AMSC 663/664 Project Proposal

Memory Efficient Signal Reconstruction from Phaseless Coefficients of a Linear Mapping

Naveed Haghani
nhaghan1@math.umd.edu

Project Advisor:
Dr. Radu Balan
rvbalan@cscamm.umd.edu
Professor of Applied Mathematics, University of Maryland
Department of Mathematics
Center for Scientific Computation and Mathematical Modeling
Norbert Weiner Center

Table of Contents

Introduction 2
Background 2
Problem Setup 2
Transformation 4
Algorithm 4
Initialization 4
Iteration 5
Implementation 6
Data Creation 6
Principal Eigenvalue (Initialization) 7
Conjugate Gradient (Iteration) 7
Coding 8
Testing 9
Validation 9
Timeline 10
Deliverables 11
References 11

Introduction

Background

A recurring problem in signal processing involves signal reconstruction using the magnitudes of the coefficients of a linear transformation. This problem has applications in the fields of speech processing and x-ray crystallography. In speech processing, it is common to work with a speech signal's spectrogram. Working with the spectrogram provides the ability to perform various audio manipulations. The challenge then becomes to retrieve a processed signal's discrete-time signal, as the spectrogram does not carry any phase information with regards to the signal. In x-ray crystallography, the diffraction pattern of an x-ray beam will deliver the magnitudes of a transformed signal of electron density levels. Obtaining the desired electron density information requires the phaseless retrieval of the original signal.

The project proposed in this paper will implement and test an iterative, recursive least squares algorithm described in Balan[5] to perform phaseless reconstruction from the magnitudes of the coefficients of a linear transformation. Testing will be done on synthetically generated input data created using random number generation. A random input vector will be generated and passed through a transformation algorithm. The transformed signal will then be passed to the iterative, recursive least squares algorithm to reconstruct the original signal. Following that, post processing will be done on the results.

The implementation will be programmed in MATLAB. The implementation will be designed to prioritize memory efficiency. Memory efficiency, in this regard, applies primarily to the storage of the resulting linear system involved in reconstruction. The linear system will be on the order of $10{,}000 \times 10{,}000$. Avoiding the costly storage of this system and deriving its contents when needed will be the primary focus during implementation of the algorithm. Following the algorithm's completion, the program's performance will be studied with regards to time efficiency, accuracy, and scalability with problem size.

Problem Setup

Given an n-dimensional complex signal, $x \in \mathbb{C}^n$, that has been passed through a linear transformation, $T(x)$, the objective is to is to reconstruct x from the element by element modulus of the transformation vector. The transformation vector will be labeled

$$T(x) = c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_m \end{bmatrix} \in \mathbb{C}^m \tag{1}$$

The transformed signal lies in the m dimensional complex space, where $m = R \cdot n$. R here represents the level of redundancy within the transformation $T(x)$.

The element by element modulus of c is represented by α:

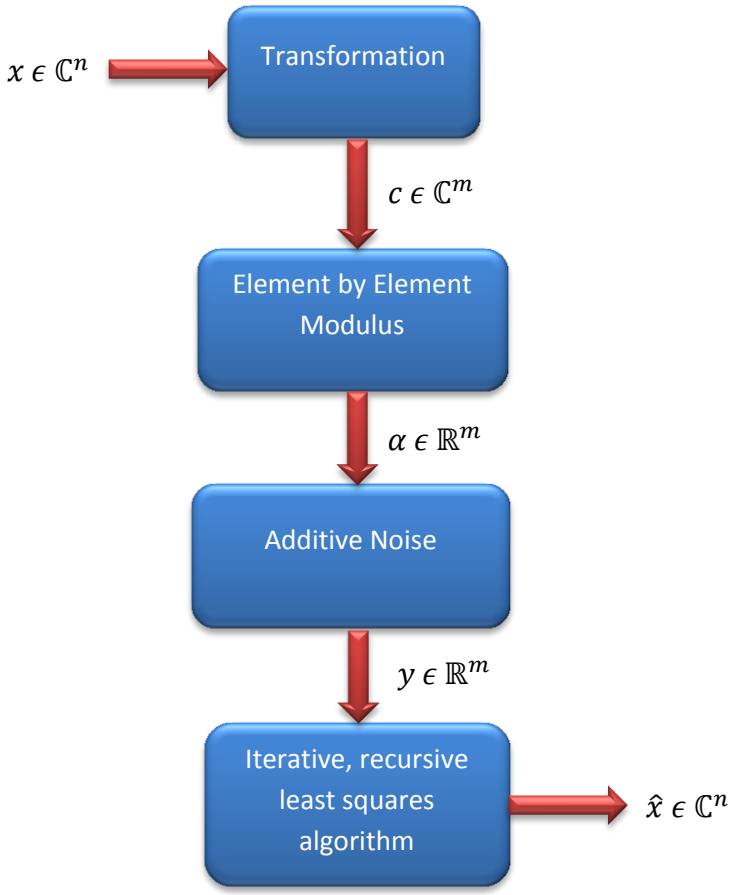
$$\alpha = \begin{bmatrix} |c_1|^2 \\ |c_2|^2 \\ \vdots \\ \vdots \\ |c_m|^2 \end{bmatrix} \epsilon \, \mathbb{R}^m \tag{2}$$

c has been transformed into the real space to produce α. Since it lies in the real space, α doesn't carry any phase information of the original signal, fitting the criterion for phaseless reconstruction.

The resulting vector will be passed into the iterative, recursive least square algorithm, but not before adding a variable amount of Gaussian noise. The resulting input to the algorithm is labeled y and is defined by:

$$y = \alpha + \sigma \cdot \nu \tag{3}$$

Where ν is Gaussian noise and σ is a weight factor used to achieve a desired signal to noise ratio for testing. y is the vector of transformation magnitudes with simulated noise. The iterative, recursive least squares algorithm will use the input y to produce an approximation of x, labeled $\hat{x}$. The entire process works as follows:



After $\hat{x}$ is obtained, the estimation is passed into a post processing framework to study certain output characteristics, namely certain trends with regards to varying signal to noise ratios in y.

Transformation

The transformation that will be used in the implementation is a weighted discrete Fourier transform. First each element of x is multiplied by a complex weight, w_i. Then the discrete Fourier transform is taken on the resulting vector. This is repeated R times, each time with an independent set of weights.

$$B_j = Discrete\ Fourier\ Transform\left\{\begin{bmatrix} w_1^{(j)} & & 0 \\ & \ddots & \\ 0 & & w_n^{(j)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}\right\}, for\ j = [1, R] \tag{4}$$

The resulting transformation output c is a composite of each of the $B_{[1,R]}$ transformations, making c lie in the $m = R \cdot n$ complex dimensional space.

$$c = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \vdots \\ B_R \end{bmatrix} \epsilon\ \mathbb{C}^m \tag{5}$$

For the sake of this study, R will be set to 8.

After c is obtained from the weighted discrete fourier transform, α is obtained by taking the modulus squared of each element of c. Finally, Gaussian noise is added to α to produce y, the input to the iterative, recursive least squares algorithm.

Algorithm

The reconstructive algorithm to be implemented has been introduced and described in Balan[5]. It consists of two primary processes, the initialization and the iterative solver. The algorithm serves as a least squares solver that is designed to minimize $\|y - \hat{\alpha}\|^2$, where $\hat{\alpha}$ is the α value in equation (2) obtained from inputting the current estimation, $\hat{x}$, into the preprocessing transformation.

Initialization

Initialization starts with finding the principal eigenvalue, a_1, and its associated eigenvector, e_1, of a matrix Q defined by:

$$Q = \sum_{k=1}^{m} y_k f_k f_k^* \ [5] \tag{6}$$

Where f_k is the k[th] frame vector from the transformation $T(x)$. For the weighted discrete Fourier transform f_k is defined by:

$$f_k = \frac{1}{\sqrt{R \cdot n}} \begin{bmatrix} w_1^{(j)} \cdot 1 \\ w_2^{(j)} e^{-i2\pi j \cdot 1/n} \\ \vdots \\ \vdots \\ w_n^{(j)} e^{-i2\pi j \cdot (n-1)/n} \end{bmatrix} where\ j = ceiling\left(\frac{k}{R}\right) \quad (7)$$

Before the principal eigenpair is retrieved, the following modification is performed on Q:

$$Q^+ = Q + q \cdot I \quad (8)$$
$$where\ q = \|y\|_\infty, \qquad I = identity$$

This modification ensures that Q^+ is positive definite, subsequently ensuring that $a_1 > 0$ and that the power method for finding the principal eigenvector will converge.

Once this eigenpair is discovered the first estimation, $\hat{x}^{(0)}$, can be initialized as:

$$\hat{x}^{(0)} = e_1 \sqrt{\frac{(1-\rho) \cdot a_1}{\sum_{k=1}^{m} |\langle e, f_k \rangle|^4}} \ [5] \quad (9)$$
$$where\ \rho\ is\ a\ constant\ bewteen\ 0\ and\ 1$$

Two additional parameters, μ and λ, are initialized as:

$$\mu_0 = \lambda_0 = \rho \cdot a_1 \ [5] \quad (10)$$

After initialization, the algorithm moves on to the iterative process.

Iteration

Through each pass of the iterative process a linear system is solved to obtain a new approximation $\hat{x}$. The linear system is constructed in the real space. Instead of working with $\hat{x}$, the algorithm works with $\xi = \begin{bmatrix} real(\hat{x}) \\ imag(\hat{x}) \end{bmatrix}$, the composite of the real values of $\hat{x}$ and the imaginary values of $\hat{x}$. The linear system which is symmetric and positive definite is defined as:

$$A\xi^{(t+1)} = b$$

$$where\ A = \sum_{k=1}^{m} \left(\Phi_k \xi^{(t)}\right) \cdot \left(\Phi_k \xi^{(t)}\right)^* + (\lambda_t + \mu_t) \cdot 1 \ [5] \quad (11)$$

$$b = \left(\sum_{k=1}^{m} y_k \cdot \Phi_k + \mu_t 1\right) \cdot \xi^t \text{ [5]} \tag{12}$$

$$\Phi_k = \varphi_k \varphi_k^T + J\varphi_k \varphi_k^T J^T, \text{ where } \varphi_k = \begin{bmatrix} real(f_k) \\ imag(f_k) \end{bmatrix} \text{ and } J = \begin{bmatrix} 0 & -I \\ I & 0 \end{bmatrix} \text{ [5]} \tag{13}$$

$\xi^{(t)}$ is the composite of the real and imaginary components of the current approximation $\hat{x}^{(t)}$, and $\xi^{(t+1)}$ is the composite of the real and imaginary components of the next approximation $\hat{x}^{(t+1)}$. Following that, the parameters are updated for the following iteration:

$$\lambda_{t+1} = \gamma \lambda_t \text{ [5]} \tag{14}$$

$$\mu_{t+1} = max(\gamma \mu_t, \mu^{min}) \text{ [5]} \tag{15}$$

$$where\ 0 < \gamma < 1 \tag{16}$$

This process is repeated until the following stopping criterion is met:

$$\sum_{k=1}^{m} \left| y_k - \left| \langle x^{(t)}, f_k \rangle \right|^2 \right|^2 \leq \kappa m \sigma^2, \text{ where } \kappa \text{ is a constant} > 1 \text{ [5]} \tag{17}$$

This stopping criterion is essentially checking whether $\|y - \hat{\alpha}\|^2$ is below a given tolerance.

Implementation

Data Creation

The complex input vector $x \in \mathbb{C}^n$ will be generated synthetically using random number generation. Each element of x will consist of a randomly generated normal component and a randomly generated imaginary component. Both random numbers will be distributed normally about 0 with variance 1. n, which is the vector length of x, will be on the order of 10,000. 10 different realizations of x will be generated and saved for repeated use.

The weights used in the weighted transformation will also be synthetically generated using random number generation. Each element of w will have a random normal component and a random imaginary component, each distributed normally about 0 with variance 1. There will be 10 different realizations of each set $w^{([1,R])}$.

The noise, ν, added to α to produce y will be generated randomly as well. Each element will be distributed normally about 0 with variance 1. There will be 10,000 different realization of noise, ν.

Principal Eigenvalue (Initialization)

During the initialization stage of the iterative, recursive least squares algorithm, the principal eigenvalue of a matrix Q^+ must be calculated. To achieve this, the power method for obtaining the principal eigenvalue will be used. The power method starts with an initial approximation of the associated eigenvector, $e^{(0)}$. For the purposes of this implementation, $e^{(0)}$ will be set to an array of random numbers. Each element will be distributed normally about 0 with variance 1.

From $e^{(0)}$ the algorithm will repeat as follows:

$$Repeat\colon e^{(t+1)} = \frac{Q^+ \cdot e^{(t)}}{\left\| Q^+ \cdot e^{(t)} \right\|}$$

$$where\ e^{(t)}\ is\ the\ current\ approximation, e^{(t+1)}\ is\ the\ following\ approximation$$

$$stop\ when\ \left\| e^{(t+1)} - e^{(t)} \right\| < tolerance$$

With the selection of an appropriate tolerance, this algorithm should produce an adequate approximation for the principal eigenvalue and its associated eigenvector that is used during the initialization stage of the least squares algorithm.

Conjugate Gradient (Iteration)

Through each iteration of the iterative, recursive least squares algorithm, a $2n \times 2n$ linear system must be solved. This would be cumbersome to solve exactly and would jeopardize the priority of memory efficiency. Instead, the conjugate gradient method of solving linear systems will be used. The conjugate gradient method is an iterative method for solving symmetric, positive definite linear systems.

The conjugate gradient method works by taking the residual of an approximate solution to a linear system and reducing it by moving the solution along several different conjugate directions. Two vectors p^1 and p^2 are considered to be conjugate with respect to a linear system A if they satisfy the following condition:

$$p^{1^T} \cdot A \cdot p^2 = 0$$

For a given matrix in $\mathbb{R}^n$, there are always n linearly independent conjugate directions. Traveling along all directions produces the exact solution to the system. However, if during that time the iterations converge to within a given tolerance of the solution, the process can be concluded at that time with a sufficient approximation.

The algorithm will be initialized as follows:

$$r^{(0)} = b - A\hat{x}^{(0)}$$

$$p^{(0)} = r^{(0)}$$

Where $\hat{x}^{(k)}$ is the approximate solution at the k[th] iteration, $r^{(k)}$ is the residual at the k[th] iteration, and $p^{(k)}$ is the k[th] conjugate direction.

Each iteration repeats as follows:

$$repeat: \alpha = \frac{\langle r^{(k)}, r^{(k)} \rangle}{{p^{(k)}}^T A p^{(k)}}$$

$$\hat{x}^{(k+1)} = \hat{x}^{(k)} + \alpha p^{(k)}$$

$$r^{(k+1)} = r^{(k)} - \alpha A p^{(k)}$$

$$p^{(k+1)} = r^{(k+1)} + p^{(k)} \frac{\langle r^{(k+1)}, r^{(k+1)} \rangle}{\langle r^{(k)}, r^{(k)} \rangle}$$

$$until \left\| r^{(k)} \right\|^2 < tolerance$$

In each iteration, the solution moves along the conjugate direction $p^{(k)}$ a distance α. The iterations are repeated until the magnitude of the residual of the current approximation is less than a given tolerance.

Coding

The entire algorithm from preprocessing through post processing will be implemented in MATLAB. The iterative, recursive least squares algorithm will be programmed to run in parallel on different input vectors. This will increase the time efficiency of the program as it runs over multiple input data sets.

To implement the discrete Fourier transform, MATLAB's $fft()$ function will be used. $fft()$ implements a fast Fourier transform. Random numbers will primarily be generated using MATLAB's $rand()$ command. For generating the random noise variants however, a linear congruential generator will be implemented. The linear congruential generator works as follows:

$$Z_{k+1} = (aZ_k + c)(mod\ M)$$

Here M is the modulus, a is the multiplier, c is the increment, and Z_0 is the seed. For a given integer Z_k, the associated random number U_k would equal:

$$U_k = \frac{Z_k}{M}$$

Using a linear congruential generator has the advantage that if the same seed is used in different testing implementations, then the same exact sequence of random numbers would be produced in each case. Since there will be 10,000 different realizations of noise vectors, it will be beneficial to generate them each time the program runs, rather than saving and loading each realization.

To preserve memory efficiency, implementing the iterative, recursive least squares algorithm will be done without storing the entire $2n \times 2n$ linear system. This is made possible by the use of the conjugate gradient solver. Whenever a specific vector of the matrix is needed for computation, that vector will be computed and delivered on the spot. It will still be necessary to store various vectors of length n or $2n$, but as long as the storage remains on that order then the desired storage goals will be preserved.

Testing

There will be 10 different sets of input data, each of which can be passed through 10 different uniquely weighted transformations to produce a unique α. The vector y is generated by adding noise to α. The noise vector v can be weighted by σ to produce a desired signal to noise ratio in y.

The goal in testing will be to test each input on a multiple of signal to noise ratio levels ranging from -30 decibels to 30 decibels, in 5 decibel increments. The appropriate signal to noise ratio will be set by adjusting σ in the following equation:

$$SNR_{dB} = 10 \cdot log_{10}\left[\frac{\sum_{k=1}^{m}|c_k|^2}{\sigma^2 \sum_{k=1}^{m}|v_k|^2}\right] \tag{18}$$

Given a certain transformed input, there are 10,000 different noise variations that can be used to for each signal to noise ratio level. This will produce 10,000 output samples for a specific input at a given signal to noise ratio level. From this data, the mean squared error of the output can be studied in relation to the signal to noise ratio. As well, the bias of the mean of the output and the variance of the output can be studied against the signal to noise ratio.

Validation

Validation for the iterative, recursive least squares implementation can be done on the individual modules within the algorithm, including the power method implementation and the conjugate gradient implementation. Using a smaller sample data set with n on the order of 100 rather than 10,000, the power method can be substituted with MATLAB's $eig()$ function. $eig()$ will reliably deliver the principal eigenvalue that was sought after by the power method implementation. The power method implementation can then be run on the same sample data in order to compare the results. If the results are comparable, the power method module will be validated.

A similar procedure can be done for the conjugate gradient implementation. On a small data set with n on the order of 100, the conjugate gradient module can be substituted with MATLAB's $mldivide()$. $mldivide()$ will provide the exact solution to the linear system. This exact solution can be used to compare with the results obtained using the conjugate gradient implementation. Comparable results would provide validation. The conjugate gradient implementation can be further validated on large data sets as well. This is done by letting the conjugate gradient run through all possible iterations. For a

system of size $n \times n$, the conjugate gradient method ensures absolute convergence to the true solution in n steps. Rather than returning a result within a certain tolerance, the implementation can be made to run through all iterations regardless. The result will serve as the true solution to validate against.

Timeline

October	▪ Post processing framework ▪ Database generation
November	▪ MATLAB implementation of iterative recursive least squares algorithm
December	▪ Validate modules written so far
February	▪ Implement power method ▪ Implement conjugate gradient
By March 15	▪ Validate power method and conjugate gradient
March 15 – April 15	▪ Test on synthetic databases ▪ Extract metrics
April 15 – end of semester	▪ Write final report

Deliverables

The project will produce several deliverables. They are listed below:

- Proposal presentation
- Written proposal
- Midterm presentation
- Final presentation
- Final report
- MATLAB program
- Input data
- Output data
- Output charts and graphs

References

[1] Allaire, Grâegoire, and Sidi Mahmoud Kaber. *Numerical linear algebra*. Springer, 2008.

[2] R. Balan, On Signal Reconstruction from Its Spectrogram, Proceedings of the CISS Conference, Princeton, NJ, May 2010.

[3] R. Balan, P. Casazza, D. Edidin, On signal reconstruction without phase, Appl.Comput.Harmon.Anal. 20 (2006), 345-356.

[4] R. Balan, Reconstruction of signals from magnitudes of redundant representations. 2012.

[5] R. Balan, Reconstruction of signals from magnitudes of redundant representations: the complex case. 2013.

[6] Christensen, Ole. "Frames in Finite-dimensional Inner Product Spaces."*Frames and Bases*. Birkhäuser Boston, 2008. 1-32.

[7] Shewchuk, Jonathan Richard. "An introduction to the conjugate gradient method without the agonizing pain." (1994).