Unmixing 4-D Ptychographic Images
Part A: Algorithmic Approach

Students: Michaela Shoffner(UTK), Zhen Zhang(CUHK), and Huanlin Zhou(CUHK)
Mentors: Dr. Richard Archibald(ORNL), Dr. Azzam Haidar(UTK), Dr. Stanimire Tomov(UTK), and Dr. Kwai Wong(UTK)

Abstract

Fast electron detectors are gaining ground in traditional high-resolution microscopy studies. In particular, 4D ptychographic datasets collected over a range of real and reciprocal space coordinates are believed to contain a wealth of information about structure and properties of materials. However, currently available data analysis methods are either too general, only allowing for analysis of simplest objects, or too reductive, effectively recreating traditional detectors from these datasets before interpretation. This project aims to explore the ways that symmetry mode analysis, the tool used to a great effect in theoretical studies of materials, can be adapted to analyze 4D datasets of materials such as multifunctional complex oxides.

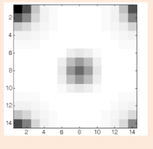
Goals

Given Matlab code using the least square model unmixing method and three sample base-model data files, we seek to improve upon the baseline program. Our primary two goals are to implement an improvement to the current unmixing algorithm, and to improve speed and performance by converting the program to C code, using LAPACK, and then having it run on a GPU, using MAGMA.

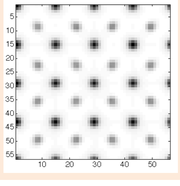
In the case of our problem, it can be formulated as

$$\alpha M_1 + \beta M_2 + \gamma M_0 = b$$

Where α, β, and γ are the coefficients we're trying to find, each M is a matrix representing one of the baseline models, and b is the given image. Represented as a grayscale picture, each image looks something like this ↓,



with the complete set of 16 here.



Research Questions

Since our example data has known coefficients, yet the original program does not find particularly close numbers, our task is to seek ways to get greater accuracy in the calculated values.

Starting from a basic least squares algorithm and a basic linear model, we began implementing various methods to solve this unmixing problem.

Process and Algorithms

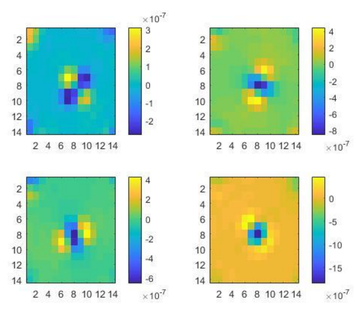
In the attempt to better solve for the coefficients, we used three different methods, but the workflow for each was still quite similar:

Least Squares

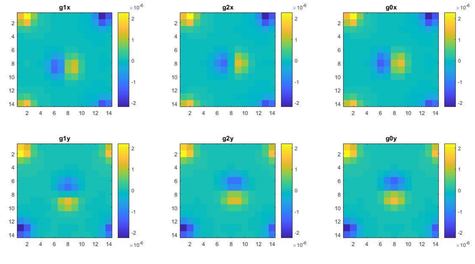
For A = [M_1 M_2 M_3] , b is the final image data, and x = $\begin{bmatrix}\alpha\\ \beta\\ \gamma\end{bmatrix}$, the basic least squares is implemented by changing $Ax = b$ into $A^TAx = A^Tb$, as this allows for the overdetermined matrix to be solved easily using LAPACK functions, but since it can be greatly affected by outliers and there is something missing in the simple linear model, it is also the least accurate method.

Simplified Least Squares with gradient

4 representative biases:
$\alpha M_1 + \beta M_2 + \gamma M_0 - b$,
α,β,γ : the true weights



Gradients:
(of the 3 modes)



As seen in the figures, both bias and gradient are highly symmetric and have similar patterns, so the gradient might be the missing part. Thus, we design a new model by including the gradients, formulated as

$$b = \alpha M_1 + \beta M_2 + \gamma M_0 + a * g1x + b * g2x + c * g0x + d * g1y + e * g2y + f * g0y$$

Split-Bregman Method

This method is used to solve the L1-regularized least squares problem: apart from solving the basic least square form, we also want to minimize the L1-norm of the gradient of the resulting weight matrices, so that the resulting weights are more piecewise constant, just as the true weights appear.

Results

All of our programs are run on the Bridges system.

Both simplified least squares with gradient and the Split-Bregman get greater accuracy than basic least squares, though they are quite similar to each other, with Split-Bregman being slightly better.

The total difference in results from actual values is:

>> 11.6405 For basic least squares
>> 2.9635 For simplified least squares with gradient
>> 2.8782 For the Split-Bregman method

As for speed, both of the latter two methods have very similar speeds and are faster than the equivalent implementation for basic least squares.

	Basic Least Squares	Simplified LS with Gradient	Split-Bregman (20 iterations)
Matlab	~ 33 seconds	~ 21 seconds	~21 seconds
CPU, using LAPACK	~ 7 seconds	~ 0.75 seconds	~ 0.76 seconds
GPU, using MAGMA	~ 50 seconds	~ 4 seconds	In-progress
CPU, using LAPACK and MPI	In-progress	~ 3 seconds	In-progress

Future Work

Next steps include:

- Working with OpenACC to parallelize the code
- Debugging and further streamlining what currently works
- Getting the MPI least squares and GPU simplified least squares plus gradient working

Acknowledgements

With thanks to all of our mentors, who have guided us every step of the way.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562.

This project was sponsored by Oak Ridge National Laboratory, the Joint Institute for Computational Sciences, the University of Tennessee, Knoxville, and the Chinese University of Hong Kong.

This project was made possible by support from the National Science Foundation.

Last but not least, thanks to Dr. Google, without whom this would have been impossible.