

Lab Assignment 6: The Deformable Mirror and the Gerchberg-Saxton Algorithm

This lab is inspired by [NASA's Nancy Grace Roman Space Telescope](#), whose objectives are “to unravel the secrets of dark energy and dark matter, search for and image exoplanets, and to explore many topics in infrared astrophysics”. A student completes a MATLAB program, in versions, to simulate the [Gerchberg-Saxton \(GS\) algorithm](#), as relevant to the control of a [deformable mirror](#) in one instrument of the space telescope. NASA has produced videos on the telescope, which include a two-minute explanation of the [coronagraph instrument](#). A related video explains a [simplified instrument](#) at the 0:18 mark.

Version 0: Getting Started

We will explore a model of the GS algorithm and its application to a simplified instrument. In “[A Tail of Two Cats](#)” by Kevin Cowtan, the GS algorithm is explained as a way to reconstruct a missing piece of a two-dimensional (2D) image. This involves the manipulation of *magnitude* and *phase* information of the 2D *Discrete Fourier Transform* (DFT). For our purposes, the necessary functionality of the 2D DFT and its inverse are encapsulated in two functions, `dft2` and `idft2`, of the `coronaSimulate` program.

Download and unzip `V0GettingStarted.zip`. In it, there are four files, including `frames_v0.mat` and `coronagraph_v0.avi`. Next, run the `coronaSimulate.m` file. In addition to outputting results to the Command Window and a Figure Window, it should also output a `frames.mat` file. The variables stored in this file, which you can load into the *base* workspace, should be identical to the variables stored in the `frames_v0.mat` file, which you can also load. Please compare the variables.

Run the `coronaAnimate.m` file. Assuming the [Computer Vision](#) toolbox is installed, it will output results to the Command Window and a Figure Window. A video, `coronagraph.avi`, is produced. Watch this video and compare it to the `coronagraph_v0.avi` video. They should appear identical.

Version 1: Simple Occultation

Download the `V1SimpleOccultation.zip` file. It has `frames_v1.mat` and `coronagraph_v1.avi` result files. After unzipping, watch the `coronagraph_v1.avi` video. When you complete this version, requiring modifications to `coronaSimulate.m` only, you should be able to reproduce the results.

Complete the `occultSquare`, `gerchbergSaxton`, and `getFrames` functions, including the comment headers. Modify `getFrames` so that the axes in your results video is grayscale. Add a title to match the given result. Use “`axis image off`” to make the pixels square and to hide ticks and labels.

Next, modify the `occultSquare` function so that the central part of the image, `im`, is black. Implement a square shape that is `width` pixels wide and tall (make `width` the second input argument). To respect the given modular organization, do not change any function other than those specified above.

The `opticalSystem` function returns a second argument, which is the true *phase aberration* in the *pupil plane* of the coronagraph. A complete GS algorithm would estimate this value independently. It would start from an initial guess, at iteration zero, and approximate the true value after a maximum number of iterations. For simplicity, our simulated algorithm assumes the aberration is known.

Modify the `gerchbergSaxton` function so that, when it invokes `idft2`, it “corrects” the aberration, `Dphi`. Do *not* modify the script, especially the statement that calls the `gerchbergSaxton` function. When `idft2` is invoked, the second argument is *phase*. Rewrite it so that the argument equals `IMp`, at iteration 0, and `IMp+Dphi`, at iteration `maxIters`. Use [linear interpolation](#) at other iterations.

Submit Version 1 of your `coronaSimulate.m` file, by its deadline, to demonstrate use of a version-based approach. Before submission, test it when the other files, from Version 0, are unchanged.

Version 2: Occultation and Plot

Download and unzip `V2Occultation&Plot.zip` to get `frames_v2.mat` and `coronagraph_v2.avi` files. The variables stored in the first (`.mat`) file, which you can load into the base workspace, ought to be identical to the variables stored in the `frames.mat` file once you complete this version. Modify your `coronaSimulate.m` file (after completing Version 1). Run `coronaAnimate`, which should produce a video. Compare it to the `coronagraph_v2.avi` video. Ideally, the videos will look the same.

Modify these functions, as well as the script: `opticalSystem`, `occultSquare`, `gerchbergSaxton`, and `getFrames`. Rename `occultSquare` to `occultCircle` and modify it so that the black central part of the image, `im`, is a circle having a diameter of `width` pixels. Use the `insertShape` function of the [Computer Vision](#) toolbox. Complete and edit, as needed, comment headers of local functions.

Have the `occultCircle` function return a second argument, `mask`, a logical 2D matrix whose rows and columns equal those of the `uint8` 2D matrix, `im`, a grayscale image. The `mask` matrix specifies occultation or simple coronagraphy. Have the `opticalSystem` function return it as a third argument. Entries are `true` for the corresponding pixels of `im` made black. Entries are `false` elsewhere.

Modify the `gerchbergSaxton` function to have a fourth input argument, `mask`, and a second output argument, `errors`. Next, modify the script so that the `mask` returned by `opticalSystem` is passed to `gerchbergSaxton`. At each iteration of the latter function, compute the sum of squared values of image, `im`, pixel entries (convert to `double` first) that correspond to `mask` entries that are `true`. Each sum is called a *sum square error*. Put them in a column vector, `errors`, of length `maxIters+1`.

Finally, modify the `getFrames` function so that the video output, `coronagraph.avi`, after running `coronaSimulate` and `coronaAnimate` in sequence, resembles the `coronagraph_v2` video. Add a second input argument to `getFrames` to `start`. Then, in each loop iteration, plot the sum square error versus iteration number. Ideally, the image and plot should overlap on the same axes, as shown.

Use `set(gca, 'YDir', <expression>)` to set the y-axis direction, if necessary, of an *active* plot axes. Use `get(gca, 'YDir')` at a command `>>` or debugger `K>>` prompt to determine direction options.

If combining plot and image in one figure, use `set(gca, 'DataAspectRatio', ratio)` to set the aspect ratio of pixels, where `ratio` is equal to `[<expression1> <expression2> 1]`. Determine suitable values for `<expression1>` and `<expression2>` so your video looks as expected. Each expression should be a simple ratio of an image matrix dimension and a plot axes limit.

Upon completion of Version 2, submit only your `coronaSimulate.m` file to eClass for marking. Ensure it works as required with the `coronaAnimate.m` file, which is unchanged from Version 0.

Additional Note

Until you [install the Computer Vision toolbox](#), you can animate `coronaSimulate` results as follows:

```
>> load frames
>> movie(gcf, frames)
```

Once it is installed, use `doc` to read about the `insertShape` function required for Version 2.

Revision History

This document and associated files were authored in 2020 by [Dileepan Joseph](#) and edited by [Wing Hoy](#). The lab assignment was revised in 2021 by Joseph and reviewed by [Edward Tiong](#) and Jason Myatt. The author gratefully acknowledges the feedback from [Dan Sirbu](#) of NASA's [Ames Research Center](#).