ECE278C Imaging Systems

Lab 4: Multi Frequency Reconstruction

February 17, 2021

Student: Ivan Arevalo
Perm Number: 5613567
Email: ifa@ucsb.com

Department of Electrical and Computer Engineering, UCSB

0 Motivation

The main objective of this report is to explore image formation by multi-frequency backward propagation for a simplified 2D case source region. We will be repeating the experiments in Lab 2 where we reconstructed a source region by back propagation, but we will do so 40 times with increasing wavelengths.

1 Image Reconstruction by Inverse Filtering

Image reconstruction involves estimating the source distribution from the wave-field pattern measured at the aperture. The wave-filed generated by a point source in 2D space is mathematically described by Green's function:

$$h(x,y) = \frac{1}{\sqrt{j\lambda_0 r}} exp\frac{j2\pi r}{\lambda_0})$$
$$r = \sqrt{x^2+y^2}$$

We can therefore express the wave-field generated by any source distribution as a convolution of the source function with the impulse response of the system (Green's function):

$$s(x,y) * h(x,y) = g(x,y)$$

In Frequency domain:

$$S(f_x,f_y)H(f_x,f_y) = G(f_x,f_y)$$

In the context of a 2D wave-field pattern, we can consider a special case of line to line propagation. For this case, the source distribution is at a line at y_1 and the data acquisition is conducted at a line at y_2. The distance in between is therefore $y_0 = y_2 - y_1$.

$$S(f_x;y_1)H(f_x,y_0) = G(f_x,y_2)$$

$$H(f_x,y_0) = exp(j2\pi y_0\sqrt{\frac{1}{\lambda}-f_x^2})$$

We can now formulate a method to recover the source distribution by reversing the wave scattering process, often referred to as inverse filtering (Lee 2016). Inverting our line to line propagation filter found previously:

$$H^{-1}(f_x;y_0) = exp(-j2\pi y_0\sqrt{\frac{1}{\lambda}-f_x^2})$$
$$= H^*(f_x;y_0)$$

Given the transfer function is all-phase, its inverse is the same as its conjugate. Transforming back into the the space domain we get the equivalent expressions:

$$H(f_x; y_0) \rightarrow h(x, y) = \frac{1}{j\lambda r} exp(j2\pi r/\lambda)$$
$$H^*(f_x; y_0) \rightarrow h^*(-x, -y) = \frac{-1}{j\lambda r} exp(-j2\pi r/\lambda)$$

We conclude that given the transfer function is the conjugate of the forward wave-field function (Green's function), then the impulse response of the image reconstruction filter is the flipped and conjugated version of the forward function function. Given that the Green's function is even, flipping it is irrelevant and we just get the conjugated forward Green's function.

We can now state the forward and backward propagation equations as:

$$\text{Forward Propagation: } s(x, y) * h(x, y) = g(x, y)$$
$$\text{Backward Propagation: } g(x, y) * h(x, y)^* = \hat{s}(x, y)$$

where $s(x, y)$ is our source distribution, $h(x, y)$ is Green's function, $g(x, y)$ is our wave-field data, and $\hat{s}(s, y)$ is our reconstructed image.

We can now reconstruct the image by convolving the wave-field data sampled at the receiver with the conjugate of Green's function.

$$\text{Full Green's: } h^*(x, y) = \frac{-1}{j\lambda r} exp(-j2\pi r/\lambda)$$
$$\text{Phase-Only Green;s: } h^*(x, y) = exp(-j2\pi r/\lambda)$$

We will repeat the reconstruction with 40 different coherent wavelengths that have the following form:

$$\lambda_n = (40\lambda_0)/(n + 20) \quad n = 1, 2, ..., 40$$

We will then sequentially superimpose the reconstructed images to observe the converging reconstruction.

2 Simulation

We begin our simulation by generating the wave field for a the following 6-point source distribution:

	scatters	*scatter locations*
1	*(x_1, y_1)*	*(0, +10 λ)*
2	*(x_2, y_2)*	*(+10 λ, 0)*
3	*(x_3, y_3)*	*(0, -10 λ)*
4	*(x_4, y_4)*	*(-10 λ, 0)*
5	*(x_5, y_5)*	*(-8 λ, -6 λ)*
6	*(x_6, y_6)*	*(+8 λ, -6 λ)*

Figure 1: Source Distribution

The receiver aperture is a centered linear receiver array with a span of $60\lambda_0$ (from x = -$30\lambda_0$ to x = +$30\lambda_0$). The linear receiver array is located at $y = yo = -60\lambda_0$. With quarter-wavelength spacing ($\lambda_0/4$) spacing, there are 241 wave-field data samples in total captured over the $60\lambda_0$-long linear aperture. Figure 2 shows the source wave-field pattern for $\lambda = 1$ along with the receiver aperture highlighted in red.

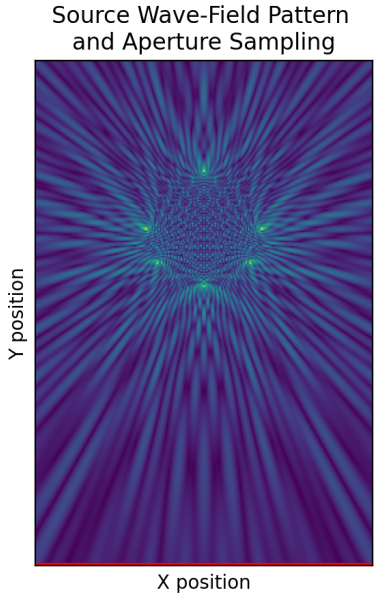

Figure 2: Source Wave-Field with Linear Receiver (y = -60) shown in red

We can now reconstruct the $60\lambda_0 \times 60\lambda_0$ source region by convolving our receiver data with Green's function. This is analogous to treating each of the 241 data samples at the receiver, as a point source and observing the resulting superimposed wave-field.

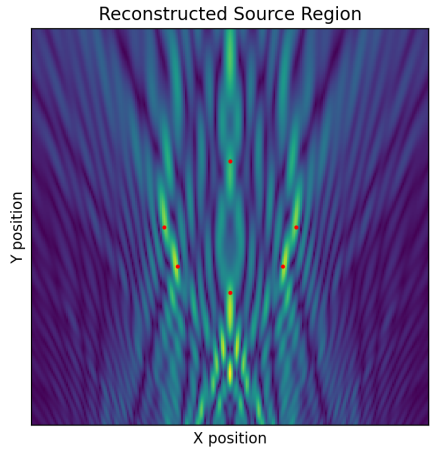

Figure 3: Single Wavelength Reconstruction

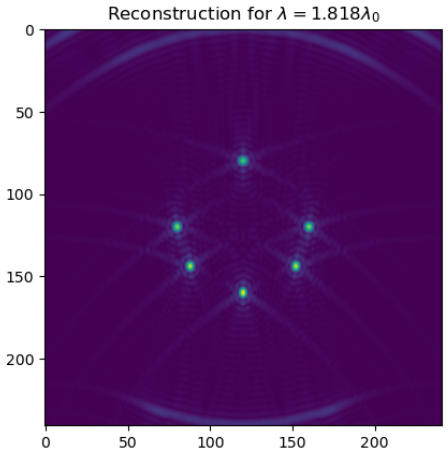

Figure 4: Multi-Wavelength Reconstruction

Figure 3 shows the reconstruction of the source region with only a single coherent wavelength of $\lambda = \lambda_0$ as done in Lab 2. We have overlay-ed the true source locations in red to observe accuracy. We can see that it has a good estimation of the source distribution in the x-direction but displays some distortion in the y-direction. Observing the superimposed reconstruction from 40 different coherent wavelengths in figure 4, we notice that the resolution in both x and y direction has significantly improved. It is clear that reconstructed the source region with more wavelengths leads to higher resolution. We will now explore why this is the case.

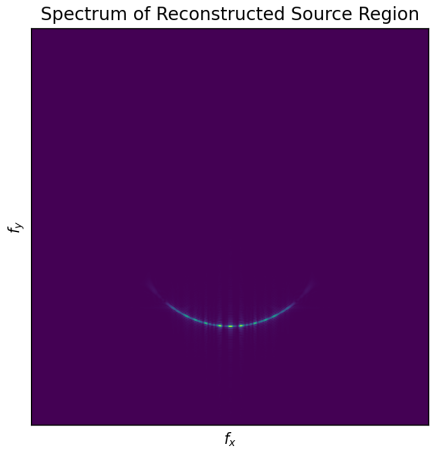

Figure 5: Single Wavelength Spectrum Reconstruction

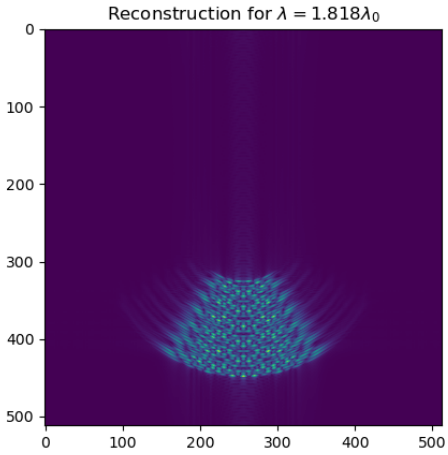

Figure 6: Multi-Wavelength Spectrum Reconstruction

Figure 5 show the spectrum of the reconstructed image with a single wavelength. We can observe that the reconstructed image spectrum has only a fraction of the original source spectrum in the direction facing the receiver aperture. We can see that spectrum being occupied along the x-direction is much higher than that occupied in the y-direction. This is why our reconstructed image has better resolution in the x-direction than the y-direction. In contrast, figure 6 shows the

spectrum of the superimposed reconstruction from the 40 wavelengths as described previously. It is clear that the spectrum occupied in the y-direction is much higher than for the single wavelength reconstruction and about the same as the spectrum occupied in the x-direction. This leads to a reconstructed image that is much sharper in both directions as shown in figure 4. The next few figures will show snapshots of the sequential reconstruction of the image and spectrum.

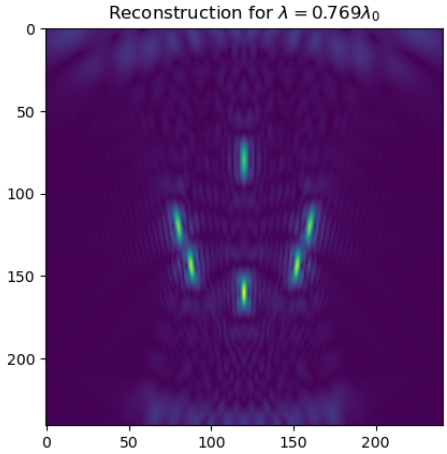

Figure 7: Sequential image reconstruction up to n = 10

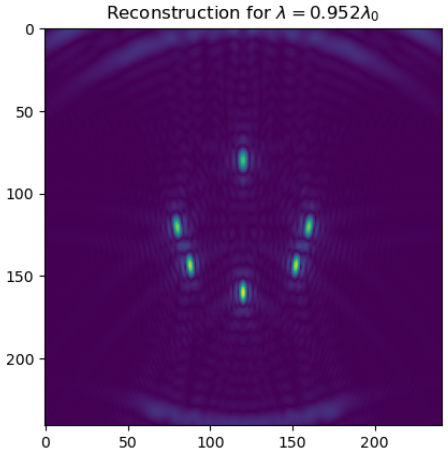

Figure 8: Sequential image reconstruction up to n = 20

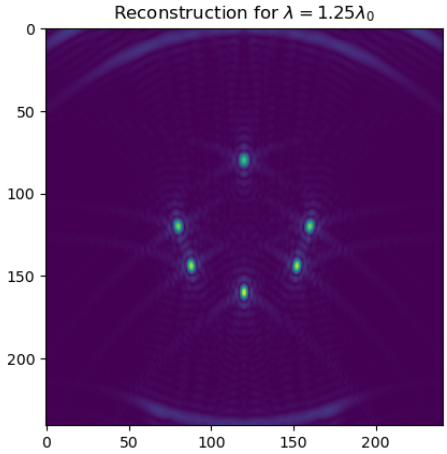

Figure 9: Sequential image reconstruction up to n = 30

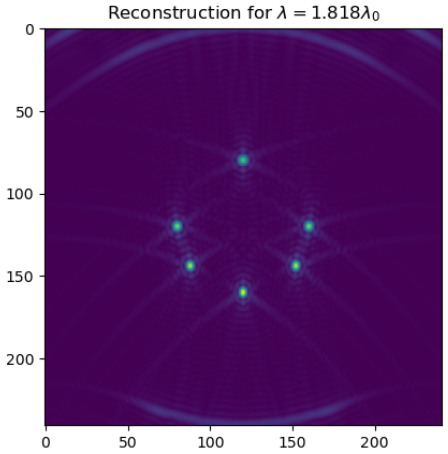

Figure 10: Sequential image reconstruction up to n = 40

The following is a link to the image reconstruction video: https://youtu.be/79yrmQFWizU
Figures 7-10 show the sequential reconstruction up to the given wavelength. We can observe how the resolution improves as we superimpose more reconstructions.

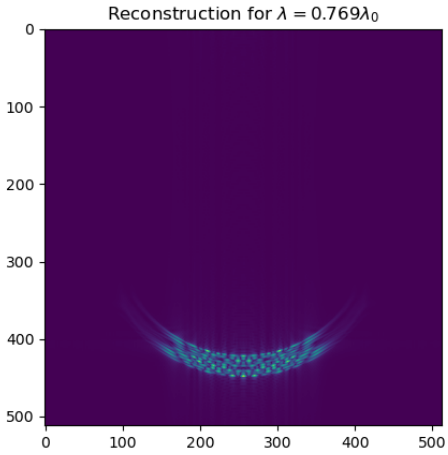

Figure 11: Sequential spectrum reconstruction up to n = 10

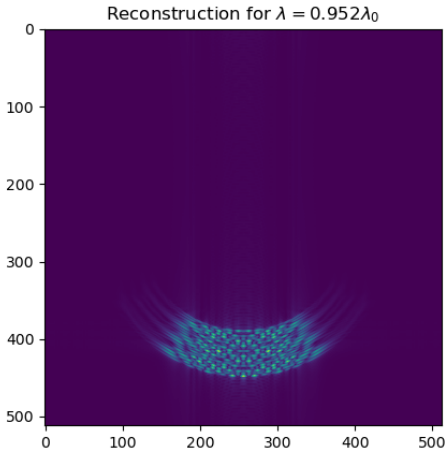

Figure 12: Sequential spectrum reconstruction up to n = 20

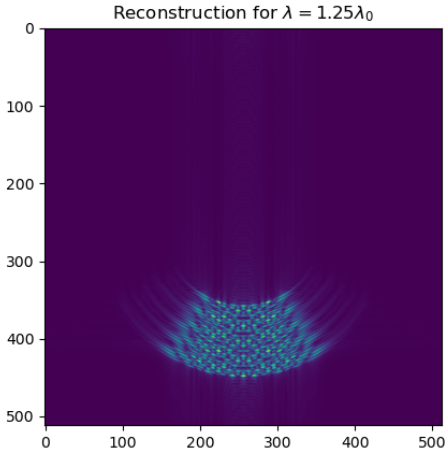

Figure 13: Sequential spectrum reconstruction up to n = 30

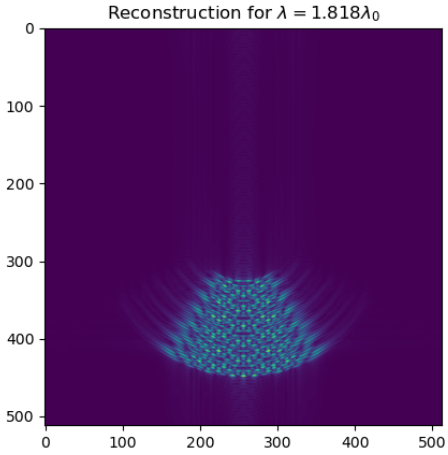

Figure 14: Sequential spectrum reconstruction up to n = 40

The following is a link to the spectrum reconstruction video: https://youtu.be/43n-PnthEfs

Figures 11-14 show the sequential reconstruction of the spectrum up to the given wavelength. We can observe how more spectrum is occupied along the y-direction as we superimpose more reconstructions.

3 Conclusions

We stated that Green's equation describes the wave-field pattern of a single source point and derived a method to reconstruct a source distribution by inverse filtering. We showed that the inverse of the forward propagation transfer function (Green's equation) is equal to its conjugate. We therefore just need to convolve the receiver data with the conjugate Green's equation to reconstruct our original

source distribution. This is analagous to treating our receiver data as the source and generating the resulting wave-field which superimposes in a way that reconstructs our image. We then compared the reconstruction from a single coherent wavelength and the superimposed reconstructions from 40 different wavelengths. It was clear from figures 3 and 4 that the multi-wavelength reconstruction had much higher resolution than the single wavelength reconstruction. Figure 5 and 6 showed that the single wavelength reconstruction displayed distortion along the y-axis given that its spectrum occupied very little along the y-direction. The spectrum of the superimposed reconstruction from the 40 wavelengths occupied much more in the y-direction and therefore had much higher resolution. We can conclude that resolution in the x axis is attributed to the length of the receiver while resolution in the y-axis is attribute to the number of frequencies used in the reconstruction.

4 References

Lee, Hua. *Acoustical Sensing and Imaging.* CRC Press, Taylor amp; Francis Group, 2016.

5 Code

```
import numpy as np
import matplotlib.pyplot as plt
import multiprocessing
import time
import os
from Lab1.wave_field import Wave_Field
from Lab2.image_reconstruction import LinearReceiver


def reconstruction(wavelength):

    receiver = LinearReceiver(y_int=-60, xlim=(-30, 30), slope=0)

    sources_xy_coordinates = np.array([[0, 10], [10, 0], [0, -10], [-10, 0], [-8, -6],
        [8, -6]])
    radius = 30
    sample_spacing = 1 / 4
    source_WF = Wave_Field(radius=radius, sample_spacing=sample_spacing)
    receiver.sample_wave_field(sources_xy_coordinates, source_WF, wavelength)

    # Reconstruct image with given wavelength
    ax = receiver.reconstruct_image(source_WF, wavelength, pattern_title=f"Reconstructed
        Source Region\n for $\lambda = {round(wavelength,3)}\lambda_0$")
    # Save image png and data npy matrix
    data_dir = os.path.join(os.path.dirname(__file__), 'data')
    plt.savefig("{}/{}_lambda.png".format(os.path.join(data_dir,
        'reconstructed_images'), wavelength))
    np.save("{}/{}_lambda".format(os.path.join(data_dir, 'reconstructed_image_data'),
        wavelength),
            receiver.reconstructed_wavefield.composite_wave_field_pattern)

    # # Reconstruct spectrum
```

```
    ax = receiver.reconstruct_spectrum(title=f"Wave Field Spectrum\n for $\lambda =
        {round(wavelength,3)}\lambda_0$")
    plt.savefig("{}/{}_lambda.png".format(os.path.join(data_dir,
        'reconstructed_spectrum'), wavelength))
    np.save("{}/{}_lambda".format(os.path.join(data_dir, 'reconstructed_spectrum_data'),
        wavelength),
            receiver.reconstructed_wavefield.wave_field_spectrum)


def multi_frequency_reconstruction(wavelengths):
    with multiprocessing.Pool() as pool:
        pool.map(reconstruction, wavelengths)


def superimpose_reconstructions(directory, vid_size, reconstruction_snapshots=None,
    video_name='reconstructed_image_video', FPS=2):
    first_im_flag = True
    superimposed_reconstructions = np.zeros((10, 10)) # Final reconstruction image
        initialization

    width = vid_size[0]
    height = vid_size[1]
    FPS = FPS

    fourcc = VideoWriter_fourcc(*'MP42')
    reconstructed_video = VideoWriter(os.path.join(directory, f"{video_name}.avi"),
        fourcc, float(FPS), (width, height), 0)

    for i, file in enumerate(sorted(os.listdir(directory))):
        if file.endswith(".npy"):
            print(os.path.join(directory, file))
            reconstruction = np.load(os.path.join(directory, file))
            if first_im_flag:
                superimposed_reconstructions = reconstruction
                first_im_flag = False
            else:
                superimposed_reconstructions += reconstruction

            if reconstruction_snapshots:
                if i+1 in reconstruction_snapshots:
                    wavelength = round(float(file.split('_')[0]), 3)
                    # fig, ax = plt.figure()
                    plt.imshow(np.abs(superimposed_reconstructions))
                    title = f"Reconstruction for $\lambda = {wavelength}\lambda_0$"
                    plt.title(title)
                    plt.savefig("{}/super_{}_lambda.png".format(directory,wavelength))

            # superimposed_reconstructions =
                (abs(superimposed_reconstructions)/np.max(superimposed_reconstructions))*255

            reconstructed_video.write((255*(np.abs(superimposed_reconstructions)/np.max(np.abs(superimpos

    reconstructed_video.release()
```

```
    plt.imshow(np.abs(superimposed_reconstructions))
    plt.title(video_name)
    plt.show()


if __name__ == "__main__":

    wavelengths = 40 / (np.arange(1,41) + 20)
    start_time = time.time()

    # multi_frequency_reconstruction(wavelengths)

    from cv2 import VideoWriter, VideoWriter_fourcc # Multiprocessing module raises
        error if imported before running
    root_dir = os.path.join(os.path.dirname(__file__), 'data/reconstructed_image_data')
    superimpose_reconstructions(root_dir, (241,241),
        reconstruction_snapshots=[10,20,30,40], video_name='reconstructed_image')
    root_dir = os.path.join(os.path.dirname(__file__),
        'data/reconstructed_spectrum_data')
    superimpose_reconstructions(root_dir, (512,512),
        reconstruction_snapshots=[10,20,30,40], video_name='reconstructed_spectrum')
    duration = time.time() - start_time
    print(f"Duration {duration} seconds")
```

```
#!/usr/bin/env

import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve2d


class Wave_Field():

    def __init__(self, radius, sample_spacing, phase_only=False):
        self.radius = radius
        self.sample_spacing = sample_spacing
        self.phase_only = phase_only

    class Point_Source():

        def __init__(self, radius, sample_spacing, wavelength=1, xshift=0, yshift=0,
            phase_only=False, amplitude=None):
            self.radius = radius
            self.sample_spacing = sample_spacing
            self.wavelength = wavelength
            self.xshift = xshift
            self.yshift = yshift
            self.phase_only = phase_only
            self.amplitude = amplitude
            self.v_wave_field_tf = np.vectorize(self.wave_field_tf, cache=False)

        def wave_field_tf(self, x, y):
```

```
        r = np.sqrt((x - self.xshift) ** 2 + (y - self.yshift) ** 2)
        if self.phase_only and self.amplitude:
            pass
        elif self.phase_only:
            amplitude = 1
        elif self.amplitude:
            amplitude = self.amplitude * (1 / np.sqrt(1j * self.wavelength * r))
        else:
            amplitude = (1 / np.sqrt(1j * self.wavelength * r))

        return amplitude * np.exp(1j * 2 * np.pi * r / self.wavelength)

    def generate_wave_field_pattern(self, xlim=None, ylim=None):

        if xlim and ylim:
            assert (isinstance(xlim, (tuple, list, np.ndarray)))
            assert (isinstance(ylim, (tuple, list, np.ndarray)))
            x = np.arange(xlim[0], xlim[1] + self.sample_spacing, self.sample_spacing)
            y = np.arange(ylim[0], ylim[1] + self.sample_spacing, self.sample_spacing)
        else:
            x = y = np.arange(-self.radius, self.radius + self.sample_spacing,
                self.sample_spacing)

        X, Y = np.meshgrid(x, y)
        wave_field_pattern = self.v_wave_field_tf(X, np.rot90(Y, 2)) # Y because of
            numpy indexing start from top left.

        return np.nan_to_num(wave_field_pattern)

    def sample_wave_field(self, coordinates):

        wave_field_pattern = self.v_wave_field_tf(coordinates[:,0], coordinates[:,1])
        return np.nan_to_num(wave_field_pattern)

def generate_composite_wave_field_pattern(self, xy_coordinates, wavelength,
    amplitude=None, xlim=None, ylim=None,
                                      pattern_title="Wave Filed Pattern",
                                          sample_mode=False,
                                      sample_coordinates=None, plot=True):

    first_pt_flag = True
    for i, xy in enumerate(np.asarray(xy_coordinates).reshape(-1,2)):

        w = wavelength[i] if isinstance(wavelength, np.ndarray) else wavelength
        a = amplitude[i] if isinstance(amplitude, np.ndarray) else amplitude
        point_source = self.Point_Source(self.radius, self.sample_spacing,
            wavelength=w, xshift=xy[0], yshift=xy[1],
                                     amplitude=a, phase_only=self.phase_only)

        if sample_mode:
            assert(sample_coordinates is not None)
            sample_pattern =
                point_source.sample_wave_field(coordinates=sample_coordinates)
            if first_pt_flag:
```

```
                self.sampled_wavefield = sample_pattern
                first_pt_flag = False
            else:
                self.sampled_wavefield += sample_pattern
        else:
            wave_field_pattern = point_source.generate_wave_field_pattern(xlim, ylim)

            if first_pt_flag:
                self.composite_wave_field_pattern = wave_field_pattern
                first_pt_flag = False
            else:
                self.composite_wave_field_pattern += wave_field_pattern
    if plot:
        ax = self.formatted_plot(np.abs(self.composite_wave_field_pattern),
            title=pattern_title, xlabel='X position',
                        ylabel='Y position')
        return ax


def generate_wave_field_spectrum(self, fft_size=512, title="Wave-Field Spectrum"):
    # Plot wave field spectrum
    self.wave_field_spectrum =
        np.fft.fftshift(np.fft.fft2(self.composite_wave_field_pattern, s=(fft_size,
        fft_size)))
    ax = self.formatted_plot(np.abs(self.wave_field_spectrum), title=title,
        xlabel='$f_x$', ylabel='$f_y$')
    return ax


def formatted_plot(self, data, title='', xlabel='', ylabel=''):
    fig, ax1 = plt.subplots(1, 1, sharex=False)
    ax1.set_title(title)
    ax1.set_xlabel(xlabel)
    ax1.set_ylabel(ylabel)
    ax1.tick_params(
        axis='both', # changes apply to the x-axis
        which='both', # both major and minor ticks are affected
        bottom=False, # ticks along the bottom edge are off
        top=False,
        left=False,
        right=False, # ticks along the top edge are off
        labelbottom=False,
        labelleft=False) # labels along the bottom edge are off
    ax1.imshow(np.abs(data))
    return ax1
```

```
import numpy as np
import matplotlib.pyplot as plt
from Lab1 import point_source
from Lab1.wave_field import Wave_Field
import time
from scipy.signal import convolve2d
```

```
class LinearReceiver():
    def __init__(self, y_int, xlim, slope=0):
        self.slope = slope
        self.y_int = y_int
        self.xlim = xlim

    def sample_wave_field(self, sources_xy_coordinates, source_WF, wavelength):
        # Sample wave field at receiver (y = mx + b)
        receiver_length = int((self.xlim[1] - self.xlim[0]) / source_WF.sample_spacing +
            1)
        # y_v = (-60) * np.ones(receiver_length)

        x_v = np.linspace(self.xlim[0], self.xlim[1], receiver_length)
        y_v = self.slope*x_v + self.y_int
        source_WF.receiver_coordinates = np.array([x_v, y_v]).transpose()

        # print(source_WF.receiver_coordinates)
        # print(source_WF.receiver_coordinates.shape)

        source_WF.generate_composite_wave_field_pattern(sources_xy_coordinates,
            wavelength=wavelength,
                                                    sample_mode=True,
                                                    sample_coordinates=source_WF.receiver_coordinates,
                                                        plot=False)

    def reconstruct_image(self, source_WF, wavelength, pattern_title = "Reconstructed
        Source Region"):
        # Reconstruct the source region with backpropagation
        assert(hasattr(source_WF, 'receiver_coordinates'))
        assert(hasattr(source_WF, 'sampled_wavefield'))
        wave_field_at_receiver = np.conjugate(source_WF.sampled_wavefield)
        self.reconstructed_wavefield = Wave_Field(source_WF.radius,
            source_WF.sample_spacing)
        self.reconstructed_wavefield.generate_composite_wave_field_pattern(source_WF.receiver_coordinate
            wavelength,
                                                                wave_field_at_receiver,
                                                                    plot=False)
        self.reconstructed_wavefield.composite_wave_field_pattern =
            np.conjugate(self.reconstructed_wavefield.composite_wave_field_pattern)
        ax =
            self.reconstructed_wavefield.formatted_plot(self.reconstructed_wavefield.composite_wave_fiel
            pattern_title)
        return ax

    def reconstruct_spectrum(self, fft_size=512, title="Wave-Field Spectrum"):
        ax = self.reconstructed_wavefield.generate_wave_field_spectrum(fft_size=fft_size,
            title=title)
        return ax


def reconstruct_image(wavelength=1):
    sources_xy_coordinates = np.array([[0, 10], [10, 0], [0, -10], [-10, 0], [-8, -6],
        [8, -6]])
    pattern_title = "6 Point Source Wave-field Pattern"
```

```
    spectrum_title = "6 Point Source Wave-field Spectrum"
    radius = 30
    sample_spacing = 1 / 4

    # Create field pattern
    source_WF = Wave_Field(radius=radius, sample_spacing=sample_spacing)
    receiver_length = int(2*radius/sample_spacing +1)
    y_v = (-60) * np.ones(receiver_length)
    x_v = np.linspace(-radius, radius, receiver_length)
    receiver_coordinates = np.array([x_v, y_v]).transpose()
    print(receiver_coordinates)
    print(receiver_coordinates.shape)
    source_WF.generate_composite_wave_field_pattern(sources_xy_coordinates,
        wavelength=wavelength,
                                                sample_mode=True,
                                                sample_coordinates=receiver_coordinates,
                                                    plot=False)

    # Reconstruct the source region with backpropagation
    wave_field_at_receiver = np.conjugate(source_WF.sampled_wavefield)
    reconstructed_image = Wave_Field(radius, sample_spacing)
    pattern_title = "Reconstructed Source Region"
    ax = reconstructed_image.generate_composite_wave_field_pattern(receiver_coordinates,
        1,
                                                                   wave_field_at_receiver,
                                                                   plot=False)
    reconstructed_image_WF =
        np.conjugate(reconstructed_image.composite_wave_field_pattern)
    ax = reconstructed_image.formatted_plot(reconstructed_image_WF, pattern_title)

def main():
    receiver = LinearReceiver(y_int=-60, xlim=(-30,30), slope=0)

    sources_xy_coordinates = np.array([[0, 10], [10, 0], [0, -10], [-10, 0], [-8, -6],
        [8, -6]])
    radius = 30
    sample_spacing = 1/4
    wavelength = 1
    source_WF = Wave_Field(radius=radius, sample_spacing=sample_spacing)
    receiver.sample_wave_field(sources_xy_coordinates, source_WF, wavelength)
    ax = receiver.reconstruct_image(source_WF, wavelength)
    ax = receiver.reconstruct_spectrum()
    return ax

if __name__ == '__main__':

    start_time = time.time()
    ax = main()
    duration = time.time()-start_time
    print(duration)
    plt.show()
```