hw09

November 30, 2021

1 Homework 09

Nathan LeRoy

[99]:
```
# better image quality
import matplotlib as mpl
%matplotlib inline
mpl.rcParams['figure.dpi'] = 200
```

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

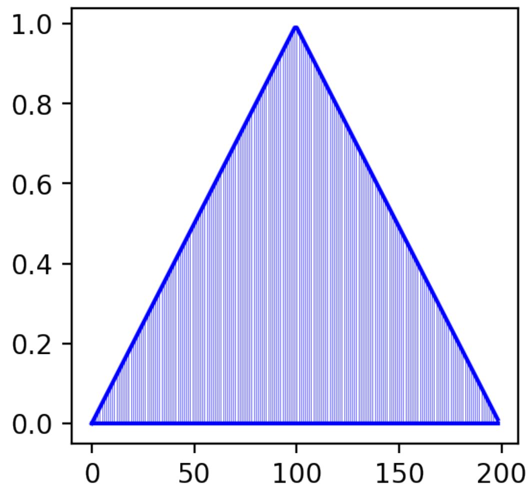
Init the discrete triangle function:

1.2.1 a.)

[100]:
```
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)
```

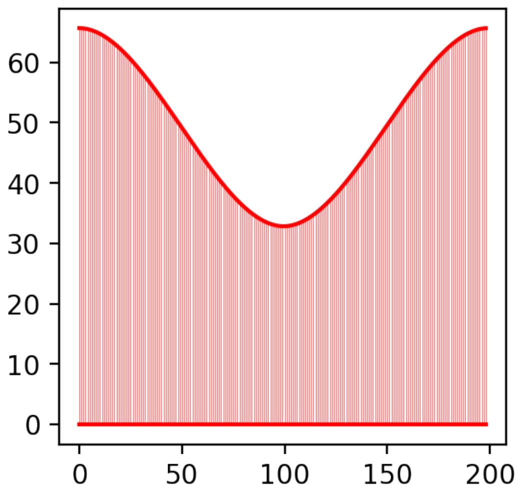
[100]:
```
[None]
```



Compute the circular convolution:

[101]:
```
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
```



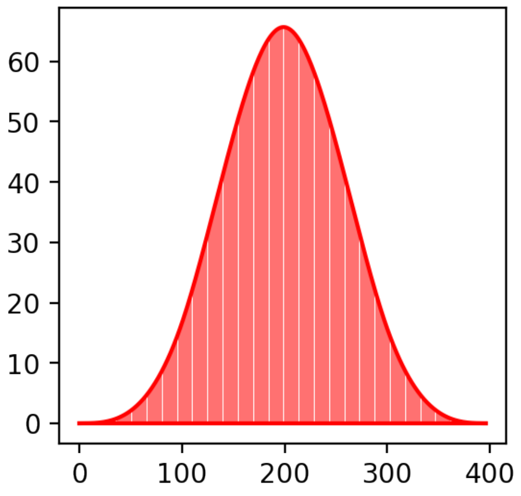
Yes, this looks like what we would expect as we are computing the circular convolution of our triangle and we can imagine them fully overlapped and as we slide over, the overlap decreases, and then goes back to normal as we slide over more.

1.2.2 b.) numpy conv

Using the `np.convolve` function:

[113]:
```
conv = np.convolve(f,f)
_, ax = plt.subplots(figsize=(3,3))
markerline, stemlines, baseline = ax.stem(
    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(conv)}")
```

```
Function total data points: 199
Circular convolution total data points: 397
```



This result looks more like one would expect from a linear convolution. If we are **not** repeating the triangle function, it makes sense that we would start with **zero** overlap and then continue on to complete overlap, and then again back down to **no** overlap. Also note how the peak of the circular convolution is identical to the peak of the linear convolution… confirming the result.

1.2.3 c.)

We can zero-pad the matrices with a python function like so:

[103]:
```python
def pad(a: np.ndarray, length: int):
    """Zero-pad an array of a certain length"""
    zeroarr = np.zeros(length)
    zeroarr[:len(a)] = a
    return zeroarr
```

If we are convolving f with itself. Thus, the total length of the linear convolution is $len(f) + len(f) - 1$. Once we accomplish this and redo the convolution theroem, we can examine the results:

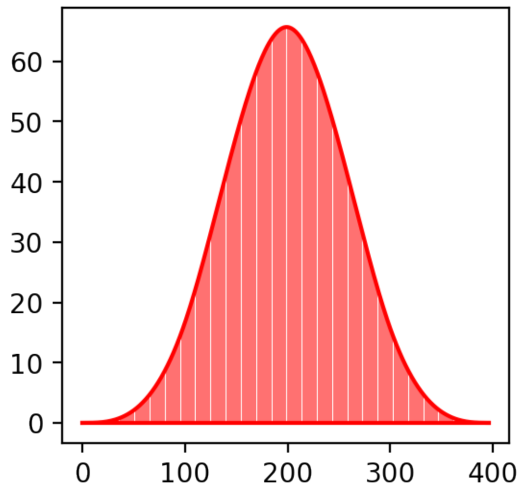
[112]:
```python
# pad f function
fpad = pad(f, 2*len(f) - 1)

# run through convolution theorem
fpad_conv = np.fft.ifft(np.fft.fft(fpad)*np.fft.fft(fpad))

_, ax = plt.subplots(figsize=(3,3))
markerline, stemlines, baseline = ax.stem(
    fpad_conv.real,
    linefmt="r-",
    markerfmt="r",
    basefmt="r-",
)
```

```python
plt.setp(stemlines, linewidth=0.2)
print(f"Function total data points: {len(fpad)}")
print(f"Circular convolution total data points: {len(fpad_conv)}")
```

```
Function total data points: 397
Circular convolution total data points: 397
```



We can see that this looks exactly like the original linear convolution.

1.3 Problem 3

1.3.1 a.)

The frequency spacing in frequency space is given as $\Delta f = \frac{f_s}{N}$... Where N is the number of samples and f_s is the sampling frequency.

$$\Delta f = \frac{3000\text{Hz}}{500} = 6\text{Hz}$$

1.3.2 b.)

If we want to know the maximum frequency spectral component... we need to know that it is got to be at least half our sampling frequency. Put another way, if we want to capture some sort of signal that is occuring at a frequency of f_a, then it must not exceed the Nyquist frequency which is precisely $\frac{f_s}{2}$. **Thus, the maximum frequency that we can sample is equal to half our sampling frequency or** $\frac{3000\text{Hz}}{2} = 1500\text{Hz}$

1.3.3 c.)

We know that in the frequency space, we will have spectral reflections that are exactly f_s apart in frequency space. For this problem, the spacing between reflections will be 3000Hz

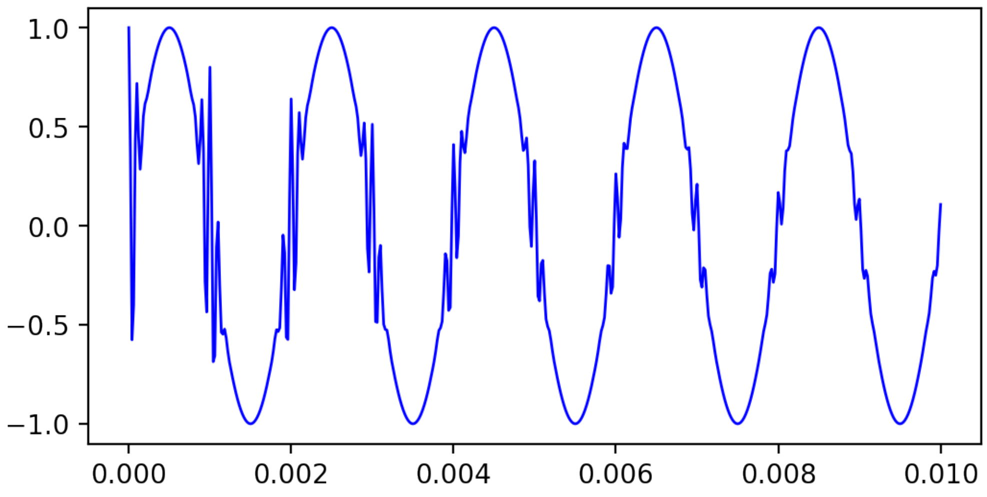
1.4 Problem 4

We can plot the data signal:

[105]:
```
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)
```

[105]:
```
[<matplotlib.lines.Line2D at 0x29c1b1700>]
```



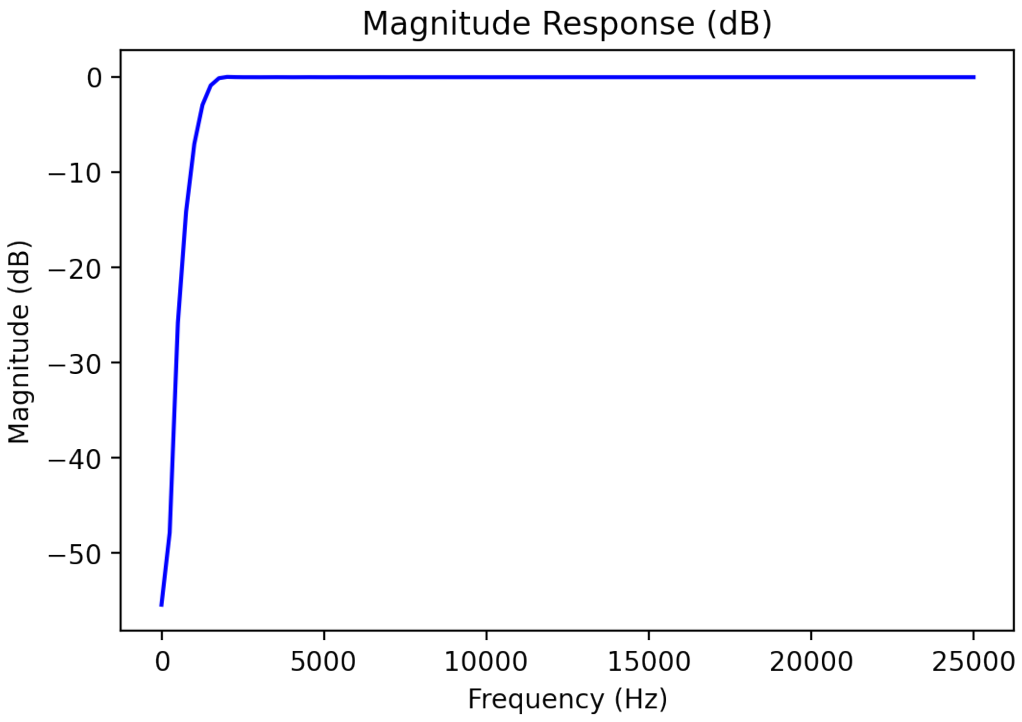
In order to stabilize and smooth out the signal we can apply a highpass filter with a cutoff frequency (f_c) around twice the frequency of the interference. In this case, that would be ~1000Hz. I am opting for a highpass filter as we know that the interference signal occurs at a low frequency of around 500Hz. Thus, the highpass filter will allow are true signal to come through while getting rid of the low frequency interference.

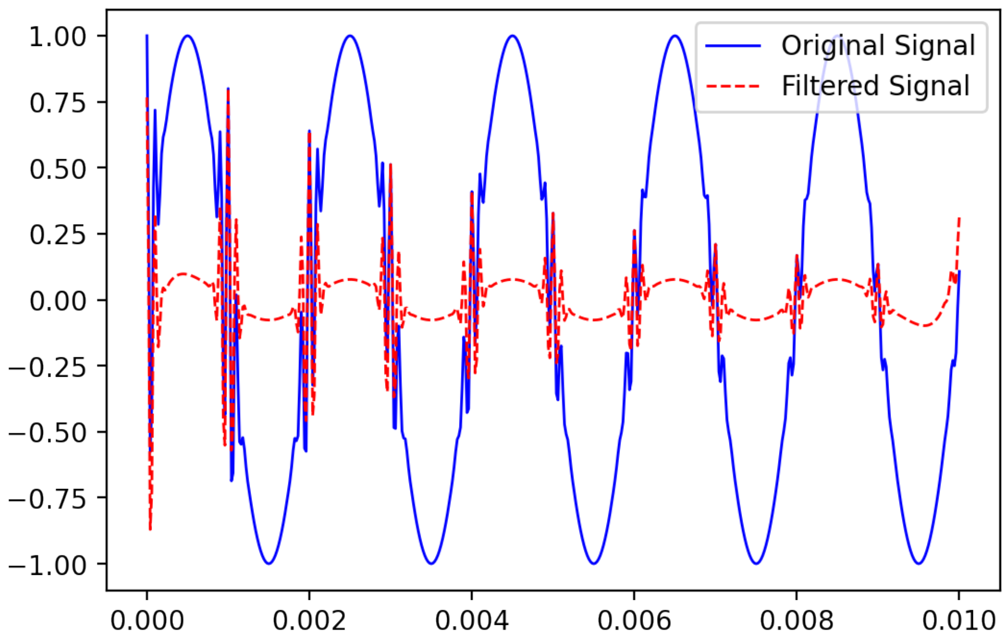
[106]:
```
# create filter
filter_order = 100
cuttoff_freq = 1000
sample_rate = 50000 #50kHz
```

```
filter_type = 'highpass' # Or highpass, bandpass, bandstop
DFILT = firwin(numtaps=filter_order+1, cutoff=cuttoff_freq,␣
↪pass_zero=filter_type, fs=sample_rate)

# plot and apply filter
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.

1.5 Problem 5

1.5.1 Part a.)

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

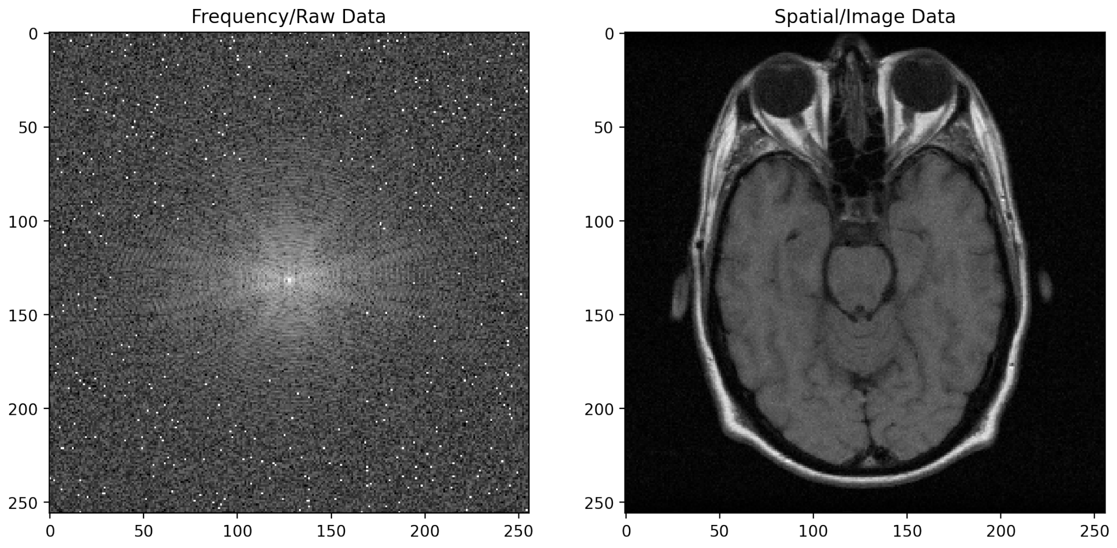
[107]:
```python
from helpers import ifft2c

# 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[0].set_title("Frequency/Raw Data")
ax[1].imshow(np.abs(d_recons), cmap="gray")
ax[1].set_title("Spatial/Image Data")
```

```
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_57512/1787536764.py:1
0: RuntimeWarning: divide by zero encountered in log
  ax[0].imshow(np.abs(np.log(d)), cmap="gray")
```

```
[107]: Text(0.5, 1.0, 'Spatial/Image Data')
```



1.5.2 part b.)

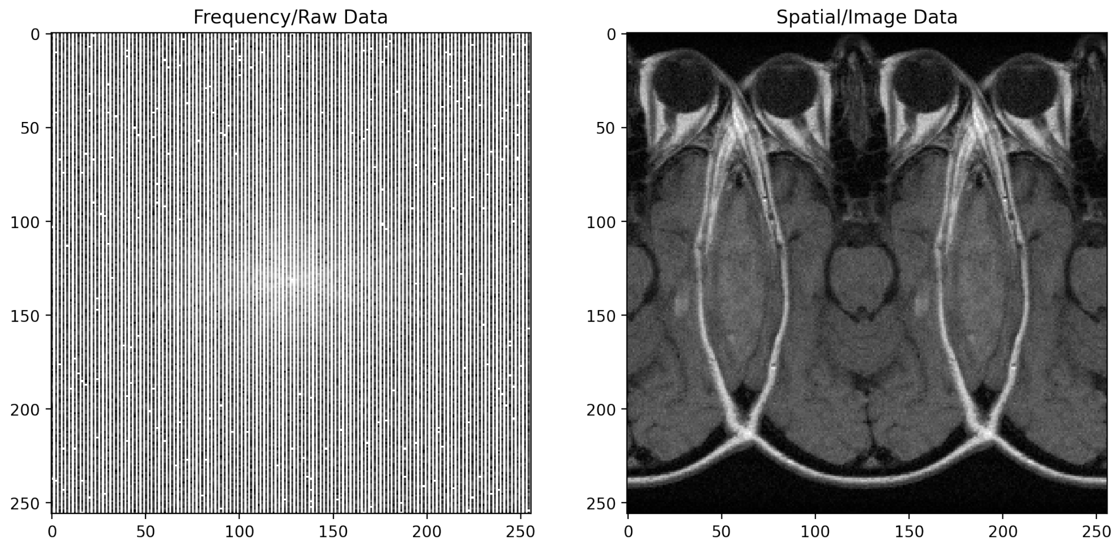
We can simulate reconstructing an image from half the original data (undersampled image):

```
[108]: # reconstruct image from half the original data
dhalf = np.zeros(d.shape, dtype=complex)
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[0].set_title("Frequency/Raw Data")
ax[1].imshow(np.abs(dhalf_recons), cmap="gray")
ax[1].set_title("Spatial/Image Data")
```

```
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_57512/1807182290.py:8
: RuntimeWarning: divide by zero encountered in log
  ax[0].imshow(np.abs(np.log(dhalf)), cmap="gray")
```

```
[108]: Text(0.5, 1.0, 'Spatial/Image Data')
```



We see a reflection of the MRI image on each side. The image is sampled in the frequency space, and thus the image is **undersampled** when we take out half the data which introduces aliasing affects to the image which causes the reflections.

1.5.3 part c.)

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

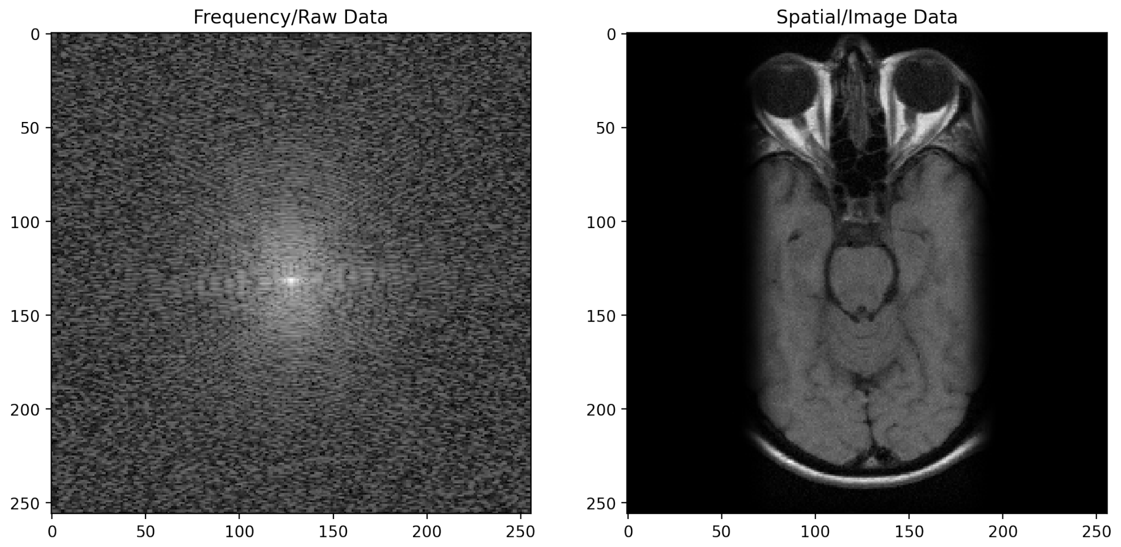
[109]:
```
cutoff = 325 # Empirically set this to a number less than 750.
FILTER = firwin(numtaps=33, cutoff=cutoff, pass_zero="lowpass", fs=1500)
dfilt = np.zeros(d.shape, dtype="complex")

# convolve along each row
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[0].set_title("Frequency/Raw Data")
ax[1].imshow(np.abs(dfilt_recons), cmap="gray")
ax[1].set_title("Spatial/Image Data")
```

[109]:
```
Text(0.5, 1.0, 'Spatial/Image Data')
```



1.5.4 part d.)

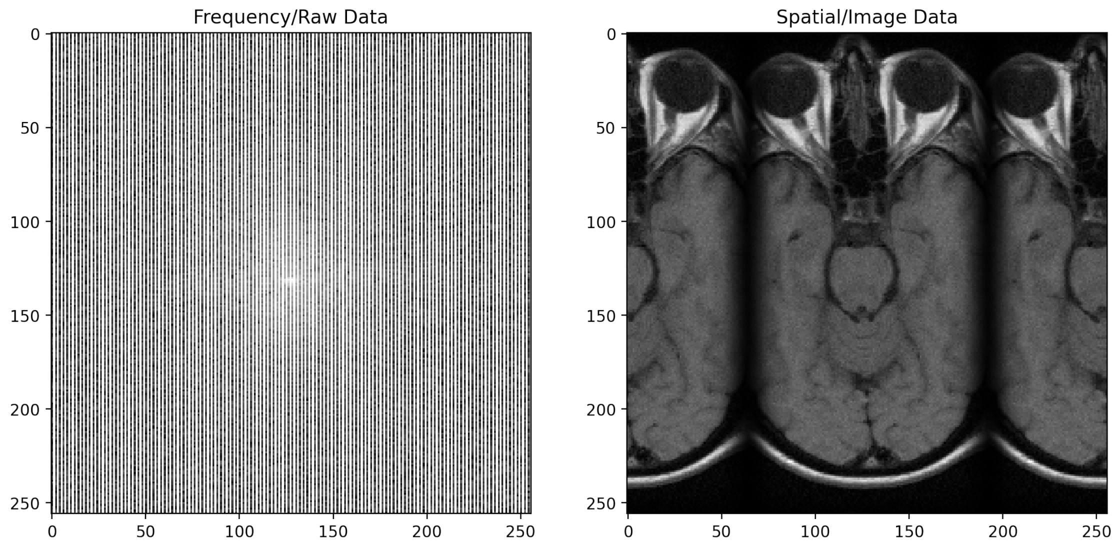
We can again simulate a subsample of the filtered data. We can see that even though the image is aliasing with itself, the frequencies that have aliased are cut off using our filter, and the image is easier to interpret.

[110]:
```
# reconstruct image from half the original data
dfilt_half = np.zeros(dfilt.shape, dtype=complex)
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[0].set_title("Frequency/Raw Data")
ax[1].imshow(np.abs(dfilt_half_recons), cmap="gray")
ax[1].set_title("Spatial/Image Data")
```

```
/var/folders/_w/yrr0qqbd52gc6jj0fnqyk8640000gn/T/ipykernel_57512/2794891123.py:8
: RuntimeWarning: divide by zero encountered in log
  ax[0].imshow(np.abs(np.log(dfilt_half)), cmap="gray")
```

[110]:
```
Text(0.5, 1.0, 'Spatial/Image Data')
```

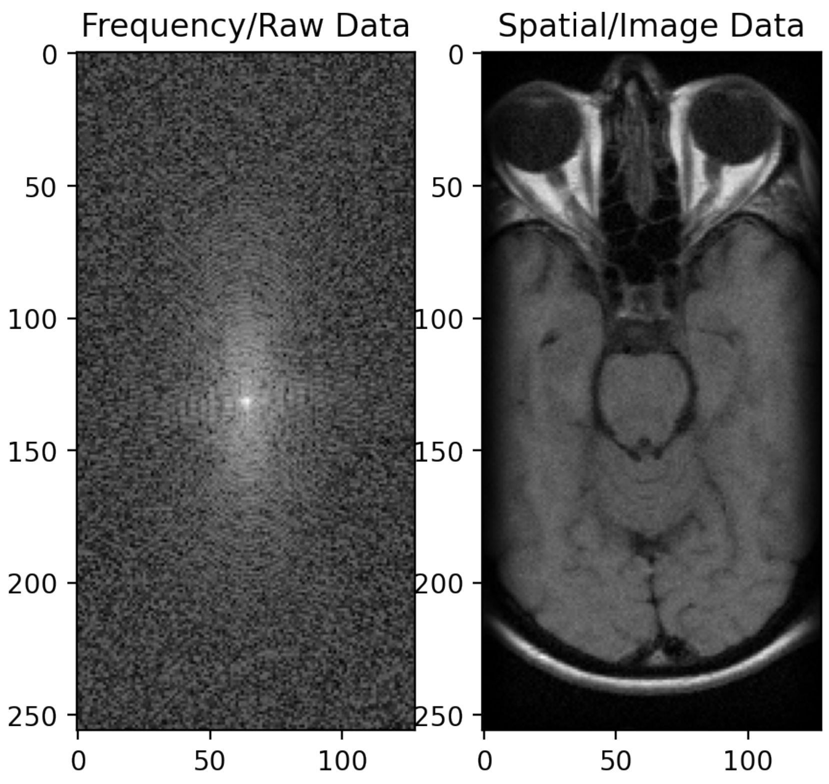


1.5.5 part e.)

[111]:
```
# generate and reconstruct a rectangular image
dfilt_rect = np.zeros((256,128), dtype=complex)
dfilt_rect[:, :]  = dfilt_half[:,0::2]
dfilt_rect_recons = ifft2c(dfilt_rect)

# plot images
_, ax = plt.subplots(1,2, figsize=(5,5))
ax[0].imshow(np.abs(np.log(dfilt_rect)), cmap="gray")
ax[0].set_title("Frequency/Raw Data")
ax[1].imshow(np.abs(dfilt_rect_recons), cmap="gray")
ax[1].set_title("Spatial/Image Data")
```

[111]:
```
Text(0.5, 1.0, 'Spatial/Image Data')
```


Frequency/Raw Data
Spatial/Image Data
0
50
100
150
200
250
0
50
100