

# 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.

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>A Brief Overview of TT-gauge Gravitational Waves</b>	<b>4</b>
<b>3</b>	<b>The Gravitational Wave Simulation</b>	<b>5</b>
3.1	A Two-Dimensional (Simple) Model . . . . .	5
3.2	A Three-Dimensional (Complex) Model . . . . .	6
3.2.1	Setting up the orientation . . . . .	6
3.2.2	Solving the frame conversion . . . . .	7
<b>4</b>	<b>Summary</b>	<b>8</b>
<b>5</b>	<b>Appendix: Code for simulating GW effect on ring</b>	<b>9</b>
	<b>References</b>	<b>17</b>

---

# 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  $P$ :

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

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

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

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

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

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

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

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,  $\xi^\sigma \partial_\sigma \Gamma_{00}^\mu = \xi^j \partial_j \Gamma_{00}^\mu$ . Further,  $R_{0j0}^i = \partial_j \Gamma_{00}^i - \partial_0 \Gamma_{0j}^i = \partial_j \Gamma_{00}^i$ . Therefore,

$$\frac{d^2 \xi^\mu}{d\tau^2} = -R_{0j0}^i \xi^j \left( \frac{dx^0}{d\tau} \right)^2 \quad (5)$$

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,

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

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

$$\ddot{\xi} = -c^2 R_{0j0}^i \xi^j \quad (7)$$

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

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

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

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

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:

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

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

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

Now considering that  $(x_0, y_0) = (\cos \alpha_i, \sin \alpha_i)$  for  $\alpha_i = 2\pi/n_i$  where  $i = 1, 2, \dots, N$  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.

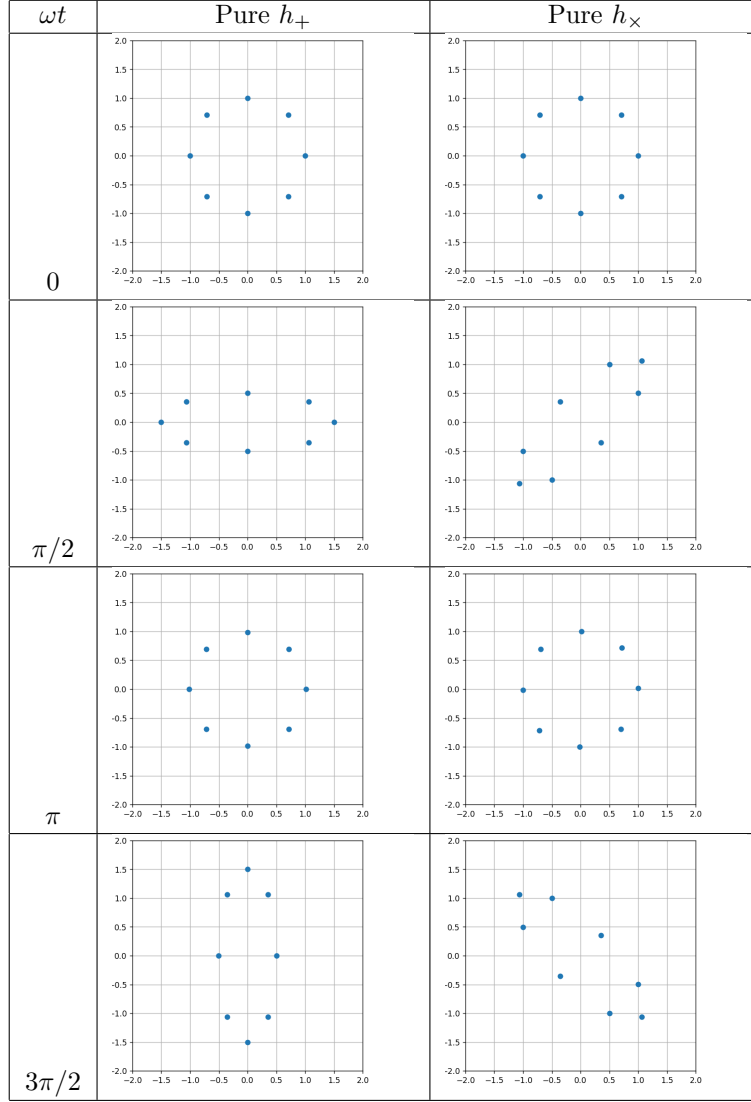


Table 1: Results from the simulation illustrating how the ring deforms due to the two different polarizations.

### 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

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

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  $\phi$ , and  $\hat{v}$  is just  $\hat{i}$  rotated along the Z-axis by an angle  $\phi$  and rotated along the Y-axis by an angle  $\theta$ . Therefore,

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

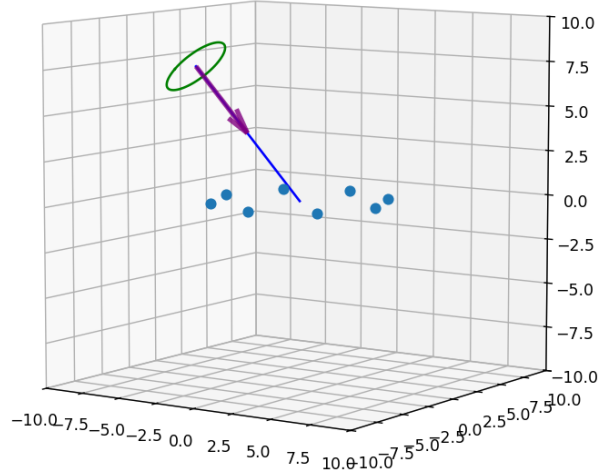


Figure 1: Figure of the 3D representation of our model. The ring of particles are positioned on the X-Y plane with the green circle representing the orbit of the compact binary. The purple arrow is the direction of  $\hat{w}$ . The circle coordinates are found using equations 12 and 13.

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  $\psi$ , 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,

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

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

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

### 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  $\phi$  about the Z-axis, followed by a rotation by  $\theta$  about the Y-axis, then another rotation about the Z-axis for the polarization angle  $\psi$ . The final result is therefore,

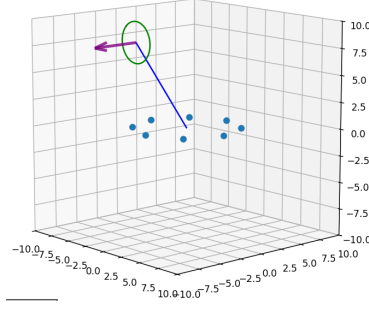


Figure 2: Same as figure 1, but the polarization angle has been changed to  $\pi$ . The circle coordinates are calculated using equation 15.

$$h = F_+ h_+ + F_\times h_\times \quad (16)$$

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

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

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

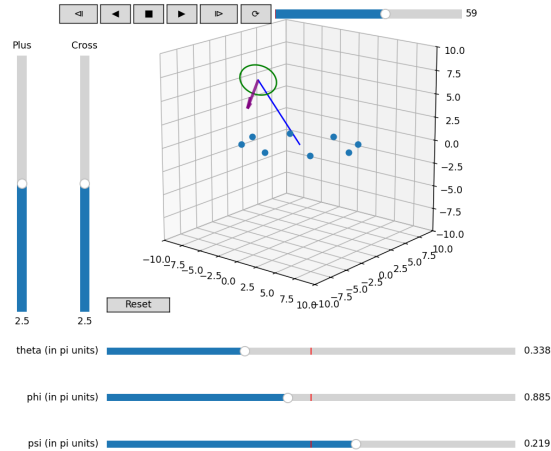


Figure 3: A snapshot of the simulation interface running on VS Code.

## 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

```
1  """
2  # Supplementary Jupyter notebook for PH 821 project
3  ### Project title: The Effect of Gravitational Waves on a Ring of Particles
4  """
5  Presented by: Aditya Pawan (23D1069)
6  -----
7  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.
8
9  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.
10
11  All effects are amplified for better visualization.
12
13  Disclaimer: If any animations or widgets break/start lagging, please restart the kernel and
   run the relevant cells again.
14  """
15  """
16  """
17  """
18  # Dependencies
19  """
20  Please check if the following modules are installed in your environment:
21  numpy v1.26.4
22  matplotlib v3.9.1
23  mpl_toolkits
24  """
25
26  import numpy as np
27  import matplotlib.pyplot as plt
28  """
29  """
30  """
31  """
32  These specific submodules, functions and classes will be used in this project to produce
   interactive, animated plots.
33  """
34  from mpl_toolkits import mplot3d
35  import matplotlib.figure as figure
36  import matplotlib.axes as axes
37  from matplotlib.animation import FuncAnimation
38  import mpl_toolkits.axes_grid1
39  import matplotlib.widgets as widgets
40  """
41  """
42  """
43  """
44  Matplotlib widget is required for the more interactive plots that contain matplotlib.widget.
   Slider and
45  matplotlib.widget.Button, as well as FuncAnimation codes. If you wish to switch to a
   different backend, you may
46  wish to enter your backend of choice. If you wish to use the default (inline) backend for
   non-interactive and
47  non-animated plots, do not run this cell.
48  """
49  %matplotlib widget
50  """
51  """
52  """
53  ## Part 1: Simulating a Two-Dimensional Ring of particles
54  """
55  ---
56  Let us start with a basic plot containing a ring of  $n = 8$  particles.
57
58  The cell below will be where our simulation will happen, so move back to it after running
```

```

    the cells below it.
59 """
60 #%%
61
62 #%%
63 fig, ax = plt.subplots()
64 plt.axis('scaled')
65
66 n = 8
67 theta = np.arange(0, 2*np.pi, 2*np.pi/n)
68 x = np.cos(theta)
69 y = np.sin(theta)
70 sc, = ax.plot(x, y, marker = 'o', ls = "")
71 ax.grid(True)
72 ax.set_xlim(-4.0, 4.0)
73 ax.set_ylim(-4.0, 4.0)
74 plt.show()
75 #%%
76
77 #%%
78 """
79 Let us now add two axes which contain sliders that enable us to vary both plus and cross
    polarizations.
80 """
81 #%%
82
83 #%%
84 fig.subplots_adjust(bottom=0.3)
85
86 axplus = fig.add_axes([0.35, 0.2, 0.55, 0.03])
87 hplus = widgets.Slider(ax=axplus, label='Plus Polarization Amplitude', valmin=0.0, valmax=1.0,
    valinit=0.5)
88
89 axcross = fig.add_axes([0.35, 0.1, 0.55, 0.03])
90 hcross = widgets.Slider(ax=axcross, label='Cross Polarization Amplitude', valmin=0.0, valmax
    =1.0, valinit=0.5)
91
92 def pos(val):
93     hp = hplus.val
94     hc = hcross.val
95
96 hplus.on_changed(pos)
97 hcross.on_changed(pos)
98 #%%
99
100 #%%
101 """
102 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.
103
104 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.
105 """
106 #%%
107
108 #%%
109 class Player(FuncAnimation):
110     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):
111         self.i = 0
112         self.min = mini
113         self.max = maxi
114         self.runs = True
115         self.forwards = True
116         self.looped = False
117         self.fig = fig
118         self.func = func
119         self.setup(pos)

```

```

120     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)

121
122     def play(self):
123         while self.runs:
124             self.i = self.i + self.forwards - (not self.forwards)
125             if self.i > self.min and self.i < self.max:
126                 yield self.i
127             else:
128                 self.stop()
129                 yield self.i
130
131     def start(self):
132         self.runs = True
133         self.event_source.start()
134
135     def stop(self, event=None):
136         if not self.looped:
137             self.runs = False
138             self.event_source.stop()
139         else:
140             self.forwards = not self.forwards
141
142     def forward(self, event=None):
143         self.forwards = True
144         self.start()
145
146     def backward(self, event=None):
147         self.forwards = False
148         self.start()
149
150     def oneforward(self, event=None):
151         self.forwards = True
152         self.onestep()
153
154     def onebackward(self, event=None):
155         self.forwards = False
156         self.onestep()
157
158     def loop(self, event=None):
159         self.looped = not self.looped
160
161     def onestep(self):
162         if self.i > self.min and self.i < self.max:
163             self.i = self.i + self.forwards - (not self.forwards)
164         elif self.i == self.min and self.forwards:
165             self.i += 1
166         elif self.i == self.max and not self.forwards:
167             self.i -= 1
168         self.func(self.i)
169         self.slider.set_val(self.i)
170         self.fig.canvas.draw_idle()
171
172     def setup(self, pos):
173         playerax = self.fig.add_axes([pos[0], pos[1], 0.64, 0.04])
174         divider = mpl_toolkits.axes_grid1.make_axes_locatable(playerax)
175         bax = divider.append_axes("right", size="80%", pad=0.05)
176         sax = divider.append_axes("right", size="80%", pad=0.05)
177         fax = divider.append_axes("right", size="80%", pad=0.05)
178         ofax = divider.append_axes("right", size="100%", pad=0.05)
179         rax = divider.append_axes("right", size="80%", pad=0.05)
180         sliderax = divider.append_axes("right", size="500%", pad=0.07)
181
182
183         self.button_oneback = widgets.Button(playerax, label="$\u29CF$")
184         self.button_back = widgets.Button(bax, label="$\u25C0$")
185         self.button_stop = widgets.Button(sax, label="$\u25A0$")
186         self.button_forward = widgets.Button(fax, label="$\u25B6$")
187         self.button_oneforward = widgets.Button(ofax, label="$\u29D0$")
188         self.button_loop = widgets.Button(rax, label='$\u27F3$')

```

```

189         self.button_oneback.on_clicked(self.onebackward)
190         self.button_back.on_clicked(self.backward)
191         self.button_stop.on_clicked(self.stop)
192         self.button_forward.on_clicked(self.forward)
193         self.button_oneforward.on_clicked(self.oneforward)
194         self.button_loop.on_clicked(self.loop)
195
196
197         self.slider = widgets.Slider(sliderax, '', self.min, self.max, valinit=self.i)
198
199         self.slider.on_changed(self.set_pos)
200
201     def set_pos(self, i):
202         self.i = int(self.slider.val)
203         self.func(self.i)
204
205     def update(self, i):
206         self.slider.set_val(i)
207
208     """
209     """
210
211     Now with the aforementioned _Player_ class in place, we construct an update function for it
212     to update our figure continuously (or as the time slider changes).
213
214     Run the cell below and go back to our plot to simulate Gravitational Waves!
215     """
216
217     """
218     """
219
220     def pos_ret_XY(val):
221         t = np.linspace(0, 20, 102)
222         hp = hplus.val*np.sin(2*np.pi*t/20)
223         hc = hcross.val*np.sin(2*np.pi*t/20)
224         dx = np.array([np.real(0.5*hp*np.cos(theta[i]) + 0.5*hc*np.sin(theta[i])) for i in range
225         (len(theta))])
226         dy = np.array([np.real(-0.5*hp*np.sin(theta[i]) + 0.5*hc*np.cos(theta[i])) for i in
227         range(len(theta))])
228         X = np.zeros((len(t), len(theta)))
229         Y = np.zeros((len(t), len(theta)))
230         X[0], Y[0] = x, y
231
232         for i in range(val + 1):
233             X[i+1], Y[i+1] = x + dx[:, i], y + dy[:, i]
234         return X, Y
235
236     def update(i):
237         X, Y = pos_ret_XY(i)
238         xt = X[i+1]
239         yt = Y[i+1]
240         sc.set_data(xt, yt)
241
242     ani = Player(fig, update, maxi=100, interval = 60)
243
244     """
245     """
246
247     """
248     """
249
250     """
251
252     """
253
254     """
255
256     """
257
258     """
259
260     """
261
262     """
263
264     """
265
266     """
267
268     """
269
270     """
271
272     """
273
274     """
275
276     """
277
278     """
279
280     """
281
282     """
283
284     """
285
286     """
287
288     """
289
290     """
291
292     """
293
294     """
295
296     """
297
298     """
299
300     """
301
302     """
303
304     """
305
306     """
307
308     """
309
310     """
311
312     """
313
314     """
315
316     """
317
318     """
319
320     """
321
322     """
323
324     """
325
326     """
327
328     """
329
330     """
331
332     """
333
334     """
335
336     """
337
338     """
339
340     """
341
342     """
343
344     """
345
346     """
347
348     """
349
350     """
351
352     """
353
354     """
355
356     """
357
358     """
359
360     """
361
362     """
363
364     """
365
366     """
367
368     """
369
370     """
371
372     """
373
374     """
375
376     """
377
378     """
379
380     """
381
382     """
383
384     """
385
386     """
387
388     """
389
390     """
391
392     """
393
394     """
395
396     """
397
398     """
399
400     """
401
402     """
403
404     """
405
406     """
407
408     """
409
410     """
411
412     """
413
414     """
415
416     """
417
418     """
419
420     """
421
422     """
423
424     """
425
426     """
427
428     """
429
430     """
431
432     """
433
434     """
435
436     """
437
438     """
439
440     """
441
442     """
443
444     """
445
446     """
447
448     """
449
450     """
451
452     """
453
454     """
455
456     """
457
458     """
459
460     """
461
462     """
463
464     """
465
466     """
467
468     """
469
470     """
471
472     """
473
474     """
475
476     """
477
478     """
479
480     """
481
482     """
483
484     """
485
486     """
487
488     """
489
490     """
491
492     """
493
494     """
495
496     """
497
498     """
499
500     """
501
502     """
503
504     """
505
506     """
507
508     """
509
510     """
511
512     """
513
514     """
515
516     """
517
518     """
519
520     """
521
522     """
523
524     """
525
526     """
527
528     """
529
530     """
531
532     """
533
534     """
535
536     """
537
538     """
539
540     """
541
542     """
543
544     """
545
546     """
547
548     """
549
550     """
551
552     """
553
554     """
555
556     """
557
558     """
559
560     """
561
562     """
563
564     """
565
566     """
567
568     """
569
570     """
571
572     """
573
574     """
575
576     """
577
578     """
579
580     """
581
582     """
583
584     """
585
586     """
587
588     """
589
590     """
591
592     """
593
594     """
595
596     """
597
598     """
599
600     """
601
602     """
603
604     """
605
606     """
607
608     """
609
610     """
611
612     """
613
614     """
615
616     """
617
618     """
619
620     """
621
622     """
623
624     """
625
626     """
627
628     """
629
630     """
631
632     """
633
634     """
635
636     """
637
638     """
639
640     """
641
642     """
643
644     """
645
646     """
647
648     """
649
650     """
651
652     """
653
654     """
655
656     """
657
658     """
659
660     """
661
662     """
663
664     """
665
666     """
667
668     """
669
670     """
671
672     """
673
674     """
675
676     """
677
678     """
679
680     """
681
682     """
683
684     """
685
686     """
687
688     """
689
690     """
691
692     """
693
694     """
695
696     """
697
698     """
699
700     """
701
702     """
703
704     """
705
706     """
707
708     """
709
710     """
711
712     """
713
714     """
715
716     """
717
718     """
719
720     """
721
722     """
723
724     """
725
726     """
727
728     """
729
730     """
731
732     """
733
734     """
735
736     """
737
738     """
739
740     """
741
742     """
743
744     """
745
746     """
747
748     """
749
750     """
751
752     """
753
754     """
755
756     """
757
758     """
759
760     """
761
762     """
763
764     """
765
766     """
767
768     """
769
770     """
771
772     """
773
774     """
775
776     """
777
778     """
779
780     """
781
782     """
783
784     """
785
786     """
787
788     """
789
790     """
791
792     """
793
794     """
795
796     """
797
798     """
799
800     """
801
802     """
803
804     """
805
806     """
807
808     """
809
810     """
811
812     """
813
814     """
815
816     """
817
818     """
819
820     """
821
822     """
823
824     """
825
826     """
827
828     """
829
830     """
831
832     """
833
834     """
835
836     """
837
838     """
839
840     """
841
842     """
843
844     """
845
846     """
847
848     """
849
850     """
851
852     """
853
854     """
855
856     """
857
858     """
859
860     """
861
862     """
863
864     """
865
866     """
867
868     """
869
870     """
871
872     """
873
874     """
875
876     """
877
878     """
879
880     """
881
882     """
883
884     """
885
886     """
887
888     """
889
890     """
891
892     """
893
894     """
895
896     """
897
898     """
899
900     """
901
902     """
903
904     """
905
906     """
907
908     """
909
910     """
911
912     """
913
914     """
915
916     """
917
918     """
919
920     """
921
922     """
923
924     """
925
926     """
927
928     """
929
930     """
931
932     """
933
934     """
935
936     """
937
938     """
939
940     """
941
942     """
943
944     """
945
946     """
947
948     """
949
950     """
951
952     """
953
954     """
955
956     """
957
958     """
959
960     """
961
962     """
963
964     """
965
966     """
967
968     """
969
970     """
971
972     """
973
974     """
975
976     """
977
978     """
979
980     """
981
982     """
983
984     """
985
986     """
987
988     """
989
990     """
991
992     """
993
994     """
995
996     """
997
998     """
999
1000    """

```

```

252  """
253  """
254  In case you needed to restart the kernel for running part 2, here are all the necessary
      package imports.
255  """
256  import numpy as np
257  import matplotlib.pyplot as plt
258  from mpl_toolkits import mplot3d
259  import matplotlib.figure as figure
260  import matplotlib.axes as axes
261  from matplotlib.animation import FuncAnimation
262  import mpl_toolkits.axes_grid1
263  import matplotlib.widgets as widgets
264  %matplotlib widget
265  """
266  """
267  """
268  """
269  Now, in order to simulate this effect in 3D space, first we need a good representation of
      the orientation of the binary source.
270
271  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.
272
273  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  $F_\times(\theta, \phi, \psi)$ 
274
275  Running the cells below will generate a 3D plot where the angles can be varied accordingly.
276  """
277  """
278  """
279  """
280  def plot_projection(theta, phi, psi): # now with an added circle with polarization angle
281      figure = plt.figure()
282
283      axes = figure.add_axes([0,0.3,1,0.7], projection='3d')
284      plt.axis('scaled')
285      radial = np.linspace(0, 10, 101)
286      xline = radial*np.sin(theta)*np.cos(phi)
287      yline = radial*np.sin(theta)*np.sin(phi)
288      zline = radial*np.cos(theta)
289
290      alpha = np.linspace(0,2*np.pi,101)
291      xloc, yloc, zloc = xline[100], yline[100], zline[100]
292      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)
293      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)
294      zcircle = zloc + 2*np.cos(alpha)*np.sin(psi)*np.cos(theta) - 2*np.sin(alpha)*np.sin(
      theta)
295
296      xvec = -5*(np.cos(psi)*np.sin(theta)*np.cos(phi) + np.sin(psi)*np.sin(phi))
297      yvec = -5*(np.cos(psi)*np.sin(theta)*np.sin(phi) - np.sin(psi)*np.cos(phi))
298      zvec = -5*np.cos(psi)*np.cos(theta)
299
300      plot1, = axes.plot(xline, yline, zline, 'b')
301      plot2, = axes.plot(xcircle, ycircle, zcircle, 'g')
302      plot3 = axes.quiver(xloc, yloc, zloc, xvec, yvec, zvec, color='purple', alpha=0.8, lw=3)
303      axes.set_xlim(-10,10)
304      axes.set_ylim(-10,10)
305      axes.set_zlim(-10,10)
306
307      return figure, axes, plot1, plot2, plot3
308  """
309  """
310  """
311  fig, ax, p1, p2, p3 = plot_projection(np.pi/2, np.pi, 0.0)
312  """

```

```

313
314 ###
315 axtheta = fig.add_axes([0.2, 0.2, 0.65, 0.03])
316 axphi = fig.add_axes([0.2, 0.1, 0.65, 0.03])
317 axpsi = fig.add_axes([0.2, 0.0, 0.65, 0.03])
318 resetax = fig.add_axes([0.2, 0.3, 0.1, 0.03])
319 theta = widgets.Slider(axtheta, 'theta (in pi units)', valmin=0.0, valmax=1.0, valinit=0.5)
320 phi = widgets.Slider(axphi, 'phi (in pi units)', valmin=0.0, valmax=2.0, valinit=1.0)
321 psi = widgets.Slider(axpsi, 'psi (in pi units)', valmin=-1.0, valmax=1.0, valinit=0.0)
322 reset_button = widgets.Button(resetax, 'Reset', hovercolor='0.975')
323 ###
324
325 ###
326 def set_theta_phi(i):
327     global ax, p1, p2, p3
328     radial = np.linspace(0, 10, 101)
329     th = theta.val*np.pi
330     ph = phi.val*np.pi
331     ps = psi.val*np.pi
332     xline = radial*np.sin(th)*np.cos(ph)
333     yline = radial*np.sin(th)*np.sin(ph)
334     zline = radial*np.cos(th)
335
336     alpha = np.linspace(0,2*np.pi,101)
337     xloc, yloc, zloc = xline[100], yline[100], zline[100]
338
339     xcircle = xloc - 2*np.cos(alpha)*np.cos(ps)*np.sin(ph) + 2*np.cos(alpha)*np.sin(ps)*np.
340     sin(th)*np.cos(ph) + 2*np.sin(alpha)*np.cos(th)*np.cos(ph)
341     ycircle = yloc + 2*np.cos(alpha)*np.cos(ps)*np.cos(ph) + 2*np.cos(alpha)*np.sin(ps)*np.
342     sin(th)*np.sin(ph) + 2*np.sin(alpha)*np.cos(th)*np.sin(ph)
343     zcircle = zloc + 2*np.cos(alpha)*np.sin(ps)*np.cos(th) - 2*np.sin(alpha)*np.sin(th)
344
345     xvec = -5*(np.cos(ps)*np.sin(th)*np.cos(ph) + np.sin(ps)*np.sin(ph))
346     yvec = -5*(np.cos(ps)*np.sin(th)*np.sin(ph) - np.sin(ps)*np.cos(ph))
347     zvec = -5*np.cos(ps)*np.cos(th)
348
349     p1.set_data(xline, yline)
350     p1.set_3d_properties(zline)
351     p2.set_data(xcircle, ycircle)
352     p2.set_3d_properties(zcircle)
353     p3.remove()
354     p3 = ax.quiver(xloc, yloc, zloc, xvec, yvec, zvec, color='purple', alpha=0.8, lw=3)
355     kwall = i
356     ax.set_xlim(-10,10)
357     ax.set_ylim(-10,10)
358     ax.set_zlim(-10,10)
359 ###
360 ###
361 def reset(i):
362     theta.reset()
363     phi.reset()
364     psi.reset()
365
366 reset_button.on_clicked(reset)
367
368 theta.on_changed(set_theta_phi)
369 phi.on_changed(set_theta_phi)
370 psi.on_changed(set_theta_phi)
371 ###
372 ###
373 """
374 Now that we have successfully made our representation of the compact binary with respect to
375 the ring located near the origin, we turn our attention to creating and animating the
376 ring.
377
378 Run the cells below to start the 3D simulation!
379 """
380 ###
381

```

```

380  """
381  beta = np.arange(0, 2*np.pi, np.pi/4)
382  x = 5*np.cos(beta)
383  y = 5*np.sin(beta)
384  z = 5*np.zeros(np.size(beta))
385  sc, = ax.plot(x, y, z, linestyle="", marker='o')
386  fig.subplots_adjust(bottom=0.3, left=0.3)
387  """
388
389  """
390  axplus = fig.add_axes([0.05, 0.3, 0.03, 0.55])
391  hplus = widgets.Slider(ax=axplus, label='Plus', valmin=0.0, valmax=5.0, valinit=2.5, orientation=
    'vertical')
392
393  axcross = fig.add_axes([0.15, 0.3, 0.03, 0.55])
394  hcross = widgets.Slider(ax=axcross, label='Cross', valmin=0.0, valmax=5.0, valinit=2.5,
    orientation='vertical')
395  """
396
397  """
398  def reset(i):
399      hplus.reset()
400      hcross.reset()
401      theta.reset()
402      phi.reset()
403      psi.reset()
404  """
405
406  """
407  reset_button.on_clicked(reset)
408  """
409
410  """
411  def pos(val):
412      hp = hplus.val
413      hc = hcross.val
414
415  hplus.on_changed(pos)
416  hcross.on_changed(pos)
417  """
418
419  """
420  class Player(FuncAnimation):
421      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):
422          self.i = 0
423          self.min = mini
424          self.max = maxi
425          self.runs = True
426          self.forwards = True
427          self.looped = False
428          self.fig = fig
429          self.func = func
430          self.setup(pos)
431          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)
432
433      def play(self):
434          while self.runs:
435              self.i = self.i + self.forwards - (not self.forwards)
436              if self.i > self.min and self.i < self.max:
437                  yield self.i
438              else:
439                  self.stop()
440                  yield self.i
441
442      def start(self):
443          self.runs = True
444          self.event_source.start()
445

```

```

446 def stop(self, event=None):
447     if not self.looped:
448         self.runs = False
449         self.event_source.stop()
450     else:
451         self.forwards = not self.forwards
452
453 def forward(self, event=None):
454     self.forwards = True
455     self.start()
456
457 def backward(self, event=None):
458     self.forwards = False
459     self.start()
460
461 def oneforward(self, event=None):
462     self.forwards = True
463     self.onestep()
464
465 def onebackward(self, event=None):
466     self.forwards = False
467     self.onestep()
468
469 def loop(self, event=None):
470     self.looped = not self.looped
471
472 def onestep(self):
473     if self.i > self.min and self.i < self.max:
474         self.i = self.i + self.forwards - (not self.forwards)
475     elif self.i == self.min and self.forwards:
476         self.i += 1
477     elif self.i == self.max and not self.forwards:
478         self.i -= 1
479     self.func(self.i)
480     self.slider.set_val(self.i)
481     self.fig.canvas.draw_idle()
482
483 def setup(self, pos):
484     playerax = self.fig.add_axes([pos[0], pos[1], 0.64, 0.04])
485     divider = mpl_toolkits.axes_grid1.make_axes_locatable(playerax)
486     bax = divider.append_axes("right", size="80%", pad=0.05)
487     sax = divider.append_axes("right", size="80%", pad=0.05)
488     fax = divider.append_axes("right", size="80%", pad=0.05)
489     ofax = divider.append_axes("right", size="100%", pad=0.05)
490     rax = divider.append_axes("right", size="80%", pad=0.05)
491     sliderax = divider.append_axes("right", size="500%", pad=0.07)
492
493     self.button_oneback = widgets.Button(playerax, label="$\u29CF$")
494     self.button_back = widgets.Button(bax, label="$\u25C0$")
495     self.button_stop = widgets.Button(sax, label="$\u25A0$")
496     self.button_forward = widgets.Button(fax, label="$\u25B6$")
497     self.button_oneforward = widgets.Button(ofax, label="$\u29D0$")
498     self.button_loop = widgets.Button(rax, label='$\u27F3$')
499
500     self.button_oneback.on_clicked(self.onebackward)
501     self.button_back.on_clicked(self.backward)
502     self.button_stop.on_clicked(self.stop)
503     self.button_forward.on_clicked(self.forward)
504     self.button_oneforward.on_clicked(self.oneforward)
505     self.button_loop.on_clicked(self.loop)
506
507     self.slider = widgets.Slider(sliderax, '', self.min, self.max, valinit=self.i)
508
509     self.slider.on_changed(self.set_pos)
510
511 def set_pos(self, i):
512     self.i = int(self.slider.val)
513     self.func(self.i)
514
515 def update(self, i):
516

```



```

517         self.slider.set_val(i)
518     """
519
520     """
521     def wave2detframe(hp, hc, theta, phi, psi):
522         Fp = 0.5*(1 + np.cos(theta)**2)*np.cos(2*phi)*np.cos(2*psi) - np.cos(theta)*np.sin(2*phi)
523             *np.sin(2*psi)
524         Fc = 0.5*(1 + np.cos(theta)**2)*np.cos(2*phi)*np.sin(2*psi) + np.cos(theta)*np.sin(2*phi)
525             *np.cos(2*psi)
526
527         return hp*Fp, hc*Fc
528     """
529     """
530     def pos_ret_XY(val):
531         t = np.linspace(0, 20, 102)
532         hp, hc = wave2detframe(hplus.val, hcross.val, theta.val, phi.val, psi.val)
533
534         hpp = hp*np.cos(2*np.pi*t/20)
535         hcc = hc*np.cos(2*np.pi*t/20)
536
537         dx = np.array([np.real(0.5*hpp*np.cos(beta[i]) + 0.5*hcc*np.sin(beta[i])) for i in range
538             (len(beta))])
539         dy = np.array([np.real(-0.5*hpp*np.sin(beta[i]) + 0.5*hcc*np.cos(beta[i])) for i in
540             range(len(beta))])
541         X = np.zeros((len(t), len(beta)))
542         Y = np.zeros((len(t), len(beta)))
543         X[0], Y[0] = x, y
544
545         for i in range(val + 1):
546             X[i+1], Y[i+1] = x + dx[:, i], y + dy[:, i]
547         return X, Y
548     """
549     """
550     def update(i):
551         X, Y = pos_ret_XY(i)
552         zt = 0
553         xt = X[i+1]
554         yt = Y[i+1]
555         sc.set_data(xt, yt)
556         sc.set_3d_properties(zt)
557
558     ani = Player(fig, update, maxi=100, interval = 60)
559     """
560     """
561     """
562     """
563     You have reached the end of this supplementary Jupyter notebook.
564
565     Thank you for your time, and hope you enjoyed running and playing with this simulation as
566         much as I did!
567
568     Comments are much appreciated.
569     """
570     """
571     """

```

## 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.