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 explore many topics in infrared astrophysics". A student completes a Python program, in versions, to simulate the Gerchberg-Saxton (GS) algorithm, as inspired by 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 coronagraph at the 0:20 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.

Unzip `V0GettingStarted.zip`. In it, there are four files, including `coronagraph_v0.avi` and a copy of an image taken from a NASA website. In the image, i.e., `300_26a_big-vlt-s.jpg`, an exoplanet called 2M1207b orbits a dwarf star called 2M1207. Next, run the `coronaSimulate.py` file. In addition to outputting text and figures to the Console and Plots panes, it outputs several `.png` files. Each `.png` file specifies one frame of a video given in the `.avi` file. Please compare the images and video.

Run the `coronaAnimate.py` file. Assuming a Spyder installation with OpenCV support, it produces a video, `coronagraph.avi`, from a sequence of `.png` files, `coronagraph0`, `coronagraph1`, *etc.*, after you enter the end frame number at the prompt. Compare `coronagraph_v0.avi` to this video.

Version 1: Simple Occultation

Unzip the `V1SimpleOccultation.zip` file. It has a `coronagraph_v1.avi` video and a subfolder of `.png` frames. Watch the video. When you complete this version, modifying `coronaSimulate.py` only, you should be able to reproduce the video after running `coronaAnimate.py` on your `.png` frames.

Complete the `occultSquare`, `gerchbergSaxton`, and `saveFrames` functions, writing their comment headers also. First, modify `saveFrames` so that each pixel shown has equal red, green, and blue parts. Edit the title to match the given result. Use a statement or statements to hide axes ticks and labels.

Next, modify the `occultSquare` function so that the central part of the image, `im`, it returns is black (zero). Implement a square shape that is exactly `width` pixels high and wide (make `width` the second input argument). Check it with the Variable Explorer. Do not change code outside `occultSquare`.

The `opticalSystem` function returns a second argument, `Dphi`, 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* of *phase*. 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. Do *not* modify the `main` function or, for the purposes of this requirement, any other function.

To follow an iterative and incremental approach, submit your Version 1 `coronaSimulate.py` file by the Version 1 deadline. Before submission, test it when other files, from Version 0, are unchanged.

Version 2: Occultation and Plot

Unzip the `V2Occultation&Plot.zip` file. When your Version 2 program is sufficiently correct, it will output `.png` frames like those in the `coronagraph_v2a` subfolder. Ideally, your `coronaSimulate` will output ones like those in the `_v2b` subfolder. You may produce videos, like `coronagraph_v2a.avi` or `_v2b.avi`, by also running the given `coronaAnimate` program, assuming an OpenCV installation.

In the `saveFrames` function, insert `plt.subplot(1,2,1)` and `plt.subplot(1,2,2)` statements so that images are shown on the right side only. Rename the `occultSquare` function to `occultCircle`, both where it is defined and also where it is invoked. Revise its definition so that instead of blacking out, or occulting, a square of the specified `width`, the function occults a circle with that diameter, or `width`. The occulted group of pixels must be at the centre of the image. Verify that your program runs.

Modify the `occultCircle` function to return a second argument, `mask`, that is a 2D NumPy array of boolean entries. Initialize it, using `np.full`, to have the same shape as the first input argument, `im`, but with the value `False`. For every entry of `im` you occult, assign the corresponding entry of `mask` to `True`. Modify the invocation in `opticalSystem` of `occultCircle` so that its output arguments are assigned to two variables, the first of which, `im`, is the same as before. Verify that your program runs.

Modify `opticalSystem` to return a third argument, `mask`, the `mask` obtained from `occultCircle`. Modify `main` to accept the third output argument from `opticalSystem` and to pass it as a fourth input argument to `gerchbergSaxton`. Modify `gerchbergSaxton` to accept a fourth input argument, `mask`, and to return a second output argument, `errors`, initialized to an empty list. Modify `saveFrames` to accept a second input argument, `errors`. So that you can verify your program runs, modify `main` to accept the `errors` argument from `gerchbergSaxton` and to pass it on to `saveFrames`.

In `gerchbergSaxton`, compute a float, `error`, from the result, `im`, of each `idft2` invocation and the fourth input argument, `mask`. Append each such float, `error`, to the list, `errors`, that the function returns. The `error` is the sum total of the squared values of `im` entries where corresponding `mask` entries are `True`. Outsource computation to a function, `occultError`, with two input arguments and one output argument, that you define and invoke appropriately. Verify that your program runs.

Examine the computed list, `errors`, with the Variable Explorer using a breakpoint inside `saveFrames`. Expected values can be read approximately from the `coronagraph_v2a\*.png` files. Compute the maximum, `maxErrors`, of `errors` within `saveFrames`. Without altering the right-side image, make and annotate the left-side plot as indicated in the given `.png` files. Verify your program works.

Examine the `coronagraph_v2b\*.png` files. To meet the implied requirements, modify `saveFrames`. Remove all `plt.subplot(…)` and any `plt.axis('off')` statements. Plot the graph first with correct limits. Using a `plt.imshow` statement with an `extent=(…)` argument, where you determine the `(…)` expression, add the image to the graph (plotted lines will show). Use `plt.gca().set_aspect(?)`, where you must compute `?` from `maxIters` and `maxErrors`, so that the image looks correct.

In your own words, write suitable comment headers for all functions, `main` aside, without comment headers. A comment header should summarize the function's purpose, input and output arguments, and side effects, if any, such as Console input or output, Plots output, and file input or output.

To follow an iterative and incremental approach, submit your Version 2 `coronaSimulate.py` file by the Version 2 deadline. Before submission, test it when other files, from Version 0, are unchanged.

Revision History

This document and associated files were authored in 2020 by Dileepan Joseph and edited by Wing Hoy. The assignment was revised in 2021 by Joseph and reviewed by Edward Tiong and Jason Myatt. Due to a programming language change, the assignment was revised in 2022 by Joseph with Hoy's help. Joseph acknowledges the support of Dan Sirbu, an imaging scientist at NASA's Ames Research Center.