Quantifying Optical Performance of James Webb Space Telescope

Eugene Lytnev*
San Jose State University, CA, USA
(Dated: December 7, 2020)

We successfully generated the Modulation Transfer Function curve of the James Webb Space Telescope's NIR Camera using its Point Spread Function.

I. MODULATION TRANSFER FUNCTION

For this project, we decided to look at the optical performance of the James Webb Space Telescope (JWST), the current successor to the Hubble Space Telescope. The most straightforward way to quantify the optical performance of an optical system would be to look at its Modulation Transfer Function (MTF). In simple terms, the MTF is a graph of how visible things or elements of various sizes would be when viewed through the system. More specifically, it is plot of contrast vs spatial frequency. This is explained further in the following sections.

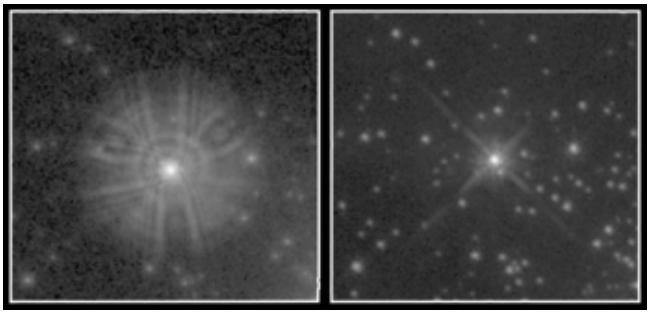

Figure 1. Images of Hubble PSF before (left) and after (right) the optical issues were corrected!

II. POINT SPREAD FUNCTION

Now that we know how we will quantify the optical performance of the telescope, we should look at how we will simulate it. One common way is to apply a convolution to the image using the Point Spread Function of the optical system. We will go into convolution in the next section. The Point Spread Function (PSF) of an optical system is the image of a point source of light, and can be used to summarize the action an optical system does to a signal. For telescopes, stars are effectively point sources of light, which means that we can experimentally find the PSF by just looking at a bright star on a blank background. (See Fig. 1)

Unfortunately, the JWST is not operational yet, so experimental values are not possible even if they would have been free of enough noise to be useful. The PSF of the James Webb telescope is provided freely online for researchers to be able to determine if a specific project will be achivable with the optical tolerances of the JWST. It is availabe as a generator called WebbPSF[1], and as pre-generated images in FITS format, which can be easily imported using AstroPy. The JWST will launch with four sensors. We were looking to benchmark the performance against the Hubble Space telescope, so we focused looking at the sensor closest in use to the Hubble's main sensor, which will be the NIR Cam.

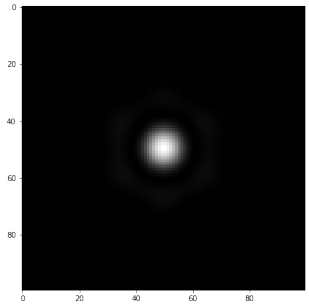

Figure 2. Pre-generated PSF of the JWST NIR Cam

Alternatively, an approximation of the PSF can be generated by taking a Fourier transform of the pupil plane, which is the region of light on one plane with the opposite focal point of the system. This method was not used and is only briefly mentioned here for the sake of completeness. (See Lightsey.)

* Correspondence email address: eugene.lytnev@sjsu.edu
[1] `https://www.stsci.edu/jwst/science-planning/proposal-planning-toolbox/psf-simulation-tool`

III. CONVOLUTION

What we can do to simulate the optical system would be to take an image and treat every pixel as a point source. This would mean we could add a PSF around every pixel in our image multiplied by the brightness of our pixel. This is essentially what we do in the `convolveLoop` function in the appendix.

The problem is that doing that with a loop is very computationally intensive and there are ways of accelerating the computation of such a process, such as using a convolution. In image processing, convolution is essentially taking a weighted average of the surrounding pixels. The matrix of weights of the pixels is called the kernel and it can be easily demonstrated that we can rotate the PSF by 180 degrees to get a kernel which will yield the same result as the loop. We used `scipy.signal.convolve` to run our convolution, which significantly reduced compute time.

IV. SAMPLE RULINGS, CONTRAST, AND GENERATION

We can create a graph of the modulation transfer function in a number of ways. The least computationally intensive way would be the Slanted Edge Method. (See Kerr.) A method that is more straightforward to understand would be to generate images with only details at a specific frequency and measuring how visible each image is after we have simulated how it will look through our system.

An image with only details of a specific frequency would be a sine wave, which is what we used. The code for generating these images can be found in the Sample Rulings section of the appendix. Note that the `X` variable is a matrix generated by either `xtall` for a horizontal signal or `xrot` if we want a rotated signal.

We can measure the visibility of an image using contrast. Contrast is a measure of how different light and dark spots on the image are when compared to similar images. It can be found using the formula below, which is implemented in the `contrast` function in the appendix.

$$Contrast = \frac{max - min}{max + min} \tag{1}$$

We generated our MTF curve by generating an image for each frequency, convolving it, and measuring its contrast. In order to ensure consistent results, the image size (in pixels) was constant, although it may not have been necessary. We took the pixel size from the WebbPSF files' README.

V. RESULTS

We found the curve in Fig. 3. Unfortunately, the actual MTF curves for the JWST are not available online, so we cannot compare them to anything. It is rather curious that the JWST cannot see significantly past an inverse arcsecond according to this graph.

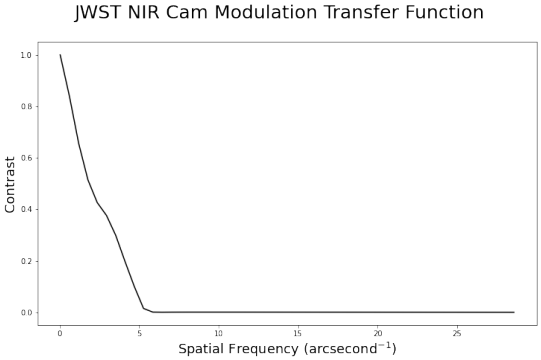

Figure 3. Generated MTF curve of the JWST NIR Cam

VI. CONCLUSIONS

All in all, we were able to find an MTF curve for the James Webb telescope. It would be interesting to see how this method compares to the slanted edge method. Additionally, this project can be extended by generating the PSF's ourselves. It may also be interesting to try and implement this project using libraries beyond NumPy, like AstroPy or HCIPy.

ACKNOWLEDGEMENTS

I would like to thank Professor Chris Smallwood for his help in creating the idea for this project and Professor Ehsan Khatami for his help with the project. Additionally, I would like to thank NASA and the developers of WebbPSF for making their information publicly available. This project has been made possible with these contributions.

[1] D. A. Kerr *Determining MTF with a Slant Edge Target* (2010).
[2] P. A. Lightsey, J. S. Knight and G. Golnik, *Status of the optical performance for the James Webb Space Telescope* (SPIE, 2014).
[3] *Hubble Space Telescope Images of a Bright Star in the Large Magellanic Cloud* (Hubble Site, 1994).

Appendix: Functions Created for Project

1. Convolution

```
def normalize(image,normal=1):
    '''Normaizes a positive function to go from 0 to norm
    '''
    image -= np.min(image)
    image /= np.max(image)
    return image * normal

def invertPSF(pattern):
    '''Returns the inverse point spread function,
    which can be used as the convolution kernel in scipy.
    Does so by rotating the pattern 180 degrees
    '''
    return np.rot90(pattern,2)

def convolveSignal(image,kernel,overcrop = False):
    '''Applies the PSF to an image using scipy.signal.con
    '''
    x,y = np.shape(kernel)
    if not overcrop:
        x //= 2
        y //= 2
    return signal.convolve(image,kernel)[x:-x,y:-y]

def convolveLoop(image,pattern):
    '''Applies the PSF to an image using a loop
    It will likely do a better job of simulating star vie
    '''
    x, y  = np.shape(image)
    xp,yp = np.shape(pattern)
    dx,dy = int(np.ceil(xp/2)),int(np.ceil(yp/2))
    newimage = np.zeros((x+xp+1,y+yp+1))
    for i in range(x):
        for j in range(y):
            newimage[i:i+xp,j:j+yp] += image[i,j] * patte
    return newimage[dx:dx+x,dy:dy+y]
```

2. Sample Rulings

```
def xtall(x, y):
    X,Y = np.meshgrid(np.arange(x),np.arange(y))
    return X * 2 * np.pi

def xrot(x, y, theta):
    X,Y = np.meshgrid(np.arange(x),np.arange(y))
```

```
    Xr  =  np.cos(theta/180*np.pi)*X - np.sin(theta/180*np.pi)*Y
    return Xr * 2 * np.pi

def ronchi(X, wavelength, duty = 0.5):
    return signal.square(X / wavelength, duty) / 2 + 0.5

def ronchif(X, frequency, duty = 0.5):
    return signal.square(X * frequency, duty) / 2 + 0.5

def sinerule(X, wavelength):
    return np.sin(X / wavelength) / 2 + 0.5

def sinerulef(X, frequency):
    return np.sin(X * frequency) / 2 + 0.5
```

3. MTF Generator

```
def contrast(image, cropx=0, cropy=0):
    cropx,cropy = int(cropx),int(cropy)
    cropped = image[cropx:-cropx,cropy:-cropy]
    maxi = np.max(cropped)
    mini = np.min(cropped)
    return (maxi-mini) / (maxi+mini)

def mtf(psf, frequencies, rotation = None):
    kernel = invertPSF(psf)
    minfreq = np.min(frequencies)
    xp,yp = np.shape(psf)
    cropx,cropy = int(2*xp),int(2*yp)
    h,w = int(2/minfreq+2*cropx),int(2/minfreq+2*cropy)
    if rotation == None:
        X = xtall(w,h)
    else:
        X = xrot(w,h,rotation)
    contrasts = np.zeros(np.shape(frequencies))
    for i,freq in enumerate(frequencies):
        contrasts[i] = contrast(convolveSignal(sinerulef(
    risk = frequencies > 0.5
    if np.any(risk):
        print("Risk of aliasing at the following frequenc
        print(frequencies[risk])
    return normalize(contrasts)

def mtfWave(psf, wavelengths, rotation = None):
    kernel = invertPSF(psf)
    maxwave = np.max(wavelengths)
    xp,yp = np.shape(psf)
    cropx,cropy = int(2*xp),int(2*yp)
    h,w = int(2*maxwave+2*cropx),int(2*maxwave+2*cropy)
    if rotation == None:
        X = xtall(w,h)
    else:
        X = xrot(w,h,rotation)
    contrasts = np.zeros(np.shape(wavelengths))
    for i,wavelength in enumerate(wavelengths):
        contrasts[i] = contrast(convolveSignal(sinerule(X
    risk = wavelengths < 2
    if np.any(risk):
        print("Risk of aliasing at the following waveleng
        print(wavelengths[risk])
    return normalize(contrasts)
```