Investigating the Gerchberg-Saxton Phase Retrieval Algorithm

Theresa Thimons[1] and Lily Wittle[2]
Sponsor: Dr. Comlan de Souza[3]

[1] *Saint Vincent College (Latrobe, PA)*
[2] *University of Miami (Coral Gables, FL)*
[3] *California State University, Fresno (Fresno, CA)*

Correspondence: theresa.thimons@gmail.com; lily.wittle@gmail.com; csouza@mail.fresnostate.edu

April 30, 2018

1 Introduction

The phase retrieval problem originates in applied physics. In optics, light wave intensities can be measured with light detectors. In electron microscopy, the intensities of electron microscope photographs can be measured [1]. However, phase information cannot be physically measured and must be estimated to recover the complete wave function. This is known as the phase retrieval problem [2].

The Gerchberg-Saxton algorithm [3] was the first efficient solution to this problem. There are two inputs to the algorithm: the amplitudes of the image and the amplitudes of the diffraction planes (the Fourier transform of the image). In other words, given the amplitude of a signal and its Fourier transform, the algorithm attempts to recover the phase information for the Fourier transform, and thereby reconstruct the signal. The algorithm alternates between Fourier and inverse Fourier transforms, using the input amplitudes at each iteration to improve the phase estimates.

As the phase estimates are indeed estimates, there is always a level of error involved. This error is defined as the difference between the known magnitudes and the estimated magnitudes, and is computed each iteration to track the algorithm's progress. We say that the algorithm is successful when the error converges to a value within a small tolerance of zero. We present a proof that this error must decrease or remain the same with each iteration.

Gerchberg and Saxton's algorithm is not flawless. One notable problem is that the error can often decrease quickly, then stagnate for several iterations before converging to zero. Another is that the algorithm is initialized with a random set of phases, which cause inconsistency in the algorithm's output. We investigated these issues in hopes of improving the phase retrieval process.

Although Gerchberg and Saxton originally wrote their algorithm in the 1980s, phase retrieval is still a relevant and active field, as evidenced by recent research such as Osherovich's description of an efficient numerical phase reconstruction method [4]. Technological advances make accurate phase retrieval even more attainable, as computations become less of a limiting factor. Therefore improvement of Gerchberg and Saxton's algorithm would be beneficial to current phase retrieval problems and related research areas.

We wrote a numerical implementation of the algorithm in MATLAB to experiment with and observe the algorithm's performance. In our experiments,

we found that functions of the form $f \times g$, where g is a Gaussian function, have better success than those of the corresponding f. We also found that a constant initial phase estimate often produces more consistent and efficient results for non-centrosymmetric input than a random initial phase estimate.

2 Definitions

The Fourier Transform is not uniformly defined in relevant literature. Hence we clarify that we define the Discrete Fourier Transform (DFT) of f as F, where

$$F[m]_{0 \leqslant m \leqslant N-1} = \frac{1}{N} \sum_{n=0}^{N-1} f[n] \exp\left(\frac{-2\pi i n m}{N}\right). \tag{1}$$

Consequently, we define the Inverse Discrete Fourier Transform (IDFT) of F as f, where

$$f[n]_{0 \leqslant n \leqslant N-1} = \sum_{m=0}^{N-1} F[m] \exp\left(\frac{2\pi i n m}{N}\right). \tag{2}$$

With these definitions, Parseval's Identity [5] is

$$\sum_{n=0}^{N-1} |f[n]|^2 = N \sum_{m=0}^{N-1} |F[m]|^2. \tag{3}$$

In general, we use lowercase to denote the object domain, representative of the image plane, and uppercase to denote the Fourier domain, representative of the diffraction plane.

Note that MATLAB's `fft` function returns only the summation in Equation 1 and $\frac{1}{N}$ must be appended. Similarly, MATLAB's `ifft` function includes the $\frac{1}{N}$ factor, so we multiplied all IDFT calculations by N to match our definitions in Equations 1 and 2.

3 The Gerchberg-Saxon Algorithm and its Numerical Implementation

We implemented the Gerchberg-Saxton algorithm in MATLAB, using function-generated points to test the algorithm's performance. We generated an array of domain points t, sampled a complex function at these domain points in an array f (representing the image plane), and took the DFT of f to obtain an array F (representing the diffraction plane). We discarded phase information to obtain $|f|$ and $|F|$, simulating the only known information in a practical application of the algorithm. A sample of this process is shown in Listing 1.

The initial estimate is computed using the magnitude of the image points $|f|$ and randomly generated phases in $(-\pi, \pi)$. The algorithm then enters an iterative loop that terminates once the error is sufficiently small or the maximum number of iterations is reached. Cooley and Tukey's Fast Fourier Transform [6] is used for computational efficiency.

In each iteration k, the phases from the previous iteration's estimate are normalized with the known image magnitudes $|f|$ to yield x_k (Equation 4). The DFT of x_k is computed to be X_k, an estimate for the diffraction plane values (Equation 5). The phases from X_k are then normalized with the known diffraction magnitudes $|F|$ to yield Y_k (Equation 6). The IDFT of Y_k is computed to be y_k, an estimate for the image plane values (Equation 7). This process can be represented as follows:

$$x_k = |f| \exp\left(i\phi_{k-1}\right) \tag{4}$$

$$X_k = DFT(x_k) = |X_k| \exp\left(i\psi_k\right) \tag{5}$$

$$Y_k = |F| \exp\left(i\psi_k\right) \tag{6}$$

$$y_k = IDFT(Y_k) = |y_k| \exp\left(i\phi_k\right) \tag{7}$$

Error is computed twice each iteration, once for each DFT- or IDFT-generated estimate. Error is calculated as the 2-norm of the difference between the estimated magnitudes and the known magnitudes. The diffraction plane error E_k is multiplied by $\sqrt{N}$ to be comparable with the image plane error e_k.

$$E_k = \left(N \sum_{n=0}^{N-1} \left(|X_k[n]| - |F[n]|\right)^2\right)^{1/2} \tag{8}$$

$$e_k = \left(\sum_{n=0}^{N-1} \left(|y_k[n]| - |f[n]|\right)^2\right)^{1/2} \tag{9}$$

Our algorithm function is seen in Listing 2.

4 Error Convergence

Gerchberg and Saxton's algorithm has previously been generalized for other Fourier applications, including x-ray crystallography and filter design. It is known as the error-reduction algorithm [7], as the error of each estimate is no greater than that of the previous estimate. In the worst case, it equals that of the previous estimate, meaning that the error will eventually converge. Error convergence within a small tolerance δ of zero indicates success; convergence to a value greater than the tolerance indicates failure. δ can be chosen as the user likes, but we used $\delta = 10^{-10}$ in our MATLAB code. We found that this value was small enough relative to the phase differences in the functions we used that it was clear that the estimated phases were a successful estimate for the actual phases. We used Fienup's proof [8] as a foundation for our proof of the error-decreasing nature of the algorithm. He proved the same conclusion that we did, but made the general argument that for any point, the estimate x_{k+1} provides the nearest value to the value given by y_k while satisfying the magnitude constraints in the image domain. We considered this concept, depicting it visually by representing estimates as vectors in Figure 1. Our result is, in our opinion, a clearer algebraic proof that is shown below.

Theorem 1. *Let E_k be the diffraction plane error (as in Equation 8) computed in iteration k, for any integer $k \geqslant 0$, of the Gerchberg-Saxton phase retrieval algorithm. Similarly, let e_k (Equation 9) be the image plane error computed later in iteration k and E_{k+1} be the diffraction plane error computed in the subsequent iteration, iteration $k+1$. Then $E_{k+1} \leqslant e_k \leqslant E_k$, i.e. the error must not increase as the algorithm progresses.*

Proof. We first prove that $e_k \leqslant E_k$. Consider

$$\left(N \sum_{n=0}^{N-1} |X_k[n] - Y_k[n]|^2 \right)^{1/2}$$

which can be rewritten as

$$\left(N \sum_{n=0}^{N-1} \left|\exp(i\psi_k) \times (|X_k[n]| - |Y_k[n]|)\right|^2 \right)^{1/2}.$$

By multiplicativity, this equals

$$\left(N \sum_{n=0}^{N-1} \left(|\exp(i\psi_k)| \times \big||X_k[n]| - |Y_k[n]|\big|\right)^2 \right)^{1/2}$$

which can be simplified to

$$\left(N \sum_{n=0}^{N-1} \left(|X_k[n]| - |Y_k[n]|\right)^2 \right)^{1/2}.$$

As $|Y_k[n]| = |F[n]|$, this is equal to E_k (Equation 8). X_k is the DFT of x_k and Y_k is the DFT of y_k. Since DFT is a linear operator, $X_k - Y_k$ is the DFT of $x_k - y_k$. By Parseval's Identity (Equation 3),

$$E_k = \left(N \sum_{n=0}^{N-1} |X_k[n] - Y_k[n]|^2 \right)^{1/2} = \left(\sum_{n=0}^{N-1} |x_k[n] - y_k[n]|^2 \right)^{1/2}.$$

Using Equations 4 and 7, this is equal to

$$\left(\sum_{n=0}^{N-1} \left| |f[n]| \exp\left(i\phi_{k-1}\right) - |y_k[n]| \exp\left(i\phi_k\right) \right|^2 \right)^{1/2}.$$

By definition of complex exponentials, this equals

$$\left(\sum_{n=0}^{N-1} \Big| |f[n]| \cos\left(\phi_{k-1}\right) + i|f[n]| \sin\left(\phi_{k-1}\right) - |y_k[n]| \cos\left(\phi_k\right) - i|y_k[n]| \sin\left(\phi_k\right) \Big|^2 \right)^{1/2}.$$

By definition of complex magnitude, this equals

$$\left(\sum_{n=0}^{N-1} \Big(\left(|f[n]| \cos\left(\phi_{k-1}\right) - |y_k[n]| \cos\left(\phi_k\right)\right)^2 + \left(|f[n]| \sin\left(\phi_{k-1}\right) - |y_k[n]| \sin\left(\phi_k\right)\right)^2 \Big) \right)^{1/2}.$$

These squares can be expanded to obtain

$$\left(\sum_{n=0}^{N-1} \big(|f[n]|^2 + |y_k[n]|^2 - 2|f[n]||y_k[n]| \left(\cos\left(\phi_{k-1}\right) \cos\left(\phi_k\right) + \sin\left(\phi_{k-1}\right) \sin\left(\phi_k\right)\right) \big) \right)^{1/2}.$$

By the cosine difference identify, E_k equals

$$\left(\sum_{n=0}^{N-1}\left(|f[n]|^2+|y_k[n]|^2 \right.\right. \\ \left.\left. -2|f[n]||y_k[n]|\cos\left(\phi_{k-1}-\phi_k\right)\right)\right)^{1/2}.$$

We know that $\cos(\theta) \leqslant 1, \forall\ \theta$. Hence,

$$|f[n]||y_k[n]|\cos\left(\phi_{k-1}-\phi_k\right) \leqslant |f[n]||y_k[n]|,$$

$\forall\ 0 \leqslant n \leqslant N-1$. It follows from this that

$$\sum_{n=0}^{N-1} -2|f[n]||y_k[n]|\cos\left(\phi_{k-1}-\phi_k\right) \geqslant \\ \sum_{n=0}^{N-1} -2|f[n]||y_k[n]|$$

and therefore

$$E_k \geqslant \left(\sum_{n=0}^{N-1}\left(|f[n]|^2+|y_k[n]|^2-2|f[n]||y_k[n]|\right)\right)^{1/2}.$$

Equation 9 for e_k can be expanded to obtain the right hand side of the above inequality, achieving our desired result of $e_k \leqslant E_k$. It can be shown that

$$e_k = \left(\sum_{n=0}^{N-1}|y_k[n]-x_{k+1}[n]|^2\right)^{1/2}.$$

By their definitions in Equations 4 and 7, y_k and x_{k+1} have the same phase of ϕ_k, so x_{k+1} cannot be any closer to y_k while retaining the same magnitude. Since $|x_k| = |x_{k+1}| = |f|$,

$$\sum_{n=0}^{N-1}|y_k[n]-x_{k+1}[n]|^2 \leqslant \sum_{n=0}^{N-1}|x_k[n]-y_k[n]|^2,$$

equivalent to $e_k^2 \leqslant E_k^2$. By their definitions in Equations 9 and 8, $E_k \geqslant 0$ and $e_k \geqslant 0$ always, so this implies $e_k \leqslant E_k$.

We next show that $E_{k+1} \leqslant e_k$. In a similar manner as above, with Parseval's Identity (Equation 3), we can show that

$$e_k = \left(\sum_{n=0}^{N-1}|y_k[n]-x_{k+1}[n]|^2\right)^{1/2} \\ = \left(N\sum_{n=0}^{N-1}|Y_k[n]-X_{k+1}[n]|^2\right)^{1/2}.$$

This can be expanded similarly as for E_k to obtain

$$e_k = \left(N\sum_{n=0}^{N-1}\left(|F[n]|^2+|X_{k+1}[n]|^2 \right.\right. \\ \left.\left. -2|F[n]||X_{k+1}[n]|\cos\left(\psi_{k+1}-\psi_k\right)\right)\right)^{1/2}.$$

The same logic of $\cos(\theta) \leqslant 1, \forall\ \theta$ can be applied for $\theta = \psi_{k+1}-\psi_k$, giving us

$$e_k \geqslant \left(N\sum_{n=0}^{N-1}\left(|F[n]|^2+|X_{k+1}[n]|^2 \right.\right. \\ \left.\left. -2|F[n]||X_{k+1}[n]|\right)\right)^{1/2}.$$

Equation 8 for E_{k+1} can be expanded to obtain the right hand side of the above inequality, achieving the result $E_{k+1} \leqslant e_k$.

We can combine these results to obtain our desired result of $E_{k+1} \leqslant e_k \leqslant E_k$. □

5 Implementation of the Low-Frequency Filter

Gerchberg and Saxton discuss their efforts to simulate situations where some data is lost to test the algorithm's practicality. Sometimes small amplitudes may not be detected, so they applied a low-frequency filter where all points below a fractional threshold ϵ are set to zero. With this filter, Gerchberg and Saxton showed that their algorithm can still succeed in situations where small frequencies are lost [3].

We implemented this low-frequency filter in MATLAB with Listing 3, to be inserted after line 20 in Listing 1, where F is defined. Figure 2 shows an example of the effects of this filter on $|F|$.

To look at the effects of the low-frequency filter, we first ran the algorithm with unfiltered magnitudes. We see the randomly-generated initial phases used to start the algorithm in Figure 3. Figure 4 displays the estimated phases that were output by the algorithm. Success is seen in this case, where we assume no information is lost. Note that although the estimated phases are not perfect, the shape of the curve can be clearly seen. Additionally, the ending error in this run was sufficiently low to be considered

a success. However, if the algorithm had continued iterating and the error decreased further, the phase estimates would have gotten more accurate as well. We then ran the program with the same set of random initial phases and low-frequency filtered magnitudes, in order to simulate a situation where data is lost. The results can be seen in Figure 5. Although its estimate is not quite as smooth, the algorithm successfully retrieves phase data when some input data are lost. The low-frequency filter emphasizes the practicality of the Gerchberg-Saxton algorithm. It nicely shows how the Gerchberg-Saxton algorithm is robust to small noise and errors in the signal.

6 Gaussian Functions

We assert that functions of the form $f \times g$, where g is a Gaussian function, have better performance using the Gerchberg-Saxton phase retrieval algorithm than those of the corresponding f. We define our Gaussian function as

$$g = \exp\left(\frac{-x^2}{2\sigma^2}\right) \tag{10}$$

where σ determines the width of the Gaussian distribution. We define 'better performance' to be more frequent success of the algorithm and error convergence to zero in a small number of iterations. Functions of the form $f \times g$ exhibit both of these qualities in comparison to their corresponding functions f.

We generated sample points of functions of the form $f \times g$, shown in Listing 4. We used MATLAB's `gaussmf` function, which evaluates a Gaussian function at a domain of defined points. σ can be used to specify the shape of the Gaussian. Its position can also be set, though we only used Gaussians centered at 0. We experimented with various σ values, and found the best performance when the Gaussian's width corresponded to the width of the interval of primary focus.

We specifically looked at the Gaussian function for three reasons. The first is that it is real. This means that phase data is not affected by the multiplication of a function by a Gaussian. Therefore if a phase retrieval algorithm recovers the phases of $f \times g$, these phases are the same as the phases of f. Secondly, it is a smoothing operation, so small magnitudes that add noise to the phase retrieval process are eliminated by multiplying by a Gaussian. Lastly, the Fourier Transform of a Gaussian Function is another Gaussian function. We consider the Convolution Theorem:

$$\begin{aligned} \mathcal{F}(f \times g) &= \mathcal{F}(f) * \mathcal{F}(g) \\ \mathcal{F}(f * g) &= \mathcal{F}(f) \times \mathcal{F}(g) \end{aligned} \tag{11}$$

(Note that $\mathcal{F}$ represents the Fourier Transform, and $*$ represents the convolution product.) Thus when we take the Fourier Transform of $f \times g$, we obtain $\mathcal{F}(f) * \mathcal{F}(g)$, where $\mathcal{F}(g)$ is also a Gaussian function. Each Fourier Transform and Inverse Fourier Transform involves a Gaussian function, which serves to smooth the phase estimate.

The success of functions of the form $f \times g$ inspired us to investigate ways to alter input magnitudes to take on this more successful form. Ideally, we would be able to manipulate all input to be of the form $f \times g$, and we believed this to be possible by the Convolution Theorem (Equation 11).

However, the initial input of the algorithm is only the magnitudes $|f|$ and $|\mathcal{F}(f)|$. When dealing with magnitudes, the convolution theorem does not give us a usable equality, but rather the inequality

$$|\mathcal{F}(f)| * |\mathcal{F}(g)| \geqslant |\mathcal{F}(f) * \mathcal{F}(g)|,$$

which does not allow for greater success via multiplying input magnitudes by a Gaussian function. Our attempt at using this strategy is in Listing 5.

Our experimentation with the Gaussian function did not open any pathways to improving Gerchberg and Saxton's algorithm, but we concluded that functions of the form $f \times g$ yield outstanding success, demonstrated by Figures 6 and 7. Although initialized with the same random phases, Figure 6 used input from a function f where Figure 7 used $f \times g$ and had much better results. A concrete mathematical proof of this observation is underway.

7 Constant Initial Phases

In Section 1, we mentioned that random initial phases cause variability in output. This is best demonstrated by Figure 8, which compares two runs of the algorithm where the same input was used, but was initialized with two different sets of random phases. The algorithm succeeded in one

case and failed in the other, dependent only upon the random phases used as an initial estimate.

The randomness of the initial phases is not entirely necessary for the algorithm's success. Gerchberg and Saxton use random phases because when both sets of input magnitudes are centrosymmetric, a constant initial phase estimate will cause the algorithm to fail [3]. In this case, the DFTs and IDFTs will not change the phase estimates, and the algorithm's error will never decrease. In all other cases, however, using a constant initial phase is acceptable and seemingly more efficient.

We experimented with the use of a constant initial phase estimate in MATLAB. We set $\phi = C$ and tried using constants $C = 0$, $C = 1$, and $C = \pi$. The retrieved phases are centered at C. That being said, the value of C is irrelevant; the fact that a constant is used produces better performance, provided that at least one set of input magnitudes is non-centrosymmetric. Figure 9 shows that the constant phase estimate produced error convergence in far fewer iterations than the random estimate. Figures 10 and 11 compare two estimates that initialized the same input with random phases and a constant phase $\phi = 0$. As can be seen, the constant phase gave a more accurate estimate. The algorithm consistently resulted in this estimate with $\phi = C$, whereas random phases resulted in varied estimates.

The examples in the aforementioned figures demonstrate that the algorithm performs better with a constant initial phase estimate. We found this to be true in all non-centrosymmetric experimental cases in which Gerchberg and Saxton's algorithm can succeed. We conjecture that it is more efficient to use a constant initial phase estimate, but have not yet formally proved this.

Open Questions

- Prove that it is more practical to begin the Gerchberg-Saxton algorithm with an initial constant phase estimate, provided that at least one of the sets of input magnitudes is non-centrosymmetric.
- Consider the use of constant initial phases with a small random perturbation. These would not cause the issues that constant initial phases precipitate in centrosymmetric problems. Their performance could be compared to that of the random initial phases.
- Thoroughly investigate the multiplication by a Gaussian using a wider variety of functions f, and prove that functions of the form $f \times g$ are more consistently successful and succeed in fewer iterations than those of the corresponding f.

Funding Information

NSF Grant #DMS-1460151.

Acknowledgments

We would like to thank the California State University, Fresno Mathematics REU and our mentor, Dr. Comlan de Souza, for his support.

References

[1] D. L. Misell, *A Method for the Solution of the Phase Problem in Electron Microscopy,* J. Phys. D: Applied Physics **6:1** (1973), pp. L6–L9.

[2] G. Taylor, *The Phase Problem*. Acta Crystallogr. **D59:11** (2003), pp. 1881–1890.

[3] R. W. Gerchberg and W. O. Saxton, *A Practical Algorithm for the Determination of Phase from Image and Diffraction Plane Picture,* Optik **35:2** (1982), pp. 237–246.

[4] E. Osherovich, *Numerical Methods for Phase Retrieval* (PhD thesis, 2011). Retrieved from Cornell University Library. (arXiv: 1203.4756)

[5] M. Parseval des Chênes, *Mémoire sur les Séries et sur l'Intégration Complète d'une Équation aux Différences Partielles Linéaire du Second Ordre, à Coefficients Constants,* Mémoires Présentés à l'Institut des Sciences, Lettres et Arts, par Divers Savants, et Lus dans ses Assemblées. Sciences, Mathématiques et Physiques **1** (1806), pp. 638–648.

[6] J. W. Cooley and J. W. Tukey, *An Algorithm for the Machine Calculation of Complex Fourier Series,* Mathematics of Computation **19** (1965), pp. 297–301.

[7] J. R. Fienup, *Reconstruction and Synthesis Applications of an Iterative Algorithm,* Transformations in Optical Signal Processing **373** (1981), pp. 147–160.

[8] J. R. Fienup, *Phase Retrieval Algorithms: A Comparison,* Appl. Optics **21:15** (1982), pp. 2758–2769.

MATLAB Code Excerpts

Listing 1: *Initializing Function Points*

```
% define sampling rate
delta = 1/256;
% max number of iterations
maxiter = 500;
% sample domain points in [-.5, .5)
t = -.5:delta:.5-delta;
% number of points (used in FT calculations)
N = length(t);

% define rect function to limit domain
syms x;
r = piecewise(x<-.5, 0, x>=-.5 & x<=.5, 1, x>.5, 0);
% sample object domain points
full_f = double(subs(r, 2*t)).*exp(30i*pi*t.^2);
% calculate known phases for comparison
phases = angle(full_f);
% |f|
f = abs(full_f);
% |F|
F = abs(fft(full_f)/N);

% random phases to start algorithm
phi = 2*pi*rand(1,N)-pi;
```

Listing 2: *Gerchberg-Saxton Algorithm*

```
function estimate = gs(f, F, phi, maxiter)
% param f : |sampled points|
% param F : |FT of sampled points|
% param phi : intial phase estimate
% param maxiter : max number of iterations
% return estimate : 3 row array
% (phase estimates, iterative error, # of iterations to success or max)

% number of points (used in FT and error calculations)
N = length(f);
% define error tolerance for success
error_tol = 1e-10;
% to keep track of error through iterations
error = zeros(2, maxiter);
% 1st estimate with given phases
x = f.*exp(1i.*phi);

k=1;
% repeat algorithm until convergence or max number of iterations
while(k == 1 || (error(k-1)> error_tol && k<maxiter+1))
    % eq 5
    X = fft(x)/N;
    % Fourier domain error
    error(1, k) = sqrt(N)*norm(abs(X) - F);
    % eq 6
    Y = F.*exp(1i.*angle(X));

    % eq 7
    y = N*ifft(Y);
    % object domain error
    error(2, k) = norm(abs(y) - f);
    % eq 4
    x = f.*exp(1i.*angle(y));

```

```
    % increment index
    k = k+1;
end

% ending estimated phases
est_phases = angle(x);
estimate(1, 1:length(est_phases))
   = est_phases;
% combine errors into single array
estimate(2,1:2*(k-1)) = reshape(
   error(1:2,1:k-1), 1, 2*(k-1));
% number of iterations it took to
   reach success/max
estimate(3,1) = k-1;
end
```

Listing 3: *Low-Frequency Filter*

```
% define threshold at which data
   is lost
F_threshold = 1/30;
% find max value of |F|
F_max = max(F);
% find indices where data is lost
indices = find(abs(F)<F_threshold*
   F_max);
% set to 0
F(indices) = 0;
```

Listing 4: *Generating Functions of the Form $f \times g$*

```
% sample a Gaussian at domain
   points
% sigma = .07, center = 0,
g = gaussmf(t, [.07 0]);
% function is of the form g x
   full_f
f_g = abs(full_f.*g);
% FT of g x full_f
F_g = abs(fft(full_f.*g)/N);
```

Listing 5: *Attempt to Replicate $f \times g$ with Convolution Product*

```
% sample a Gaussian at domain
   points
% centered at 0, sigma = .07
g = gaussmf(t, [0 .07]);
% f and F defined as usual
f = abs(full_f);
F = abs(fft(full_f)/N);
% pointwise multiplication of
   object domain points
f_g = f.*g;
% FT of Gaussian
G = fftshift(fft(g)/N);
% convolution product
C = conv(F, G);
% wrap convolution product to get
   F_g of correct length
for k=1:N-1
    F_g(k) = abs(C(k)+C(N+k));
end
F_g(N) = abs(C(N));
```

Figures

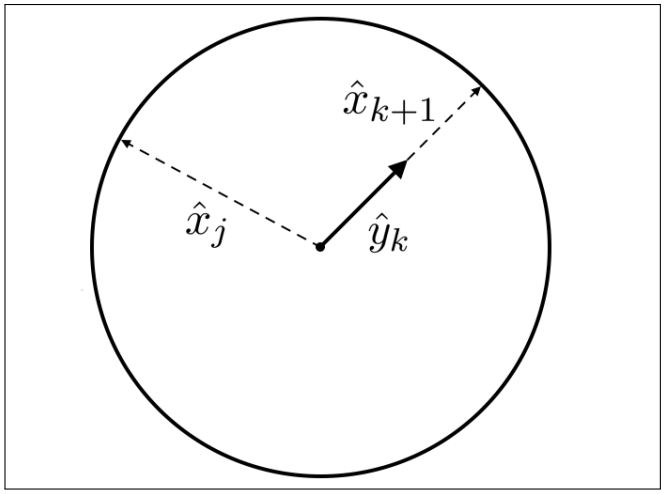

Figure 1: *As estimates y_k and x_{k+1} have the same phase ϕ_k, they can be graphically depicted as vectors pointing in the same direction. Any vector having the same magnitude as $\hat{x}_{k+1}$ must lie on this circle. None of these vectors $\hat{x}_j$ for any $j \neq k+1$ can be closer to $\hat{y}_k$ than $\hat{x}_{k+1}$.*

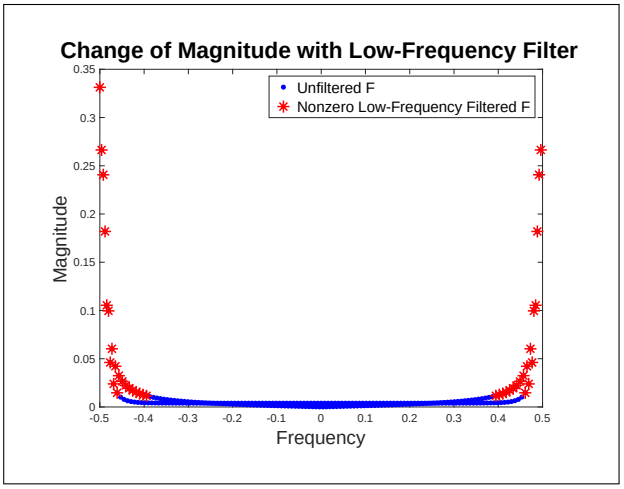

Figure 2: *Input magnitudes $|F|$ in the diffraction plane (Fourier Transform of $f(t) = \text{rect}(2t)\exp\left(15i\pi t^2\right)$). When a low-frequency filter ($\epsilon = 1/30$) was applied, only red points ($\ast$) were retained. All blue points ($\cdot$) were set to zero.*

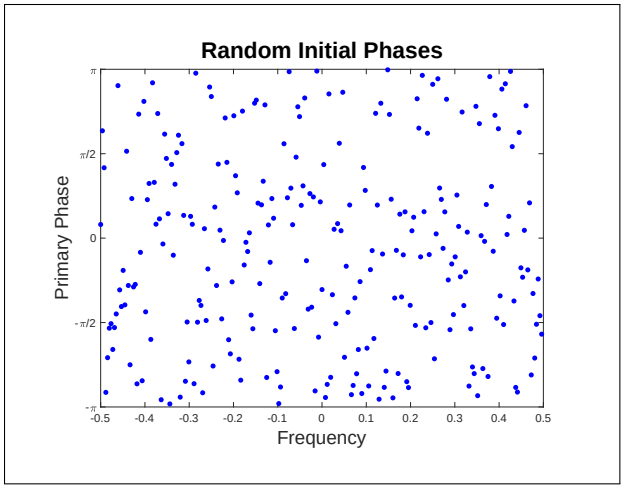

Figure 3: *The algorithm was initialized with a set of initial phases in $(-\pi, \pi)$ that are randomly generated.*

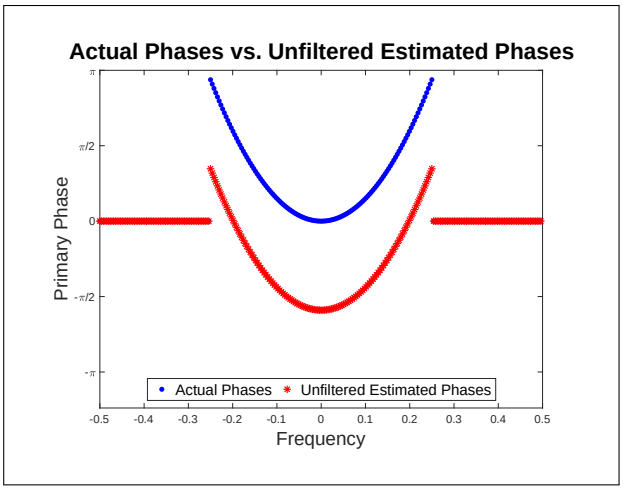

Figure 4: *A successful unfiltered estimate for input magnitudes corresponding to points generated by the function $f(t) = \text{rect}(2t)\exp\left(15i\pi t^2\right)$.*

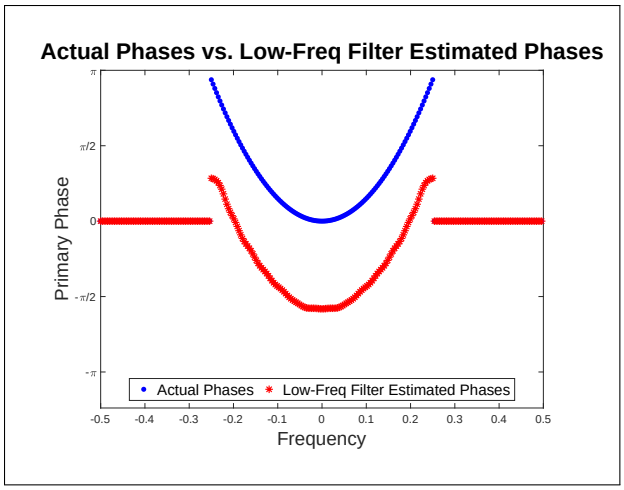

Figure 5: *Phase estimate for magnitudes corresponding to points generated by the function $f(t) = \text{rect}(2t)\exp\left(15i\pi t^2\right)$, when magnitudes below threshold $\epsilon = 1/30$ were filtered by our low-frequency filter.*

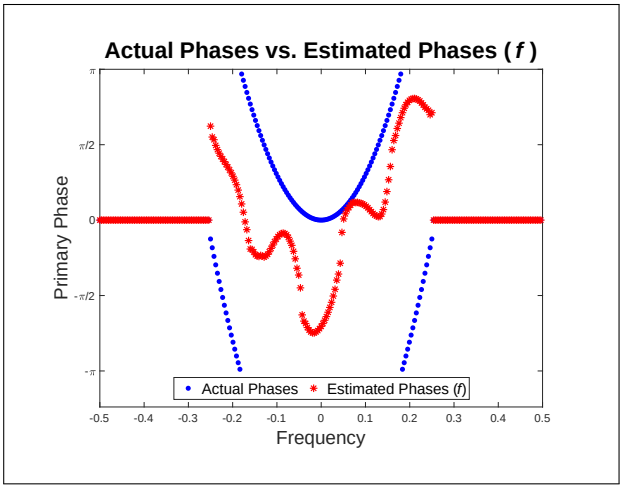

Figure 6: *Phase retrieval estimate for an input function* $f(t) = \text{rect}(2t)\exp\left(30i\pi t^2\right)$ *not multiplied by* g.

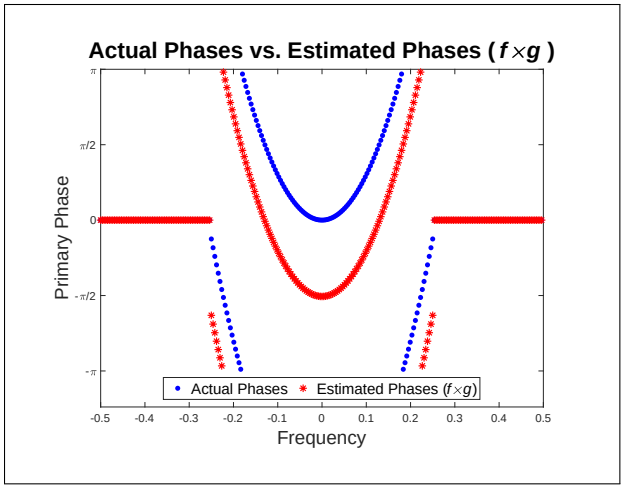

Figure 7: *Phase retrieval estimate for an input function of the form* $f \times g$, *where* $f(t) = \text{rect}(2t)\exp\left(30i\pi t^2\right)$, *and* $g = \exp\left(\frac{-x^2}{2(.07)^2}\right)$.

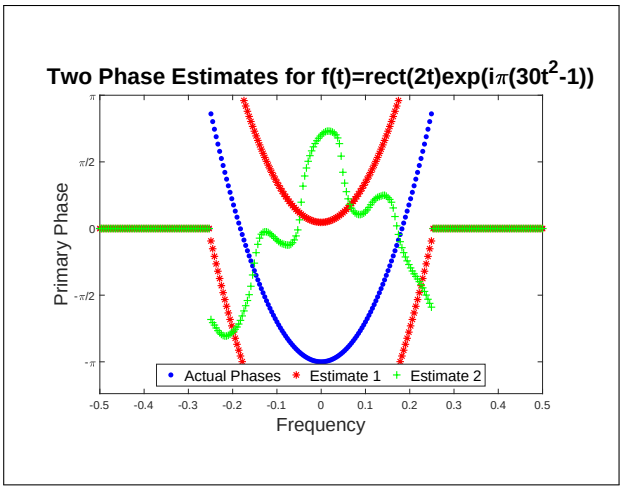

Figure 8: *A comparison of the results of the algorithm from two runs initialized with the same function input but two different sets of random phases. The algorithm produced success in one run and failure in another, based only upon the initial phase estimates.*

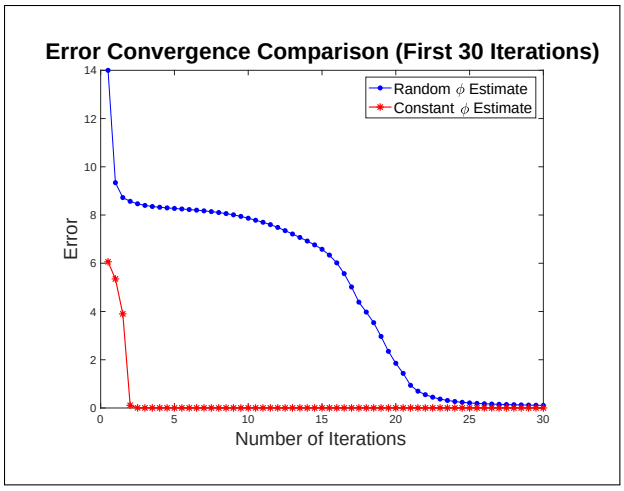

Figure 9: *A comparison of error convergence of the same input* ($f(t) = \text{rect}(2t)\exp\left(20i\pi t^3\right)$), *when one set used an initial random phase estimate (blue* $\cdot$*) and the other used an initial constant phase estimate (red* $*$*). Both succeeded in this case, but the constant initial phase produced success in fewer iterations.*

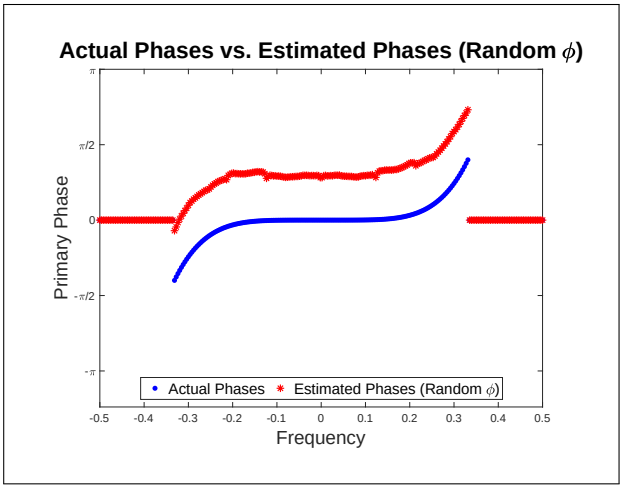

Figure 10: *Phase estimate for magnitudes corresponding to points generated by the function* $f(t) = \mathrm{rect}(1.5t)\exp\left(100i\pi t^5\right)$*, when the algorithm used an initial random phase estimate.*

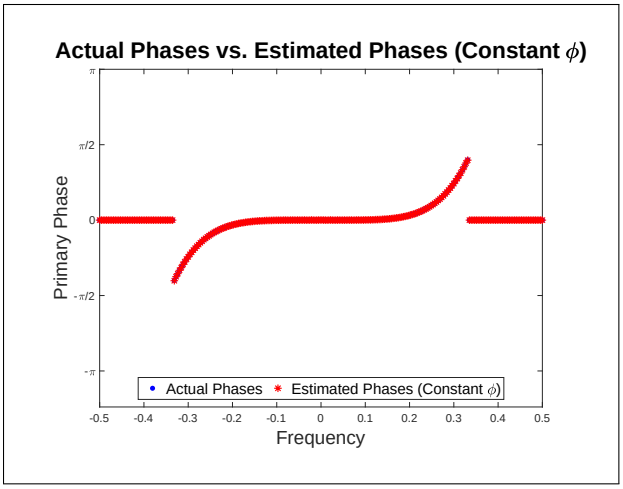

Figure 11: *Phase estimate for magnitudes corresponding to points generated by the function* $f(t) = \mathrm{rect}(1.5t)\exp\left(100i\pi t^5\right)$*, when the algorithm used an initial constant phase estimate.*