, the following is the calculation for the number of rows in a sample block:

$$\text{rows} = (16 / 4) = 4$$

Therefore, from the above calculations the memory in extended heap is parsed using an array of size *blocks* that holds a pointer to extended heap. This pointer in extended heap points to an array which is the size of *max_samples* by *rows* (column by row). The main data structure in extended heap is called *data*. It is an array of size *blocks* that contains the pointers to arrays in extended heap. Each entry in the arrays in extended heap has a record with two fields. The first field is called *real_pt* for the real component of that pixel or sample and the second field is called *image_pt* for the imaginary component of that pixel or sample.

For a 16 by 16 matrix, *max_samples* is 16, *sample_blocks* is 8 and *blocks* is 4. Figure 3-3 depicts the memory requirements of a 16 by 16 matrix. In this case there are four blocks of memory in extended heap, which have pointers from *data*. The reason for using an array in extended heap is that the access to the data through an array is much faster than access through a linked list. Moreover, Turbo Pascal is unable to compile a data structure such as a matrix in the form of an array that is larger than 64 KB. Therefore, pointers to extended heap to matrices less than 64 KB were needed to accelerate the algorithms. Even though it is not necessary to implement this kind of memory structure for a 16 by 16 matrix, it is necessary for a 64 by 64 and 128 by 128 matrix. For a 64 by 64 matrix, *max_samples* is 64, *sample_blocks* is 16 and *blocks* is 4. Figure 3-4 depicts the memory requirements for a 64 by 64 matrix. Finally, for a 128 by 128 matrix, *max_samples* is 128, *sample_blocks* is 16 and *blocks* is 8. In order to execute this algorithm with a 128 by 128 input matrix, changes must be made to the source code. Five data structures of type *fftp*

and of *size* 128 by 128 have two real field each of size 8 bytes. These five data structures are needed to perform the added MSE and hybrid input-output analysis. This would require 1.35 MB of memory which is not possible with only 640 KB of memory for a conventional computer. If the two real fields in the data structures of type 'fftp' are changed to short integers (4 bytes each) and the MSE analysis is removed from the source code, only four data structures are needed to execute the source code. This would only require 525 KB of memory. Theoretically, a even more streamlined version of the source code can be created to run a 256 by 256 input matrix; however, it is probably wiser to transfer the code to a workstation for an input matrix of this size. Other memory combinations are possible by setting *max_samples*, *sample_blocks* and *blocks* to different values to meet the requirements. However, the original matrix must be square. When executing the program SIMUL.PAS with a 64 by 64, or larger matrix application, the operator should compile and executable version of the code to the hard disk and leave Turbo Pascal; otherwise, the computer will run out of memory with Turbo Pascal in the background.

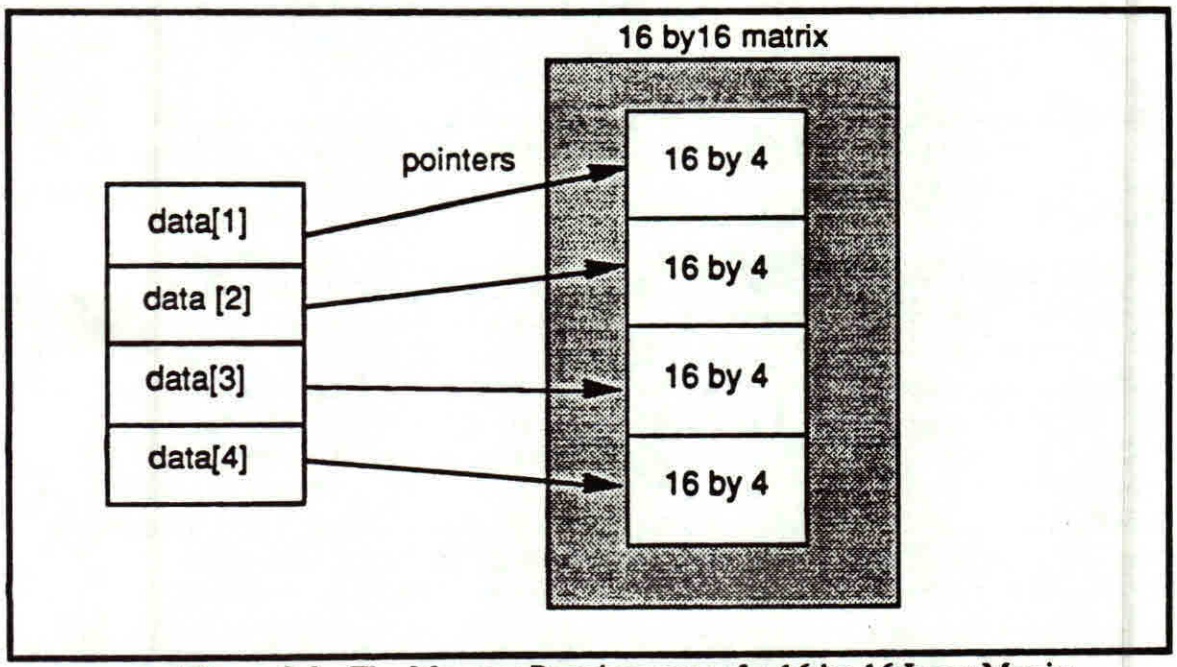

Figure 3-3. The Memory Requirements of a 16 by 16 Input Matrix

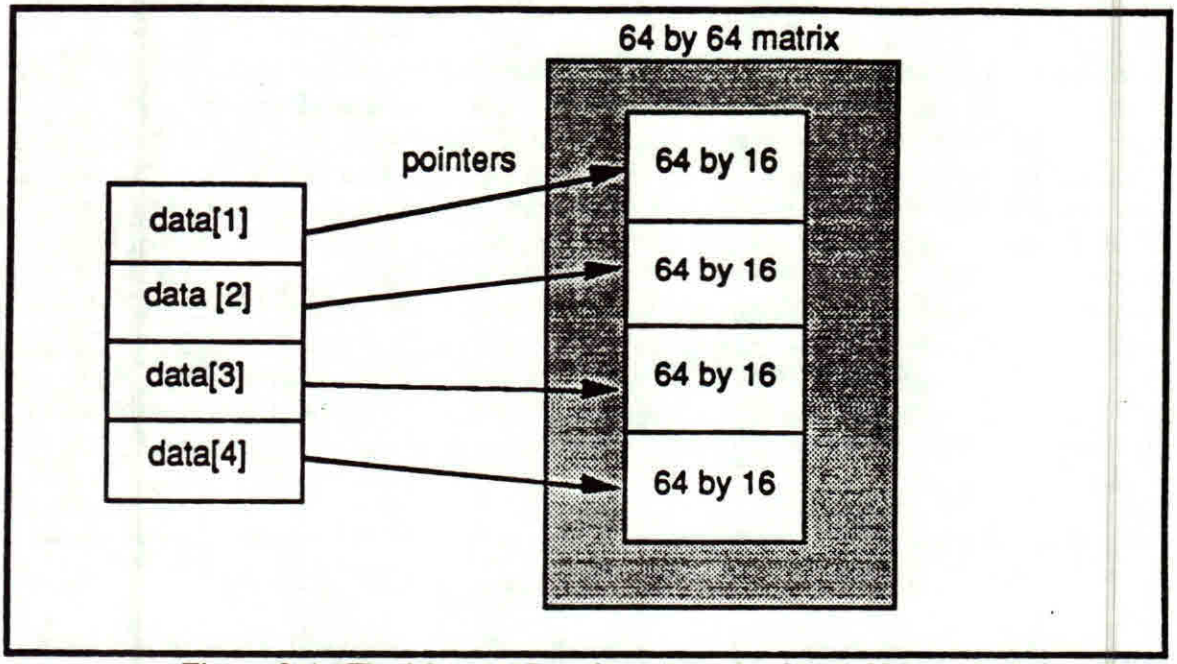

Figure 3-4. The Memory Requirements of a 64 by 64 Input Matrix

3.3.2 Fast Fourier Transform (FFT)

The discrete Fourier transform analysis used in the final algorithm was implemented using the Fast Fourier Transform (FFT).[19] In this application, the data of the two-dimensional object was stored in a two-dimensional matrix, 16 by 16. To take the FFT of a two-dimensional matrix, first Fourier transform all of the rows in the matrix; second, transpose the matrix; third, Fourier transform all the rows again; and fourth, transpose the matrix again. This essentially reduces to taking the Fourier transform of the rows and then the Fourier transform of the columns. Transposition merely turns the rows into columns or the columns into rows. Figure 3-5 depicts the Fourier transform process using the FFT. Figure 3-6 shows how a 16 by 16 matrix is transposed. Therefore, the (column,row) entry of (1,1) in the matrix stays in the same position but now the (row,column) entry is (1,1). However, the (column,row) entry of (2,1) changes to the (row,column) entry of (2,1). This effectively turns the rows into columns or the columns into rows. Transposition is complicated even further in the source code because of the way data is stored in extended heap (see section 3.3.1).

The FFT analysis is located in the FFT module called FFT_LIB.PAS (see appendix D) and contains all the functions and procedures that perform an discrete FFT on an input matrix. Function power_down determines the power of a base number that created the number sent to the function power_down. It is useful in determining the power of 2. It is used to determine the number of stages in the FFT algorithm. Procedure find_trans_entry is used to find the transpose entry within the memory technique implemented in program

SIMUL.PAS (see section 3.3.1). Procedure transpose transposes the two-dimensional matrix for the FFT algorithm. Procedure nfft takes the FFT of one row in a matrix. Procedure two_d_fft breaks the matrix into individual rows to be sent to procedure nfft and also calls procedure transpose.

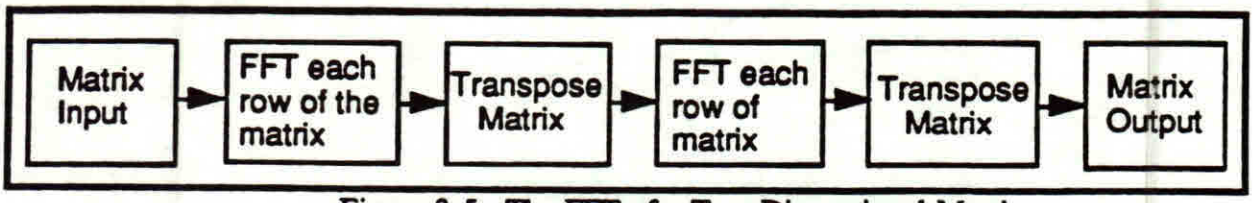

Figure 3-5. The FFT of a Two-Dimensional Matrix

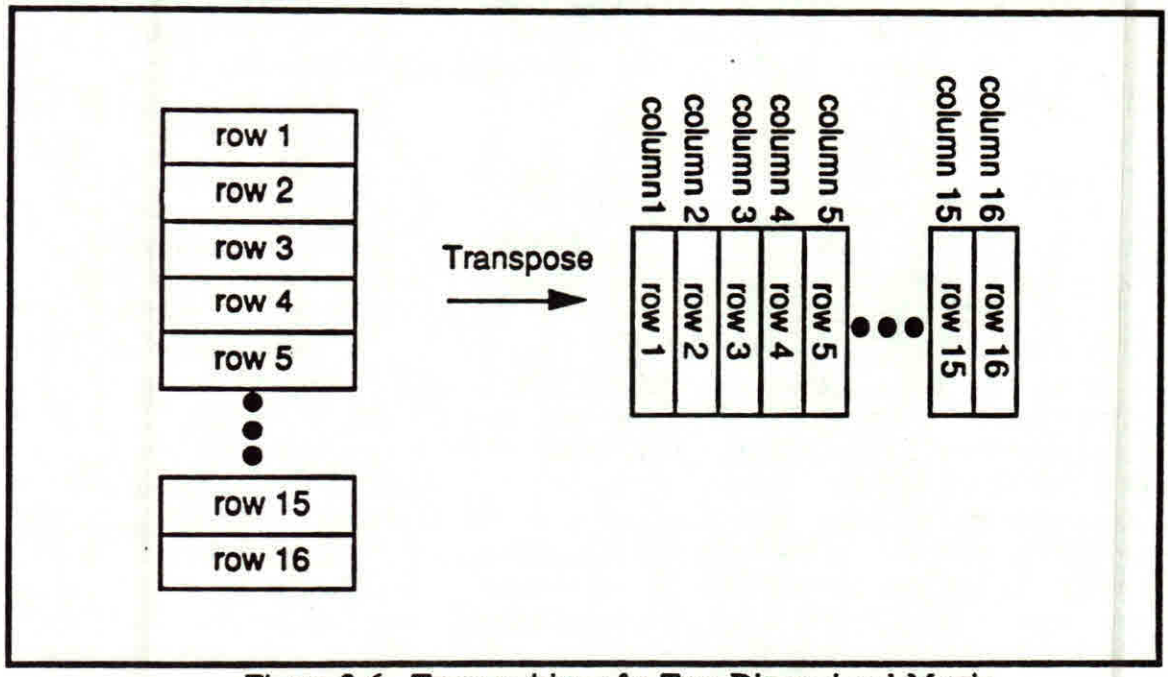

Figure 3-6. Transposition of a Two-Dimensional Matrix

3.3.3 Operations

Many of the operations performed in the final algorithm are located in the operations library, which is called module OP_LIB.PAS (see appendix E). Procedure dispose_pointers disposes of all the pointers that point to matrices in extended heap. Procedure save_info saves information by writing it to the hard disk in files previously described in section 3.2. Procedure get_input reads the INPUT.DAT file, which contains the original object. Procedure make_sine_cos_tbl makes the sine and cosine tables for the analysis used to retrieve the object. This allows the FFT to run more rapidly because the FFT does not have to calculate the sine and cosine function. Instead, the FFT merely

uses a look up array or table to access the sine and cosine values. Procedure multiply_2_d multiplies the two-dimensional arrays together that have both real and imaginary components. Multiplying one matrix of real and complex components with another is accomplished using the following process:

$$(x + jy) \cdot (w + jz) = (xw - yz) + j(yw + xz) \tag{39}$$

$$\text{real component} = (xw - yz) \tag{40}$$

$$\text{complex component} = (yw + xz) \tag{41}$$

Procedure random_phase distorts the original object by calculating random phase matrix of real and imaginary components and then by multiplying the random phase matrix with the Fourier transform of the original object matrix. Random phase is created in the following manner:

$$\cos(2 \cdot \pi \cdot r) + j \cdot [\sin(2 \cdot \pi \cdot r)] \tag{42}$$

where *r* is a randomly generated number created by using the internal random number generation program in Turbo Pascal. This was total distortion of the image or the worst case scenario. An alternate technique is a random phase generation program which program SIMUL.PAS and the final algorithm use (see section 3.1). Either form of random phase generation is valid. The alternate random phase generation program merely accelerates the convergence process, but even with an established random phase generation process, the final algorithm is capable of retrieving any object. Procedure theta_mod applies the U-domain constraints to a matrix for both the error-reduction and hybrid input-output algorithms. Procedure save_prev_data saves the data of a matrix for the next iteration to be used in the MSE analysis or the hybrid input-output algorithm. Procedure save_error_data saves the data of a matrix for the MSE analysis. Procedure x_domain_contraints applies the X-domain constraints of the error-reduction or the hybrid input-output algorithm depending on which algorithm is executing during an iteration. In addition, this procedure performs preliminary calculations for the MSE analysis of the hybrid input-output algorithm. Procedure calculate_error is explained in section 3.3.4. Procedure save_mod_file saves a particular iteration on the hard disk.

3.3.4 Mean Squared Error (MSE) Analysis

The mean squared error (MSE) analysis is performed in the operations library in module OP_LIB.PAS in procedure calculate_error. The mean squared error analysis is only used with the error-reduction algorithm to determine if it converged or stagnated. From section 2.2, the normalized MSE is expressed in the following equations:

$$E_{Ok}^2 = \frac{\int_{-\infty}^{\infty} |g_{k+1}(x) - g'_k(x)|^2 \, dx}{\int_{-\infty}^{\infty} |g'_k(x)|^2 \, dx} \tag{27}$$

$$= \frac{e_{Ok}^2}{\int_{-\infty}^{\infty} |g'_k(x)|^2 \, dx} \tag{43}$$

and

$$E_{Fk}^2 = \frac{\int_{-\infty}^{\infty} |G_k(u) - G'_k(u)|^2 \, du}{\int_{-\infty}^{\infty} |G'_k(u)|^2 \, du} \tag{28}$$

$$= \frac{e_{Fk}^2}{\int_{-\infty}^{\infty} |G'_k(u)|^2 \, du} \tag{44}$$

and therefore using equations 19 and 20:

$$e_{Ok}^2 \leq e_{Fk}^2 \tag{21}$$

The unnormalized MSE causes the algorithm to stop when the following is true:

$$e_{Ok}^2 = e_{Fk}^2 \tag{46}$$

Applying this analysis to a discrete implementation of the MSE analysis, the following must be true, given that $g_{k+1}(x)$ is $g_k(x)$ and $g_k(x)$ is $g_s(x)$ in Figure 3-1 of section 3.2.

$$e_{Fk}^2 = \sum_x |g_k(x) - g'_k(x)|^2 \qquad = \sum_x |g_s(x) - g'_k(x)|^2 \tag{47}$$

$$e_{Ok}^2 = \sum_x |g_{k+1}(x) - g'_k(x)|^2 \qquad = \sum_x |g_k(x) - g'_k(x)|^2 \tag{48}$$

$$e_{Fk}^2 = \int_{-\infty}^{\infty} |g_k(x) - g'_k(x)|^2 \, dx = \sum_x |g_s(x) - g'_k(x)|^2$$

$$= \sum_x \{ [Re(g_s(x)) - Re(g'_k(x))]^2 + [Im(g_s(x)) - Im(g'_k(x))]^2 \} \quad (49)$$

$$e_{Ok}^2 = \int_{-\infty}^{\infty} |g_{k+1}(x) - g'_k(x)|^2 \, dx = \sum_x |g_k(x) - g'_k(x)|^2$$

$$= \sum_x \{ [Re(g_k(x)) - Re(g'_k(x))]^2 + [Im(g_k(x)) - Im(g'_k(x))]^2 \} \quad (50)$$

Finally, to normalize the error, it is also necessary to calculate the following:

$$n_k = \int_{-\infty}^{\infty} |g'_k(x)|^2 \, dx = \sum_x |g'_k(x)|^2 \quad (51)$$

$$= \sum_x \{[Re(g'_k(x))]^2 + [Im(g'_k(x))]^2\} \quad (52)$$

Therefore, the error reduces to the following:

$$\text{Error} = \frac{e_{Fk}^2 - e_{Ok}^2}{n_k} \quad (53)$$

And

$$0 \leq \text{Error} \leq \text{Maximum Value} \quad (54)$$

When this occurs, the error-reduction algorithm halts and prompts the operator for new information. The maximum value is typically from 0.00001 to 0.0001. Figure 3-7 shows the error analysis over 40 iterations on a 16 by 16 input test matrix using only the error-reduction algorithm with the maximum value set to 0.00005. However, the results were poor because as you can see the MSE actually increased during some of the iterations during the execution of the error-reduction algorithm. Again, this analysis seemed to disprove the claims of Fienup until it was realized that using e_{Fk} has no meaning when

applied to a single intensity measurement for the error-reduction algorithm. This analysis was design for dual intensity measurements, which explains the increase in the normalized MSE at about 25 iterations. Therefore, using this error analysis on the error-reduction algorithm with only a single intensity input only shows that the MSE gradually decreases as the number of iterations continues. There is a rapid decrease in the MSE in the first few iterations. However, as Fienup, Hayes and myself experienced, this rapid decrease is followed by a longer period (thousands of iterations) of stagnation that will last long past 40 iterations. Moreover, the decrease initially in the normalized MSE does not necessarily mean that the image has been retrieved only that it has converged. Using this MSE analysis on the hybrid input-output algorithm is useless because e_{Fk} has no meaning and would cause the MSE analysis to produce large negative and positive values. The error e_{Fk} is meaningless since $g_k(x)$ is no longer a estimate of the object as

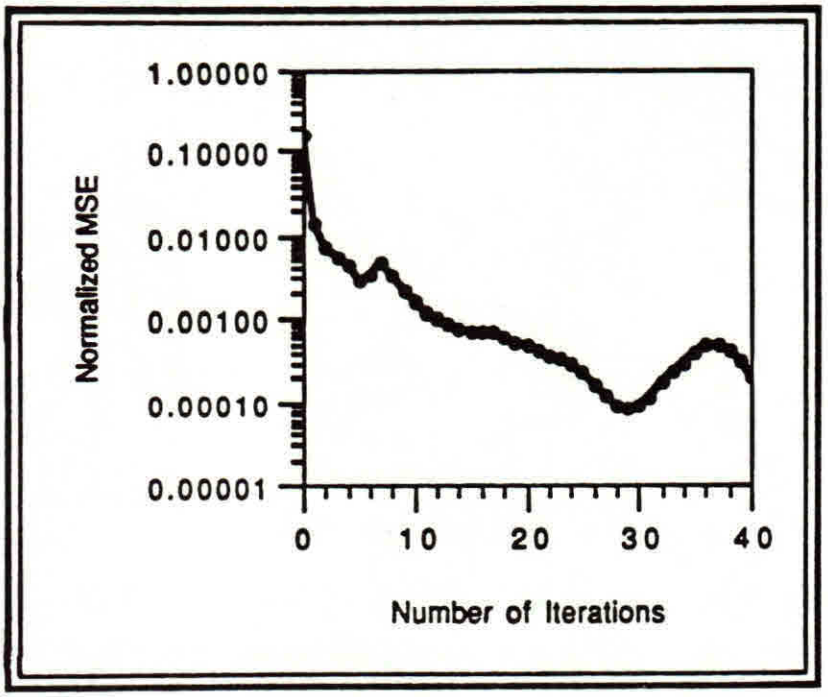

Figure 3-7. Error vs. Number of Iterations (Log Graph)

discussed in section 2.3. This explanation also is true for this particular application of the error-reduction algorithm because it does not use the modulus of the original object, $|f(x)|$, in the X-domain constraints to retrieve the image. Instead, the X-domain constraints is that the object is real and nonnegative. Therefore, for a single intensity measurement, the following evaluation of the e_{Ok} was derived for the error-reduction algorithm to determine if the image converged or stagnated using equation 48:

$$e_{Ok}^2 = \sum_x |g_k(x) - g'_k(x)|^2 \qquad (48)$$

E_{Ok} is the root mean square (RMS) value of e_{Ok}^2 using equation 51.

$$n_k = \sum_x |g'_k(x)|^2 \qquad (51)$$

where from equation 52,

$$\sum_x |g'_k(x)|^2 = \sum_x \{[Re(g'_k(x))]^2 + [Im(g'_k(x))]^2\} \qquad (52)$$

Using equations 48 and 50 for e_{Ok}^2 and equations 51 and 52, E_{Ok} is:

$$E_{Ok} = \sqrt[2]{\frac{e_{Ok}^2}{n_k}} \quad \text{,error reduction} \qquad (55)$$

and

$$0 < E_{Ok} < \text{Maximum Value} \qquad (56)$$

If E_{Ok} is less than the maximum value, then the algorithm halts and prompts the operator for input. This RMS application works successfully for the error-reduction algorithm. Again modification of the original MSE analysis in section 2.2 is necessary for the single intensity application of this algorithm. The maximum value is set by the variable *max_error* in program SIMUL.PAS and can be changed if the code is recompiled. *Max_error* was set to 0.18. During the iterations of the error-reduction algorithm, the final algorithm will halt three iterations after the RMS error of E_{Ok} becomes less than 0.18. This was done to allow the error to decrease during the iterations of the error-reduction algorithm. As will be shown in figure 3-8, the RMS error is typically much less than 0.18 when the error-reduction and hybrid input-output algorithms are combined. Therefore, at least three iterations of the error-reduction algorithm are performed after the iterations of the hybrid input-output algorithm have completed.

The following is the RMS analysis for the hybrid input-output algorithm using equation 17:

$$e_{Ok}^2 = \sum_{x \varepsilon \gamma} |g'_k(x)|^2 \qquad (17)$$

where,

$$\sum_{x \varepsilon \gamma} |g'_k(x)|^2 = \sum_{x \varepsilon \gamma} \{[Re(g'_k(x))]^2 + [Im(g'_k(x))]^2\} \qquad (57)$$

When x is a member of γ, then it violates the X-domain constraints and must be re-evaluated using the hybrid input-output algorithm. The normalized equation for the RMS of E_{Ok} is the following using equations 17 and 57 for e_{Ok}^2 and equations 51 and 52 for n_k:

$$E_{Ok} = \sqrt[2]{\frac{e_{Ok}^2}{n_k}} \quad \text{, hybrid input-output} \qquad (58)$$

Figure 3-8 shows a comparison of the RMS of E_{Ok} vs. the number of iteration for the error-reduction algorithm and for the combination of the error-reduction and hybrid input-output algorithms. The original object was a 16 by 16 input test matrix for the 40 iterations of the algorithm. Line 1 represents only the error-reduction algorithm. Line 2 represents an alternating strategy of 5 iterations of the error-reduction algorithm followed by 15 iterations of the hybrid input-output algorithm until 40 iterations are reached. Line 3 represents an alternating strategy of 2 iterations of the error-reduction algorithm followed by 18 iterations of the hybrid input-output algorithm until 40 iterations are reached. A β of 5 was chosen as the optimum β after some analysis. Line 1 is represented by circles; line 2, squares; and line 3 diamonds. Each test was given the same starting distorted object before the iterations began. For the error-reduction algorithm, the error levels off and begins to stagnate after only 2 iterations. It will take many more before the image will converge to the original object. In this case after 40 iterations, the image not only stagnates but is a long way from converging to the image or even from improving the quality of the image. For line 2, after the five iterations of the error-reduction algorithm, initially the error begins to decrease, stagnates at the tenth iteration, increases after the tenth iteration and then decreases sharply. It converges to the original image at iteration number 29. For line 2, the error sharply decreases at a faster rate than line 3 and then levels off. Line 3 converged to the original image in 35 iterations. For different objects, there are perhaps many optimum strategies that can be devised to retrieve the image in the fewest number of iterations. Line 2 and line 3 of figure 3-8 represent only two of thousands of possibilities.

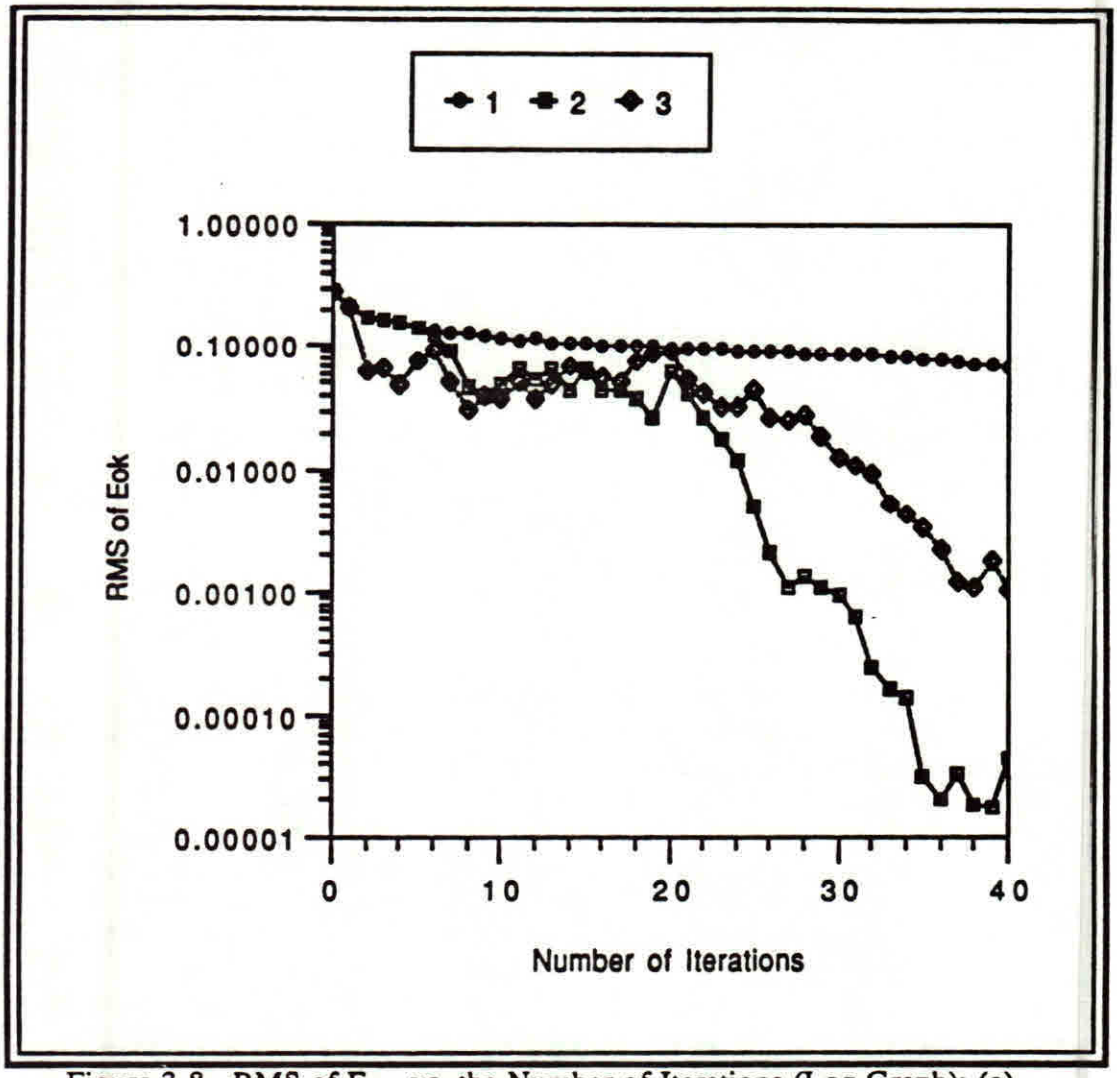

Figure 3-8. RMS of E_{Ok} vs. the Number of Iterations (Log Graph); (a) line 1 is the error reduction algorithm; (b) line 2 is 5 iteration of the error-reduction algorithm followed by 15 iterations of the hybrid input-output algorithm; (c) line 3 is 2 iteration of the error-reduction algorithm followed by 18 iterations of the hybrid input-output algorithm.

However, it is a good recommendation that the combining of the error-reduction and hybrid input-output algorithm is the best way to retrieve an image because the error-reduction algorithm prevents the hybrid input-output algorithm from increasing the error significantly in the original image. For the case of the hybrid input-output algorithm, the visual quality improves but E_{Ok} does not. However, if a few iterations of the error-reduction algorithm are combined with the hybrid input-output algorithm, the visual quality during the iterations of the error-reduction algorithm improves the quality of the distorted image very

little but causes E_{Ok} to decrease rapidly until it is consistent with the quality of the image. E_{Ok} in line 2 decreases sharply during the iterations 20 to 25 of the error-reduction algorithm even though in the previous 15 iterations the error increases and decreases during the iterations of the hybrid input-output algorithm. The same is true for line 3 at iterations 20 and 21, although it is less noticeable. Again alternating between the two algorithms prevents the error in the hybrid input-output algorithm from eventually increasing due to the value of β even after the distorted image has converged to the original object. As line 2 shows at iterations 36 to 40, the error begins to increase even after the image has converged to the original object. If the algorithm was allowed to continue the next five iterations of the error-reduction algorithm would sharply decrease the error in the image and would prevent the introduction of any added distortion. Clearly, the strategy in line 2 or 3 is far superior than only using the error-reduction algorithm.

This analysis reinforces the claims of Fienup and dispels those of Hayes for the error-reduction algorithm; therefore, an algorithm exists that reduces the error and distortion in a image when only the magnitude of the Fourier transform of the object is known and when the object is real and nonnegative.

3.3.5 Plot

The plot library is contained in module PLOT_LIB.PAS (see appendix F). It creates the patterns and displays the images on the screen of a PC computer with color graphics. The object or image is displayed on the screen of the computer in five colors which created the 36 grey scale levels. Light blue is for the value zero in the bit pattern; white is for the value 1; light grey is for the value 3; and black is for the value 4. Only the object plane of the matrix is displayed; therefore, the zero planes are not displayed because they are of no importance to the operator. Procedure display_pix plots one pixel on the screen of the computer. Procedure create_patterns creates all 36 grey scale patterns. Procedure display_image scales each data point of the matrix that is to be displayed. The possible values procedure display_image can scale to are 0 to 36. Procedure display_image calls procedure display_pix to map it to the screen of the computer. Procedure pause stops the program, gives instructions to the operator and determines what keys have been pressed. Procedure beep_op causes the computer to beep at the operator. Procedure dip_dis locates, sets the x-y axis for the image to be displayed and determines how many images to display on the screen.

The 36 grey scales can be modified to create 64 grayscale levels. However, it can be used with only the 16 by 16 and 64 by 64 matrices that have object planes of 8 by 8 or 32 by 32 respectively; otherwise, for a 128 by 128 input matrix, a 64 by 64 object plane with 64 grey scale levels will exceed the number of pixels on a conventional PC monitor. The implementation of 64 grey scale levels can be done for a 16 by 16 and 64 by 64 input matrix, if the operator enters module program SIMUL.PAS and changes the bit_size constant from 3 to 4. Then the operator must enter module OP_LIB.PAS and make changes to procedure x_domain_constraints. In procedure x_domain_contraints, the

maximum bound should be changes from 36 to 64. Finally, the appropriate changes should be made to the module PLOT_LIB.PAS.

SECTION 4

CONCLUSIONS AND RECOMMENDATIONS

There are many conclusions and recommendations to make concerning the final algorithm. First, there is an algorithm for the reconstruction of a multidimensional sequence or image from the estimated modulus or magnitude of its Fourier transform given that the original two-dimensional object is real and nonnegative. However, based upon its initial results, the error-reduction algorithm takes far too many iterations to converge. This document outlines an acceleration technique to be used with the error-reduction algorithm to retrieve the image. Alternating between the error-reduction and hybrid input-output algorithms provides the optimal phase retrieval technique for the application of objects that are real and nonnegative. Second, the use of these algorithms for two-dimensional image retrieval is of great importance in astronomy and pupil synthesis. As described in section 1, there have been many other successful applications of both algorithms to other problems but perhaps the greatest potential for these algorithms is in optical applications. Both the error-reduction and hybrid input-output algorithms are powerful algorithms which can retrieve the worst distorted images using information on the Fourier modulus. The error-reduction algorithm reduces the error in the distorted image with each iteration but only changes the quality of the distorted image slightly. The hybrid input-output algorithm improves the quality of the distorted image but can increase the error in the distorted image if the iterations continue for too long. Therefore, these two algorithms complement each other perfectly when implemented together. Appendix A provides examples of the ability of the final algorithm to retrieve distorted images.

However, there are still many problems to be addressed in the use of the final algorithm outlined in this document. First, if an estimate of the object is known, then it can be applied to retrieve a corrupted image. Monitoring the RMS in the application of the error-reduction algorithm can not only determine whether or not the image has converged or stagnated but can also provide information as to whether or not the estimate of the object represents the corrupted image. In addition, there should be a further investigation into the effects that a Fourier modulus has on the retrieval process if it is only an estimate of the original modulus. The sensitivity of the final algorithm is caused by the error-reduction and hybrid input-output algorithm acting as the driving force behind the retrieval process using the Fourier modulus. However, more work is need to investigate the sensitivity of the error-reduction and hybrid input-output algorithms to the estimate of the Fourier modulus of the original object as well as the sensitivity of the hybrid input-output algorithm to the β input. Although appendix B provides a mathematical interpretation of the hybrid input-output algorithm, more work is needed to determine the mathematical relationship of β to the final algorithm especially to the FFT analysis. At the present time, there does not appear to be any general convergence rules that could lead to a choice of β and the number of iterations for the hybrid input-output algorithm without the

subjective and apriori input from the operator before the final algorithm returns to iterations of the error-reduction algorithm.

Others have found some similar problems in the optical implementation and computer simulation of other error-reduction algorithms. First, Hayes points to some of the problems associated with using an estimate of the Fourier modulus in his discussion of his magnitude-only reconstruction algorithms.[17] He discovers it is not always possible to know the boundaries of the distorted image from just the Fourier modulus. However, Hayes discusses the application of the algorithms to the phase-retrieval problem for a discrete multidimensional sequence. He develops a similar iterative procedure as the error-reduction algorithm for the reconstruction of a signal from the modulus of its Fourier transform. The information necessary to use his recursion algorithm is the boundary values of the distorted image which help in determining an estimate of the Fourier modulus. However, Hayes is not always able to determine these boundary values from the Fourier modulus alone. Information in the X-domain as to the boundary values and in the U-domain as to an estimated Fourier modulus are needed. If the distored image has a region of support with a certain geometry, then it is possible to determine the boundary values.[20] However, this area needs more research. Second, Paxman and Fienup analyzed an optical implementation of an algorithm similar to the error-reduction algorithm based on the phase-diversity method of Professor Gonsalves.[21] The technique was used on a mirrored telescope that suffered from phase errors due to unaligned segments which had to be within a fraction of a wavelength. Paxman and Fienup analyzed this algorithms with the effects of added noise. These results are important because in the final algorithm the estimate of the Fourier modulus was noise free and therefore an exact estimate of the object. However, if there is added noise to the estimate, then the noise will effect the convergence of the image because the noisy estimate of the Fourier modulus is generally the driving force behind the algorithms. They encountered many problems in this optical implementation. It suffices to say that although the computer simulation of these algorithms worked successfully, more work is needed for an optical implementation.

Other applications of this algorithm exist as mentioned in section one. There are other applications of this particular algorithm than merely optical applications such as kinoform and radar applications. For example in radar applications, the final algorithm developed in this report can be used to bandlimit the sidelobe interference in the Fourier or U-domain of a radar signal or could be used to place the nulls of an antenna pattern at prescribed locations. Moreover, if a complex radar signal, $g(x) = |f(x)| \exp[i\varphi]$, is a pure phase function so that $|f(x)| = 1$, then $G(u)$ is a delta function in the Fourier or U-domain and therefore, its Fourier spectrum would be constant over the bandwidth of interest. In this way the algorithm could synthesize a radar signal.

There are many recommendations to make about this algorithm. First, persistence and patience in the pursuit of a working algorithm is necessary. Second, the analysis in section 3 was hindered by the memory contraints of 640 KB on a PC. A Sun or Apollo workstation with a larger memory and faster speed is definitely needed especially for the

FFT analysis. The Pascal code should be transferred to and re-programmed on a workstation in C++ to increase the speed and program ease. Third, a β that automatically changes itself in the algorithm is needed rather than a β that is manually entered. Although this was investigated, the work in this area should be addressed again. More error analysis and investigation is needed beyond that which has been outlined in appendix B to develop an automatically adjusting β. However, since image retrieval still remains very subjective for the basis of this algorithm, it still may be difficult to implement such an analysis as mentioned in section 2.3. Fourth, work needs to be done on the establishment of boundary conditions on distorted images as well as the optical implementation of the algorithms. Finally, an optical implementation is perhaps the only way of improving the processing speed of the algorithm and achieve real time analysis due to the intensive FFT calculations; otherwise, for anything above a 64 by 64 image a Pixar or workstation (even a Cray) is needed to perform the analysis to avoid days of calculations on an IBM PS/2 - 80 with a 25 MHz clock speed. However, with minor modifications to the final algorithm, it can be applied to ongoing projects here at MITRE such as the optical computer being developed by Dr. Marvin D. Drake in D-53. The fifth problem outlined in section one describes a problem in which the modulus of a complex function is given and one wants to find an associated phase function that results in a Fourier transform whose complex values are part of a prescribed set of quantized complex values. This application is especially useful for the reduction of quantized noise in computer generated holography, in coded signal transmissions and in optical computing. Therefore, the final algorithm can be used to determine the phase values for a particular picture stored in a spatial light modulator (SLM) used in the optical computer in D-53. Furthermore, it can be used to help focus and restore the phase values of incoming images that could be processed by an optical computer. Although this algorithm is deterministic in nature, it provides an accurate method of reconstructing or retrieving the phase of an image.

The area of phase reconstruction and phase retrieval deserves further investigation. Not only are the Gerchberg-Saxton, error-reduction, and input-output algorithms capable of one-dimensional and two-dimensional phase reconstruction and retrieval but also other approaches such as the steepest-descent method and gradient search methods can be investigated for applications to one-dimensional and two-dimensional phase reconstruction or retrieval. The author hopes to continue this area of research in the future using different phase reconstruction and retrieval methods as applied to radar and image processing applications as well as optical applications.

LIST OF REFERENCES

1. R. W. Gerchberg and W. O. Saxton, "A practical algorithm for the determination of phase from image and diffraction plane pictures," *Optik*, Vol. 35, No. 2, 1972, p.237-246.

2. W. O. Saxton, *Computer Techniques for Image Processing in Electron Microscopy*, NewYork: Academic, 1978.

3. R. A. Gonsalves, "Phase Retrieval from modulus data," *Journal of the Optical Society of America* , Vol. 66, No. 9, September 1976, p.961-964.

4. Adrain Walther, "The question of phase retrieval in optics," *Optic Acta*, Volume 10, 1963, p.41-49.

5. D. Kohler and L. Mandel, "Source reconstruction from the modulus of the correlation function: a practical approach to the phase problem of optical coherence theory," *Journal of the Optical Society of America*, Volume 63, No. 2, February 1973, p.126-134.

6. J. W. Goodman and A. M. Silvestri, "Some effects of Fourier-domain phase quantization," *IBM Journal of Research and Development*, Volume 14, No. 5, September 1970, p.478-484.

7. J. R. Fienup, "Phase retrieval algorithms: a comparison," *Applied Optics*, Volume 21, No. 15, August 1982, p.2758-2769.

8. J. R. Fienup, "Iterative method applied to image reconstruction and to computer-generated holograms," *Optical Engineering*, Volume 19, No. 3, May/June 1980, p.297-305.

9. J. R. Fienup, T. R. Crimmins, and W. Holsztynski, "Reconstruction of the support of an object from the support of its autocorrelation," *Journal of the Optical Society of America*, Volume 72, No. 5, May 1982, p.610-624.

10. J. R. Fienup, "Reconstruction of an object from the modulus of its Fourier transform," *Optics Letters*, Volume 3, No. 1, July 1978, p.27-29.

11. J. R. Feinup, "Space object imaging through the trubulent atmosphere," *Optical Engineering*, Volume 18, No.5, September-October 1979, p.529-534.

12. G. B. Feldkamp and J. R. Feinup, "Noise properties of images reconstructed from Fourier modulus," in *Proceedings of 1980 International Optical Computing Conference*, SPIE, Volume 231, 1980, p. 84.

13. J. R. Feinup and et. al., see *Advanced Institute on Transformations in Optical Signal Processing* (1981: Seattle, Washington), "Proceedings of the SPIE Advanced Institute on Transformations in Optical Signal Processing," Bellingham Washington, SPIE (International Society for), v.373.

14. J. R. Feinup, "Reconstruction and synthesis applications of an iterative algorithm," in *Transformations in Optical Signal Processing*, W. T. Rhodes, J. R. Fienup and B. E. A. Saleh, Eds., v.373, Society of Photo-Optical Instrumentation Engineers: Bellingham, Washington, 1983, p.147-160.

15. E. O. Brigham, *The Fast Fourier Transform*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1974, p.64.

16. N. C. Gallagher and B. Lui, "Convergence of a spectrum shaping algorithm," *Applied Optics*, Volume 13, No. 11, November 1974, p.2470-2471.

17. Monson H. Hayes, "The reconstruction of a multidimensional sequence from the phase or Magnitude of its Fourier transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, Volume ASSP-30, No. 2, April 1982, p.140-154.

18. J. R. Feinup, Comments on "The reconstruction of a multidimensional sequence from the phase or Magnitude of its Fourier transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, Volume ASSP-31, No. 3, June 1983, p.738-739.

19. James W. Cooley and John W. Tukey, "An algorithm for the machine calculation of a complex Fourier series," *Mathematics of Computations*, Volume 19, No. 90, April 1965, p.297-301.

20. Monson H. Hayes and Thomas F. Quatieri, "Recursive phase retrieval using boundary conditions," *Journal of the Optical Society of America*, Volume 73, No. 11, November 1983, p.1427-1433.

21. R. G. Paxman and J. R. Fienup, "Optical misalignment sensing and image reconstruction using phase diversity," *Journal of the Optical Society of America*, Volume 5, No. 6, June 1988, p.914-923.

22. Simon Haykin, *Communication Systems*, Second Edition, New York: John Wiley & Sons, 1983.

23. Alan V. Oppenheim and Ronald W. Schafer, *Digital Signal Processing*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1975.

APPENDIX A

EXAMPLES OF RETRIEVED IMAGES

This appendix provides three concrete examples of images that are retrieved and discusses some of the results in the application of this algorithm to single intensity phase retrieval using the Fourier modulus. Here a 32 by 32 pixel image which is real and nonnegative is placed in a 64 by 64 array in a similar fashion as described in section 3.3 and as shown in figure 3-2. Again in this appendix, the 32 by 32 pixel image is placed in a bed of zeros that is twice the size of the image. Each iteration of the algorithm for the 64 by 64 array with a 32 by 32 image takes approximately 40 seconds to process on an IBM PS/2 - 80 with a 16 MHz clock of which the forward and inverse Fourier transforms each take 15 seconds. Therefore, the two Fourier transforms take 30 seconds and the other error and algorithm analysis takes about 10 seconds.

There are two known applications of this algorithm in two dimensions. First, in pupil synthesis, it is possible to retrieve the pupil function of a diffraction limited optical system given a point spread function at a particular point in the image plane. The point spread function is the square of the Fourier modulus of the pupil function and the optical transfer function is the autocorrelation of the pupil function. Alternatively, an optical system could be designed to synthesize a pupil function that would produce a desired point spread function. Second, in astronomy, atmospheric turbulence causes a distortion in the resolution of the image but it is possible to measure the Fourier modulus of the image out to the diffraction limits of the telescope using interferometer data. Using the data on the modulus of the Fourier transform of the image, F(u) can be retrieved as well as the object. Furthermore, the autocorrelation function can be computed from F(u) allowing the diameter of the object to be determined.

The first image is a simple optical aperture consisting of a white rectangle inside a black rectangle which is in turn inside a gray rectangle. The image is totally distorted by a random phase. Figures A-1 to A-15 shows the image before and after entering the final algorithm. As in figure 3-1 in section 3.2, the original image is Fourier transformed, a random phase is added in the Fourier-domain and the image is then inverse Fourier transformed to get an initial estimate. The final algorithm continues until a solution is reached in the final iteration. In this case, the distortion prior to entering the iterative portion of algorithm was total. Notice that it took many extra iterations to retrieve the last few pixels of the image and that the original image was retrieved and not the mirrored image.

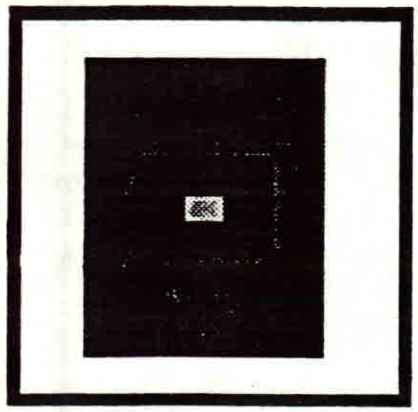

Figure A-1. The Original Image, f(x)

Figure A-2. The Fourier Transform of the Original Image, F(u)

Figure A-3. The Modulus of the Fourier Transform of the Original Image, |F(u)|

Figure A-4. The Modulus of the Image after the Random Phase is Added to |F(u)|

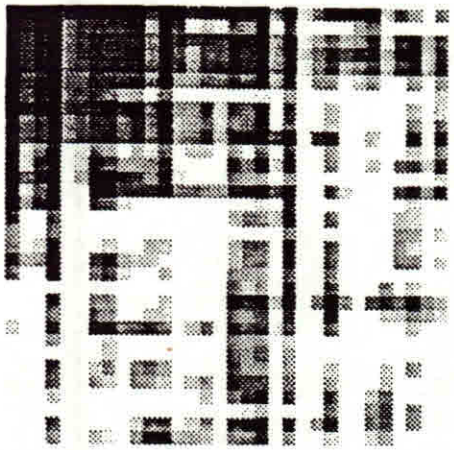

Figure A-5. The Initial Estimate of the Original Image

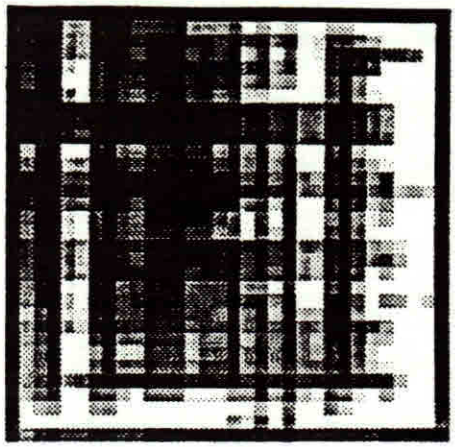

Figure A-6. The Retrieved Image after 5 Iterations

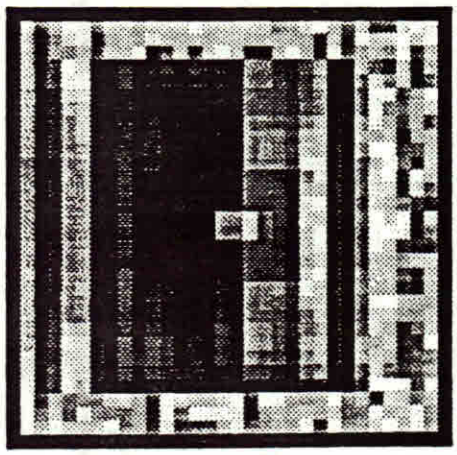

Figure A-7. The Retrieved Image after 20 Iterations

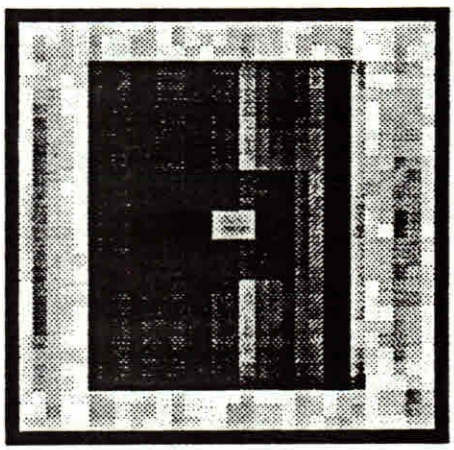

Figure A-8. The Retrieved Image after 40 Iterations

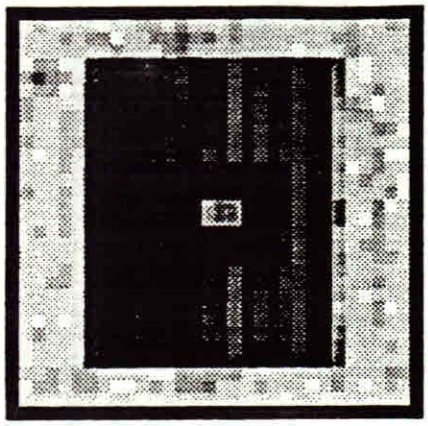

Figure A-9. The Retrieved Image after 100 Iterations

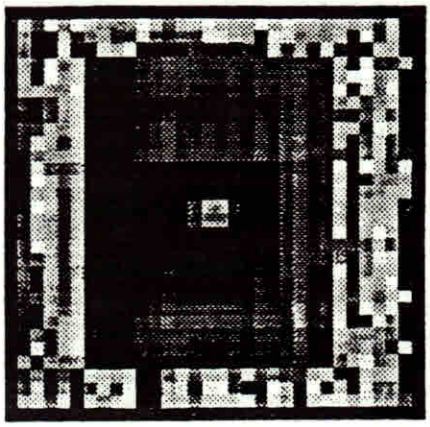

Figure A-10. The Retrieved Image after 226 Iterations

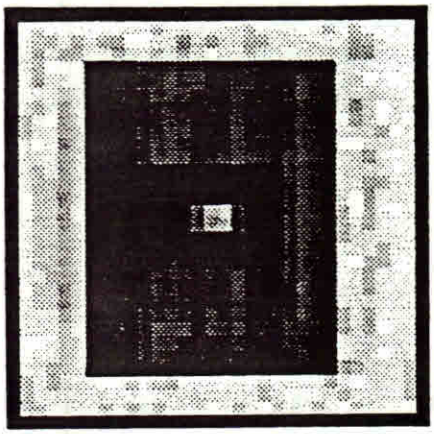

Figure A-11. The Retrieved Image after 250 Iterations

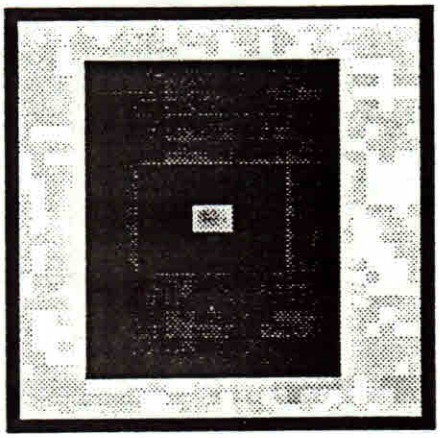

Figure A-12. The Retrieved Image after 500 Iterations

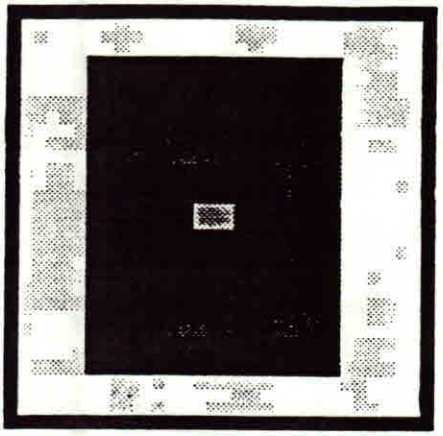

Figure A-13. The Retrieved Image after 689 Iterations

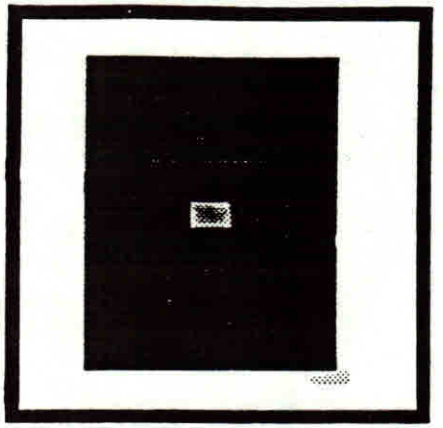

Figure A-14. The Retrieved Image after 773 Iterations

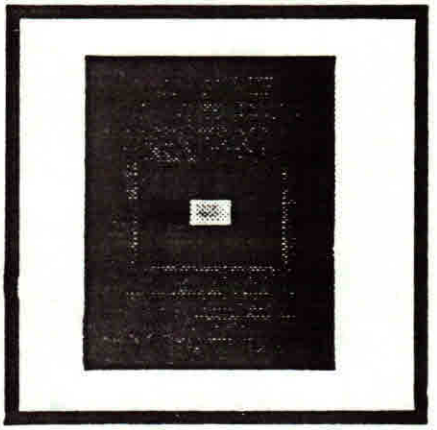

Figure A-15. The Retrieved Image after 895 Iterations (The Final Iteration)

The second image is the same optical aperture as the first image (shown in figure A-1) but this time the initial estimate is closer to the original image. The same Fourier transform and modulus of the Fourier transform from the first image are used as shown in figures 2 and 3 of this appendix respectively. Figures A-16 to A-22 show the image before and after entering the final algorithm. In this case, the final algorithm takes fewer iterations to retrieve the image because the initial estimate has much less distortion and was close to the original image. Moreover, it only took five iterations of the algorithm for

a fairly good quality image to appear as opposed to the 250 iterations of the algorithm it took for the first image to achieve about the same quality (shown in figure A-11). However, in the second image it still took many extra iterations to retrieve the last few pixels of the image.

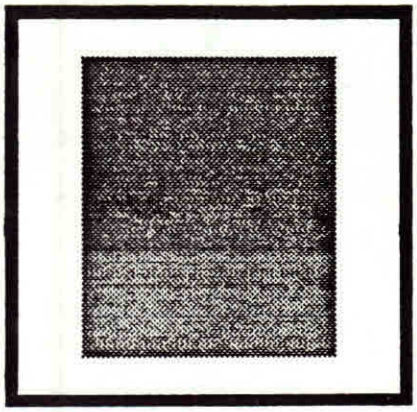

Figure A-16. The Initial Estimate of the Original Image

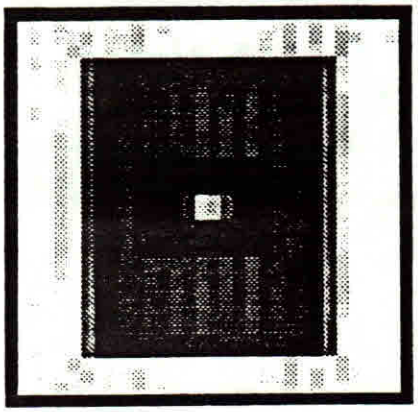

Figure A-17. The Retrieved Image after 5 Iterations

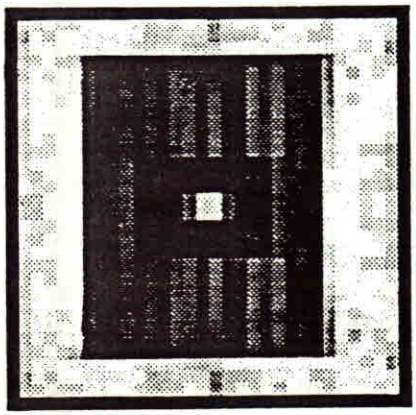

Figure A-18. The Retrieved Image after 20 Iterations

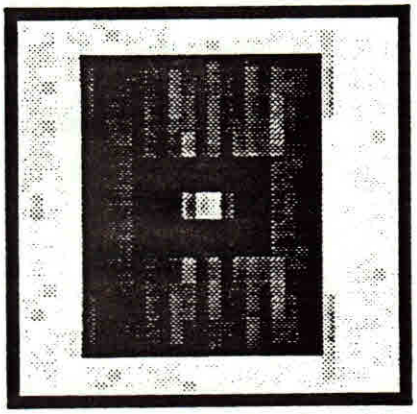

Figure A-19. The Retrieved Image after 100 iterations

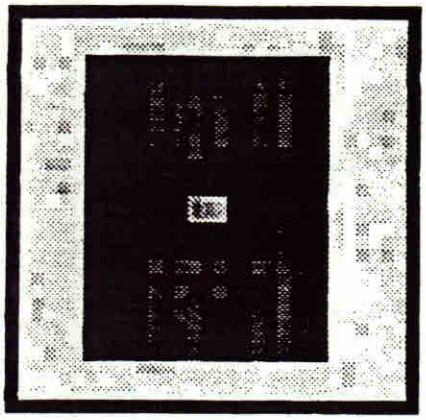

Figure A-20. The Retrieved Image after 250 Iterations

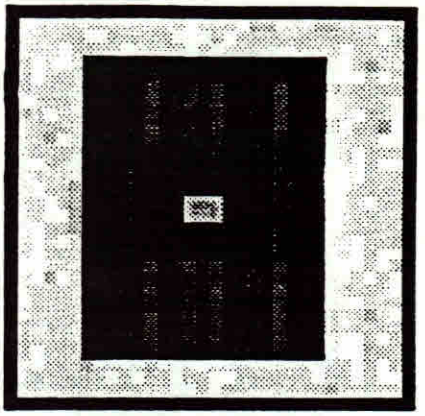

Figure A-21. The Retrieved Image after 358 iterations

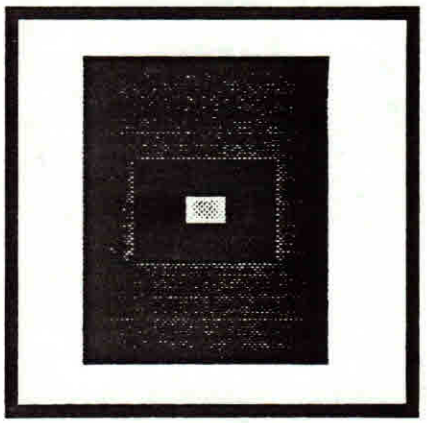

Figure A-22. The Retrieved Image after 523 Iterations (The Final Iteration)

The third image is a more complicated image, a fighter. Figures A-23 to A-35 show the image before and after entering the final algorithm. It took many more iterations to retrieve the fighter. In this case, the distortion prior to entering the iterative portion of the algorithm was total as in the first image. Figure A-26 is added to show that when the inverse Fourier transform of |F(u)| is taken, one does not get the original image. Notice that the mirrored image of the original image was retrieved.

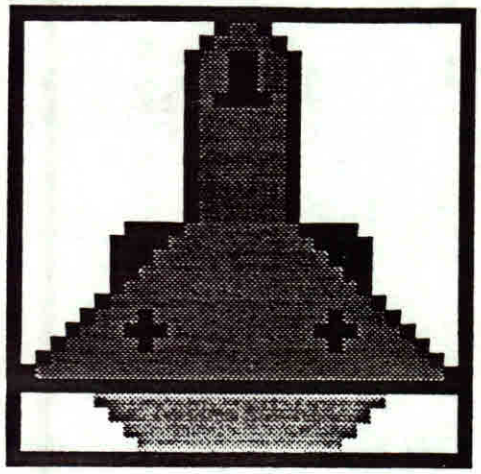

Figure A-23. The Original Image, f(x)

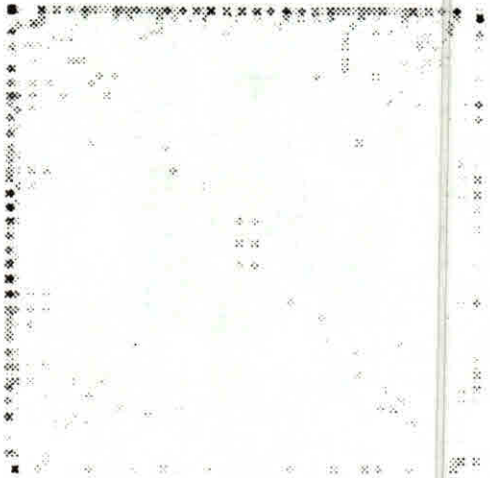

Figure A-24. The Fourier Transform of the Original Image, F(u)

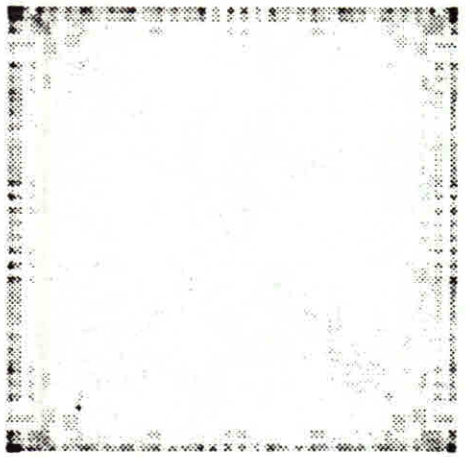

Figure A-25. The Modulus of the Fourier Transform of the Original Image, |F(u)|

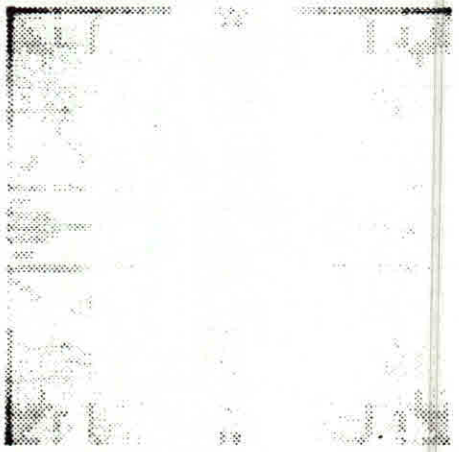

Figure A-26. The Inverse Fourier Transform of |F(u)|

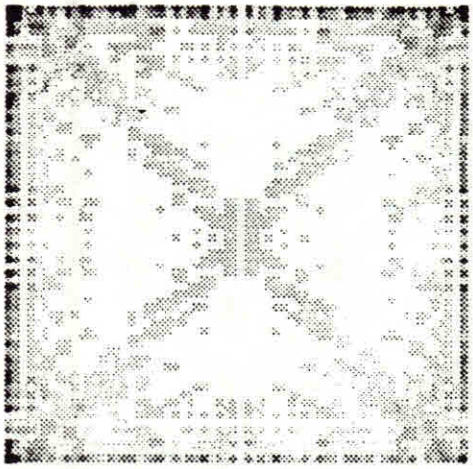

Figure A-27. The Modulus of the Image after the Random Phase is Added to |F(u)|

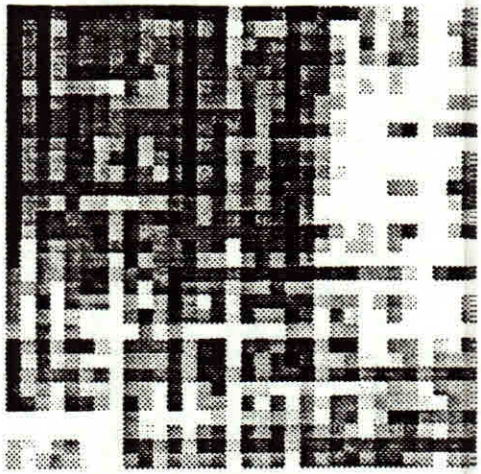

Figure A-28. The Initial Estimate of the Original Image

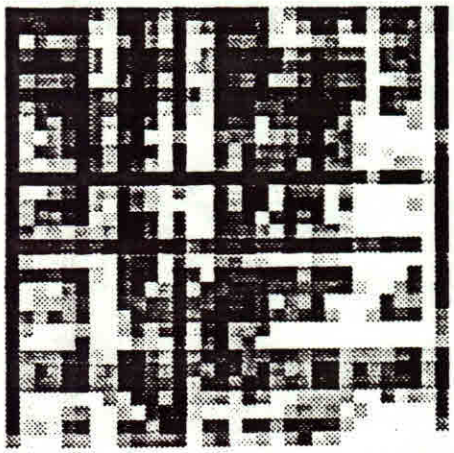

Figure A-29. The Retrieved Image after 5 Iterations

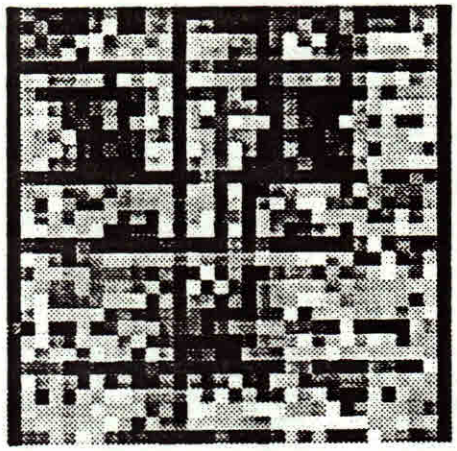

Figure A-30. The Retrieved Image after 40 Iterations

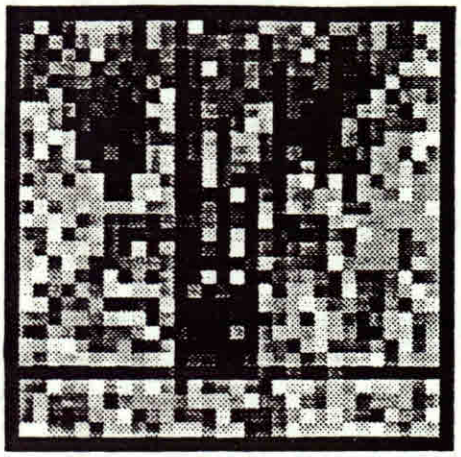

Figure A-31. The Retrieved Image after 100 Iterations

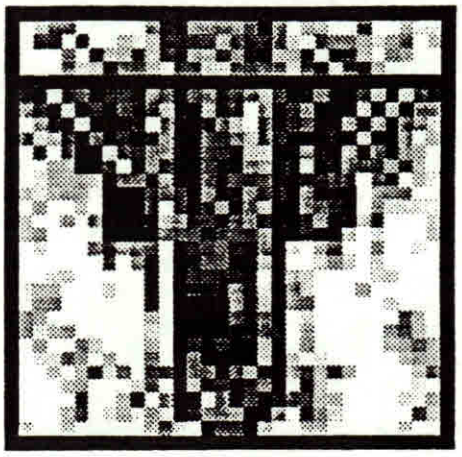

Figure A-32. The Retrieved Image after 288 Iterations

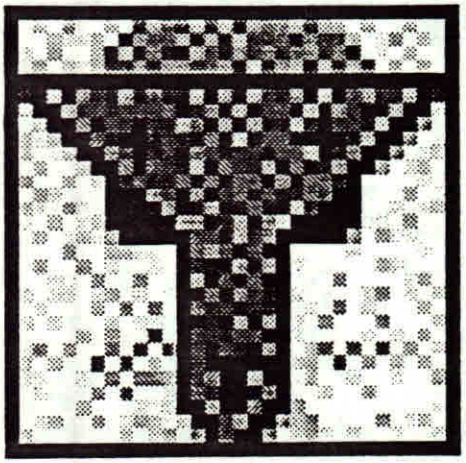

Figure A-33. The Retrieved Image after 656 Iterations

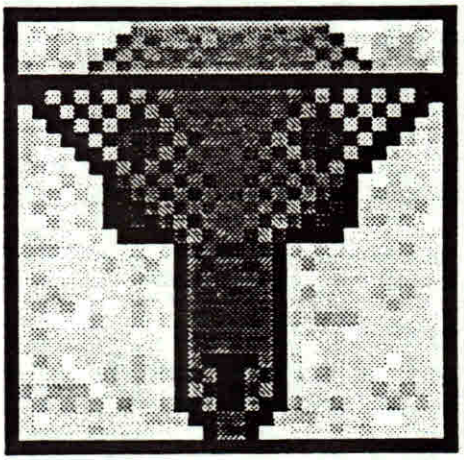

Figure A-34. The Retrieved Image after 1000 Iterations

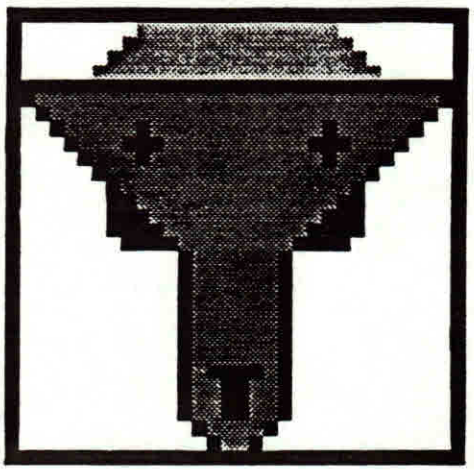

Figure A-35. The Retrieved Image after 1342 Iterations (The Final Iteration)

It is clear that if the initial estimate is close to the original image, it will take fewer iterations to retrieve the image. If there is less phase distortion, then the image will converge faster. However, it seems to take many extra iterations to completely retrieve the image in full detail even after the retrieved image is close to the original image. Finally, choosing the correct β for the hybrid input-output algorithm can clearly accelerate or delay the retrieval process. Other attempts at retrieving the same two images can add or subtract from the number of iterations given the same distortion or the same initial estimate if a different β is used during different iterations.

APPENDIX B

INTERPRETATION OF THE HYBRID INPUT-OUTPUT ALGORITHM

If the Fourier modulus of the image is a constant, then we can use the hybrid input-output algorithm to retrieve the image. The estimate of G(u) can be represented by the following:

$$G'(u) = K \exp[i\phi(u)] \tag{B-1}$$

where $\phi(u)$ is the phase of G(u):

$$G(u) = |G(u)| \exp[i\phi(u)] = \mathcal{FT}[g(x)] \tag{B-2}$$

and K is a constant, $K = |F(u)|$. The resulting image is g'(x):

$$g'(x) = \mathcal{FT}^{-1}[G'(u)] \tag{B-3}$$

When a change, $\Delta g(x)$, is made to the input, the change, $\Delta g(x)$, causes a change in the output, $\Delta G(u)$ which in turn causes a change in $\Delta G'(u)$ and a corresponding change in $\Delta g'(x)$ of the output image. This is shown in figures B-1 and B-2 using an analysis of a kinoform:

$$\Delta g'(x) = \mathcal{FT}^{-1}[\Delta G'(u)] \tag{B-4}$$

In order to justify the use of the hybrid input-output algorithm, a relationship between $\Delta g'(x)$ and $\Delta g(x)$ must be determined. Figure B-3 shows the relationship between $\Delta G'(u)$ and the two components that create $\Delta G(u)$. These components are orthogonal. Using similar triangles where $|\Delta G(u)| << |G(u)|$, the following relationships are true:

$$\Delta G'(u) \cong \Delta G^{s}(u) \frac{K}{|G_k(u)|} \tag{B-5}$$

$$\Delta G^{c}(u) = |\Delta G(u)| \cos\beta(u) \exp[i\phi(u)] \tag{B-6}$$

$$\Delta G^{s}(u) = |\Delta G(u)| \sin\beta(u) \exp[i(\phi(u) + \pi/2)] \tag{B-7}$$

Notice that ΔG^{s} is rotated 90° or $\pi/2$ as shown in figure B-3.

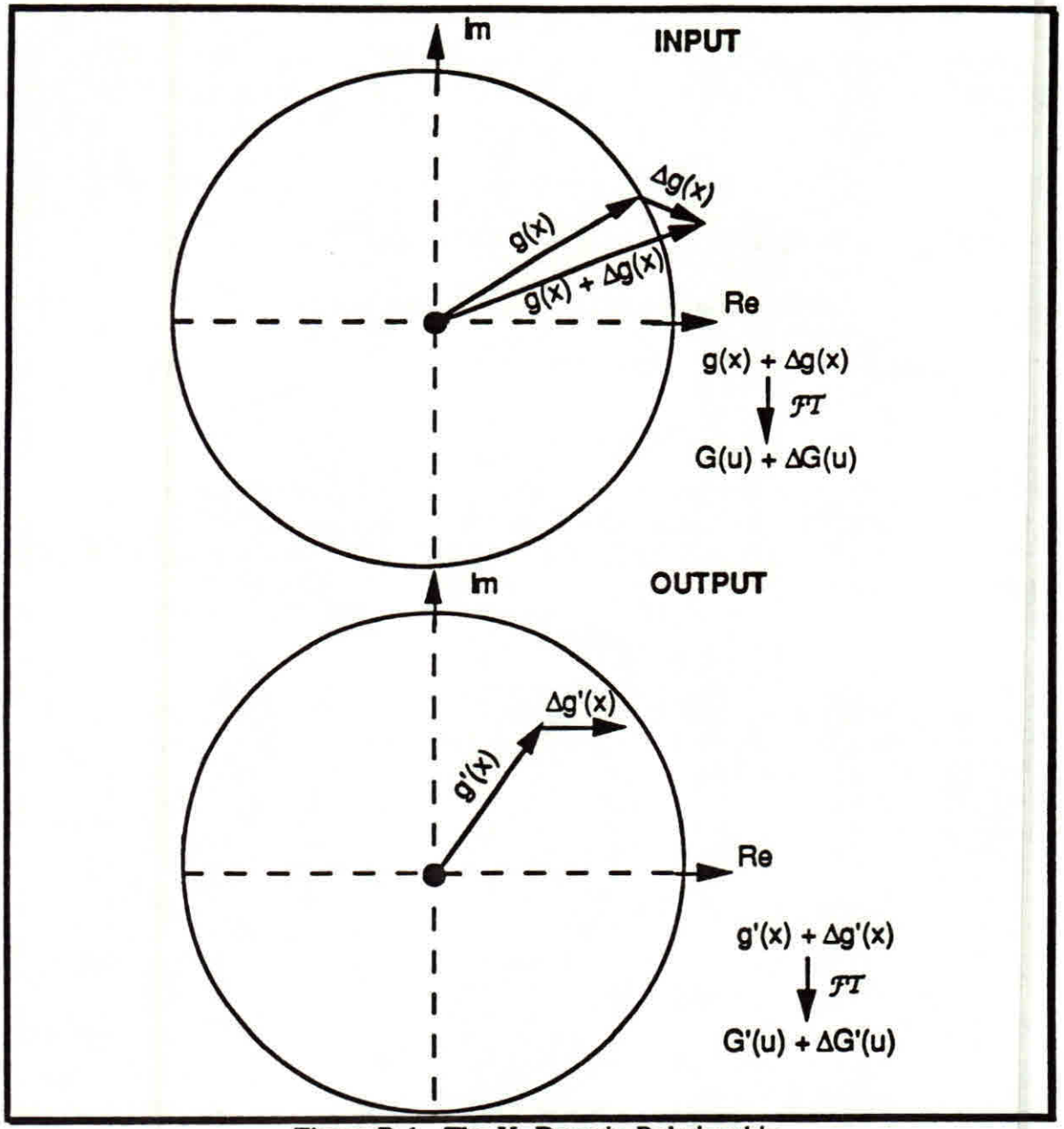

Figure B-1. The X- Domain Relationship

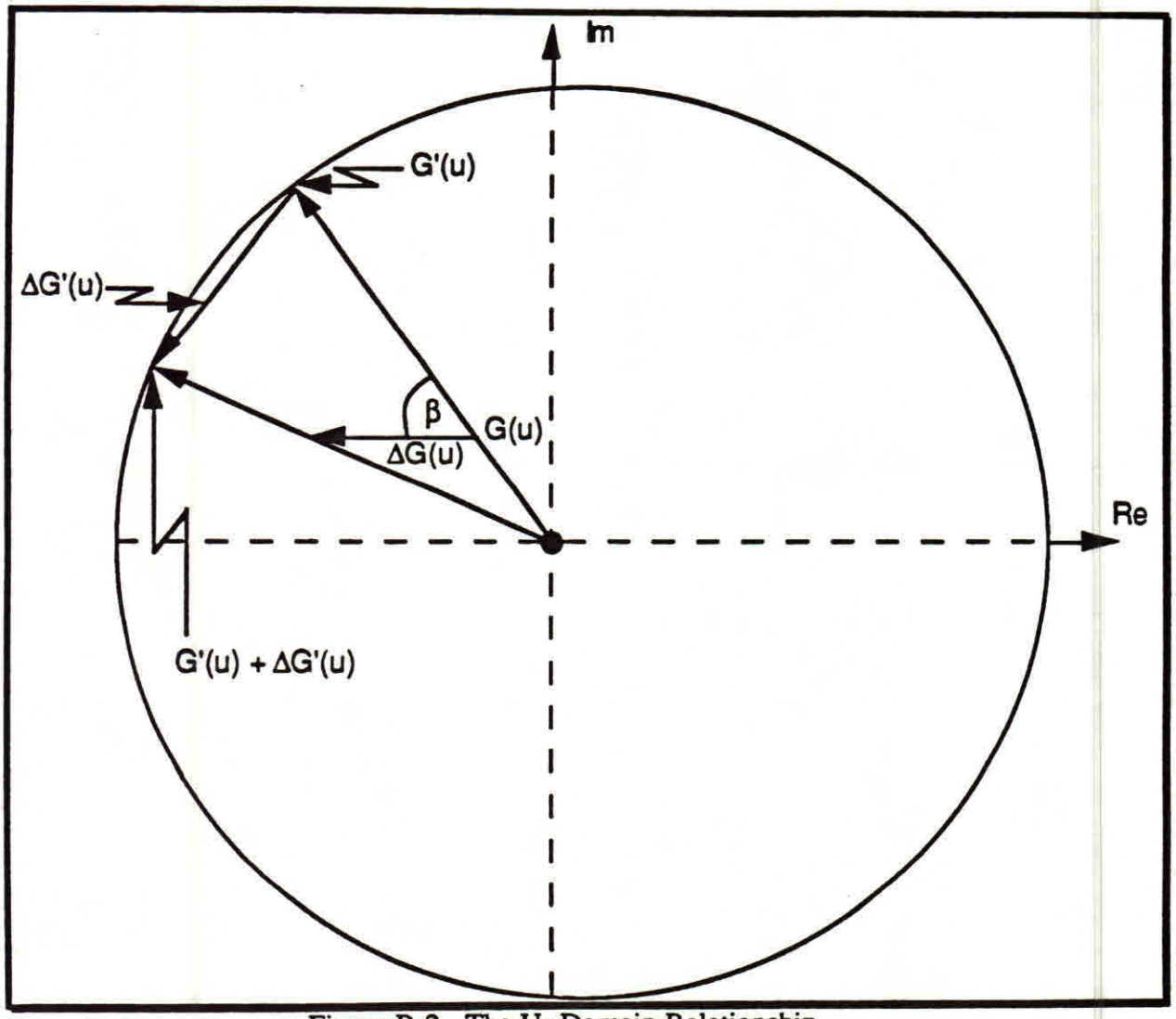

Figure B-2. The U- Domain Relationship

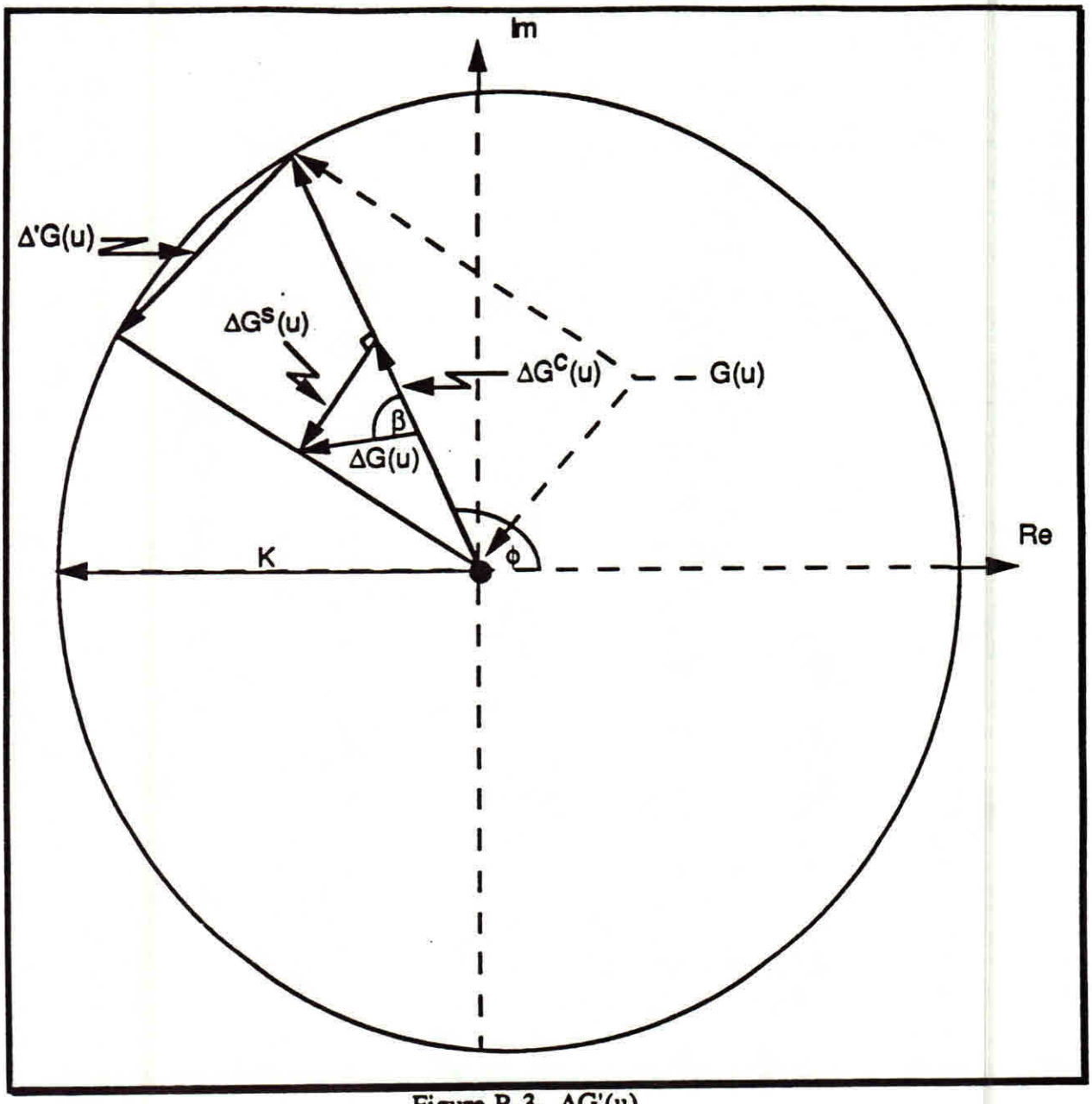

Figure B-3. ΔG'(u)

The two orthogonal components of ΔG(u) are parallel to G(u). Therefore, ΔG(u) is equalled to the following:

$$\Delta G(u) = \Delta G^{c}(u) + \Delta G^{s}(u) \qquad \text{(B-8)}$$

$$\Delta G(u) = |\Delta G(u)| \cos\beta(u) \exp[i\phi(u)] + |\Delta G(u)| \sin\beta(u) \exp[i(\phi(u)+ \pi/2)]$$

$$= |\Delta G(u)| [\cos\beta(u) (\cos\phi(u) + i \sin\phi(u)) + \sin\beta(u) (\cos\phi(u) + i \sin\phi(u)) (\cos\pi/2 + i \sin\pi/2)]$$

$$= |\Delta G(u)| [\cos\beta(u) (\cos\phi(u) + i \sin\phi(u)) + \sin\beta(u) (\cos\phi(u) + i \sin\phi(u))]$$

$$= |\Delta G(u)| (\cos\phi(u) + i \sin\phi(u)) (\cos\beta(u) + i \sin\beta(u))$$

$$\Delta G(u) = |\Delta G(u)| \exp[i(\phi(u) + \beta(u))] \quad \text{(B-9)}$$

$\beta(u)$ is the angle between $\Delta G(u)$ and $G(u)$. However, only $\Delta G^s(u)$ contributes to $\Delta G'(u)$.

The expected change in the output, $E[\Delta g'(x)]$, should be calculated. $E[\Delta g'(x)]$ is calculated by treating the phase angle $\beta(u)$ and the $|G(u)|$ as random variables.

$$\Delta G^s(u) = |\Delta G(u)| \sin\beta(u) \exp[i(\phi(u) + \pi/2)]$$

As previously derived in equation B-9,

$$\Delta G(u) = |\Delta G(u)| \exp[i(\phi(u) + \beta(u))]$$

Therefore,

$$|\Delta G(u)| = \Delta G(u) \exp[-i(\phi(u) + \beta(u))] \quad \text{(B-10)}$$

and

$$\Delta G^s(u) = \Delta G(u) \exp[-i(\phi(u) + \beta(u))] \sin\beta(u) \exp[i(\phi(u) + \pi/2)]$$

$$= \Delta G(u) \exp[-i\phi(u)] \exp[-i\beta(u)] \sin\beta(u) \exp[i\phi(u)] \exp[i\pi/2]$$

$$= \Delta G(u) \exp[-i\beta(u)] \sin\beta(u) \exp[i\pi/2]$$

$$= \Delta G(u) [\cos(-\beta(u)) + i \sin(-\beta(u))] \sin\beta(u) (\cos\pi/2 + i \sin\pi/2)$$

$$= \Delta G(u) [\sin\beta(u) \cos(-\beta(u)) - i \sin\beta(u) \sin\beta(u)] \text{ (i)}$$

$$\Delta G^s(u) = \Delta G(u)\ (\sin\beta(u)\cos\beta(u) - i\sin^2\beta(u))$$

$$= \Delta G(u)\ [\sin^2\beta(u) + i\sin\beta(u)\cos\beta(u)] \qquad \text{(B-11)}$$

If $\beta(u)$ is uniformly distributed over $[0, 2\pi]$, the expected value of $G^s(u)$ is the following:

$$E[\Delta G^s(u)] = E\{|\Delta G(u)|\ \sin\beta(u)\ \exp[i(\phi(u) + \pi/2)]\} \qquad \text{(B-12)}$$

where from equation B-7,

$$\Delta G^s(u) = |\Delta G(u)|\ \sin\beta(u)\ \exp[i(\phi(u) + \pi/2)]$$

Applying equations B-10 to B-7, the following is derived:

$$\Delta G^s(u) = \Delta G(u)\ \exp[-i(\phi(u) + \beta(u))]\ \sin\beta(u)\ \exp[i(\phi(u) + \pi/2)]$$

$$= \Delta G(u)\ \exp[-i\phi(u)]\ \exp[-i\beta(u)]\ \sin\beta(u)\ \exp[i\phi(u)]\ \exp[i\pi/2]$$

$$= \Delta G(u)\ \exp[-i\beta(u)]\ \sin\beta(u)\ \exp[i\pi/2]$$

Since $\sin\beta(u) = \dfrac{\exp[i\beta(u)] - \exp[-i\beta(u)]}{2i}$, then

$$\Delta G^s(u) = \Delta G(u)\ \exp[i\pi/2]\ \exp\ [-i\beta(u)]\ \frac{\exp[i\beta(u)] - \exp[-i\beta(u)]}{2i}$$

If $1/i = (1/i)(i/i) = -i = \exp\ [-i\pi/2]$, then

$$\Delta G^s(u) = \Delta G(u)\ \exp[i\pi/2]\ \exp\ [-i\beta(u)]\ \exp[-i\pi/2]\ \frac{\exp[i\beta(u)] - \exp[-i\beta(u)]}{2}$$

$$= 1/2\ \Delta G(u)\ (1 - \exp\ [-2i\beta(u)]) \qquad \text{(B-13)}$$

If a dummy variable is used for $\beta(u)$, since $\beta(u)$ is a random variable (see p.180 and pp.244-250 in reference 22), then

$$\beta(u) = \theta \quad \text{and} \quad \beta(u) = \begin{cases} 1 & \text{for } [0 \text{ to } 2\pi] \\ 0 & \text{otherwise} \end{cases}$$

where $d\beta(u) = d\theta$

$$E[\Delta G^s(u)] = E[1/2\ \Delta G(u)\ (1- \exp[-2i\theta])]$$

$$= 1/2\ \Delta G(u)\ E[1-\exp[-i2\theta]] \qquad \text{(B-14)}$$

The density function for $\beta(u)$ is the following for a uniform distribution:

$$\beta(u) = \begin{cases} \frac{1}{2\pi-0} \text{ for } [0 \text{ to } 2\pi] \\ 0 \quad \text{otherwise} \end{cases}$$

$$\beta(u) = \begin{cases} \frac{1}{2\pi} \text{ for } [0 \text{ to } 2\pi] \\ 0 \quad \text{otherwise} \end{cases} \qquad \text{(B-15)}$$

Therefore,

$$E[\Delta G^s(u)] = 1/2\ \Delta G(u) \frac{1}{2\pi} [\int_0^{2\pi} 1 d\theta - \int_0^{2\pi} \exp[-2i\theta]\ d\theta]$$

$$= 1/2\ \Delta G(u) \frac{1}{2\pi} [\theta\ |_0^{2\pi} - \frac{\exp[-2i\theta]}{d(-2i\theta)/d\theta}\ |_0^{2\pi}]$$

$$= 1/2\ \Delta G(u) \frac{1}{2\pi}\ [(2\pi - 0) - \frac{(\cos 2\pi - i \sin 2\pi) - (\cos 0 - i \sin 0)}{d(-2i\theta)/d\theta}]$$

$$= 1/2\ \Delta G(u) \frac{1}{2\pi}\ [(2\pi - 0)]$$

$$= 1/2\ \Delta G(u) \frac{2\pi}{2\pi}$$

$$\therefore \quad E[\Delta G^s(u)] = 1/2\ \Delta G(u) \qquad \text{(B-16)}$$

The expected value of the output, using equations B-5 and B-16 and the fact that $|G(u)|$ are identically distributed random variables which are independent of $\beta(u)$, is the following:

$$E[\Delta g'(x)] = E\{ \mathcal{FT}^{-1}[\Delta G'(u)] \} = \mathcal{FT}^{-1}\{E(\Delta G'(u)\}$$

$$\cong \mathcal{FT}^{-1}\{ E[\Delta G^s(u)] \; E[\frac{K}{|G(u)|}] \}$$

$$\cong \mathcal{FT}^{-1} \{ 1/2 \; \Delta G(u) \; E[\frac{K}{|G(u)|}] \}$$

where $E[\frac{K}{|G(u)|}]$ is a constant:

$$E[\Delta g'(x)] \cong \mathcal{FT}^{-1} [1/2 \; \Delta G(u) \; E[\frac{K}{|G(u)|}]$$

$$\therefore \quad E[\Delta g'(x)] \cong 1/2 \; \Delta g(x) \; E[\frac{K}{|G(u)|}] \qquad \text{(B-17)}$$

In other words, the expected change in the output is 1/2 E[K/|G(u)|] multiplied by the input. After a few iterations, |G(u)| will not differ greatly from K and 1/2 E[K/ |G(u)|] ≅ 1/2 E[1] ≅ 1/2.

Moreover, the variance can be shown to be derived as the following:

$$\text{Variance} = \sigma = E\{ (|\Delta g'(x)| - |E[\Delta g'(x)]|)^2 \}$$

$$= E\{ |\Delta g'(x)|^2 - 2 |\Delta g'(x)| \, |E[\Delta g'(x)]| - |E [\Delta g'(x)]|^2 \}$$

$$= E[|\Delta g'(x)|^2] - 2 |E[\Delta g'(x)]| \, |E[\Delta g'(x)]| - |E[\Delta g'(x)]|^2]$$

where $E[|\Delta g'(x)|] = |E[\Delta g'(x)]|$,

$$\text{Variance} = E [|\Delta g'(x)|^2] - 2 |E[\Delta g'(x)]|^2 - |E[\Delta g'(x)]^2|$$

$$= E [|\Delta g'(x)|^2] - |E [\Delta g'(x)]|^2 \qquad \text{(B-18)}$$

Using equation B-17,

$$E [\Delta g'(x)] = \mathcal{FT}^{-1}[1/2 \; \Delta G] \; E[\frac{K}{|G(u)|}]$$

and

$$E\,[|\Delta g'(x)|^2] = E\{\ \mathcal{FT}^{-1}[\ (\Delta G'(u))^2]\ \}$$

$$= \mathcal{FT}^{-1}\ \{\ E[(\Delta G'(u))^2]\ \}$$

$$= \mathcal{FT}^{-1}\{\ E[(\Delta G^s(u))^2\ \frac{K^2}{|G(u)|^2}\ ]\ \}$$

$$= \mathcal{FT}^{-1}\{\ E[(\Delta G^s(u))^2]\ E[\ \frac{K^2}{|G(u)|^2}\ ]\ \} \qquad \text{(B-19)}$$

Applying equation B-7,

$$E[(\Delta G^s(u))^2] = E\{|\Delta G(u)|^2\ \sin^2\beta(u)\ \exp[2i(\phi(u) + \pi/2)]\ \}$$

Using equation B-10,

$$E[(\Delta G^s(u))^2] = E\{\Delta G^2(u)\ \exp[-2i(\phi(u) + \beta(u))]\ [\ \frac{\exp[i\beta(u)] - \exp[-i\beta(u)]}{2i}\ ]^2$$
$$\exp[2i\phi(u)]\ \exp\ [i\pi]\ \}$$

$$= E\{\Delta G^2(u)\ \exp[-2i\beta(u)]\ \exp[i\pi]$$
$$[\ \frac{(\exp[2i\beta(u)] + \exp[-2i\beta(u)] - 2)}{-4}\ ]\ \}$$

$$= \Delta G^2(u)\ E\ \{(\cos\pi\ +\ i\ \sin\pi)\ \frac{1\ +\ \exp[-4\beta(u)]\ -\ 2\exp[-2i\beta(u)]}{-4}\}$$

$$= \Delta G^2(u)\ E\ \{(-1)\ (-1/4\ -\ \frac{\exp[-4i\beta(u)]}{4}\ +\ 2\frac{\exp[-2i\beta(u)]}{4})\ \}$$

$$= \Delta G^2(u)\ E\ \{-1/4\ +\ \frac{\exp[-4i\beta(u)]}{4}\ -\ \frac{\exp[-2i\beta(u)]}{2}\ \}$$

If a dummy variable is used for $\beta(u)$ and equation B-15 is used for $\beta(u)$ as a uniformly distributed variable, then

$$\beta(u) = \theta \quad \text{and} \quad \beta(u) = \begin{cases} \frac{1}{2\pi} & \text{for } [0 \text{ to } 2\pi] \\ 0 & \text{otherwise} \end{cases}$$

where $d\beta(u) = d\theta$. Therefore,

$$E[(\Delta G^s(u))^2] = \Delta G^2(u)\, E\left\{ 1/4 + \frac{\exp[-4i\theta]}{4} - \frac{\exp[-2i\theta]}{2} \right\}$$

$$= \Delta G^2(u) \frac{1}{2\pi} \left\{ \frac{1}{2\pi} \left[\int_0^{2\pi} 1/4\, d\theta + \int_0^{2\pi} \frac{\exp[-4i\theta]}{4} d\theta \right\} \right.$$

$$\left. - \int_0^{2\pi} \frac{\exp[-2i\theta]}{2} d\theta \right\}$$

$$= \Delta G^2(u) \frac{1}{2\pi} \left[\frac{\theta}{4} \Big|_0^{2\pi} + \frac{\exp[-4i\theta]}{4\, d(-4i\theta)/d\theta} \Big|_0^{2\pi} - \frac{\exp[-2i\theta]}{2\, d(-2i\theta)/d\theta} \Big|_0^{2\pi} \right]$$

$$= \Delta G^2(u) \frac{1}{2\pi} \left[\frac{2\pi}{4} \right]$$

$$= 1/4\, \Delta G^2(u) \qquad \text{(B-20)}$$

Using equation B-19 and B-20,

$$E[|\Delta g'(x)|^2] \equiv \mathcal{FT}^{-1} \left[E[(\Delta G^s(u)^2] \, E\left[\frac{K^2}{|G(u)|^2} \right] \right.$$

$$\equiv \mathcal{FT}^{-1} \left[1/4\, \Delta G^2(u)\, E\left[\frac{K^2}{|G(u)|^2} \right] \right.$$

where $E\left[\frac{K^2}{|G(u)|^2} \right]$ is a constant

$$E[|\Delta g'(x)|^2] \equiv E\left[\frac{K^2}{|G(u)|^2} \right] \mathcal{FT}^{-1}[1/4\, \Delta G^2(u)] \qquad \text{(B-21)}$$

Using equations B-18, B-19, B-20, and B-21,

$$\text{Variance} = \sigma = E[|\Delta g'(x)|^2] - |E\,[\Delta g'(x)]|^2$$

$$\equiv 1/4\; E\left[\frac{K^2}{|G(u)|^2} \right] \mathcal{FT}^{-1}[\Delta G^2\,(u)] - |\mathcal{FT}^{-1}[1/2\, \Delta G(u)]\, E\left[\frac{K}{|G(u)|} \right]|^2$$

$$\sigma \equiv 1/4\ E[\frac{K^2}{|G(u)|^2}] \int_{-\infty}^{\infty} \Delta G^2(u)du - 1/2\ E[\frac{K}{|G(u)|}]^2 \left|\int_{-\infty}^{\infty} \Delta G(u)\, du\right|^2$$

$$\equiv 1/4\ \{ E[\frac{K^2}{|G(u)|^2}] - 2\,E[\frac{K}{|G(u)|}]^2 \} \int_{-\infty}^{\infty} |\Delta G(u)|^2\, du$$

By Parseval's theorem,

$$\int_{-\infty}^{\infty} |\Delta G(u)|^2\, du = \int_{-\infty}^{\infty} |\Delta g(x)|^2\, dx$$

$$\therefore \quad \sigma \equiv 1/4\ \{E[\frac{K^2}{|G(u)|^2}] - 2\ E[\frac{K}{|G(u)|}]^2 \}\ \frac{1}{A} \int_{-\infty}^{\infty} |\Delta g(x)|^2\, dx \quad \text{(B–22)}$$

where A is the area of the image. In other words this equation states that the variance of the change of the output $\Delta g'(x)$ at any point x is proportional to the integral squared of the change in the input. Therefore as one makes larger changes in the input, the predictability and control of $\Delta g'(x)$ decreases. However, as shown in sections 2 and 3 of this document sometimes large changes in the input are needed to surge the necessary energy into the image even though the predictability and control over $\Delta g'(x)$ decreases.

The difference between the change in the output and the expected change in the output is given by equation B-17:

$$E[\Delta g'(x)] \equiv 1/2\ \Delta g(x)\ E[\frac{K}{|G(u)|}]$$

Using equations B-17 and B-22, we can see that small changes in the input result in similar changes in the output; therefore, the output can be manipulated to satisfy the constraints by changing the input and surging positive energy into the image.

APPENDIX C

PROGRAM SIMUL.PAS

```
{*************************************************************************}
{*************************************************************************}
{*                  P R O G R A M   S I M U L . P A S                    *}
{*************************************************************************}
{*************************************************************************}

{*************************************************************************}
{*            Author        : James J. Hosker                            *}
{*            Created       : June 12, 1989                              *}
{*            Version       : 1.0                                        *}
{*            Compiler      : Borland TurboPascal Version 5.0            *}
{*************************************************************************}

{*    The purpose of this program is to integrate a simulation the
   Gerchberg-Saxton, error-reduction, and hybrid input-output
   algorithms into one final algorithm.  This algorithm is able to
   retrieve distorted images using the Fourier modulus of the original
   object and the fact that the original object is real and nonnegative.
   For a further explanation of the algorithm, see my paper
   'Investigation into the Gerchberg-Saxton and Error-Reduction
   Algorithms'.  This program and the paper were submitted as credit for
   the completion of a Master's Degree from Tufts University in Electrical
   Engineering.

      There are four modules that are need to compile the program:

            1. Module 'FFT_LIB.PAS' which contains the implementation of
               the Fast Fourier Transform.

            2. Module 'OP_LIB.PAS' which contains all the operations
               involved in the retrieval process.

            3. Module 'PLOT_LIB.PAS' which contains all the plotting
               procedures needed to plot the images on the screen of the
               operator.

   This program reads the original image from 'INPUT.DAT' and during the
   retrieval process saves certain iterations to the hard disk.  For more
   information on how program simulation works see my paper on the retrieval
   process.  There is also a 'NPLOT.PAS' program that plots the images
   that are saved to the hard disk.  *}


{*************************************************************************}
{* MAIN PROGRAM                                                          *}
{*************************************************************************}
program simul(input,output);

{*************************************************************************}
{* Set Memory Requirements and Stack Size for Program SIMUL              *}
```

```
{*************************************************************************}
{$M 40000,0,600000}

{*************************************************************************}
{* Call in Internal Turbo Pascal Units                                   *}
{* Unit DOS supports DOS functions and program execution.                *}
{* Unit CRT supports screen mode controls, windows, sound and color      *}
{*      for the plotting procedures in the program.                      *}
{* Unit GRAPH supports CGA, EGA and VGA graphics for the plotting        *}
{*      procedures in the program.                                       *}
{*************************************************************************}
uses dos,crt,graph;

{*************************************************************************}
{* CONSTANTS and TYPE and VARIABLE DECLARATION                           *}
{*************************************************************************}
const

  max_samples = 64;
  sample_block = 16;
  blocks = 4;
  max_error = 0.18;
  sqr_sample_block = sample_block * sample_block;
  sqr_max_samples = max_samples * max_samples;
  rows = ((sqr_max_samples div blocks) div max_samples);
  nc = ((max_samples div 2) + 1);

  bit_size = 3;
  sqr_bit_size = bit_size * bit_size;
  max_pats = (sqr_bit_size * 4);

type

  fft_type =
    record
      real_pt : real;
      image_pt : real;
    end;

  fft_array = array[1..max_samples,1..rows] of fft_type;

  fft_ptr = ^fft_array;

  fftp = array[1..blocks] of fft_ptr;

  real_array = array[1..max_samples] of real;
  image_array = array [1..max_samples] of real;

  pattern_rec =
    record
      x1 : byte;
      x2 : byte;
      x3 : byte;
      x4 : byte;
      x5 : byte;
      x6 : byte;
      x7 : byte;
```

```
      x8 : byte;
      x9 : byte;
      x10 : byte;
      x11 : byte;
      x12 : byte;
      x13 : byte;
      x14 : byte;
      x15 : byte;
      x16 : byte;
    end;

  pat = array[0..max_pats] of pattern_rec;

  cos_sin_array = array [-nc..nc] of real;

var
    s,save_block : longint;
    save_key,no_store,yes_store,for_ward,back_ward : boolean;
    error_data,prev_data,sav_ptr,save_mod,data,mag : fftp;
    find_j,find_i : integer;
    error_block,prev_block,sav_block,fft_block : fft_ptr;
    final_iteration,new_iteration : string;
    cnt,wwx,new_xx,new_yy,iteration,new_x,new_y : integer;
    neww,newz,errorcode,graphdriver,graphmode : integer;
    stagnated,no_delete,continue,quit : boolean;
    inp,error_file : text;
    nx,ny : longint;
    cs,sn : cos_sin_array;
    al_name,gbl,fftname : string;
    bit_pat : pat;
    fftr_block : real_array;
    ffti_block : image_array;
    stop_it,dip_all : boolean;
    add_inc,of1,of2,y_count,dip_count : integer;
    ngp,eok,sneo,gp,beta,nmm : real;
    save_in_out,increment,a1x,a1j,a1i : integer;


{*************************************************************************}
{* Call in External Modules                                              *}
{*************************************************************************}
{$I fft_lib.pas}
{$I plot_lib.pas}
{$I op_lib.pas}

begin

  {* Main of Program Simul *}

  s := sizeof(fft_array);

  for_ward := false;
  back_ward := true;

  no_store := false;
  yes_store := true;
```

```
dip_all := true;

create_patterns;

clrscr;
continue := true;
quit := false;
no_delete := false;
new_iteration := '0';
graphdriver := 0;
graphmode := 0;
detectgraph(graphdriver,graphmode);
initgraph(graphdriver,graphmode,'c:\tp');
errorcode := graphresult;
if errorcode <> grOk then
  begin
    continue := false;
    assign(error_file,'error.dat');
    rewrite(error_file);
    writeln(error_file,errorcode);
    close(error_file);
  end;

if(continue = true)then
  begin

    make_sin_cos_tbl;

    nx := getmaxx div 2;
    ny := getmaxy;
    settextstyle(defaultfont,horizdir,1);
    setfillstyle(solidfill,blue);
    setbkcolor(blue);
    get_input(data);

    {* display f(x) *}
    display_image(data,(getmaxx div 2),(getmaxy div 2));
    new_x := nx - 88;
    new_y := ny - 24;
    outtextxy(new_x,new_y,'This is a plot of f(x).');
    pause;

    if (quit = false)then
      begin

        fftname := 'original.dat';
        save_info(data,fftname,new_iteration);

        setcolor(blue);
        bar(1,1,getmaxx,getmaxy);

        {* take 2-d FFT of f(x) *}
        two_d_fft(data,for_ward);

        {* take the magnitude of |F(u)| = | FFT[f(x)] | *}
        magnitude(data,mag,yes_store);
```

```
    display_image(data,(getmaxx div 2),(getmaxy div 2));
    new_x := nx - 156;
    new_y := ny - 24;
    outtextxy(new_x,new_y,'This is a plot of the magnitude of F(u).');
    pause;

  end; {* if quit = false *}


if (quit = false)then
  begin

    setcolor(blue);
    bar(1,1,getmaxx,getmaxy);

    save_mod_file(data);

    {* add random phase by calling r_phase *}
    random_phase(data);
    {* plot the magnitude of R(u) = |F(u)| times the random phase *}
    magnitude(data,mag,no_store);
    display_image(mag,(getmaxx div 2),(getmaxy div 2));
    new_x := nx - 136;
    new_y := ny - 24;
    outtextxy(new_x,new_y,'This is a plot of R(u) which equals');
    new_x := nx - 116;
    new_y := ny - 16;
    outtextxy(new_x,new_y,'|F(u)| times the random phase.');
    pause;

    dispose_pointers(mag);

  end; {* if quit = false *}

setcolor(black);
bar(1,1,getmaxx,getmaxy);

iteration := 1;
dip_all := false;
dip_count := 0;
y_count := 1;
increment := 5;
save_in_out := 1;
al_name := 'GS=';


if(max_samples = 16)then
  begin
    if(bit_size = 3)then
      begin
        new_x := 32;
        new_y := 32;
        new_xx := new_x - 16;
        new_yy := new_y + 18;
      end
    else if(bit_size = 4)then
      begin
```

```
        new_x := 60;
        new_y := 60;
        new_xx := new_x - 20;
        new_yy := new_y + 22;
      end;
  end
else if(max_samples = 64)then
  begin
    if(bit_size = 3)then
      begin
        new_x := 74;
        new_y := 84;
        new_xx := new_x - 64;
        new_yy := new_y + 66;
      end
    else if(bit_size = 4)then
      begin
        new_x := 90;
        new_y := 90;
        new_xx := new_x - 80;
        new_yy := new_y + 81;
      end;
  end;

gotoxy(1,1);

if(quit = false)then
  begin
    assign(inp,'original.dat');
    reset(inp);
    readln(inp,gbl);
    readln(inp,gbl);

    for alx := 1 to blocks do
      begin
        getmem(sav_block,s);
        for alj := 1 to rows do
          begin
            for ali := 1 to max_samples do
              begin
                readln(inp,nmm);
                with sav_block^[ali,alj] do
                  begin
                    real_pt := nmm;
                    image_pt := 0;
                  end;
              end;
          end;
        sav_ptr[alx] := sav_block;
      end;
    close(inp);

    {* Display the original image in the left conor of the screen as
       GS=0.  This image will not be overwritten.  This way the
       operator can remember what the original looks like. *}
    final_iteration := 'GS=0.';
    display_image(sav_ptr,new_x,new_y);
```

```
    outtextxy(new_xx,new_yy,final_iteration);
    dispose_pointers(sav_ptr);
    setcolor(black);
    bar(1,getmaxy - 8,getmaxx,getmaxy);
    setcolor(yellow);
    outtextxy(((getmaxx div 2) - 96),(getmaxy - 8),'Algorithm is now running.');
  end; {* if quit = false *}

stagnated := false;
cnt := 0;
while ((iteration <= 10000) and (quit = false))do
  begin

    {* take 2-d inverse FFT of R(u) to get a(x) *}
    two_d_fft(data,back_ward);

    {* save present data for mse analysis *}
    save_error_data(data);

    {* put in x-domain contraints *}
    x_domain_contraints(data);

    if(iteration <> 1) then calculate_error(data);

    if(iteration = increment)then
      begin

        increment := increment + 5;
        new_iteration := '';
        str(iteration,new_iteration);

        fftname := 'NUL';
        if(iteration = 5)then fftname := 'd5.dat'
        else if(iteration = 10)then fftname := 'd10.dat'
        else if(iteration = 20)then fftname := 'd20.dat'
        else if(iteration = 40)then fftname := 'd40.dat'
        else if(iteration = 60)then fftname := 'd60.dat'
        else if(iteration = 80)then fftname := 'd80.dat'
        else if(iteration = 100)then fftname := 'd100.dat'
        else if(iteration = 250)then fftname := 'd250.dat'
        else if(iteration = 500)then fftname := 'd500.dat'
        else if(iteration = 1000)then fftname := 'd1000.dat'
        else if(iteration = 1500)then fftname := 'd1500.dat'
        else if(iteration = 2000)then fftname := 'd2000.dat'
        else if(iteration = 3000)then fftname := 'd3000.dat'
        else if(iteration = 4000)then fftname := 'd4000.dat'
        else if(iteration = 5000)then fftname := 'd5000.dat'
        else if(iteration = 7500)then fftname := 'd7500.dat'
        else if(iteration = 10000)then fftname := 'd10000.dat';

        if(fftname <> 'NUL')then save_info(data,fftname,new_iteration);

        stop_it := false;
        dip_dis;

      end; {* if iteration *}
```

```
        if((keypressed = true) or (stagnated = true))then
          begin
            stop_it := true;
            if(stagnated = true)then
              begin
                setcolor(blue);
                bar(1,getmaxy - 8,getmaxx,getmaxy);
                setcolor(yellow);
                outtextxy((getmaxx div 2 - 114),(getmaxy - 16),'Image has stagnated or
converged.');
                stagnated := false;
              end;
            dip_dis;
            setcolor(blue);
            bar(1,getmaxy - 8,getmaxx,getmaxy);
            setcolor(yellow);
            outtextxy((getmaxx div 2 - 60),(getmaxy - 8),'Enter new data.');
            if(quit = false)then
              begin
                setcolor(yellow);
                wwx := getmaxx div 2;
                outtextxy(1,1,'BETA is now : ');
                gotoxy(14,1);
                readln(beta);
                setcolor(blue);
                bar(1,1,getmaxx,15);
                if(beta <> 1)then
                  begin
                    setcolor(yellow);
                    outtextxy(1,1,'Iterations = ');
                    gotoxy(13,1);
                    readln(add_inc);
                    setcolor(blue);
                    bar(1,1,getmaxx,15);
                    setcolor(yellow);
                    save_in_out := iteration + add_inc;
                    al_name := 'IO=';
                  end {* if beta <> 1 *}
                else al_name := 'GS=';
                setcolor(blue);
                bar(1,getmaxy - 16,getmaxx,getmaxy);
                setcolor(blue);
                bar(1,getmaxy - 8,getmaxx,getmaxy);
                setcolor(yellow);
                outtextxy((wwx - 108),(getmaxy - 8),'Algorithm is now continuing.');
              end; {* if quit = false *}
            if(save_key = true)then
              begin
                fftname := 'd';
                fftname := concat(fftname,new_iteration);
                fftname := concat(fftname,'.dat');
                save_info(data,fftname,new_iteration);
              end;
          end; {* if keypressed *}

        if(iteration >= save_in_out)then al_name := 'GS=';
```

```
        if((quit = false) and (iteration <> 10000))then
          begin

            save_prev_data(data);

            {* take FFT of of a(x) with x-domain constaints to get to U-domain *}
            two_d_fft(data,for_ward);

            {* calculate theta correct modulus to impose on A(u) *}
            theta_mod(data);

            {* multiply A(u) by the theta modulus to get a new A(u) *}

            no_delete := true;
            multiply_2_d(data,save_mod);
            no_delete := false;

          end; {* if quit = false *}

        if(quit = false)then inc(iteration);

      end; {* while *}

    closegraph;

    fftname := 'lastit.dat';
    save_info(data,fftname,new_iteration);

  end; {* if continue = true *}

clrscr;

end. {* End Program SIMUL *}

{* END PROGRAM SIMUL *}
{*************************************************************************}
```

APPENDIX D

MODULE FFT_LIB.PAS

```
{*************************************************************************}
{*************************************************************************}
{*              M O D U L E   F F T _ L I B . P A S                      *}
{*************************************************************************}
{*************************************************************************}

{*************************************************************************}
{*          Author        : James J. Hosker                              *}
{*          Created       : June 12, 1989                                *}
{*          Version       : 1.0                                          *}
{*          Compiler      : Borland TurboPascal Version 5.0              *}
{*************************************************************************}

{* This module is part of program simulation which was submitted to fulfull
   the requirements of a Master's Degree in Electrical Engineering at
   Tufts University.  For more information see the main program module
   'SIMUL.PAS' and my paper that was submitted along with the
   program. *}

{*************************************************************************}
{*              F U N C T I O N   P O W E R _ D O W N                    *}
{*************************************************************************}
{* This function has two arguments passed to it, 'base' and 'number'.  It
   calculates the power or exponent value of a 'base' number sent to it
   given it is a power of a certain 'number'.  This is used in many ways
   but one way is to caluculate the number of stages for the FFT. *}
function power_down(var base : integer; number : longint) : integer;
  var
    temp_entries : longint;
    exp_val : integer;
    i : integer;

  begin
    temp_entries := number;
    exp_val := 0;
    REPEAT
      begin
       temp_entries := temp_entries div base;
       exp_val := exp_val + 1;
      end;
    UNTIL temp_entries = 1;

    power_down := exp_val;

  end; {* function power_down *}


{*************************************************************************}
{*       P R O C E D U R E   F I N D _ T R A N S _ E N T R Y            *}
{*************************************************************************}
{* This procedure finds the transpose entry within the memory implementation
```

```
   that was outline in my paper.  The procedure is sent the two value ti
   and tj which represent the row and column entry in the input matrix and
   it finds the entry within the memory structure outlined in my paper. *)
procedure find_trans_entry(var ti,tj : integer);
var
  min_i,min_j,max_j,x : integer;
  end_it : boolean;

begin

  x := 1;
  min_j := 1;
  max_j := rows;
  end_it := false;

  while((x <= blocks) and (end_it = false))do
    begin
      if((tj <= max_j) and (tj >= min_j))then
        begin
          end_it := true;
          find_j := tj - min_j + 1;
          find_i := ti;
          save_block := x;
        end
      else
        begin
          min_j := min_j + rows;
          max_j := max_j + rows;
          inc(x);
        end;
    end; {* while *}

end; {* procedure find_trans_entry *}


{*************************************************************************}
{*                P R O C E D U R E   T R A N S P O S E                  *}
{*************************************************************************}
{* This procedure transposes a matrix of type fftp that is sent to it using
   the procedure find_trans_entry. *}
procedure transpose(var trans_array : fftp);
var
  i,j,x : integer;
  save_i,save_j : integer;
  new_block : fft_ptr;
  temp_real,temp_image : real;

begin

  for i := 1 to max_samples do
    begin
      for j := i to max_samples do
        begin

          find_trans_entry(i,j);
          new_block := trans_array[save_block];
          save_i := find_i;
```

```
            save_j := find_j;
            temp_real := new_block^[save_i,save_j].real_pt;
            temp_image := new_block^[save_i,save_j].image_pt;

            find_trans_entry(j,i);
            fft_block := trans_array[save_block];
            new_block^[save_i,save_j].real_pt := fft_block^[find_i,find_j].real_pt;
            new_block^[save_i,save_j].image_pt := fft_block^[find_i,find_j].image_pt;
            fft_block^[find_i,find_j].real_pt := temp_real;
            fft_block^[find_i,find_j].image_pt := temp_image;

        end; {* for i *}
    end; {* for j  *}

end; {* procedure transpose *}


{*************************************************************************}
{*                    P R O C E D U R E   N F F T                        *}
{*************************************************************************}
{* This procedure calculates the FFT of a row of a two dimensional matrix.
   It is sent one argument inverse which tells the procedure if it is taking
   the forward or inverse Fourier transform.  If inverse is true, then it
   take the inverse FFT.  If inverse is false then it take the forward FFT.
   Certain section of this procedure have been commented out because they are
   not used in a two dimensional implementation as opposed to the
   one-dimensional implementation. Two data structures hold the rows that the
   FFT is to be taken - fftr_block for the real information and ffti_block
   for the imaginaray information.  The are set by procedure two_d_fft. *}
procedure nfft(var inverse : boolean);

var
   nd2f,nm1f,iff,iif,jf,j2f,kf,rf,df,argf,twff : longint;
   t1f, t2f, cf, sf : real;
   a,index,stages,w,indexf : integer;
   entries : longint;
   pif : real;

  begin

    {* Find the power of two that represents the number of entries into the *}
    {* array                                                                *}
    pif := 3.1415926536;
    stages := 0;
    a := 2;
    entries := max_samples;
    stages := power_down(a,entries);

    if(inverse = true)then index := -1
    else index := 1;

{*********************************************************}
{*    indexf := -1;
    if(inverse = false)then
      begin
        for jf := 1 to entries do
```

```
          begin
            indexf := -indexf;
            ffti_block[jf] := ffti_block[jf] * indexf;
            fftr_block[jf] := fftr_block[jf] * indexf;
          end;
      end;  *)
{*********************************************************}

    nd2f := entries div 2;
    nmlf := entries - 1;
    jf := 1;
    for iff := 1 to nmlf do
      begin
        if (iff < jf) then
          begin
            t1f := fftr_block[jf];
            fftr_block[jf] := fftr_block[iff];
            fftr_block[iff] := t1f;
            t2f := ffti_block[jf];
            ffti_block[jf] := ffti_block[iff];
            ffti_block[iff] := t2f;
          end;
        kf := nd2f;
        while (kf < jf) do
          begin
            jf := jf - kf;
            kf := kf div 2;
          end;
        jf := jf + kf;
      end;

    for iff := 1 to stages do
      begin
        rf := 1;
        for iif := 1 to iff do rf := rf*2;
        df := rf div 2;

        {* Calculate FFT *)

        {*****************************************}
        {* If not use cos and sin table then     *}
        {*****************************************}

        {*****************************************}
        {* Note : index can be negative or positive *}
        {*****************************************}

        argf := -index * entries div rf;
        for jf := 1 to df do
          begin
            twff := (jf - 1) * argf;

            {*************************************}
            {* If not use cos and sin table then *}
            {* cf := cos(twff);                  *}
```

```
            {* sf := sin(twff);                    *)
            {*************************************}

            cf := cs[twff];
            sf := sn[twff];
            kf := jf;
            while( kf <= entries) do
              begin
                j2f := kf + df;
                t1f := (cf * fftr_block[j2f] + sf * ffti_block[j2f]);
                t2f := (-sf * fftr_block[j2f] + cf * ffti_block[j2f]);
                fftr_block[j2f] := fftr_block[kf] - t1f;
                ffti_block[j2f] := ffti_block[kf] - t2f;
                fftr_block[kf] := fftr_block[kf] + t1f;
                ffti_block[kf] := ffti_block[kf] + t2f;
                kf := kf + rf;
              end; {* while *}
          end; {* for jf *}
      end; {* for iff *}

    if (inverse = true) then
      begin
        for w := 1 to entries do
          begin
            fftr_block[w] := fftr_block[w] / entries;
            ffti_block[w] := ffti_block[w] / entries;
          end; {* for w *}
      end; {* if inverse is true *}


{*****************************************************************}
{*    If -indexf is used then replace if inverse = true with    *}
{*            indexf := -indexf;
            fftr_block[w] := fftr_block[w] * indexf / entries;
            ffti_block[w] := ffti_block[w] * indexf / entries; *}
{*****************************************************************}

  end; {* procedure nfft *}



{*************************************************************************}
{*              P R O C E D U R E   T W O _ D _ F F T                    *}
{*************************************************************************}
{* This procedure takes the two-dimensional FFT of an input of type fftp
   sent to it (new_array).  It is also sent inverse, which tells the
   procedure if it is taking the forward or inverse FFT of the matrix. *}
procedure two_d_fft(var new_array : fftp; inverse : boolean);

var
  x,i,j,w_count,n : integer;

begin

  for n := 1 to 2 do
    begin
```

```
      for x := 1 to blocks do
        begin
          fft_block := new_array[x];
          for j := 1 to rows do
            begin

              w_count := 0;
              for i := 1 to max_samples do
                begin
                  inc(w_count);
                  fftr_block[w_count] := fft_block^[i,j].real_pt;
                  ffti_block[w_count] := fft_block^[i,j].image_pt;
                end; {* for i *}

              nfft(inverse);

              w_count := 0;
              for i := 1 to max_samples do
                begin
                  inc(w_count);
                  fft_block^[i,j].real_pt := fftr_block[w_count];
                  fft_block^[i,j].image_pt := ffti_block[w_count];
                end; {* for i *}

            end; {* for j *}
        end; {* for x *}

      {* Transpose entries in the real and complex arrays *}
      transpose(new_array);

    end; {* for n *}

end; {* procedure 2_d_fft) *}


{*************************************************************************}
{*                  P R O C E D U R E   M A G N I T U D E                *}
{*************************************************************************}
{* This procedure takes the magnitude of an array of type fftp (new_array)
   and stores in a data structure of type fftp (new_mag) provided the varible
   store is true.  Otherwise, if store is false, then the magnitude is stored
   back into new_array.  The variable store is used when the image is to be
   displayed but the data is not to be corrupted. *}
procedure magnitude(var new_array,new_mag : fftp; store : boolean);

var
  i,j,x : integer;
  new_block : fft_ptr;

begin

   for x := 1 to blocks do
     begin
       fft_block := new_array[x];
       if (store = false)then getmem(new_block,s);
       for j := 1 to rows do
         begin
```

```
            for i := 1 to max_samples do
              begin
                 if(store = true) then
                   begin
                     fft_block^[i,j].real_pt := sqrt( sqr(fft_block^[i,j].real_pt) +
                         sqr(fft_block^[i,j].image_pt) );
                     fft_block^[i,j].image_pt := 0;
                   end
                 else
                   begin
                     new_block^[i,j].real_pt := sqrt( sqr(fft_block^[i,j].real_pt) +
                         sqr(fft_block^[i,j].image_pt) );
                     new_block^[i,j].image_pt := 0;
                   end;
              end; {* for i *}
          end; {* for j *}
        if(store = false)then new_mag[x] := new_block;
      end; {* for x *}

end; {* procedure magnitude *}

{* END MODULE FFT_LIB.PAS *}
{*************************************************************************}
```

APPENDIX E

MODULE OP_LIB.PAS

```
{*************************************************************************}
{*************************************************************************}
{*                M O D U L E   O P _ L I B . P A S                      *}
{*************************************************************************}
{*************************************************************************}

{*************************************************************************}
{*            Author        : James J. Hosker                            *}
{*            Created       : June 12, 1989                              *}
{*            Version       : 1.0                                        *}
{*            Compiler      : Borland TurboPascal Version 5.0            *}
{*************************************************************************}

{* This module is part of program simulation which was submitted to fulfull
   the requirements of a Master's Degree in Electrical Engineering at
   Tufts University.  For more information see the main program module
   'SIMUL.PAS' and my paper that was submitted along with the
   program. *}

{*************************************************************************}
{*       P R O C E D U R E   D I S P O S E _ P O I N T E R S             *}
{*************************************************************************}
{*  This procedure disposes of all the pointers created by type fftp.
    A data array of type fftp is passed to this procedure. *}
procedure dispose_pointers(var any_array : fftp);
var
  x : integer;

begin

  for x := 1 to blocks do freemem(any_array[x],s);

end; {* procedure dispose_pointers *}



{*************************************************************************}
{*                P R O C E D U R E   S A V E _ I N F O                  *}
{*************************************************************************}
{* This procedure saves the information of one iteration to the hard disk
   of a PC computer.  Three pieces of information are passed to it.  An
   array of type fftp (sdata), the name of the file the information is
   to be saved in (fname), and the iteration number (itname).  *}
procedure save_info(var sdata : fftp; fname,itname: string);
var
  nw,nnum,x,i,j : longint;
  nn_dat : text;


begin
```

```
      assign(nn_dat,fname);
      rewrite(nn_dat);
      writeln(nn_dat,'GS OUTPUT FILE');
      writeln(nn_dat,itname);
      for x := 1 to blocks do
        begin
          fft_block := sdata[x];
          for j := 1 to rows do
            begin
              for i := 1 to max_samples do
                begin
                  writeln(nn_dat,fft_block^[i,j].real_pt);
                end; {* for i *}
            end; {* for j *}
        end; {* for x *}
      close(nn_dat);

end; {* procedure save_info *}



{*************************************************************************}
{*              P R O C E D U R E   G E T _ I N P U T                    *}
{*************************************************************************}
{* This procedure reads the file 'INPUT.DAT',which contain the original
   object and all the zero planes used to retrieve the object.  It also
   creates three data structures of type fftp:  data, error_data and
   prev_data.  Data holds the present image.  Error_data saves g'(x) for
   the MSE analysis.  Prev_data holds the last value of the image g(x) to
   be used in the hybrid input_output algorithm.  A sturcture of type fftp
   is passed to the procedure. *}
procedure get_input(var  block_ptr : fftp);
var
   i,j,x,a : integer;
   num : integer;
   garbl : char;
   fft_bl : fft_ptr;
   input_file : text;

begin

  assign(input_file,'input.dat');
  reset(input_file);

  for x := 1 to blocks do
    begin
      getmem(fft_block,s);
      getmem(prev_block,s);
      getmem(error_block,s);
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              read(input_file,num);
              read(input_file,garbl);
              with fft_block^[i,j] do
                begin
```

```
                real_pt := num;
                image_pt := 0;
              end;
            with prev_block^[i,j] do
              begin
                real_pt := 0;
                image_pt := 0;
              end;
            with error_block^[i,j] do
              begin
                real_pt := 0;
                image_pt := 0;
              end;
          end; {* for  i *}
      end; {* for j *}
    block_ptr[x] := fft_block;
    prev_data[x] := prev_block;
    error_data[x] := error_block;
  end; {* for x *}

  close(input_file);

end; {* procedure get_input *}



{*************************************************************************}
{*       P R O C E D U R E   M A K E _ S I N _ C O S _ T B L         *}
{*************************************************************************}
{* This procedure makes the sin and cosine table that is used in the
   calculation of the FFT. *}
procedure make_sin_cos_tbl;
var
 j : longint;
 pif : real;
 gain : real;

begin

  pif := 3.1415926536;

  gain := 2 * pif / max_samples;

  for j := -nc to nc do
    begin
      cs[j] := cos(j * gain);
      sn[j] := sin(j * gain);
    end; {* for j *}

end; {* procedure make_sin_cos_tbl *}



{*************************************************************************}
{*          P R O C E D U R E   M U L T I P L Y _ 2 _ D             *}
{*************************************************************************}
{*  This procedure multiplies to data structures of type fftp together and
```

```
    stores the result into the first data structure (new_data) that was passed
    to the procedure. It also deletes the second data structure (ph_data)
    by calling dispose_pointers provided no_delete is true; otherwise, it
    saves the information in the second data structure ph_data.*)
procedure multiply_2_d(var new_data,ph_data : fftp);
var
  x,i,j : integer;
  image_part,real_part : real;
  new_block,ph_block : fft_ptr;

begin

  for x := 1 to blocks do
    begin
      new_block := new_data[x];
      ph_block := ph_data[x];
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              real_part := ((new_block^[i,j].real_pt * ph_block^[i,j].real_pt) -
                            (new_block^[i,j].image_pt * ph_block^[i,j].image_pt));
              image_part := ((new_block^[i,j].image_pt * ph_block^[i,j].real_pt) +
                             (new_block^[i,j].real_pt * ph_block^[i,j].image_pt));
              new_block^[i,j].real_pt := real_part;
              new_block^[i,j].image_pt := image_part;
            end; {* for i *}
        end; {* for j *}
    end; {* for x *}

  if(no_delete = false)then  dispose_pointers(ph_data);

end; {* procedure multiply_2_d *}



{*************************************************************************}
{*          P R O C E D U R E   R A N D O M _ P H A S E                   *}
{*************************************************************************}
{* This procedure calculates the random_phase that is used to distort the
   original image. The section that is commented out is the established
   way of calcuating the random phase.  However, this program uses a random
   phase technique developed by Prof. Gonsalves.  Either technique will work
   with the final algorithm. A data structure of type fftp is passed to
   this procedure. *}
procedure random_phase(var n_data : fftp);
var
   newj,newk,temp_i,temp_j,x,i,j,k : integer;
   nx11,noise,x11,r : real;
   temprand_block,random_block : fft_ptr;
   random_data : fftp;
   a11 : array[1..max_samples] of real;
   k1,n : integer;

begin

  n := max_samples;
```

```
  for x := 1 to blocks do
    begin
      fft_block := n_data[x];
      getmem(random_block,s);
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              random_block^[i,j].real_pt := 0;
              random_block^[i,j].image_pt := 0;
            end; {* for i *}
        end; {* for j *}
      random_data[x] := random_block;
    end; {* for x *}

  noise:=1.0*2*pi;
  randomize;
  fft_block := n_data[1];
  x11 := fft_block^[1,1].real_pt;

{* Caculation of established random phase *}
{*  for x := 1 to blocks do
    begin
      random_block := random_data[x];
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              r := random;
              random_block^[i,j].real_pt := cos(2*pi*r);
              random_block^[i,j].image_pt := sin(2*pi*r)
            end;
        end;
      random_data[x] := random_block;
    end; *}

  for j:=1 to nc do
    for k:=1 to n do
      begin

        find_trans_entry(k,j);

        fft_block := n_data[save_block];
        random_block := random_data[save_block];
        nx11 := fft_block^[find_i,find_j].real_pt;

        a11[k]:=noise*(random-0.5)*(1.0-nx11/x11);
          { +pi*(k-1)/2 +pi*(j-1)/2; }
        if j =1 then
          begin
            if k = 1 then a11[k]:=0;
            if k = nc then a11[k]:=0;
            if k > nc then a11[k]:=-a11[nc-(k-nc)];
          end;
        if j =nc then
           begin
```

```
            if k = 1 then all[k]:=0;
            if k = nc then all[k]:=0;
            if k > nc then all[k]:=-all[nc-(k-nc)];
          end;
        random_block^[find_i,find_j].real_pt := cos(all[k]);
        random_block^[find_i,find_j].image_pt := sin(all[k]);
      end;


  for k:=1 to n do
    for j:=nc+1 to n do
      begin
        if k >1 then
          begin

            find_trans_entry(k,j);
            temprand_block := random_data[save_block];
            temp_i := find_i;
            temp_j := find_j;

            newj := n+2-j;
            newk := n+2-k;
            find_trans_entry(newk,newj);
            random_block := random_data[save_block];

            temprand_block^[temp_i,temp_j].real_pt :=
               random_block^[find_i,find_j].real_pt;
            temprand_block^[temp_i,temp_j].image_pt :=
               random_block^[find_i,find_j].image_pt;

          end;

        if k =1 then
          begin

            find_trans_entry(k,j);
            temprand_block := random_data[save_block];
            temp_i := find_i;
            temp_j := find_j;

            newj := n+2-j;
            newk := k;
            find_trans_entry(newk,newj);
            random_block := random_data[save_block];

            temprand_block^[temp_i,temp_j].real_pt :=
               random_block^[find_i,find_j].real_pt;
            temprand_block^[temp_i,temp_j].image_pt :=
               random_block^[find_i,find_j].image_pt;

          end;
      end;

  multiply_2_d(n_data,random_data);

end; {* procedure random_phase *}
```

```
{*************************************************************************}
{*              P R O C E D U R E   T H E T A _ M O D                    *}
{*************************************************************************}
{* This procedure imposed the U-domain contraints on the G(u) to create
   G'(u).  It imposes the theta_modulus on the data structure fftp. *}
procedure theta_mod(var new_data : fftp);
var
   mag_data : fftp;
   mag_block : fft_ptr;
   i,j,x : integer;

begin

  magnitude(new_data,mag_data,no_store);

  for x := 1 to blocks do
    begin
      fft_block := new_data[x];
      mag_block := mag_data[x];
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              fft_block^[i,j].real_pt := (fft_block^[i,j].real_pt /
mag_block^[i,j].real_pt);
              fft_block^[i,j].image_pt := (fft_block^[i,j].image_pt /
mag_block^[i,j].real_pt);
            end; {* for i *}
        end; {* for j *}
      end; {* for x *}

   dispose_pointers(mag_data);

end; {* theta_mod *}


{*************************************************************************}
{*        P R O C E D U R E   S A V E _ P R E V _ D A T A                 *}
{*************************************************************************}
{*  This procedure saves the previous iteration of g(x) to be used in the
    calculations of the hybrid input-output algorithm in the data structure
    prev_data. A data structure of type fftp is passed to this procedure. *}
procedure save_prev_data(var  block_ptr : fftp);
var
   i,j,x,a : integer;

begin

  for x := 1 to blocks do
    begin
      fft_block := block_ptr[x];
      prev_block := prev_data[x];
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
```

```
          begin
            prev_block^[i,j].real_pt := fft_block^[i,j].real_pt;
            prev_block^[i,j].image_pt := fft_block^[i,j].image_pt;
          end; {* for  i *}
        end; {* for j *}
    end; {* for x *}

end; {* procedure save_prev_data *}


{************************************************************************}
{*       P R O C E D U R E   S A V E _ E R R O R _ D A T A              *}
{************************************************************************}
{* This procedure saves g'(x) before the X-domain constraints are applied
   to create g(x) in data structure error_data.  This data is used in
   the calculation of the RMS of Eok.  The data structure that is passed
   to this procedure is of type fftp. *}
procedure save_error_data(var  block_ptr : fftp);
var
   i,j,x,a : integer;

begin

  for x := 1 to blocks do
    begin
      fft_block := block_ptr[x];
      error_block := error_data[x];
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              error_block^[i,j].real_pt := fft_block^[i,j].real_pt;
              error_block^[i,j].image_pt := fft_block^[i,j].image_pt;
            end; {* for  i *}
        end; {* for j *}
    end; {* for x *}

end; {* procedure save_error_data *}


{************************************************************************}
{*    P R O C E D U R E   X _ D O M A I N _ C O N S T R A I N T S      *}
{************************************************************************}
{* This procedure applies all the X-domain contraints to the object for
   the error-reduction and hybrid input-output algorithms including the
   some calculation for the hybrid input-output algorithm used in the
   calculation of the RMS of Eok. This procedure has a data structure of
   type fftp passed to it. *}
procedure x_domain_contraints(var new_data : fftp);
var
  x,i,j,first,end_row : integer;
  neg_one : real;
  mag_block : fft_ptr;

begin

  if((max_samples = 16) or (max_samples = 64))then
```

```
  begin

    for x := 3 to blocks do
      begin
        fft_block := new_data[x];
        for j := 1 to rows do
          begin
            for i := 1 to max_samples do
              begin
                fft_block^[i,j].real_pt := 0;
                fft_block^[i,j].image_pt := 0;
              end; {* for i *}
          end; {* for j *}
      end; {* for x *}

    for i := nc to max_samples do
      begin
        for x := 1 to 2 do
          begin
            fft_block := new_data[x];
            for j := 1 to rows do
              begin
                fft_block^[i,j].real_pt := 0;
                fft_block^[i,j].image_pt := 0;
              end; {* for j *}
          end; {* for x *}
      end; {* for i *}

  end {* if 16 or 64 *}

else
  begin

    for x := 1 to 4 do
      begin
        fft_block := new_data[x];
        for j := 1 to rows do
          begin
            for i := 1 to max_samples do
              begin
                fft_block^[i,j].real_pt := 0;
                fft_block^[i,j].image_pt := 0;
              end; {* for i *}
          end; {* for j *}
      end; {* for x *}

    for x := 13 to 16 do
      begin
        fft_block := new_data[x];
        for j := 1 to rows do
          begin
            for i := 1 to max_samples do
              begin
                fft_block^[i,j].real_pt := 0;
                fft_block^[i,j].image_pt := 0;
              end; {* for i *}
          end; {* for j *}
```

```
      end; {* for x *}

    for i := 1 to 32 do
      begin
        for x := 1 to blocks do
          begin
            fft_block := new_data[x];
            for j := 1 to rows do
              begin
                fft_block^[i,j].real_pt := 0;
                fft_block^[i,j].image_pt := 0;
              end; {* for j *}
          end; {* for x *}
      end; {* for i *}

    for i := 97 to max_samples do
      begin
        for x := 1 to blocks do
          begin
            fft_block := new_data[x];
            for j := 1 to rows do
              begin
                fft_block^[i,j].real_pt := 0;
                fft_block^[i,j].image_pt := 0;
              end; {* for j *}
          end; {* for x *}
      end; {* for i *}

  end; {* else max_samples = 128 *}

gp := 0;
for x := 1 to blocks do
  begin
    fft_block := new_data[x];
    prev_block := prev_data[x];
    for j := 1 to rows do
      begin
        for i := 1 to max_samples do
          begin

            if(al_name = 'GS=')then
              begin
                if (fft_block^[i,j].real_pt < 0.001) then
                    fft_block^[i,j].real_pt := 0;
                if (fft_block^[i,j].real_pt > 36) then
                    fft_block^[i,j].real_pt := 36;
                fft_block^[i,j].image_pt := 0;
              end
            else if(al_name = 'IO=')then
              begin
                if (fft_block^[i,j].real_pt < 0.001) then
                  begin
                    gp := (gp + sqr(fft_block^[i,j].real_pt) +
                          sqr(fft_block^[i,j].image_pt));
                    fft_block^[i,j].real_pt := prev_block^[i,j].real_pt -
                    (beta * fft_block^[i,j].real_pt);
                  end;
```

```
                if (fft_block^[i,j].real_pt > 36) then
                    fft_block^[i,j].real_pt := 36;
                fft_block^[i,j].image_pt := 0;
              end;

          end; {* for i *}
      end; {* for j *}
  end; {* for x *}

end; {* procedure x_domain_constraints *}


{*************************************************************************}
{*        P R O C E D U R E   C A L C U L A T E _ E R R O R          *}
{*************************************************************************}
{* This procedure calcuates the RMS of Eok for the error-reduction and
   hybrid input-output algorithms.  It is used for the error-reduction
   algorithm to determine if the algorithm has stagnated. A data structure
   of type fftp is passed to it. Finally it writes certain information to
   the screen of the operator. *}
procedure calculate_error(var new_data : fftp);
var
  neo : real;
  x,i,j : integer;
  itnm,errornm,er_nm : string;

begin

  eok := 0;
  ngp := 0;

  for x := 1 to blocks do
    begin
      fft_block := new_data[x];
      error_block := error_data[x];
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              eok := (eok +
                   sqr(fft_block^[i,j].real_pt - error_block^[i,j].real_pt) +
                   sqr(fft_block^[i,j].image_pt - error_block^[i,j].image_pt));
              ngp := (ngp + sqr(error_block^[i,j].real_pt) +
                     sqr(error_block^[i,j].image_pt));
            end; {* for i *}
        end; {* for j *}
    end; {* for x *}

  if(al_name = 'GS=')then neo := eok / ngp;
  if(al_name = 'IO=')then neo := gp / ngp;
  sneo := sqrt(neo);

  if((sneo >= 0) and (sneo <= max_error) and (al_name = 'GS='))then
    begin
      cnt := cnt + 1;
      if(cnt = 3)then
        begin
```

```
          cnt := 0;
          stagnated := true;
        end;
    end
  else if(al_name = 'IO=')then cnt:= 0;

  errornm := '';
  er_nm := 'RMS error of Eok is ';
  str(sneo,errornm);
  str(iteration,itnm);
  er_nm := concat(er_nm,errornm);
  er_nm := concat(er_nm,' - ');
  if(al_name = 'GS=')then er_nm := concat(er_nm,'GS iteration ');
  if(al_name = 'IO=')then er_nm := concat(er_nm,'IO iteration ');
  er_nm := concat(er_nm,itnm);
  er_nm := concat(er_nm,'.');
  setcolor(blue);
  bar(1,getmaxy - 8,getmaxx,getmaxy);
  setcolor(yellow);
  outtextxy((getmaxx div 2 - 220),(getmaxy - 8),er_nm);

end; {* procedure calculate_error *}


{*************************************************************************}
{*          P R O C E D U R E   S A V E _ M O D _ F I L E                 *}
{*************************************************************************}
{* This procedure saves the modulus of an object or image to be used in
   a calculation or to be displayed on the screen.  A data structure of
   type fftp is passed to this procedure. *}
procedure save_mod_file(var new_array : fftp);
var
  x,j,i : integer;
  new_block : fft_ptr;

begin


  for x := 1 to blocks do
    begin
      fft_block := new_array[x];
      getmem(new_block,s);
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              new_block^[i,j].real_pt := fft_block^[i,j].real_pt;
              new_block^[i,j].image_pt := fft_block^[i,j].image_pt;
            end; {* for i *}
        end; {* for j *}
      save_mod[x] := new_block;
    end; {* for x *}

end; {* procedure save_mod_file *}

{* END MODULE OP_LIB.PAS *}
{*************************************************************************}
```

APPENDIX F

MODULE PLOT_LIB.PAS

```
{*************************************************************************}
{*************************************************************************}
{*              M O D U L E   P L O T _ L I B . P A S                    *}
{*************************************************************************}
{*************************************************************************}

{*************************************************************************}
{*           Author        : James J. Hosker                            *}
{*           Created       : June 12, 1989                              *}
{*           Version       : 1.0                                        *}
{*           Compiler      : Borland TurboPascal Version 5.0            *}
{*************************************************************************}

{* This module is part of program simulation which was submitted to fulfull
   the requirements of a Master's Degree in Electrical Engineering at
   Tufts University.  For more information see the main program module
   'SIMUL.PAS' and my paper that was submitted along with the
   program. *}

{*************************************************************************}
{*            P R O C E D U R E   D I S P L A Y _ P I X                  *}
{*************************************************************************}
{* This procedure displays the pixel on the screen of the operator.  It is
   sent pblk that indicates the greyscale pixel to be diplayed and ni and
   nj indicate the x andy y position to begin displaying the pixel on the
   screen. *}
procedure display_pix(var pblk,ni,nj : integer);
var
  ipix,y,count : integer;
  new_ni,new_nj : integer;
  t1,t2,t3,t4,t5,t6,t7,t8,t9,tem : byte;

begin

  new_ni := ni;
  new_nj := nj;
  ipix := 1;
  count := 1;

  for y := 1 to sqr_bit_size do
    begin

      if(bit_size = 3)then
        begin
          if (y = 1)then tem := bit_pat[pblk].x1
          else if (y = 2)then tem := bit_pat[pblk].x2
          else if (y = 3)then tem := bit_pat[pblk].x3
          else if (y = 4)then tem := bit_pat[pblk].x4
          else if (y = 5)then tem := bit_pat[pblk].x5
          else if (y = 6)then tem := bit_pat[pblk].x6
          else if (y = 7)then tem := bit_pat[pblk].x7
```

```
    else if (y = 8)then tem := bit_pat[pblk].x8
    else if (y = 9)then tem := bit_pat[pblk].x9
  end
else if(bit_size = 4)then
  begin
    if (y = 1)then tem := bit_pat[pblk].x1
    else if (y = 2)then tem := bit_pat[pblk].x2
    else if (y = 3)then tem := bit_pat[pblk].x3
    else if (y = 4)then tem := bit_pat[pblk].x4
    else if (y = 5)then tem := bit_pat[pblk].x5
    else if (y = 6)then tem := bit_pat[pblk].x6
    else if (y = 7)then tem := bit_pat[pblk].x7
    else if (y = 8)then tem := bit_pat[pblk].x8
    else if (y = 9)then tem := bit_pat[pblk].x9
    else if (y = 10)then tem := bit_pat[pblk].x10
    else if (y = 11)then tem := bit_pat[pblk].x11
    else if (y = 12)then tem := bit_pat[pblk].x12
    else if (y = 13)then tem := bit_pat[pblk].x13
    else if (y = 14)then tem := bit_pat[pblk].x14
    else if (y = 15)then tem := bit_pat[pblk].x15
    else if (y = 16)then tem := bit_pat[pblk].x16
  end;

if (tem = 0)then
  begin
    setfillstyle(solidfill,lightblue);
    setcolor(lightblue);
  end
else if (tem = 1)then
  begin
    setfillstyle(solidfill,white);
    setcolor(white);
  end
else if (tem = 2)then
  begin
    setfillstyle(solidfill,lightgray);
    setcolor(lightgray);
  end
else if (tem = 3)then
  begin
    setfillstyle(solidfill,darkgray);
    setcolor(darkgray);
  end
else if (tem = 4)then
  begin
    setfillstyle(solidfill,black);
    setcolor(black);
  end;

bar(new_ni,new_nj,(new_ni + 1),(new_nj + 1));

new_ni := new_ni + 1;

if( ((bit_size = 3 ) and (count = 3)) or ((bit_size = 4) and
     (count = 4)) )then
  begin
    new_ni := ni;
```

```
          new_nj := new_nj + 1;
          count := 1;
        end
      else inc(count);

    end; {* for y *}


end; {* procedure display_pix *}


{*************************************************************************}
{*        P R O C E D U R E   C R E A T E _ P A T T E R N S          *}
{*************************************************************************}
{* This procedure creates the 36 greyscale level bit patterns that are
   used to display on the screen of the operator.  *}
procedure create_patterns;
var
  a,b,k,c : integer;
  t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16 : byte;

begin

  t1 := 0;
  t2 := 0;
  t3 := 0;
  t4 := 0;
  t5 := 0;
  t6 := 0;
  t7 := 0;
  t8 := 0;
  t9 := 0;
  t10 := 0;
  t11 := 0;
  t12 := 0;
  t13 := 0;
  t14 := 0;
  t15 := 0;
  t16 := 0;

  k := 1;
  for c := 0 to max_pats do
    begin

      if(bit_size = 3)then
        begin

          bit_pat[c].x1 := t1;
          bit_pat[c].x2 := t2;
          bit_pat[c].x3 := t3;
          bit_pat[c].x4 := t4;
          bit_pat[c].x5 := t5;
          bit_pat[c].x6 := t6;
          bit_pat[c].x7 := t7;
          bit_pat[c].x8 := t8;
          bit_pat[c].x9 := t9;
```

```
    if(k = 1)then inc(t1)
    else if(k = 2)then inc(t2)
    else if(k = 3)then inc(t3)
    else if(k = 4)then inc(t4)
    else if(k = 5)then inc(t5)
    else if(k = 6)then inc(t6)
    else if(k = 7)then inc(t7)
    else if(k = 8)then inc(t8)
    else if(k = 9)then
      begin
        inc(t9);
        k := 0;
      end;
  end {* if bit_size = 3 *}

else if(bit_size = 4)then
  begin

    bit_pat[c].x1 := t1;
    bit_pat[c].x2 := t2;
    bit_pat[c].x3 := t3;
    bit_pat[c].x4 := t4;
    bit_pat[c].x5 := t5;
    bit_pat[c].x6 := t6;
    bit_pat[c].x7 := t7;
    bit_pat[c].x8 := t8;
    bit_pat[c].x9 := t9;
    bit_pat[c].x10 := t10;
    bit_pat[c].x11 := t11;
    bit_pat[c].x12 := t12;
    bit_pat[c].x13 := t13;
    bit_pat[c].x14 := t14;
    bit_pat[c].x15 := t15;
    bit_pat[c].x16 := t16;

    if(k = 1)then inc(t1)
    else if(k = 2)then inc(t2)
    else if(k = 3)then inc(t3)
    else if(k = 4)then inc(t4)
    else if(k = 5)then inc(t5)
    else if(k = 6)then inc(t6)
    else if(k = 7)then inc(t7)
    else if(k = 8)then inc(t8)
    else if(k = 9)then inc(t9)
    else if(k = 10)then inc(t10)
    else if(k = 11)then inc(t11)
    else if(k = 12)then inc(t12)
    else if(k = 13)then inc(t13)
    else if(k = 14)then inc(t14)
    else if(k = 15)then inc(t15)
    else if(k = 16)then
      begin
        inc(t16);
        k := 0;
      end;
```

```
        end; {* if bit_size = 4 *}

      inc(k);

    end; {* for c *}

end; {* procedure create_patterns *}


{*************************************************************************}
{*          P R O C E D U R E   D I S P L A Y _ I M A G E                *}
{*************************************************************************}
{* This procedure finds the x and y pixel to begin displaying the image
   based on its size and normalizes the values of g(x) within the 36
   possible greyscale levels.  It displays each individual pixel by calling
   display_pix.  A data structure of fftp and the x and y axis to begin
   displaying the image are sent to the procedure. *}
procedure display_image(var new_data : fftp;x_axis,y_axis : integer);
var
  savei,savej,p,i,j,x,offy,offx,newi,newj,mult_factor,pix : integer;
  levels,local_max,local_min,auto_scale : real;
  n_quit : boolean;
  nnx : integer;
  new_real : longint;
  end_block,end_sample : integer;

begin

  local_max := 0;
  fft_block := new_data[1];
  local_min := fft_block^[1,1].real_pt;

  for x := 1 to blocks do
    begin
      fft_block := new_data[x];
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              if fft_block^[i,j].real_pt > local_max then
                  local_max := fft_block^[i,j].real_pt;
              if fft_block^[i,j].real_pt < local_min then
                  local_min := fft_block^[i,j].real_pt;
            end; {* for i *}
        end; {* for j *}
    end; {* for x *}

  auto_scale := local_max - local_min;
  if (auto_scale = 0) then auto_scale := 1E00;

  if((max_samples = 16) or (max_samples = 64))then
    begin
      if(bit_size = 3)then
        begin
          mult_factor := 4;
          pix := 3;
```

```
        levels := 36;
      end
    else if(bit_size = 4)then
      begin
        mult_factor := 5;
        pix := 4;
        levels := 64;
      end;
  end
else if(max_samples = 128)then
  begin
    if(bit_size = 3)then
      begin
        mult_factor := 4;
        pix := 3;
        levels := 36;
      end;
  end;

if(dip_all = false)then
  begin
    offy := y_axis - ((mult_factor * (max_samples div 2)) div 2);
    offx := x_axis - ((mult_factor * (max_samples div 2)) div 2);
    end_block := 2;
    end_sample := max_samples div 2;
  end
else
  begin
    setfillstyle(solidfill,black);
    setbkcolor(black);
    bar(1,1,getmaxx,getmaxy);
    offy := y_axis - ((mult_factor * max_samples) div 2);
    offx := x_axis - ((mult_factor * max_samples) div 2);
    end_block := blocks;
    end_sample := max_samples;
  end;

savei := offx;
savej := offy;


for x := 1 to end_block do
  begin
    fft_block := new_data[x];
    for j := 1 to rows do
      begin
        for i := 1 to end_sample do
          begin

            new_real := round((fft_block^[i,j].real_pt / auto_scale) * levels);

            newi := savei;
            newj := savej;

            if(new_real <= 0)then p := 0
            else if(new_real = max_pats)then p := max_pats
            else
```

```
                begin
                  n_quit := false;
                  nnx := 1;
                  Repeat
                    if(new_real = nnx)then
                      begin
                        p := nnx;
                        n_quit := true;
                      end;
                    inc(nnx);
                  Until(n_quit = true);
                end; {* else *}

              display_pix(p,newi,newj);

              savei := (savei + pix + 1);
            end; {* for i *}

          savej := (savej + pix + 1);
          savei := offx;
        end; {* for j *}
    end; {* for x *}

  setfillstyle(solidfill,blue);
  setcolor(yellow);

end; {* procedure display_image *}



{*************************************************************************}
{*                    P R O C E D U R E   P A U S E                      *}
{*************************************************************************}
{* This procedure halts the program by generating a software interrupt $16
   and prompt the operator to decide whether to continue, quit or continue but
   save the present iteration to the hard disk.  If the wrong key is pressed,
   then beep the operator. *}
procedure pause;
const
  lowmask = $00ff;
var
  key : char;
  res : registers;
  return_key : boolean;
  ww : integer;

begin

   if(dip_all = false)then
     begin
       setcolor(blue);
       bar(1,getmaxy - 8,getmaxx,getmaxy);
       setcolor(yellow);
     end;
   ww := getmaxx div 2;
   outtextxy((ww - 300),(getmaxy - 8),'Press "return" to continue, press "s" to save and
continue, or "q" to quit.');
```

```
return_key := false;
save_key := false;

{* Clear keyboard buffer *}
res.AX := $0c06;
res.DL := $00;
intr($21,res);

Repeat

  res.AH := $00;
  intr($16,res);

  {* check to see if return is pressed *}
  if (res.AL = $0D) then
    begin
      return_key := true;
      res.AX := $0000;
      res.DX := res.AX;
      res.AX := $0600;
      setcolor(blue);
      if(dip_all = false)then
        begin
          setcolor(blue);
          bar(1,getmaxy - 8,getmaxx,getmaxy);
          setcolor(yellow);
          ww := getmaxx div 2;
          outtextxy((ww - 108),(getmaxy - 8),'Algorithm is now continuing.');
        end;
    end

  {* check to see if 'q' or 'Q' is pressed *}
  else if ((res.AL = $71) or (res.AL= $51))then
    begin
      quit := true;
      res.AX := $0000;
      res.DX := res.AX;
      res.AX := $0600;
    end

  {* check to see if 's' or 'S' is pressed *}
  else if ((res.AL = $73) or (res.AL= $53))then
    begin
      save_key := true;
      return_key := true;
      res.AX := $0000;
      res.DX := res.AX;
      res.AX := $0600;
    end

  {* else beep the operator *}
  else
    begin
      res.DX := $0007;
      res.AX := $0600;
      intr($21,res);
    end;
```

```
   Until((quit = true) or (return_key = true));

end; {* procedure pause *}


{*************************************************************************}
{*                    P R O C E D U R E   B E E P _ O P                   *}
{*************************************************************************}
{* This procedure generates a software interrupt $21 to beep the operator. *}
procedure beep_op;
var
  res : registers;

begin
  res.DX := $0007;
  res.AX := $0600;
  intr($21,res);
end; {* procedure beep_op *}


{*************************************************************************}
{*                    P R O C E D U R E   D I P _ D I S                   *}
{*************************************************************************}
{* This procedure finds the x and y axis and pixel to begin displaying the
   images from. It adjusts depending on the size of the matrix and
   determines how many of this size it can display on the screen of the
   operator. It will overwrite all except the original image if it runs
   out of space on the screen. It also writes the type of iteration and
   the iteration number to the screen of the operator. *}
procedure dip_dis;
begin

  inc(dip_count);

  if(max_samples = 16)then
    begin

      if(bit_size = 3)then
        begin
          of1 := 17;
          of2 := 47;
          new_x := new_x + 64;
          if((dip_count / 7) = 1)then
            begin
              dip_count := 0;
              if(y_count = 5)then
                begin
                  new_y := 32;
                  new_x := 96;
                  y_count := 1;
                  dip_count := 1;
                end
              else
                begin
                  inc(y_count);
                  new_y := new_y + 64;
```

```
                new_x := 32;
              end;
          end; {* if dip_count / 8 = 1 *}
        new_xx := new_x - 16;
        new_yy := new_y + 18;
      end {* if bit_size = 3 *}

    else if(bit_size = 4)then
      begin
        of1 := 21;
        of2 := 39;
        new_x := new_x + 80;
        if((dip_count / 6) = 1)then
          begin
            dip_count := 0;
            if(y_count = 4)then
              begin
                new_y := 60;
                new_x := 140;
                y_count := 1;
                dip_count := 1;
              end
            else
              begin
                inc(y_count);
                new_y := new_y + 80;
                new_x := 60;
              end;
          end; {* if dip_count / 6 = 1 *}
        new_xx := new_x - 20;
        new_yy := new_y + 22;
      end; {* if bit_size = 4 *}

  end {* if max_samples = 16 *}

else if(max_samples = 64)then
  begin

    if(bit_size = 3)then
      begin
        of1 := 65;
        of2 := 73;
        new_x := new_x + 138;
        if((dip_count / 3) = 1)then
          begin
            dip_count := 0;
            if(y_count = 2)then
              begin
                new_y := 84;
                new_x := 212;
                y_count := 1;
                dip_count := 1;
              end
            else
              begin
                inc(y_count);
                new_y := new_y + 148;
```

```
                new_x := 74;
              end;
          end; {* if dip_count / 3 = 1 *}
        new_xx := new_x - 64;
        new_yy := new_y + 66;
      end {* if bit_size = 3 *}

    else if(bit_size = 4)then
      begin
        of1 := 81;
        of2 := 89;
        new_x := new_x + 170;
        if((dip_count / 2) = 1)then
          begin
            dip_count := 0;
            if(y_count = 2)then
              begin
                new_y := 90;
                new_x := 260;
                y_count := 1;
                dip_count := 1;
              end
            else
              begin
                inc(y_count);
                new_y := new_y + 170;
                new_x := 90;
              end;
          end; {* if dip_count / 2 = 1 *}
        new_xx := new_x - 80;
        new_yy := new_y + 81;
      end; {* if bit_size = 4 *}

  end; {* if max_samples = 64 *}

  final_iteration := al_name;
  new_iteration := '';
  str(iteration,new_iteration);
  final_iteration := concat(final_iteration,new_iteration);
  final_iteration := concat(final_iteration,'.');

  setcolor(blue);
  bar(new_x-of1,new_y-of1,new_x+of2,new_y+of2);

  display_image(data,new_x,new_y);
  outtextxy(new_xx,new_yy,final_iteration);
  if(stop_it = true)then
    begin
      pause;
      setcolor(blue);
      bar(1,1,8,16);
      gotoxy(1,1);
    end;

end; {* procedure dip_dis *}

{* END MODULE PLOT_LIB.PAS *}
```

{**********}

APPENDIX G

PROGRAM NPLOT.PAS

```
{*************************************************************************}
{*************************************************************************}
{*                  P R O G R A M   N P L O T . P A S                    *}
{*************************************************************************}
{*************************************************************************}

{*************************************************************************}
{*            Author        : James J. Hosker                            *}
{*            Created       : June 12, 1989                              *}
{*            Version       : 1.0                                        *}
{*            Compiler      : Borland TurboPascal Version 5.0            *}
{*************************************************************************}

{* This program complements program simulation by displaying the save images
   from the hard disk to the screen of the operator. If it is not a
   Gerchberg-Saxton file, then an error message to that effect is displayed
   for the operator to see.  For more information see my paper and see
   the main module of program simulation, 'SIMUL.PAS'. *}

{*************************************************************************}
{* MAIN PROGRAM                                                          *}
{*************************************************************************}
program nplot(input,output);

{*************************************************************************}
{* Set Memory Requirements and Stack Size for Program NPLOT              *}
{*************************************************************************}
{$M 20000,0,640000}

{*************************************************************************}
{* Call in Internal Turbo Pascal Units                                   *}
{* Unit DOS supports DOS functions and program execution.                *}
{* Unit CRT supports screen mode controls, windows, sound and color      *}
{*      for the plotting procedures in the program.                      *}
{* Unit GRAPH supports CGA, EGA and VGA graphics for the plotting        *}
{*      procedures in the program.                                       *}
{*************************************************************************}
uses dos,crt,graph;

{*************************************************************************}
{* CONSTANTS and TYPE and VARIABLE DECLARATION                           *}
{*************************************************************************}
const

  max_samples = 64;
  sample_block = 16;
  blocks = 4;
  sqr_sample_block = sample_block * sample_block;
  sqr_max_samples = max_samples * max_samples;
  rows = ((sqr_max_samples div blocks) div max_samples);
```

```
  bit_size = 3;
  sqr_bit_size = bit_size * bit_size;
  max_pats = (sqr_bit_size * 4);

type

  fft_type =
    record
      real_pt : real;
      image_pt : real;
    end;

  fft_array = array[1..max_samples,1..rows] of fft_type;

  fft_ptr = ^fft_array;

  fftp = array[1..blocks] of fft_ptr;

  pattern_rec =
    record
      x1 : byte;
      x2 : byte;
      x3 : byte;
      x4 : byte;
      x5 : byte;
      x6 : byte;
      x7 : byte;
      x8 : byte;
      x9 : byte;
      x10 : byte;
      x11 : byte;
      x12 : byte;
      x13 : byte;
      x14 : byte;
      x15 : byte;
      x16 : byte;
    end;

  pat = array[0..max_pats] of pattern_rec;


var
    s : longint;
    data,mag : fftp;
    fft_block : fft_ptr;
    checkline,filename,final_iteration,iteration : string;
    nx,ny,new_x,new_y : integer;
    errorcode,graphdriver,graphmode : integer;
    try_again,continue,quit : boolean;
    opfile,error_file : text;
    ch : char;
    bit_pat : pat;


{*************************************************************************}
{*       P R O C E D U R E   D I S P O S E _ P O I N T E R S         *}
{*************************************************************************}
```

```
{*  This procedure disposes of all the pointers created by type fftp.
    A data array of type fftp is passed to this procedure. *}
procedure dispose_pointers(var any_array : fftp);
var
  x : integer;

begin

  for x := 1 to blocks do freemem(any_array[x],s);

end; {* procedure dispose_pointers *}



{*************************************************************************}
{*              P R O C E D U R E   G E T _ I N P U T                    *}
{*************************************************************************}
{* This procedure reads the file the operator selected and if it is a
   Gerchberg-Saxton file it plots it to the screen of the operator. *}
procedure get_input(var  block_ptr : fftp;fftdname : string);
var
   i,j,x,a : integer;
   num : real;
   garbl : string;
   input_file : text;

begin

  assign(input_file,fftdname);
  reset(input_file);
  readln(input_file,garbl);
  readln(input_file,iteration);

  for x := 1 to blocks do
    begin
      getmem(fft_block,s);
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin
              readln(input_file,num);
              with fft_block^[i,j] do
                begin
                  real_pt := num;
                  image_pt := 0;
                end;
            end; {* for  i *}
        end; {* for j *}
      block_ptr[x] := fft_block;
    end; {* for x *}

  close(input_file);

end; {* procedure get_input *}
```

```
{*************************************************************************}
{*            P R O C E D U R E   D I S P L A Y _ P I X                  *}
{*************************************************************************}
{* This procedure displays the pixel on the screen of the operator.  It is
   sent pblk that indicates the greyscale pixel to be diplayed and ni and
   nj indicate the x andy y position to begin displaying the pixel on the
   screen. *}
procedure display_pix(var pblk,ni,nj : integer);
var
  ipix,y,count : integer;
  new_ni,new_nj : integer;
  t1,t2,t3,t4,t5,t6,t7,t8,t9,tem : byte;

begin

  new_ni := ni;
  new_nj := nj;
  ipix := 1;
  count := 1;
  for y := 1 to sqr_bit_size do
    begin

      if(bit_size = 3)then
        begin
          if (y = 1)then tem := bit_pat[pblk].x1
          else if (y = 2)then tem := bit_pat[pblk].x2
          else if (y = 3)then tem := bit_pat[pblk].x3
          else if (y = 4)then tem := bit_pat[pblk].x4
          else if (y = 5)then tem := bit_pat[pblk].x5
          else if (y = 6)then tem := bit_pat[pblk].x6
          else if (y = 7)then tem := bit_pat[pblk].x7
          else if (y = 8)then tem := bit_pat[pblk].x8
          else if (y = 9)then tem := bit_pat[pblk].x9
        end
      else if(bit_size = 4)then
        begin
          if (y = 1)then tem := bit_pat[pblk].x1
          else if (y = 2)then tem := bit_pat[pblk].x2
          else if (y = 3)then tem := bit_pat[pblk].x3
          else if (y = 4)then tem := bit_pat[pblk].x4
          else if (y = 5)then tem := bit_pat[pblk].x5
          else if (y = 6)then tem := bit_pat[pblk].x6
          else if (y = 7)then tem := bit_pat[pblk].x7
          else if (y = 8)then tem := bit_pat[pblk].x8
          else if (y = 9)then tem := bit_pat[pblk].x9
          else if (y = 10)then tem := bit_pat[pblk].x10
          else if (y = 11)then tem := bit_pat[pblk].x11
          else if (y = 12)then tem := bit_pat[pblk].x12
          else if (y = 13)then tem := bit_pat[pblk].x13
          else if (y = 14)then tem := bit_pat[pblk].x14
          else if (y = 15)then tem := bit_pat[pblk].x15
          else if (y = 16)then tem := bit_pat[pblk].x16
        end;

      if (tem = 0)then
        begin
          setfillstyle(solidfill,lightblue);
```

```
          setcolor(lightblue);
        end
      else if (tem = 1)then
        begin
          setfillstyle(solidfill,white);
          setcolor(white);
        end
      else if (tem = 2)then
        begin
          setfillstyle(solidfill,lightgray);
          setcolor(lightgray);
        end
      else if (tem = 3)then
        begin
          setfillstyle(solidfill,darkgray);
          setcolor(darkgray);
        end
      else if (tem = 4)then
        begin
          setfillstyle(solidfill,black);
          setcolor(black);
        end;

      bar(new_ni,new_nj,(new_ni + 1),(new_nj + 1));

      new_ni := new_ni + 1;

      if( ((bit_size = 3 ) and (count = 3)) or ((bit_size = 4) and
           (count = 4)) )then
        begin
          new_ni := ni;
          new_nj := new_nj + 1;
          count := 1;
        end
      else inc(count);

    end; {* for y *}


end; {* procedure display_pix *}



{*************************************************************************}
{*        P R O C E D U R E   C R E A T E _ P A T T E R N S          *}
{*************************************************************************}
{* This procedure creates the 36 greyscale level bit patterns that are
   used to display on the screen of the operator.  *}
procedure create_patterns;
var
  a,b,k,c : integer;
  t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16 : byte;

begin

  t1 := 0;
  t2 := 0;
```

```
t3 := 0;
t4 := 0;
t5 := 0;
t6 := 0;
t7 := 0;
t8 := 0;
t9 := 0;
t10 := 0;
t11 := 0;
t12 := 0;
t13 := 0;
t14 := 0;
t15 := 0;
t16 := 0;

k := 1;
for c := 0 to max_pats do
  begin

    if(bit_size = 3)then
      begin

        bit_pat[c].x1 := t1;
        bit_pat[c].x2 := t2;
        bit_pat[c].x3 := t3;
        bit_pat[c].x4 := t4;
        bit_pat[c].x5 := t5;
        bit_pat[c].x6 := t6;
        bit_pat[c].x7 := t7;
        bit_pat[c].x8 := t8;
        bit_pat[c].x9 := t9;

        if(k = 1)then inc(t1)
        else if(k = 2)then inc(t2)
        else if(k = 3)then inc(t3)
        else if(k = 4)then inc(t4)
        else if(k = 5)then inc(t5)
        else if(k = 6)then inc(t6)
        else if(k = 7)then inc(t7)
        else if(k = 8)then inc(t8)
        else if(k = 9)then
          begin
            inc(t9);
            k := 0;
          end;
      end {* if bit_size = 3 *}

    else if(bit_size = 4)then
      begin

        bit_pat[c].x1 := t1;
        bit_pat[c].x2 := t2;
        bit_pat[c].x3 := t3;
        bit_pat[c].x4 := t4;
        bit_pat[c].x5 := t5;
        bit_pat[c].x6 := t6;
        bit_pat[c].x7 := t7;
```

```
            bit_pat[c].x8 := t8;
            bit_pat[c].x9 := t9;
            bit_pat[c].x10 := t10;
            bit_pat[c].x11 := t11;
            bit_pat[c].x12 := t12;
            bit_pat[c].x13 := t13;
            bit_pat[c].x14 := t14;
            bit_pat[c].x15 := t15;
            bit_pat[c].x16 := t16;

            if(k = 1)then inc(t1)
            else if(k = 2)then inc(t2)
            else if(k = 3)then inc(t3)
            else if(k = 4)then inc(t4)
            else if(k = 5)then inc(t5)
            else if(k = 6)then inc(t6)
            else if(k = 7)then inc(t7)
            else if(k = 8)then inc(t8)
            else if(k = 9)then inc(t9)
            else if(k = 10)then inc(t10)
            else if(k = 11)then inc(t11)
            else if(k = 12)then inc(t12)
            else if(k = 13)then inc(t13)
            else if(k = 14)then inc(t14)
            else if(k = 15)then inc(t15)
            else if(k = 16)then
              begin
                inc(t16);
                k := 0;
              end;

          end; {* if bit_size = 4 *}

        inc(k);

      end; {* for c *}

end; {* procedure create_patterns *}


{*************************************************************************}
{*          P R O C E D U R E   D I S P L A Y   I M A G E              *}
{*************************************************************************}
{*  This procedure finds the x and y pixel to begin displaying the image
    based on its size and normalizes the values of g(x) within the 36
    possible greyscale levels.  It displays each individual pixel by calling
    display_pix.  A data structure of fftp and the x and y axis to begin
    displaying the image are sent to the procedure. *)
procedure display_image(var new_data : fftp;x_axis,y_axis : integer);
var
  savei,savej,p,i,j,x,offy,offx,newi,newj,mult_factor,pix : integer;
  levels,new_real,local_max,local_min,auto_scale : real;
  n_quit : boolean;
  nnx : integer;

begin
```

```
local_max := 0;
fft_block := new_data[1];
local_min := fft_block^[1,1].real_pt;

for x := 1 to blocks do
  begin
    fft_block := new_data[x];
    for j := 1 to rows do
      begin
        for i := 1 to max_samples do
          begin
            if fft_block^[i,j].real_pt > local_max then
                local_max := fft_block^[i,j].real_pt;
            if fft_block^[i,j].real_pt < local_min then
                local_min := fft_block^[i,j].real_pt;
          end; {* for i *}
      end; {* for j *}
  end; {* for x *}

auto_scale := local_max - local_min;
if (auto_scale = 0) then auto_scale := 1E00;

setfillstyle(solidfill,black);
setbkcolor(black);
bar(1,1,getmaxx,getmaxy);

if((max_samples = 16) or (max_samples = 64))then
  begin
    if(bit_size = 3)then
      begin
        mult_factor := 4;
        pix := 3;
        levels := 36;
      end
    else if(bit_size = 4)then
      begin
        mult_factor := 5;
        pix := 4;
        levels := 64;
      end;
  end
else if(max_samples = 128)then
  begin
    if(bit_size = 3)then
      begin
        mult_factor := 4;
        pix := 3;
        levels := 36;
      end;
  end;

offy := y_axis - ((mult_factor * max_samples) div 2);
offx := x_axis - ((mult_factor * max_samples) div 2);

savei := offx;
savej := offy;
```

```
  for x := 1 to blocks do
    begin
      fft_block := new_data[x];
      for j := 1 to rows do
        begin
          for i := 1 to max_samples do
            begin

              new_real := round((fft_block^[i,j].real_pt / auto_scale) * levels);

              newi := savei;
              newj := savej;

              if(new_real <= 0)then p := 0
              else if(new_real = max_pats)then p := max_pats
              else
                begin
                  n_quit := false;
                  nnx := 1;
                  Repeat
                    if(new_real = nnx)then
                      begin
                        p := nnx;
                        n_quit := true;
                      end;
                    inc(nnx);
                  Until(n_quit = true);
                end; {* else *}

              display_pix(p,newi,newj);

              savei := (savei + pix + 1);
            end; {* for i *}

          savej := (savej + pix + 1);
          savei := offx;
        end; {* for j *}
    end; {* for x *}

  setfillstyle(solidfill,blue);
  setcolor(yellow);

end; {* procedure display_image *}


{*****************************************************************************}
{*                    P R O C E D U R E   P A U S E                          *}
{*****************************************************************************}
{* This procedure halts the program by generating a software interrupt $16
   and prompt the operator to decide whether to continue, or quit.  If
   the wrong key is pressed, then beep the operator. *}
procedure pause;
var
  key : char;
  res : registers;
```

```
  return_key : boolean;
  ww : integer;

begin

   ww := getmaxx div 2;
   outtextxy((ww -156),(getmaxy - 8),'Press "return" to continue or "q" to quit.');
   return_key := false;

   {* Clear keyboard buffer *}
   res.AX := $0c06;
   res.DL := $00;
   intr($21,res);

   Repeat

     res.AH := $00;
     intr($16,res);

     if (res.AL = $0D) then
       begin
         return_key := true;
         res.AX := $0000;
         res.DX := res.AX;
         res.AX := $0600;
       end

     else if ((res.AL = $71) or (res.AL= $51))then
       begin
         quit := true;
         res.AX := $0000;
         res.DX := res.AX;
         res.AX := $0600;
       end

     else
       begin
         res.DX := $0007;
         res.AX := $0600;
         intr($21,res);
       end;

  Until((quit = true) or (return_key = true));

end; {* procedure pause *}


{*************************************************************************}
{*                    P R O C E D U R E   B E E P _ O P                   *}
{*************************************************************************}
{* This procedure generates a software interrupt $21 to beep the operator. *}
procedure beep_op;
var
  res : registers;

begin
```

```
  res.DX := $0007;
  res.AX := $0600;
  intr($21,res);
end; {* procedure beep_op *}



begin

  {* Main of Program NPLOT *}

  s := sizeof(fft_array);
  create_patterns;

  Repeat

    {* Make sure that the file is a Gerchberg-Saxton file and that it has
       GS on the first line and the iteration number on the second line.
       If is not a Gerchberg-Saxton file, then display that fact to the
       operator by an error message. *}
    clrscr;
    gotoxy(1,15);
    writeln('      Type in the name of the Gerchberg-Saxton output file you ');
    write('      wish to display : ');
    textbackground(white);
    textcolor(black);
    readln(filename);
    textbackground(black);
    textcolor(white);

    if (filename = '') then
      begin
        try_again := true;
        continue := false;
      end
    else try_again := false;

    if (try_again = false)then
      begin
        assign(opfile,filename);
        {$I-}
        reset(opfile);
        {$I+}
        if IOResult = 0 then continue := true
        else continue := false;
      end; {* if *}

    if(continue = true) then
      begin
        readln(opfile,checkline);
        close(opfile);
        try_again := false;
      end {* if *}
    else
      begin
        clrscr;
```

```
        gotoxy(1,15);
        write('         File not found.  Do you want to try again (y or n)? ');
        ch := readkey;
        if((ch = 'Y') or (ch = 'y'))then try_again := true
        else if((ch = 'N') or (ch = 'n'))then try_again := false
        else
          begin
            beep_op;
            try_again := true;
          end; {* else *}
      end; {* else *}

    if (continue = true) then
      begin
        if(checkline = 'GS OUTPUT FILE')then continue := true
        else
          begin
            continue := false;
            clrscr;
            gotoxy(1,15);
            writeln('       The file your selected, ',filename,', is not a Gerchberg-Saxton
');
            write('      output file.  Do you want to try again (y or n)? ');
            ch := readkey;
            if((ch = 'Y') or (ch = 'y'))then try_again := true
            else if((ch = 'N') or (ch = 'n'))then try_again := false
            else
              begin
                beep_op;
                try_again := true;
              end; {* else *}
          end; {* else *}
      end; {* if continue *}

  if (continue = true) then
    begin
      clrscr;
      quit := false;
      graphdriver := 0;
      graphmode := 0;
      detectgraph(graphdriver,graphmode);
      initgraph(graphdriver,graphmode,'c:\tp');
      errorcode := graphresult;
      if errorcode <> grOk then
        begin
          continue := false;
          assign(error_file,'error.dat');
          rewrite(error_file);
          writeln(error_file,errorcode);
          close(error_file);
        end;
    end; {* if continue *}

  if(continue = true)then
    begin

      nx := getmaxx div 2;
```

```
      ny := getmaxy;
      settextstyle(defaultfont,horizdir,1);
      setfillstyle(solidfill,blue);
      setbkcolor(blue);
      get_input(data,filename);

      final_iteration := 'The iteration number is ';
      final_iteration := concat(final_iteration,iteration);
      final_iteration := concat(final_iteration,'.');
      display_image(data,(getmaxx div 2),(getmaxy div 2));
      new_x := nx - 124;
      new_y := ny - 32;
      outtextxy(new_x,new_y,'This is a plot of g(x), which is');
      new_x := nx - 76;
      new_y := ny - 24;
      outtextxy(new_x,new_y,'the recovered image.');
      new_x := nx - 116;
      new_y := ny - 16;
      outtextxy(new_x,new_y,final_iteration);
      pause;

      closegraph;

      dispose_pointers(data);

      clrscr;
        gotoxy(1,15);
        write('   Do you want to select another Gerchberg-Saxton output file (y or n)? ');
        ch := readkey;
        if((ch = 'Y') or (ch = 'y'))then try_again := true
        else if((ch = 'N') or (ch = 'n'))then try_again := false
        else
          begin
            beep_op;
            try_again := true;
          end; {* else *}

    end; {* if continue = true *}

  Until (try_again = false);

  clrscr;

end. {* End Program NPLOT *}

{* END PROGRAM NPLOT.PAS *}
{*************************************************************************}
```