hw09

November 29, 2021

1 Homework 09

Nathan LeRoy

[1]:
```
# better image quality
import matplotlib as mpl
%matplotlib inline
mpl.rcParams['figure.dpi'] = 300
```

1.1 Problem 1

1.1.1 a.) $\{0,1,0,0\} \circledast \{0,0,1,0\}$

The easiest method is using the circulant matrix method:

$$\{0,1,0,0\} \circledast \{0,0,1,0\} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

1.1.2 b.) $\{1,1,0,0\} \circledast \{0,0,1,1\}$

$$\{1,1,0,0\} \circledast \{0,0,1,1\} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

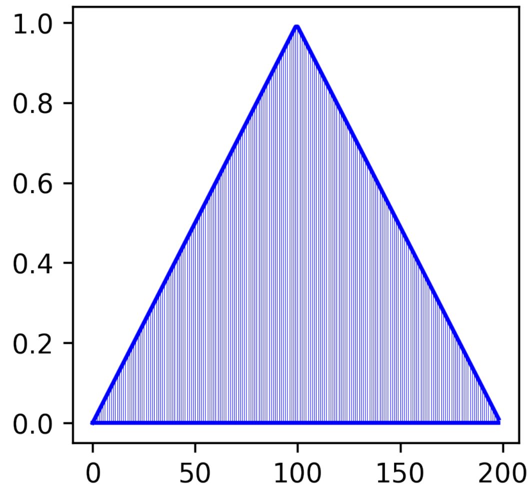
1.2 Problem 2

Init the discrete triangle function: ### a.)

[2]:
```
import numpy as np
import matplotlib.pyplot as plt

f = np.concatenate((np.arange(0,1,0.01), np.arange(0.99,0,-0.01)))
_, ax = plt.subplots(figsize=(3,3))
markerline, stemlines, baseline = ax.stem(
    f,
    linefmt="b-",
```

```
    markerfmt="b",
    basefmt="b-",
)
plt.setp(stemlines, linewidth=0.2)
```

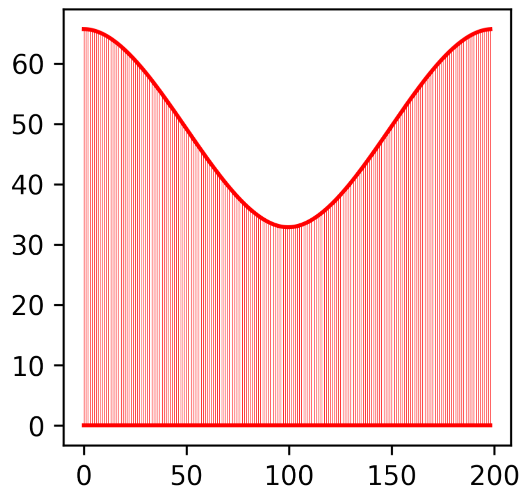
[2]: [None]



Compute the circular convolution:

[3]:
```
circ_conv = np.fft.ifft(np.fft.fft(f)*np.fft.fft(f))
_, ax = plt.subplots(figsize=(3,3))
markerline, stemlines, baseline = ax.stem(
    circ_conv.real,
    linefmt="r-",
    markerfmt="r",
    basefmt="r-",
)
plt.setp(stemlines, linewidth=0.2)
print(f"Function total data points: {len(f)}")
print(f"Circular convolution total data points: {len(circ_conv)}")
```

```
Function total data points: 199
Circular convolution total data points: 199
```

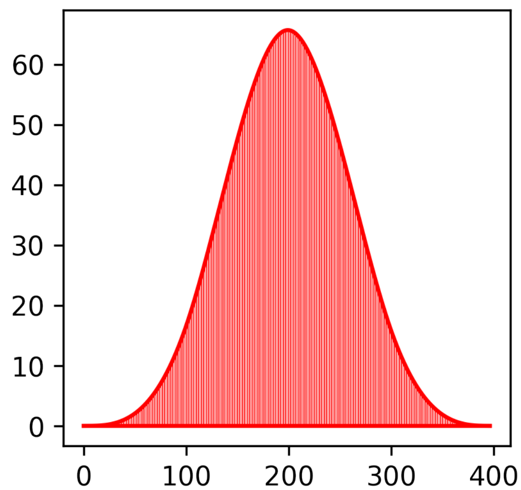


1.2.1 b.) numpy conv

Using the `np.convolve` function:

[4]:
```
conv = np.convolve(f,f)
_, ax = plt.subplots(figsize=(3,3))
markerline, stemlines, baseline = ax.stem(
    conv,
    linefmt="r-",
    markerfmt="r",
    basefmt="r-",
)
plt.setp(stemlines, linewidth=0.2)
```

[4]: [None]

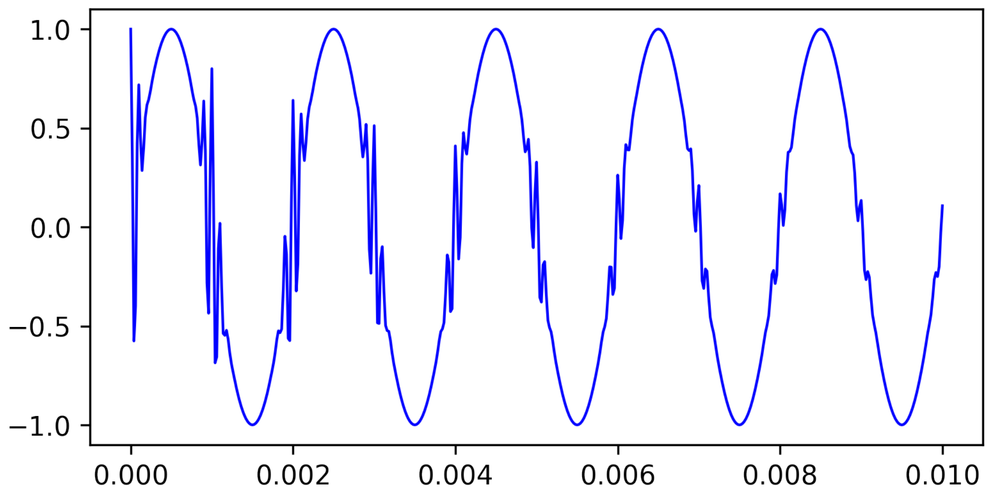


1.3 Problem 4

We can plot the data signal:

[5]:
```python
from scipy.io import loadmat
from scipy.signal import firwin
from helpers import plot_filt

data = loadmat("data/hw9prob4data.mat")
g = data['g'][0,:]
t = data['t'][0,:]
plt.subplots(figsize=(6,3))
plt.plot(t,g, 'b-', linewidth=1)
```

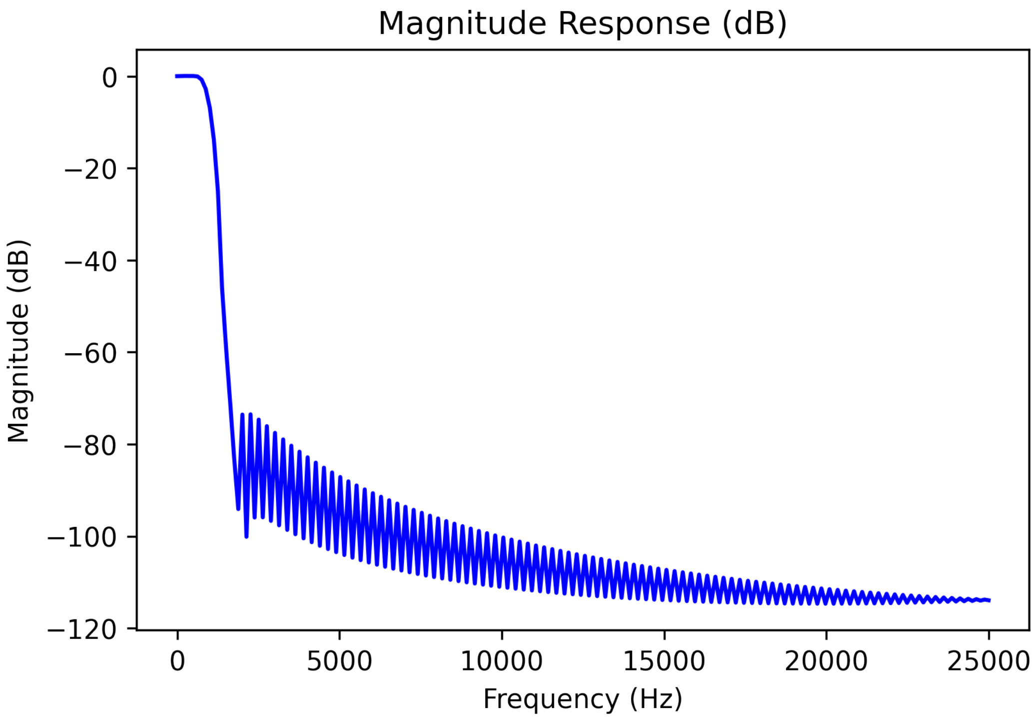
[5]: `[<matplotlib.lines.Line2D at 0x13907caf0>]`

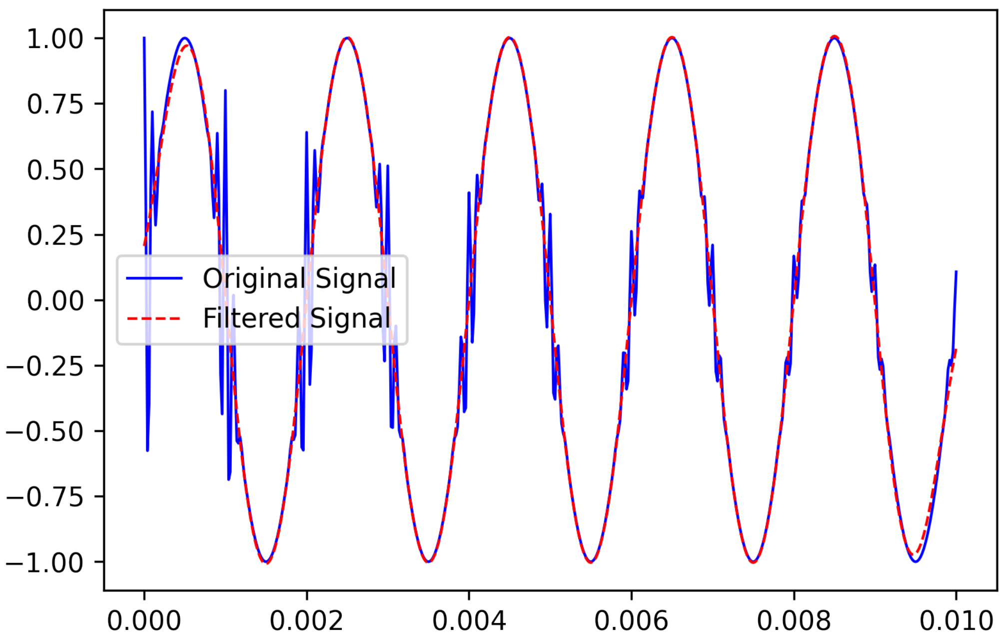


In order to stabilize and smooth out the signal we can apply a lowpass filter with a cutoff frequency (f_c) around twice the frequency of the interference. In this case, that would be ~1000Hz.

[35]:
```python
# create filter
filter_order = 200
cuttoff_freq = 1000
sample_rate = 50000 #50kHz
filter_type = 'lowpass' # Or highpass, bandpass, bandstop
DFILT = firwin(numtaps=filter_order+1, cutoff=cuttoff_freq,␣
 ↪pass_zero=filter_type, fs=sample_rate)

# plot and apply filter
```

```python
plot_filt(DFILT, sample_rate)
g_filtered = np.convolve(DFILT, g, "same")

# view newly filtered signal
_, ax = plt.subplots(figsize=(6,4))
ax.plot(t,g, 'b-', linewidth=1)
ax.plot(t,g_filtered,'r--', linewidth=1)
ax.legend(['Original Signal','Filtered Signal'])
plt.show()
```





We can see after the filter is applied that we get a **much** smoother signal that removes a lot of the aliasing and interference.

1.4 Problem 5

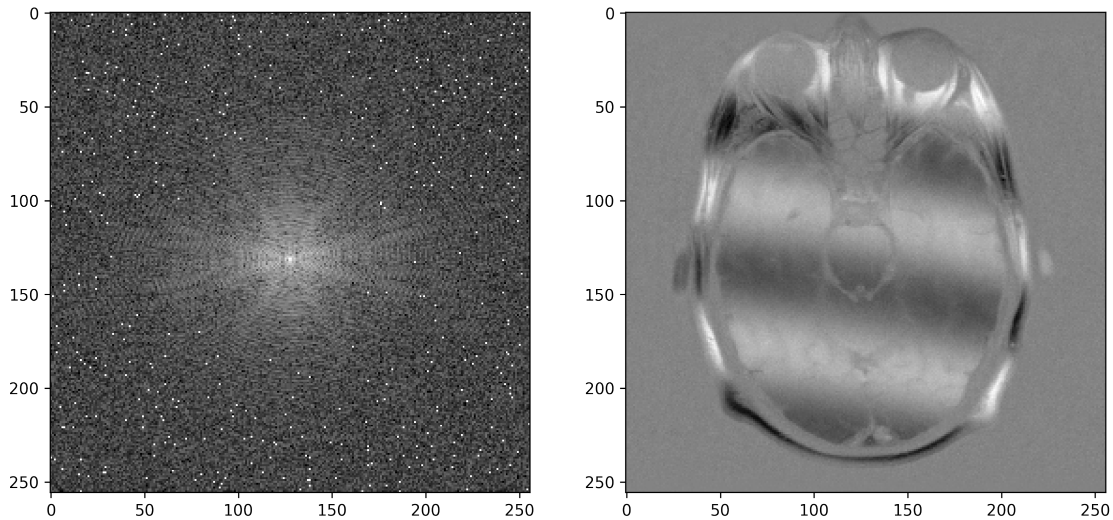
1.4.1 Part a.)

We will plot the raw data with the reconstructed image next to it

[93]:
```python
from helpers import ifft2c

# extract data and inverse fft
data = loadmat("data/fse_t1_ax_data.mat")
d = data['d']
d_recons = ifft2c(d)

# plot images
_, ax = plt.subplots(1,2, figsize=(12,6))
ax[0].imshow(np.abs(np.log(d)), cmap="gray")
ax[1].imshow(d_recons.real, cmap="gray")
```

```
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_38912/1892396003.py:1
0: RuntimeWarning: divide by zero encountered in log
  ax[0].imshow(np.abs(np.log(d)), cmap="gray")
```

[93]:
```
<matplotlib.image.AxesImage at 0x17774fd90>
```



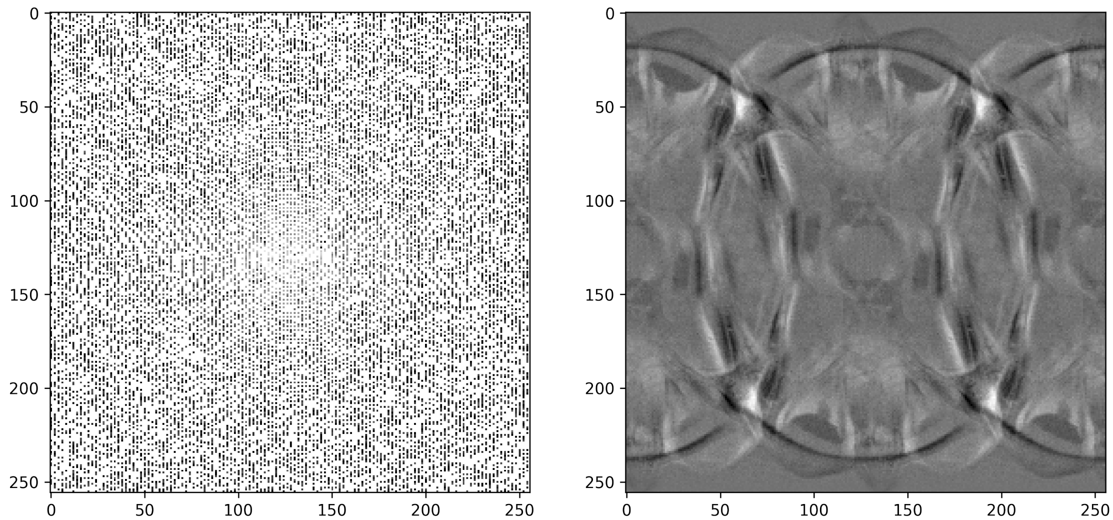
1.4.2 part b.)

We can simulate reconstructing an image from half the original data (undersampled image):

[92]:
```python
# reconstruct image from half the original data
dhalf = np.zeros(d.shape)
dhalf[:,::2] = d[:,::2]
dhalf_recons = ifft2c(dhalf)

# plot images
_, ax = plt.subplots(1,2, figsize=(12,6))
ax[0].imshow(np.abs(np.log(dhalf)), cmap="gray")
ax[1].imshow(dhalf_recons.real, cmap="gray")
```

```
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_38912/2291779447.py:3
: ComplexWarning: Casting complex values to real discards the imaginary part
  dhalf[:,::2] = d[:,::2]
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_38912/2291779447.py:8
: RuntimeWarning: divide by zero encountered in log
  ax[0].imshow(np.abs(np.log(dhalf)), cmap="gray")
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_38912/2291779447.py:8
: RuntimeWarning: invalid value encountered in log
  ax[0].imshow(np.abs(np.log(dhalf)), cmap="gray")
```

[92]:
```
<matplotlib.image.AxesImage at 0x1776324f0>
```



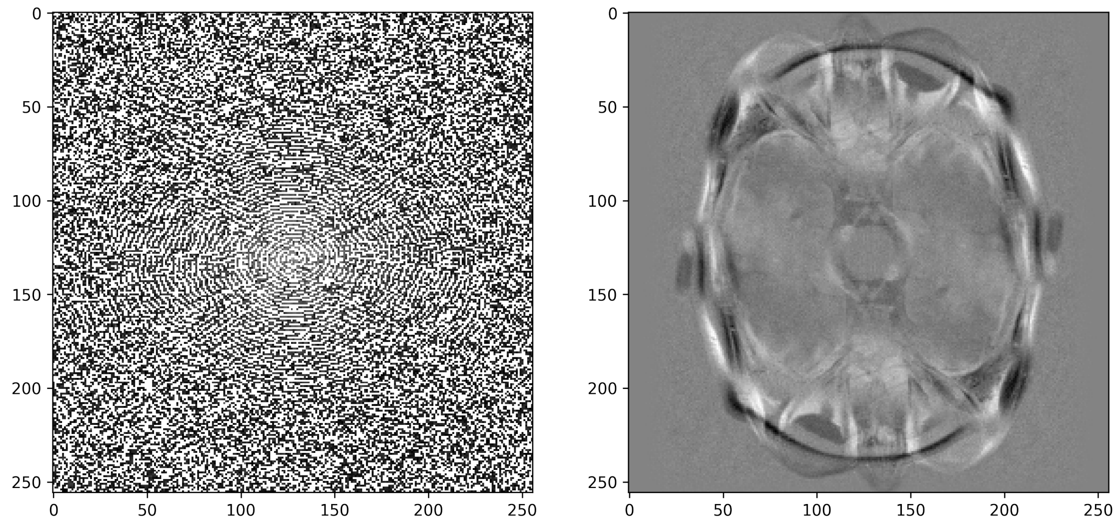
1.4.3 part c.)

Lets build and apply a filter to the MRI data using the `scipy.signals.firwin` package.

[95]:
```
cutoff = 650 # Empirically set this to a number less than 750.
FILTER = firwin(numtaps=33, cutoff=cutoff, pass_zero="lowpass", fs=1500)
dfilt = np.zeros(d.shape)

for n in range(d.shape[0]):
    dfilt[n,:] = np.convolve(FILTER, d[n,:], "same")

dfilt_recons = ifft2c(dfilt)

# plot images
_, ax = plt.subplots(1,2, figsize=(12,6))
ax[0].imshow(np.abs(np.log(dfilt)), cmap="gray")
ax[1].imshow(dfilt_recons.real, cmap="gray")
```

```
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_38912/3908254211.py:6
: ComplexWarning: Casting complex values to real discards the imaginary part
  dfilt[n,:] = np.convolve(FILTER, d[n,:], "same")
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_38912/3908254211.py:1
2: RuntimeWarning: invalid value encountered in log
  ax[0].imshow(np.abs(np.log(dfilt)), cmap="gray")
```

[95]: <matplotlib.image.AxesImage at 0x28fe4c940>



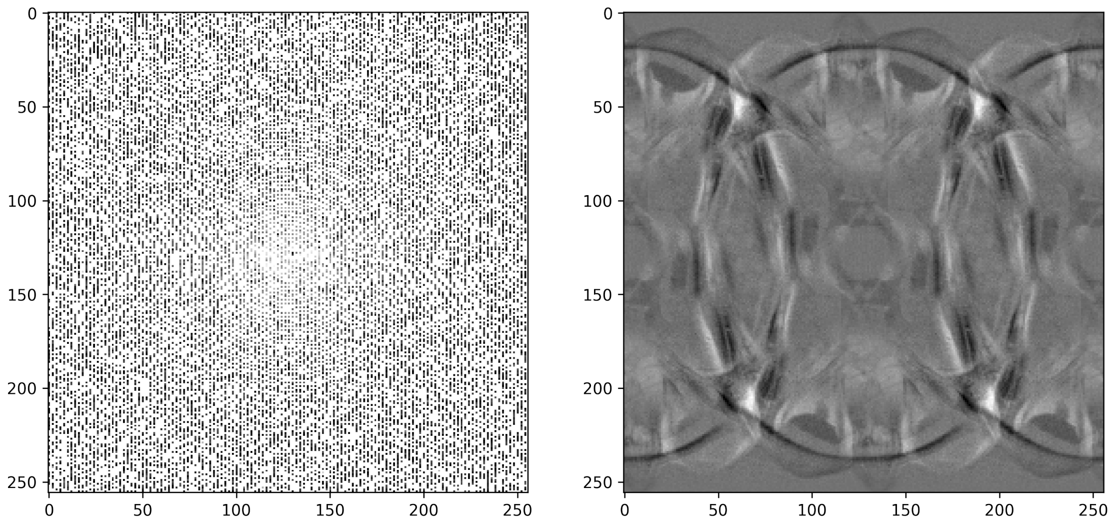
1.4.4 part d.)

We can again simulate a subsample of the filtered data:

[91]:
```
# reconstruct image from half the original data
dfilt_half = np.zeros(dfilt.shape)
dfilt_half[:,::2] = dfilt[:,::2]
dfilt_half_recons = ifft2c(dfilt_half)

# plot images
_, ax = plt.subplots(1,2, figsize=(12,6))
ax[0].imshow(np.abs(np.log(dfilt_half)), cmap="gray")
ax[1].imshow(dfilt_half_recons.real, cmap="gray")
```

```
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_38912/2220414860.py:8
: RuntimeWarning: divide by zero encountered in log
  ax[0].imshow(np.abs(np.log(dfilt_half)), cmap="gray")
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_38912/2220414860.py:8
: RuntimeWarning: invalid value encountered in log
  ax[0].imshow(np.abs(np.log(dfilt_half)), cmap="gray")
```

[91]:
```
<matplotlib.image.AxesImage at 0x17756dd00>
```



1.4.5 part e.)