PH 821: Gravitational Wave Physics and Astronomy

The Effect of Gravitational Waves on a Ring of particles

Aditya Pawan
23D1069
Department of Physics
Indian Institute of Technology Bombay

25 November, 2024

ACKNOWLEDGEMENT

I would like thank everyone who helped me with this report. First, I would like to express my gratitude to my supervisor Varun Bhalerao, my lab members and my classmates in this course who helped clear doubts and test important components of the simulation in this project.

I am indebted to my friends and family for their guidance, assistance, and time. I am lastly grateful to the Indian Institute of Technology Bombay for providing the necessary resources and facilities to learn about this field in a suitable manner.

Mumbai - 400 076 **Aditya Pawan Saikia**
November 2024

Contents

1 Introduction

Gravitational Waves are one of the most widely known and popular phenomena that is mathematically predicted by Einstein's General theory of Relativity. Although their existence has been predicted for well over 120 years starting with Oliver Heaviside in 1893, indirect evidence for gravitational waves has only been around for 50 years from observations of the Hulse-Taylor binary orbital decay, and direct evidence for just 9 years from the LIGO gravitational wave detectors.

In this project we will focus on the effect of monochromatic gravitational waves on matter, with a focus on visualization of this effect on a ring of particles using a simulation. Through the use of basic coordinate transformations, we will show how the particles get affected by a compact binary source and its orientation with respect to the ring.

Accompanying this project document will be a Jupyter notebook (ADITYA-PAWAN.ipynb) where all the plots given in this term paper are clearly outlined, along with some interesting interactive plots. This code is also attached at the end of this report in section 5, a *.ipynb.txt* file is also attached that can be directly renamed to run as a Jupyter notebook. The code can also be found at the github repository Grav-Ring-Effect-2024.

Disclaimer: This project will not go into detail on how gravitational waves are generated from compact binary sources, and how the amplitudes and the waveforms are generated. As such, the orbit of the compact binary is simplified to be a perfect circle. In addition, all amplitudes are amplified for better visualization.

2 A Brief Overview of TT-gauge Gravitational Waves

We will follow the derivation as followed in chapter 2 of *Michele Maggiore - Gravitational Waves, Volume 1 (Theory and Experiment)* [1].

Consider the metric tensor of flat space at the detector placed at point P:

$$ds^2 \simeq c^2 dt^2 - \delta_{ij} dx^i dx^j \tag{1}$$

In the presence of gravitational waves, assume that the corrections to this metric (that are proportional to the first derivative of $g_{\mu\nu}$) to linear order in $|x^i|$ vanish at P due to this being a freely falling frame. When this metric is expanded to second order, the second derivatives of $g_{\mu\nu}$ are expressed in terms of the Riemann tensor, so the metric from equation 1 is now

$$ds^2 \simeq c^2 dt^2 [1 + R_{0i0j} x^i x^j] + 2cdt\ dx^i \left(\frac{2}{3} R_{0jik} x^j x^k\right) - dx^i dx^j \left[\delta_{ij} - \frac{1}{3} R_{ijkl} x^k x^l\right] \tag{2}$$

Define the deviation in geodesics from the flat frame to this new frame as ξ^μ defined in the coordinate transformation $x^\mu \to x'^\mu = x^\mu + \xi^\mu(x)$ we can use the form of the geodesic equation given below and substitute equation 2 into it:

$$\frac{d^2\xi^\mu}{d\tau^2} + 2\Gamma^\mu_{\nu\rho}(x) \frac{dx^\nu}{d\tau} \frac{dx^\rho}{d\tau} + \xi^\sigma \partial_\sigma \Gamma^\mu_{\nu\rho}(x) \frac{dx^\nu}{d\tau} \frac{dx^\rho}{d\tau} = 0 \tag{3}$$

Using the form of the metric given in equation 2, and assuming that the detector only moves non-relativistically so $dx^i/d\tau \ll dx^0/d\tau$, we get

$$\frac{d^2\xi^\mu}{d\tau^2} + \xi^\sigma \partial_\sigma \Gamma^\mu_{00} \left(\frac{dx^0}{d\tau}\right)^2 = 0 \tag{4}$$

Since $g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu}$ and $h \sim \mathcal{O}(x^i x^j)$, only the spatial derivatives will produce a non-zero contribution. In other words, $\xi^\sigma \partial_\sigma \Gamma^\mu_{00} = \xi^j \partial_j \Gamma^i_{00}$. Further, $R^i_{0j0} = \partial_j \Gamma^i_{00} - \partial_0 \Gamma^i_{0j} = \partial_j \Gamma^i_{00}$. Therefore,

$$\frac{d^2\xi^\mu}{d\tau^2} = -R^i_{0j0} \xi^j \left(\frac{dx^0}{d\tau}\right)^2 \tag{5}$$

Our test mass is initially at rest, and through the passage of the gravitational wave gains a velocity $dx^i/d\tau = c\ \mathcal{O}(h)$. Now,

$$dt = d\tau\sqrt{1 + \frac{1}{c^2}\frac{dx^i}{d\tau}\frac{dx^j}{d\tau}} = d\tau\sqrt{1 + \mathcal{O}(h^2)} \tag{6}$$

and we only consider linear order terms in h, $d\tau$ can be replaced by dt in equation 5, so $dx^0/d\tau = c$, and we get

$$\ddot{\xi} = -c^2 R^i_{0j0}\xi^j \tag{7}$$

Since we are working in the TT gauge, $R^i_{0j0} = R_{i0j0} = -(1/2c^2)\ddot{h}_{ij}^{TT}$. Therefore, the geodesic equation we are working with in the proper detector frame is simplified as

$$\ddot{\xi^i} = \frac{1}{2}\ddot{h}_{ij}^{TT}\xi^j \tag{8}$$

In the next section we can apply equation 8 to our simulation of the ring of particles.

3 The Gravitational Wave Simulation

The ring of particles that we are working with as our detector can be easily modeled using equation 8. We will assume for simplicity that this ring sits in the X-Y plane. First, we will construct a model for the simplified case of the gravitational wave traveling perpendicular to the plane of the ring, i.e. along the Z-axis. Later, we will consider the general case for a wave being produced by a compact binary source in an arbitrary direction to the ring.

3.1 A Two-Dimensional (Simple) Model

For a gravitational wave traveling along the Z-axis, the perturbation to the metric is given as

$$h_{ab}^{TT} = \sin\omega t\begin{pmatrix} h_+ & h_\times \\ h_\times & -h_+ \end{pmatrix} \tag{9}$$

Where h_+ and $h_\times$ are the individual amplitudes of the gravitational wave polarizations, and $a, b = 1, 2$ are the indices in the transverse plane. The above equation chooses $h_{ab}^{TT} = 0$ at $t = 0$. Writing $\xi_a(t) = (x_0 + \delta x(t), y_0 + \delta y(t))$ where (x_0, y_0) are the unperturbed positions and $(\delta x(t), \delta y(t))$ are the changes in the coordinates induced by the gravitational wave, equation 8 is split into two as:

$$\begin{aligned}\delta\ddot{x} &= -\frac{h_+}{2}(x_0 + \delta x)\omega^2\sin\omega t - \frac{h_\times}{2}(y_0 + \delta y)\omega^2\sin\omega t \\ \delta\ddot{y} &= \frac{h_+}{2}(y_0 + \delta y)\omega^2\sin\omega t - \frac{h_\times}{2}(x_0 + \delta x)\omega^2\sin\omega t\end{aligned} \tag{10}$$

The $\delta x, \delta y$ terms in equation 10 can be safely ignored since those are linear in h_+ and $h_\times$. Therefore, they can be integrated, and we get

$$\begin{aligned}\delta x(t) &= \frac{1}{2}(h_+ x_0 + h_\times y_0)\sin\omega t \\ \delta y(t) &= \frac{1}{2}(h_\times x_0 - h_+ y_0)\sin\omega t\end{aligned} \tag{11}$$

Now considering that $(x_0, y_0) = (\cos\alpha_i, \sin\alpha_i)$ for $\alpha_i = 2\pi/n_i$ where $i = 1, 2, \ldots N$ for N particles, we can easily model the behavior of a ring of N particles. The table given in the following page illustrates how the motion of the particles vary with the phase of the passing wave.

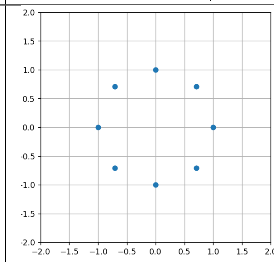
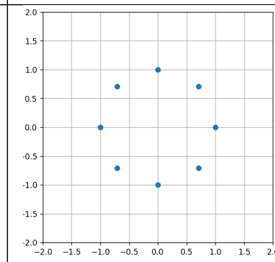
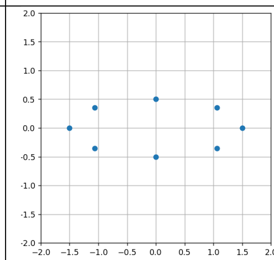
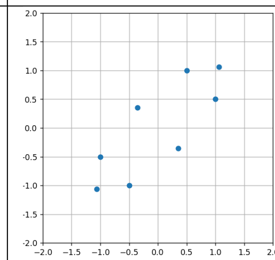
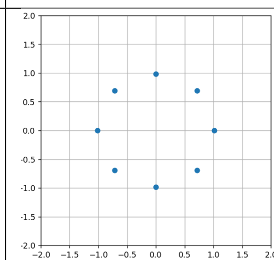
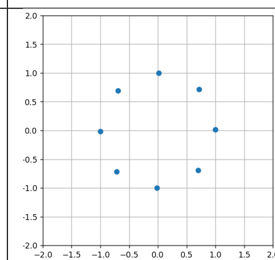
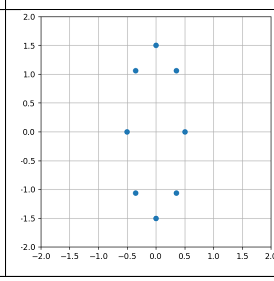
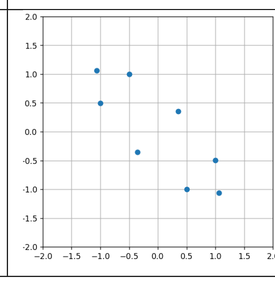
ωt	Pure h_+	Pure $h_\times$
0		
$\pi/2$		
π		
$3\pi/2$		

Table 1: Results from the simulation illustrating how the ring deforms due to the two different polarizations.

3.2 A Three-Dimensional (Complex) Model

In this simulation we are considering the general case where our gravitational wave is arriving at the ring of particles (which is still spread over the X-Y plane) from an arbitrary direction. Not only that, since the source of these gravitational waves are compact binary objects, the orientation of the orbit will influence the effect that is observed on the ring.

The challenge is to set up the orientation of the orbit with respect to the ring of particles, then use coordinate transformations to convert h_{ij}^{TT} from the source frame to the detector frame.

3.2.1 Setting up the orientation

The three dimensional coordinates of a circle of radius a is given by the parametrization (x, y, z) $= (a\cos\alpha, a\sin\alpha, 0)$ for $\alpha \in [0, 2\pi]$. However, our circle is not centered at the origin, but at an arbitrary point
$\vec{P} = (r\sin\theta\cos\phi, r\sin\theta\sin\phi, r\cos\theta)$. We define our circle as the set of points $\vec{C}$ such that

$$\vec{C} = \vec{P} + a\cos(\alpha)\hat{u} + a\sin(\alpha)\hat{v} \tag{12}$$

Where the unit vectors $\hat{u}$ and $\hat{v}$ form an orthogonal basis. Another way of interpreting these unit vectors is that $\hat{u}$ is the tangent vector, and $\hat{v}$ is the normal vector. With this interpretation, we can deduce that $\hat{u}$ is just $\hat{j}$ rotated along the Z-axis by an angle ϕ, and $\hat{v}$ is just $\hat{i}$ rotated along the Z-axis by an angle ϕ and rotated along the Y-axis by an angle θ. Therefore,

$$\begin{aligned} \hat{u} &= \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -\sin\phi \\ \cos\phi \\ 0 \end{pmatrix} \\ \hat{v} &= \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -\cos\theta\cos\phi \\ \cos\theta\sin\phi \\ \sin\theta \end{pmatrix} \end{aligned} \tag{13}$$

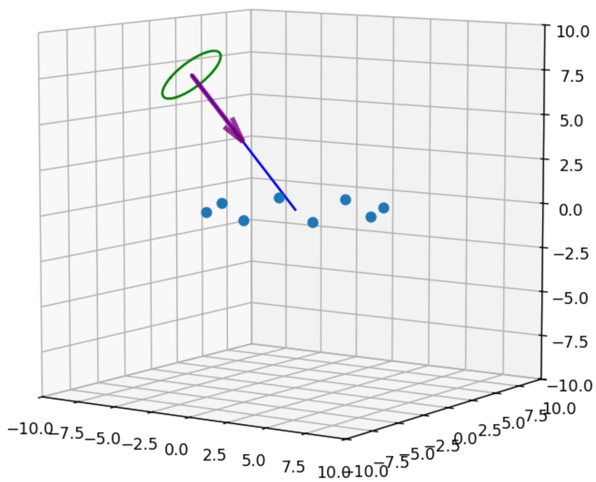


Figure 1: Figure of the 3D representation of our model. The ring of particles are positioned on the X-Y plane with the green circle representing the orbit of the compact binary. The purple arrow is the direction of $\hat{w}$. The circle coordinates are found using equations 12 and 13.

However, as we mentioned at the beginning of the section, the orbit of the compact binary may not be inclined towards the ring of particles. In other words, the normal to the plane of the circle (defined as $\hat{w} = \hat{u} \times \hat{v}$, as shown in figure 1) may be pointing away from $-\vec{P}$ by an angle ψ, called the *Polarization angle*.

The choice of axis around which $\hat{w}$ is rotated may be arbitrary, but for this project we assume that the orbit is rotated while keeping the normal ($\hat{v}$) constant. The new tangent unit vector may thus be calculated using Rodrigues' rotation formula as,

$$\begin{aligned} \hat{u}' &= \hat{u}\cos\psi + (\hat{v} \times \hat{u})\sin\psi + \hat{v}(\hat{v}.\hat{u})(1 - \cos\psi) \\ &= \hat{u}\cos\psi - \hat{w}\sin\psi \end{aligned} \tag{14}$$

Hence, using equations 12, 13 and 14, we finally obtain the equation for the coordinates of a circle for an arbitrary polarization angle:

$$\vec{C} = \begin{cases} x(\alpha) = & r\sin\theta\cos\phi - a\cos\alpha\cos\psi\sin\phi + a\cos\alpha\sin\psi\sin\theta\cos\phi + a\sin\alpha\cos\theta\cos\phi \\ y(\alpha) = & r\sin\theta\sin\phi + a\cos\alpha\cos\psi\cos\phi + a\cos\alpha\sin\psi\sin\theta\sin\phi + a\sin\alpha\cos\theta\sin\phi \\ z(\alpha) = & r\cos\theta + a\cos\alpha\sin\psi\cos\theta - a\sin\alpha\sin\theta \end{cases} \tag{15}$$

3.2.2 Solving the frame conversion

On a physics level, what happens to h_{ij}^{TT}? Equations 13 and 14 provide us with a clue: h_{ij}^{TT} is first rotated by an angle ϕ about the Z-axis, followed by a rotation by θ about the Y-axis, then another rotation about the Z-axis for the polarization angle ψ. The final result is therefore,

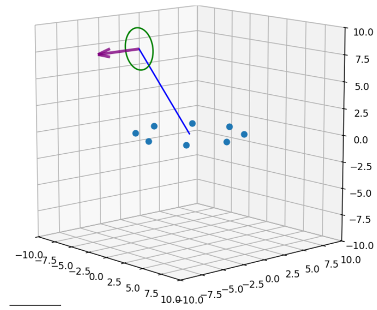


Figure 2: Same as figure 1, but the polarization angle has been changed to π. The circle coordinates are calculated using equation 15.

$$h = F_+ h_+ + F_\times h_\times \tag{16}$$

Where F_+ and $F_\times$ are termed as "antenna" functions, and they are defined as,

$$\begin{aligned} F_+(\theta, \phi, \psi) =& \frac{1}{2}(1 + \cos^2\theta)\cos 2\phi \cos 2\psi - \cos\theta \sin 2\phi \sin 2\psi \\ F_\times(\theta, \phi, \psi) =& \frac{1}{2}(1 + \cos^2\theta)\cos 2\phi \sin 2\psi + \cos\theta \sin 2\phi \cos 2\psi \end{aligned} \tag{17}$$

From here, h from the above equation is substituted into equation 11 to calculate δx and δy. A more involved visualization is found in the Jupyter notebook submitted along with this report, as well in section 5.

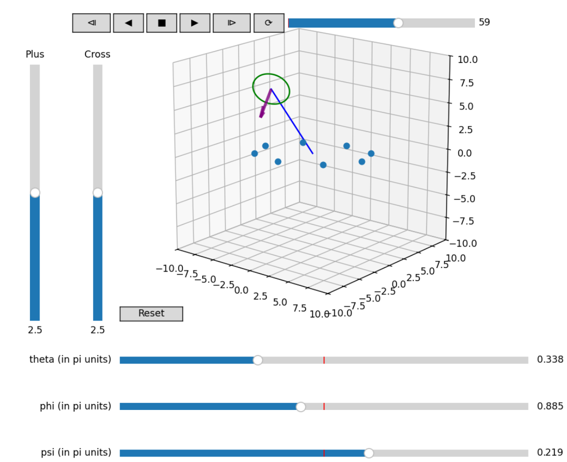


Figure 3: A snapshot of the simulation interface running on VS Code.

4 Summary

In this report we have derived the coordinate deviations of individual particles of the ring under action from a monochromatic gravitational wave originating from either a plane perpendicular to the ring of particles, or from an arbitrarily oriented plane. As stated, further work may be done to include the orbital parameters, but they are beyond the scope of this project.

Thank you for reading this project report.

5 Appendix: Code for simulating GW effect on ring

```
#%%
# Supplementary Jupyter notebook for PH 821 project
### Project title: The Effect of Gravitational Waves on a Ring of Particles
"""
Presented by: Aditya Pawan (23D1069)
-----------------------------------
In this notebook, I will demonstrate how to simulate the effects of gravitational waves on a
     ring of particles. We will first look at a two-dimensional representation of the ring
    of particles, then simulate the response of a monochromatic gravitational wave with the
    polarization amplitudes being interactable.

Later, we shall move to a three-dimensional representation, which allows us to simulate the
    effect of these gravitational waves originating from a compact binary source with
    arbitrary position and binary inclination with respect to the ring frame.

All effects are amplified for better visualization.

Disclaimer: If any animations or widgets break/start lagging, please restart the kernel and
    run the relevant cells again.
"""
#%%

#%%
# Dependencies
"""
Please check if the following modules are installed in your environment:
numpy v1.26.4
matplotlib v3.9.1
mpl_toolkits
"""

import numpy as np
import matplotlib.pyplot as pl
#%%

#%%
"""
These specific submodules, functions and classes will be used in this project to produce
    interactive, animated plots.
"""
from mpl_toolkits import mplot3d
import matplotlib.figure as figure
import matplotlib.axes as axes
from matplotlib.animation import FuncAnimation
import mpl_toolkits.axes_grid1
import matplotlib.widgets as widgets
#%%

#%%
"""
Matplotlib widget is required for the more interactive plots that contain matplotlib.widget.
    Slider and
matplotlib.widget.Button, as well as FuncAnimation codes. If you wish to switch to a
    different backend, you may
wish to enter your backend of choice. If you wish to use the default (inline) backend for
    non-interactive and
non-animated plots, do not run this cell.
"""
%matplotlib widget
#%%

#%%
## Part 1: Simulating a Two-Dimensional Ring of particles
"""

---
Let us start with a basic plot containing a ring of n = 8 particles.

The cell below will be where our simulation will happen, so move back to it after running
```

```python
    the cells below it.
"""
#%%

#%%
fig, ax = pl.subplots()
pl.axis('scaled')

n = 8
theta = np.arange(0,2*np.pi,2*np.pi/n)
x = np.cos(theta)
y = np.sin(theta)
sc, = ax.plot(x, y, marker = 'o', ls = "")
ax.grid(True)
ax.set_xlim(-4.0, 4.0)
ax.set_ylim(-4.0, 4.0)
pl.show()
#%%

#%%
"""
Let us now add two axes which contain sliders that enable us to vary both plus and cross polarizations.
"""
#%%

#%%
fig.subplots_adjust(bottom=0.3)

axplus = fig.add_axes([0.35, 0.2, 0.55, 0.03])
hplus = widgets.Slider(ax=axplus,label='Plus Polarization Amplitude',valmin=0.0,valmax=1.0,valinit=0.5)

axcross = fig.add_axes([0.35, 0.1, 0.55, 0.03])
hcross = widgets.Slider(ax=axcross,label='Cross Polarization Amplitude',valmin=0.0,valmax=1.0,valinit=0.5)

def pos(val):
    hp = hplus.val
    hc = hcross.val

hplus.on_changed(pos)
hcross.on_changed(pos)
#%%

#%%
"""
We would not be able to visualize anything on this plot currently since to do that we would need to see it in motion. Fortunately, Matplotlib's animation sub-module contains the class FuncAnimation that allows us to animate a figure using an update function.

However, the ability to manipulate the passage of time cannot be changed while the code is running. To provide more control to the user, I introduce the "Player" class, which calls the FuncAnimation function and automatically adds an axis that has a slider and buttons for time manipulation.
"""
#%%

#%%
class Player(FuncAnimation):
    def __init__(self, fig, func, frames = None, init_func = None, fargs = None, save_count = None, mini = 0, maxi = 100, pos = (0.125, 0.92), cache_frame_data = False, **kwargs):
        self.i = 0
        self.min = mini
        self.max = maxi
        self.runs = True
        self.forwards = True
        self.looped = False
        self.fig = fig
        self.func = func
        self.setup(pos)
```

```
        FuncAnimation.__init__(self, self.fig, self.update, frames=self.play(), init_func=init_func, fargs=fargs, save_count=save_count, cache_frame_data=cache_frame_data, **kwargs)

    def play(self):
        while self.runs:
            self.i = self.i + self.forwards - (not self.forwards)
            if self.i > self.min and self.i < self.max:
                yield self.i
            else:
                self.stop()
                yield self.i

    def start(self):
        self.runs = True
        self.event_source.start()

    def stop(self, event=None):
        if not self.looped:
            self.runs = False
            self.event_source.stop()
        else:
            self.forwards = not self.forwards

    def forward(self, event=None):
        self.forwards = True
        self.start()

    def backward(self, event=None):
        self.forwards = False
        self.start()

    def oneforward(self, event=None):
        self.forwards = True
        self.onestep()

    def onebackward(self, event=None):
        self.forwards = False
        self.onestep()

    def loop(self, event=None):
        self.looped = not self.looped

    def onestep(self):
        if self.i > self.min and self.i < self.max:
            self.i = self.i + self.forwards - (not self.forwards)
        elif self.i == self.min and self.forwards:
            self.i += 1
        elif self.i == self.max and not self.forwards:
            self.i -= 1
        self.func(self.i)
        self.slider.set_val(self.i)
        self.fig.canvas.draw_idle()

    def setup(self, pos):
        playerax = self.fig.add_axes([pos[0], pos[1], 0.64, 0.04])
        divider = mpl_toolkits.axes_grid1.make_axes_locatable(playerax)
        bax      = divider.append_axes("right", size="80%",  pad=0.05)
        sax      = divider.append_axes("right", size="80%",  pad=0.05)
        fax      = divider.append_axes("right", size="80%",  pad=0.05)
        ofax     = divider.append_axes("right", size="100%", pad=0.05)
        rax = divider.append_axes("right", size="80%", pad=0.05)
        sliderax = divider.append_axes("right", size="500%", pad=0.07)


        self.button_oneback    = widgets.Button(playerax,  label="$\u29CF$")
        self.button_back       = widgets.Button(bax,       label="$\u25C0$")
        self.button_stop       = widgets.Button(sax,       label="$\u25A0$")
        self.button_forward    = widgets.Button(fax,       label="$\u25B6$")
        self.button_oneforward = widgets.Button(ofax,      label="$\u29D0$")
        self.button_loop       = widgets.Button(rax,       label='$\u27F3$')
```

```
        self.button_oneback.on_clicked(self.onebackward)
        self.button_back.on_clicked(self.backward)
        self.button_stop.on_clicked(self.stop)
        self.button_forward.on_clicked(self.forward)
        self.button_oneforward.on_clicked(self.oneforward)
        self.button_loop.on_clicked(self.loop)

        self.slider = widgets.Slider(sliderax, '', self.min, self.max, valinit=self.i)

        self.slider.on_changed(self.set_pos)

    def set_pos(self, i):
        self.i = int(self.slider.val)
        self.func(self.i)

    def update(self, i):
        self.slider.set_val(i)
#%%

#%%
"""
Now with the aforementioned _Player_ class in place, we construct an update function for it
    to update our figure continuously (or as the time slider changes).

Run the cell below and go back to our plot to simulate Gravitational Waves!
"""
#%%

#%%
def pos_ret_XY(val):
    t = np.linspace(0,20,102)
    hp = hplus.val*np.sin(2*np.pi*t/20)
    hc = hcross.val*np.sin(2*np.pi*t/20)
    dx = np.array([np.real(0.5*hp*np.cos(theta[i]) + 0.5*hc*np.sin(theta[i])) for i in range
    (len(theta))])
    dy = np.array([np.real(-0.5*hp*np.sin(theta[i]) + 0.5*hc*np.cos(theta[i])) for i in
    range(len(theta))])
    X = np.zeros((len(t), len(theta)))
    Y = np.zeros((len(t), len(theta)))
    X[0], Y[0] = x, y

    for i in range(val + 1):
        X[i+1], Y[i+1] = x + dx[:,i], y + dy[:,i]
    return X, Y


def update(i):
    X, Y = pos_ret_XY(i)
    xt = X[i+1]
    yt = Y[i+1]
    sc.set_data(xt, yt)

ani = Player(fig, update, maxi=100, interval = 60)
#%%

#%%
## Part 2: Simulating the effect of a compact binary source in Three Dimensions
"""
---
Now that we have seen how gravitational waves affect a ring of particles propagated
    perpendicular to the plane where the ring sits, the next step is to of course ask how
    the same gravitational wave's effect will affect the ring when the wave is propagating
    from an arbitrary direction.

Not only that, but as we are simulating compact binary sources, we would like to see how the
     deviation of the orbit normal to the source vector (which we will refer to as the "
    _polarization_" angle) affects what is seen in the ring.
"""
#%%
```

```python
#%%
"""
In case you needed to restart the kernel for running part 2, here are all the necessary package imports.
"""
import numpy as np
import matplotlib.pyplot as pl
from mpl_toolkits import mplot3d
import matplotlib.figure as figure
import matplotlib.axes as axes
from matplotlib.animation import FuncAnimation
import mpl_toolkits.axes_grid1
import matplotlib.widgets as widgets
%matplotlib widget
#%%

#%%
"""
Now, in order to simulate this effect in 3D space, first we need a good representation of the orientation of the binary source.

The binary source is located at an arbitrary point _P_ $(r \sin(\theta)\cos(\phi), r \sin(\theta)\sin(\phi), r \cos(\theta))$. But that's not all.

Imagine that the normal to this plane, that points in the opposite direction of $\vec{OP}$ where _O_ is the origin, is now rotated by an angle $\psi$ about the normal direction. This is the polarization angle, and it is the third Euler angle that allows for us to convert our gravitational wave amplitude from the source frame to the ring frame using the Antenna functions $F_{+}(\theta, \phi, \psi)$ and $F_{\times}(\theta, \phi, \psi)$

Running the cells below will generate a 3D plot where the angles can be varied accordingly.
"""
#%%

#%%
def plot_projection(theta, phi, psi): # now with an added circle with polarization angle
    figure = pl.figure()

    axes = figure.add_axes([0,0.3,1,0.7],projection='3d')
    pl.axis('scaled')
    radial = np.linspace(0, 10, 101)
    xline = radial*np.sin(theta)*np.cos(phi)
    yline = radial*np.sin(theta)*np.sin(phi)
    zline = radial*np.cos(theta)

    alpha = np.linspace(0,2*np.pi,101)
    xloc, yloc, zloc = xline[100], yline[100], zline[100]
    xcircle = xloc - 2*np.cos(alpha)*np.cos(psi)*np.sin(phi) + 2*np.cos(alpha)*np.sin(psi)*np.sin(theta)*np.cos(phi) + 2*np.sin(alpha)*np.cos(theta)*np.cos(phi)
    ycircle = yloc + 2*np.cos(alpha)*np.cos(psi)*np.cos(phi) + 2*np.cos(alpha)*np.sin(psi)*np.sin(theta)*np.sin(phi) + 2*np.sin(alpha)*np.cos(theta)*np.sin(phi)
    zcircle = zloc + 2*np.cos(alpha)*np.sin(psi)*np.cos(theta) - 2*np.sin(alpha)*np.sin(theta)

    xvec = -5*(np.cos(psi)*np.sin(theta)*np.cos(phi) + np.sin(psi)*np.sin(phi))
    yvec = -5*(np.cos(psi)*np.sin(theta)*np.sin(phi) - np.sin(psi)*np.cos(phi))
    zvec = -5*np.cos(psi)*np.cos(theta)

    plot1, = axes.plot(xline, yline, zline,'b')
    plot2, = axes.plot(xcircle, ycircle, zcircle, 'g')
    plot3 = axes.quiver(xloc, yloc, zloc, xvec, yvec, zvec, color='purple', alpha=0.8, lw=3)
    axes.set_xlim(-10,10)
    axes.set_ylim(-10,10)
    axes.set_zlim(-10,10)

    return figure, axes, plot1, plot2, plot3
#%%

#%%
fig, ax, p1, p2, p3 = plot_projection(np.pi/2, np.pi, 0.0)
#%%
```

```
#%%
axtheta = fig.add_axes([0.2, 0.2, 0.65, 0.03])
axphi = fig.add_axes([0.2, 0.1, 0.65, 0.03])
axpsi = fig.add_axes([0.2, 0.0, 0.65, 0.03])
resetax = fig.add_axes([0.2, 0.3, 0.1, 0.03])
theta = widgets.Slider(axtheta, 'theta (in pi units)', valmin=0.0, valmax=1.0, valinit=0.5)
phi = widgets.Slider(axphi, 'phi (in pi units)', valmin=0.0, valmax=2.0, valinit=1.0)
psi = widgets.Slider(axpsi, 'psi (in pi units)', valmin=-1.0, valmax=1.0, valinit=0.0)
reset_button = widgets.Button(resetax, 'Reset', hovercolor='0.975')
#%%

#%%
def set_theta_phi(i):
    global ax, p1, p2, p3
    radial = np.linspace(0, 10, 101)
    th = theta.val*np.pi
    ph = phi.val*np.pi
    ps = psi.val*np.pi
    xline = radial*np.sin(th)*np.cos(ph)
    yline = radial*np.sin(th)*np.sin(ph)
    zline = radial*np.cos(th)

    alpha = np.linspace(0,2*np.pi,101)
    xloc, yloc, zloc = xline[100], yline[100], zline[100]

    xcircle = xloc - 2*np.cos(alpha)*np.cos(ps)*np.sin(ph) + 2*np.cos(alpha)*np.sin(ps)*np.
    sin(th)*np.cos(ph) + 2*np.sin(alpha)*np.cos(th)*np.cos(ph)
    ycircle = yloc + 2*np.cos(alpha)*np.cos(ps)*np.cos(ph) + 2*np.cos(alpha)*np.sin(ps)*np.
    sin(th)*np.sin(ph) + 2*np.sin(alpha)*np.cos(th)*np.sin(ph)
    zcircle = zloc + 2*np.cos(alpha)*np.sin(ps)*np.cos(th) - 2*np.sin(alpha)*np.sin(th)

    xvec = -5*(np.cos(ps)*np.sin(th)*np.cos(ph) + np.sin(ps)*np.sin(ph))
    yvec = -5*(np.cos(ps)*np.sin(th)*np.sin(ph) - np.sin(ps)*np.cos(ph))
    zvec = -5*np.cos(ps)*np.cos(th)

    p1.set_data(xline, yline)
    p1.set_3d_properties(zline)
    p2.set_data(xcircle, ycircle)
    p2.set_3d_properties(zcircle)
    p3.remove()
    p3 = ax.quiver(xloc, yloc, zloc, xvec, yvec, zvec, color='purple', alpha=0.8, lw=3)
    kwall = i
    ax.set_xlim(-10,10)
    ax.set_ylim(-10,10)
    ax.set_zlim(-10,10)
#%%

#%%
def reset(i):
    theta.reset()
    phi.reset()
    psi.reset()

reset_button.on_clicked(reset)

theta.on_changed(set_theta_phi)
phi.on_changed(set_theta_phi)
psi.on_changed(set_theta_phi)
#%%

#%%
"""
Now that we have successfully made our representation of the compact binary with respect to
    the ring located near the origin, we turn our attention to creating and animating the
    ring.

Run the cells below to start the 3D simulation!
"""
#%%
```

```
#%%
beta = np.arange(0,2*np.pi,np.pi/4)
x = 5*np.cos(beta)
y = 5*np.sin(beta)
z = 5*np.zeros(np.size(beta))
sc, = ax.plot(x, y, z, linestyle="", marker='o')
fig.subplots_adjust(bottom=0.3, left=0.3)
#%%

#%%
axplus = fig.add_axes([0.05, 0.3, 0.03, 0.55])
hplus = widgets.Slider(ax=axplus,label='Plus',valmin=0.0,valmax=5.0,valinit=2.5,orientation=
    'vertical')

axcross = fig.add_axes([0.15, 0.3, 0.03, 0.55])
hcross = widgets.Slider(ax=axcross,label='Cross',valmin=0.0,valmax=5.0,valinit=2.5,
    orientation='vertical')
#%%

#%%
def reset(i):
    hplus.reset()
    hcross.reset()
    theta.reset()
    phi.reset()
    psi.reset()
#%%

#%%
reset_button.on_clicked(reset)
#%%

#%%
def pos(val):
    hp = hplus.val
    hc = hcross.val

hplus.on_changed(pos)
hcross.on_changed(pos)
#%%

#%%
class Player(FuncAnimation):
    def __init__(self, fig, func, frames = None, init_func = None, fargs = None, save_count
    = None, mini = 0, maxi = 100, pos = (0.125, 0.92), cache_frame_data = False, **kwargs):
        self.i = 0
        self.min = mini
        self.max = maxi
        self.runs = True
        self.forwards = True
        self.looped = False
        self.fig = fig
        self.func = func
        self.setup(pos)
        FuncAnimation.__init__(self, self.fig, self.update, frames=self.play(), init_func=
    init_func, fargs=fargs, save_count=save_count, cache_frame_data=cache_frame_data, **
    kwargs)

    def play(self):
        while self.runs:
            self.i = self.i + self.forwards - (not self.forwards)
            if self.i > self.min and self.i < self.max:
                yield self.i
            else:
                self.stop()
                yield self.i

    def start(self):
        self.runs = True
        self.event_source.start()

```

```
    def stop(self, event=None):
        if not self.looped:
            self.runs = False
            self.event_source.stop()
        else:
            self.forwards = not self.forwards

    def forward(self, event=None):
        self.forwards = True
        self.start()

    def backward(self, event=None):
        self.forwards = False
        self.start()

    def oneforward(self, event=None):
        self.forwards = True
        self.onestep()

    def onebackward(self, event=None):
        self.forwards = False
        self.onestep()

    def loop(self, event=None):
        self.looped = not self.looped

    def onestep(self):
        if self.i > self.min and self.i < self.max:
            self.i = self.i + self.forwards - (not self.forwards)
        elif self.i == self.min and self.forwards:
            self.i += 1
        elif self.i == self.max and not self.forwards:
            self.i -= 1
        self.func(self.i)
        self.slider.set_val(self.i)
        self.fig.canvas.draw_idle()

    def setup(self, pos):
        playerax = self.fig.add_axes([pos[0], pos[1], 0.64, 0.04])
        divider = mpl_toolkits.axes_grid1.make_axes_locatable(playerax)
        bax     = divider.append_axes("right", size="80%",  pad=0.05)
        sax     = divider.append_axes("right", size="80%",  pad=0.05)
        fax     = divider.append_axes("right", size="80%",  pad=0.05)
        ofax    = divider.append_axes("right", size="100%", pad=0.05)
        rax = divider.append_axes("right", size="80%", pad=0.05)
        sliderax = divider.append_axes("right", size="500%", pad=0.07)


        self.button_oneback    = widgets.Button(playerax,  label="$\u29CF$")
        self.button_back       = widgets.Button(bax,       label="$\u25C0$")
        self.button_stop       = widgets.Button(sax,       label="$\u25A0$")
        self.button_forward    = widgets.Button(fax,       label="$\u25B6$")
        self.button_oneforward = widgets.Button(ofax,      label="$\u29D0$")
        self.button_loop       = widgets.Button(rax,       label='$\u27F3$')

        self.button_oneback.on_clicked(self.onebackward)
        self.button_back.on_clicked(self.backward)
        self.button_stop.on_clicked(self.stop)
        self.button_forward.on_clicked(self.forward)
        self.button_oneforward.on_clicked(self.oneforward)
        self.button_loop.on_clicked(self.loop)

        self.slider = widgets.Slider(sliderax, '', self.min, self.max, valinit=self.i)

        self.slider.on_changed(self.set_pos)

    def set_pos(self, i):
        self.i = int(self.slider.val)
        self.func(self.i)

    def update(self, i):
```

```python
            self.slider.set_val(i)
#%%

#%%
def wave2detframe(hp,hc,theta,phi,psi):
    Fp = 0.5*(1 + np.cos(theta)**2)*np.cos(2*phi)*np.cos(2*psi) - np.cos(theta)*np.sin(2*phi)*np.sin(2*psi)
    Fc = 0.5*(1 + np.cos(theta)**2)*np.cos(2*phi)*np.sin(2*psi) + np.cos(theta)*np.sin(2*phi)*np.cos(2*psi)

    return hp*Fp, hc*Fc
#%%

#%%
def pos_ret_XY(val):
    t = np.linspace(0,20,102)
    hp, hc = wave2detframe(hplus.val, hcross.val, theta.val, phi.val, psi.val)

    hpp = hp*np.cos(2*np.pi*t/20)
    hcc = hc*np.cos(2*np.pi*t/20)

    dx = np.array([np.real(0.5*hpp*np.cos(beta[i]) + 0.5*hcc*np.sin(beta[i])) for i in range(len(beta))])
    dy = np.array([np.real(-0.5*hpp*np.sin(beta[i]) + 0.5*hcc*np.cos(beta[i])) for i in range(len(beta))])
    X = np.zeros((len(t), len(beta)))
    Y = np.zeros((len(t), len(beta)))
    X[0], Y[0] = x, y

    for i in range(val + 1):
        X[i+1], Y[i+1] = x + dx[:,i], y + dy[:,i]
    return X, Y
#%%

#%%
def update(i):
    X, Y = pos_ret_XY(i)
    zt = 0
    xt = X[i+1]
    yt = Y[i+1]
    sc.set_data(xt, yt)
    sc.set_3d_properties(zt)

ani = Player(fig, update, maxi=100, interval = 60)
#%%

#%%
"""
You have reached the end of this supplementary Jupyter notebook.

Thank you for your time, and hope you enjoyed running and playing with this simulation as much as I did!

Comments are much appreciated.
"""
#%%
```

References

[1] Michele Maggiore. *Gravitational Waves. Vol. 1: Theory and Experiments*. Oxford University Press, 2007. ISBN: 978–0–19–857074–5. DOI: 10.1093/acprof:oso/9780198570745.001.0001.