Statistical Comparison Between Observational Data and Simulated Gravitational Waves

Matthew Gordon

Statistical Comparison Between Observational Data and Simulated Gravitational Waves

by

Matthew Gordon

Advisor: Dr. Deva O’Neil

Physics Department

Bridgewater College

April 12, 2022

Abstract:

The goal of this project is to determine the effective accuracy of a simulation that graphs the gravitational waves emitted from a binary system. The Python program that creates these graphs uses different analytical models to approximate each phase of the model, inspiral and merger. Data from the Gravitational Wave Open Science Center (GWOSC) website regarding an observed event is used to compare to the output of the simulation. Another Python program matches up the two waveforms and calculates a correlation coefficient between the two, which is close to 1, as desired. Additional graphs are simulated using masses that differ from the observed event and when compared with it, produce much lower correlation coefficients, supporting the accuracy of the simulation.

Introduction:

In 2015, the first gravitational waves were detected by the Laser Interferometer Gravitational-Wave Observatory (LIGO). They were previously predicted by Albert Einstein, in his general theory of relativity, almost a hundred years prior, but proof of their existence had never been observed. Gravitational waves are created by violently powerful sources called binary systems, which are composed of two black holes or neutron stars revolving around each other. As these large objects accelerate, they emit waves that travel through spacetime, distorting it as they go. Their importance is in the new data this gives us about the universe and the various different celestial bodies that inhabit it.

These waves travel at the speed of light and as they oscillate, they push and pull at spacetime, producing a measurable outcome that is used to assign properties to a waveform. This outcome is called strain, which can be found by dividing the change in length of an object by the original length of that object. One can picture an object being stretched out into a long shape and a wave passing through it. As the wave travels over the object, it stretches it out and when measured, lets the observer calculate the strain, which is graphed as the amplitude of the waveform and is determined by how large the masses of the objects are. The LIGO facilities used to measure this strain utilize two mirrors that are spaced four kilometers apart and reflect a laser that is aimed at them [1]. As gravitational waves come into contact with the laser inside the vacuum chamber, the length of the laser changes, allowing the strain to be measured, and gives data on the objects that caused those waves.

Strain is calculated differently depending on which phase the wave is in. For the inspiral phase, where the two objects are revolving around each other, there are two separate calculations required to find the overall strain. These two involve different forms of strain, plus polarization and cross polarization, that distinguish which directions the wave is being stretched and compressed in. If the wave is traveling into the page, then plus polarization occurs where the wave is being stretched in the x and y directions (resembling a plus sign). Cross polarization is similar to plus polarization, but just rotated 45 degrees (resembling a multiplication cross). The equations for finding the strain for plus polarization and cross

polarization strain are as follows:

$$h_+ = -\frac{M\eta}{r}\left[(\cos^2\theta + 1)\left[\left(-\dot{r}^2 + r^2\dot{\Phi}^2 + \frac{M}{r}\right)\cos 2\Phi + 2r\dot{r}\dot{\Phi}\sin 2\Phi\right] + \left(-\dot{r}^2 + r^2\dot{\Phi}^2 + \frac{M}{r}\right)\sin^2\theta\right], \quad (1)$$

$$h_\times = -2\frac{M\eta}{r}\cos\theta\left[\left(-\dot{r}^2 + r^2\dot{\Phi}^2 + \frac{M}{r}\right)\sin 2\Phi - 2r\dot{r}\dot{\Phi}\cos 2\Phi\right], \quad (2)$$

where $\boldsymbol{\Phi}$ is the azimuthal, or rotational, angle in spherical coordinates. M_η and M are the masses of the two objects, and r is the distance between the two objects. Any variable with a dot above it is the derivative of the respective variable. Often, to make things easier, it is assumed that the inclination angle, $\boldsymbol{\theta}$, which is the angle of observation, is set to 0, which simplifies the plus (3)

polarization equation to be the following:

$$h_+ = -\frac{M\eta}{r}\left[\left(-\dot{r}^2 + r^2\dot{\Phi}^2 + \frac{M}{r}\right)\cos 2\Phi + 2r\dot{r}\dot{\Phi}\sin 2\Phi\right].$$

To calculate the cumulative strain on a wave at any point during the inspiral phase, this equation is used:

$$h_{ins}(t) = h_+(t) - ih_\times(t). \quad (4)$$

For the merger phase, which is everything after the two objects collide, strain is calculated using a different equation. A similar approach, with the real and imaginary parts of the waveform separated, applies to this phase when calculating the strain, however, it is outside the scope of this project.

One of the formulas used in graphing waveforms is Euler’s Formula:

$$Ae^{i\theta} = A\cos\theta + Ai\sin\theta \, . \tag{5}$$

In it, there are two separate terms that define different parts of a wave’s behavior. The term with $\cos\theta$ explains the real portion of the wave, while the $i\sin\theta$ term describes the imaginary behavior. Both terms are orthogonal projections of the unit circle, however the cosine term is 90 degrees out of phase [2]. A good example of this would be electro-magnetic waves where the electric and magnetic fields are oscillating in the same direction, but orthogonal to each other. If you look back at the formula for cumulative strain (inspiral), you can see that it resembles Euler’s Formula. The two equations are off by a negative sign, but there are two different terms, one real and one imaginary.

Strain is a significant part of this project since simulated and observed gravitational waveforms are being compared. The strain is what is measured on the y-axis, with respect to time on the x-axis. By simulating a binary system that has the same object masses as an observed event, one can test the predictive power of the simulation. This project aims to do so and the following sections define how this process was carried out.

Procedure:

My project involves the use of a Python program [3] adapted by Daniel Hancock from another program designed by Dr. Maria Babiuc and Dillon Buskirk [4]. The original Mathematica code was transferred over to a Jupyter notebook. The program takes mass values for the two objects and then performs calculations to find the strain for both the inspiral and the merger phases. This is done for both the real and imaginary portions of the wave. After lists are made that carry the strain values, the two portions are graphed together.

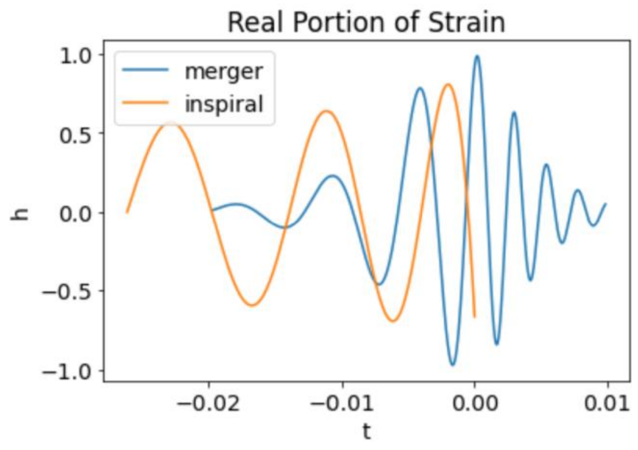


Figure 1: The inspiral and merger phases graphed together before matching them up.

At first, the inspiral and merger phases do not match, so a factor is determined that gets added to the merger strain, effectively moving it to the right, in order to match up with the inspiral strain. After this change is made, the two plots look like this:

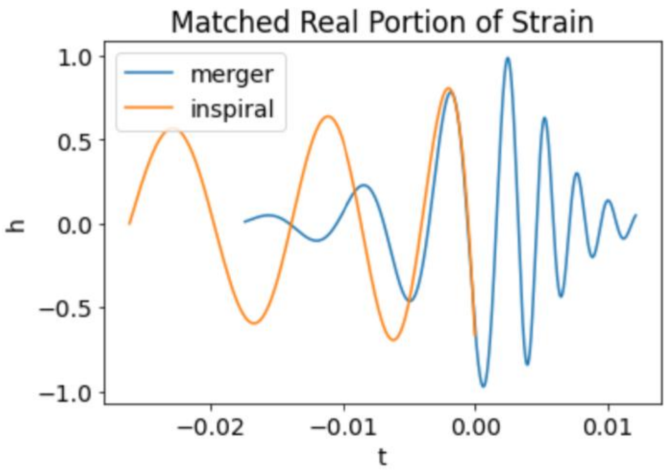


Figure 2: The inspiral and merger phases graphed together after matching them up.

Once this is done, the graph is cleaned up by cutting out the merger plot where it meets up with the inspiral phase.

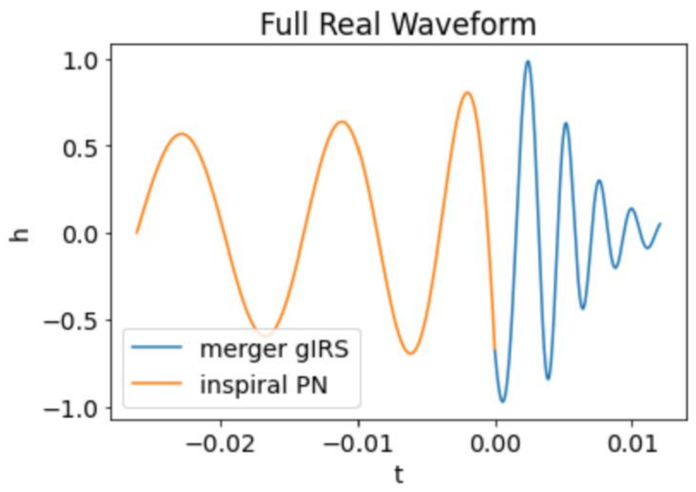
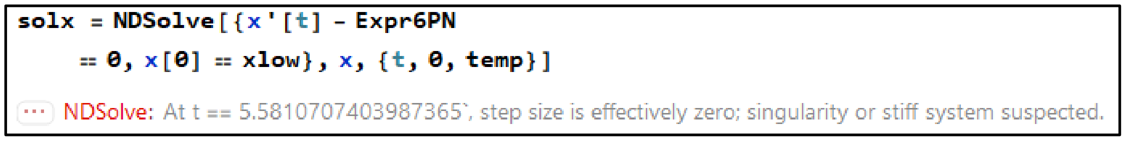


Figure 3: The full real waveform after cutting out the merger phase from before it meets the inspiral phase.

The same process is completed for the imaginary portion of the waveform as well.

This program works well for the originally designed scenario where both objects have a mass of 20 solar mass units. However, the first task was to find out if it worked when those mass values were altered. Upon changing these values, there were certain regions of code that gave a “divide by zero” error, due to a variable called stiff time that helps predict the behavior of the inspiral phase. This portion of the waveform is approximated using a differential equation. This equation is really only effective before gravitational forces get too strong since it requires a small time step. As the merger phase begins, those forces get too strong for the differential equation to handle, and the program fails to execute. The stiff time’s purpose is to let the simulation know when to stop calculating the differential equation, signaling the end of the inspiral phase. To find the appropriate stiff time, the mass values were inputted into a Mathematica file developed by Dr. Babiuc and Dillon Buskirk, and all of the cells were evaluated. One of those cells provided an error message stating the time value where the program could no longer run. This time was the proper stiff time that would allow the differential equation to be solved effectively.



```
solx = NDSolve[{x'[t] - Expr6PN
    == 0, x[0] == xlow}, x, {t, 0, temp}]
··· NDSolve: At t == 5.5810707403987365`, step size is effectively zero; singularity or stiff system suspected.
```

Figure 4: A sample of the Mathematica file used to find the stiff time for the simulations. The time value in the “NDSolve:” section would change depending on the mass values, and would subsequently be used in the Python program.

With the code able to successfully run, the next obstacle encountered was the inability for the simulation to match the inspiral and merger phases on the graph. This is where a specific variable was found in the code that moved the merger points to the proper time value. This value was inserted in as a guess and check variable where the user manually modifies it to see if the merger is in the correct position. That value needs to be changed depending on the masses in the binary system that are included at the beginning of the program. After finding the appropriate value for this variable, the code executed correctly, allowing for work with the observational data to begin.

The Gravitational Wave Open Science Center (GWOSC) is a website that hosts and updates information regarding various events detected by LIGO. The first observed LIGO event was used to make comparisons with the simulation [5]. Data was downloaded from one of the observatories as a text file, however it was not formatted in a way that would make it easy to manipulate. This file had two columns, one with time values and the other with strain values. They were lined up vertically so that each strain was next to its corresponding point in time. Because of this, it was not possible to simply copy and paste each column into two separate lists. Instead, the .split() command was used to separate each time and strain into its own element of a list and then run a loop to determine which new list to put it in. This required the use of the modulus operator, which gives the remainder when a number is divided by another number. If the index of the list “mod” 2 equaled 0, making it an even number, it was from the time column.

If the index of the list “mod” 2 equaled 1, meaning it was an odd number, that value came from the strain column. By doing this, the values were able to be separated into two different lists for time and strain. Once this was complete, the lists were graphed and produced a waveform for the observed event.

After the observed data and independent lists were obtained, the simulated waveform needed to be exported out of the original program. The plan was to print the time and strain values to separate text files so that the data could be copied to any program needed. To do this, each value had to be converted to a string so that it could be printed to an external file (see Appendix A). Once these files were created, the values were copied back to make sure the plot was still graphable. Let it also be noted that only the inspiral portion of the simulated waveforms were taken to initially see if the waves matched up well. Seeing as how the inspiral portion is approximated similarly to how the data is gathered, it made sense to test just this portion first. Following that, the same was done with the inspiral portion of the imaginary strain values.

Now it was time to test the simulation’s accuracy with actual observed events. In a new program (see Appendix B), both sets of data were plotted on the same graph and produced this result:

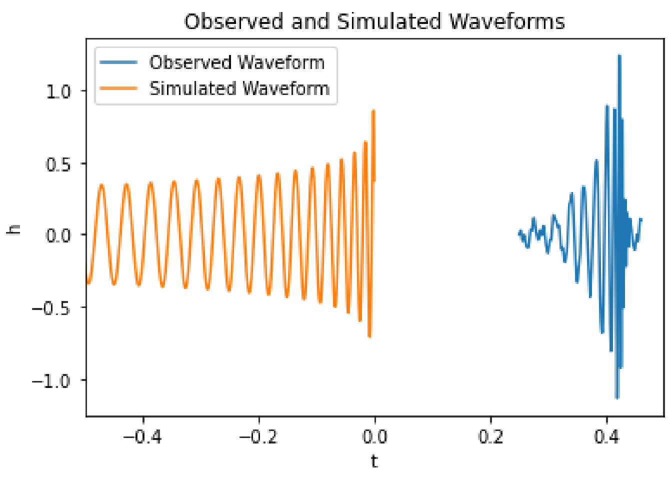


Figure 5: Both the observed and simulated (inspiral) waveforms graphed on the same plot.

As one can see, the time lists include vastly different values. This is because in the simulation, the inspiral and merger phases were adjusted so that they met at $t = 0$. Therefore, all of the inspiral time values were negative numbers. In the case of the observed data, the time values start slightly after t = 0 so none of those values overlap with the simulation.

To fix this problem, the maximum value of the observed strain was found. This gave the peak of that waveform, which needed to be set to t = 0 so that it gave a reference point for lining up both waveforms. Next, the index of that point in the list (its place in the list) was obtained and used to find the corresponding time value, which was labeled as the difference. After that, a “for loop” was used to run through each element in the time list. If the time value for the max strain in the list was less than 0, the loop would add that time value to each entry in the list, effectively moving the entire wave to the right enough to where the peak was at t = 0. If the time value for the max strain was greater than 0, it would subtract that time value from the entire list, moving the wave to the left. This process was done for the simulated time values as well, bringing both waves into the same time frame.

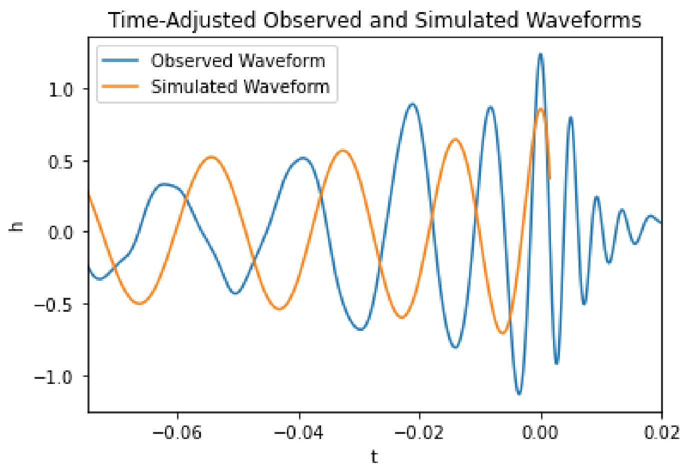


Figure 6: The observed and simulated waveforms together after moving each one to meet at t = 0.

While the waves were definitely much closer to each other, they still did not match up well. The highest peak of the observed waveform was considerably higher than that of the simulated waveform. After looking at the graphs, it seemed like the peak before the observational maximum better resembled the maximum of the simulated waveform. This could be because that was the last peak during the inspiral phase of the observed event. Another possible explanation is that the largest peak of the waveform occurred at the start of the merger phase, meaning that the observed peak would not match up with any value in the simulated waveform. A value was then added to move the observed wave back to the right in the hopes of matching up the two peaks closest in value. With the new list, the graph looked like this:

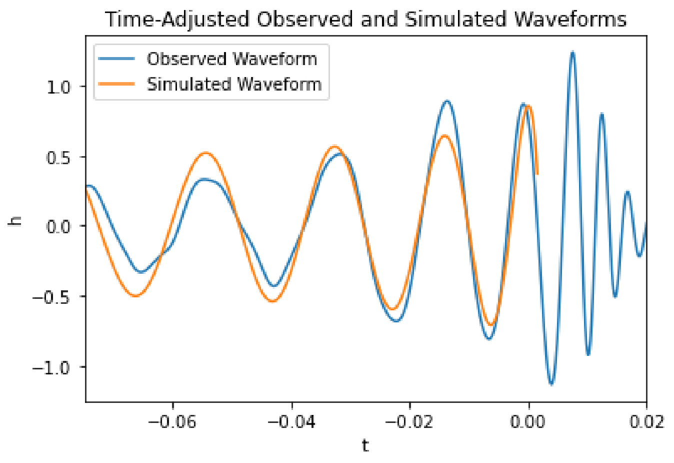


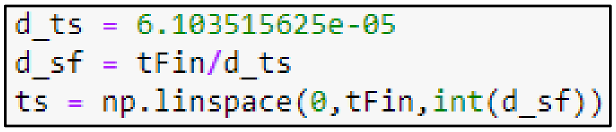
Figure 7: The observed and simulated waveforms after the extra factor is added to match them up better.

With that factor added in, the waves appear to look more similar, both in amplitude and frequency.

Now that the two waveforms lined up, it was time to see how well they matched up by finding the correlation coefficient between the two. They look to be fairly close when the two are simply observed, but this coefficient will deliver more precise results. The Pearson correlation coefficient is calculated using this equation:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2}} \quad , \tag{6}$$

where x_i and y_i are the values of the variable for their respective sample and $\bar{x}$ and $\bar{y}$ are the means for their respective groups. The closer the coefficient is to 1, the more strongly the two variables are positively correlated. The inverse is true for when the coefficient approaches -1. As the coefficient gets closer to 0, that implies that the two groups are weakly correlated. For a more visual outcome, you can graph the observed strain with the simulated strain. Ideally, the output would be a straight line 45° above the x-axis. The first attempt created a graph that had no reasonable structure, just a squiggly line. This was likely caused by the time steps being off, and after testing this theory, it was found to be correct. To fix this, the time scale of the observed event was determined, since that is something that cannot be changed. For the simulation, a list was created and filled with equally separated time values based on the end time, calculated earlier in the program, and the desired number of entries. That number was set at ten thousand, so a function was developed that separated each number by the same time scale as the observed event. To determine the proper amount of entries, the end time was divided by the desired time scale found earlier.



```
d_ts = 6.103515625e-05
d_sf = tFin/d_ts
ts = np.linspace(0,tFin,int(d_sf))
```

Figure 8: The code added to Daniel's program in order to modify the time step and make it a function of the desired time step.

This worked well with the rest of the program, but when compared with the observed strain values, a graph was produced very similar to the one obtained earlier:

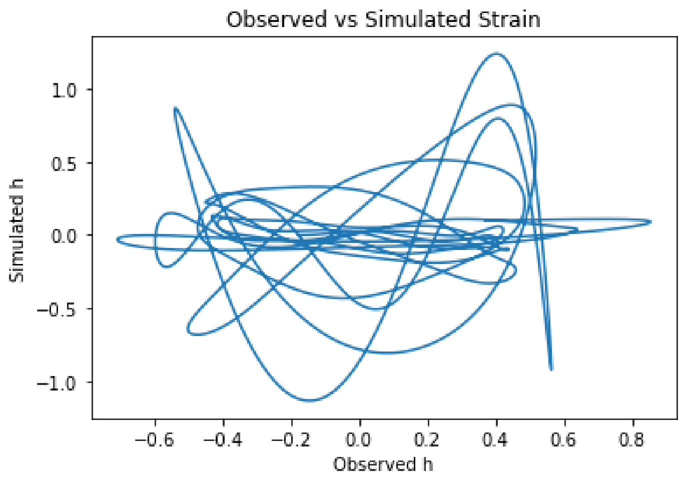


Figure 9: Correlation graph between the observed and simulated strain after the simulated time scale was set to be the same as the observed event.

The next attempt involved doing away with the list of time values altogether. Instead, the time list was inputted straight from the observed event, forcing the simulation to run with that set of times. This however, created multiple errors in the rest of the program, keeping it from even finishing.

To counter this, the previous time step, outlined in Figure 8 and demonstrated in Figure 9, was used again. While it did not perfectly replicate the time step of the observed event, it was very close and would have to suffice for this project. The code from Appendix B was used, but was modified to scale down the comparison to the portion of code that matched the best, since “noise” from observed measurements would make things match up less well. Since the time steps were not the same, it was not possible to simply pick a start and end time to limit the lists to. Instead, desired start and end times were chosen, and a loop looked through each time list for those values. However, it was highly unlikely that the exact time being searched for was in the list, so the code rounded each element to a number of decimal places before determining if that time should be the cutoff. Once the cutoff points were obtained for the beginning and end of the time list, new lists for time and strain were created that would just include the desired interval. This was done for the observed and simulated values, creating four new lists that were used to make a correlation graph. Before the graph was successfully produced, an error occurred due to the lists being different lengths. This must have come from the loops that were assigned to find the cutoff points and the rounding that they were forced to make. To counteract this, a certain number of entries was subtracted from the simulated lists so as to force them into having the same number of elements. Then, a graph was produced that looks like this:

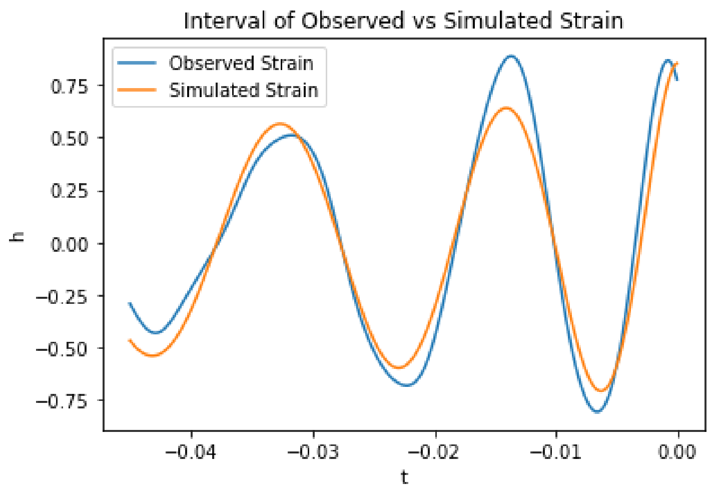


Figure 10: Graph of the two waveforms on the interval used to calculate correlation coefficient (-0.05 < t < 0).

After cutting the time and strain lists down to a more manageable size, the correlation between the observed and simulated strain was then graphed:

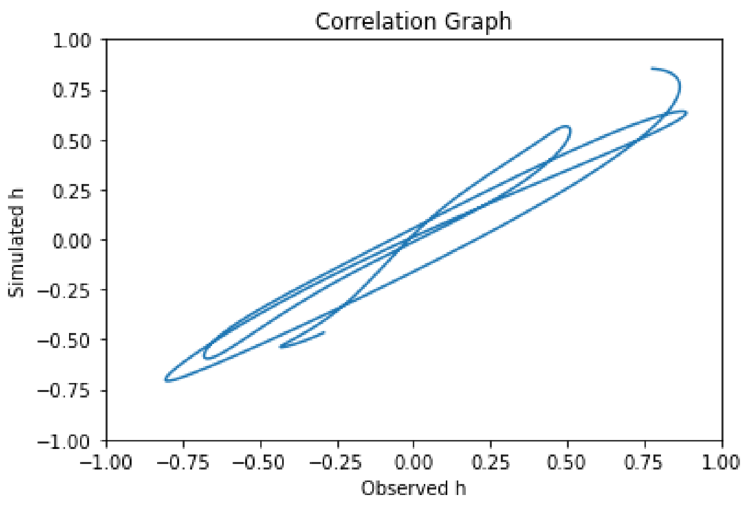


Figure 11: Correlation graph consisting of observed strain plotted against simulated strain.

While a straight 45°, as desired, was not produced, it still shows promising results. The graph seems to oscillate, much like the waveforms do, between the lower left and upper right corners of the plot. It is not a perfect line, but it does resemble the desired graph. Satisfied by the results in Figure 11, a built-in Python function was used to actually calculate a coefficient that describes how well the two lists match up. To do this, the function corrcoef() from the library NumPy was utilized. When two lists are used as parameters for this function, its output is an array that gives the coefficients. For the observed and simulated waveforms, when the mass values were set at 36.2 and 29.1 solar mass units, the correlation coefficient was 0.977, a relatively high value. To get a better sense of just how accurate the simulation is, the observed data was compared to data simulated with different masses as shown in the table below:

Observed Masses (in solar mass units)	Simulated Masses (in solar mass units)	Correlation Coefficient
M1 = 36.2 M2 = 29.1	M1 = 36.2 M2 = 29.1	.977
M1 = 36.2 M2 = 29.1	M1 = 33 M2 = 29	.933
M1 = 36.2 M2 = 29.1	M1 = 25 M2 = 25	.231

Figure 12: Table of correlation coefficients depending on differing simulated masses.

Conclusion:

To confirm the simulation's accuracy, it would ideally produce a low coefficient when the masses of the simulated waveform were starkly different. To test this, mass values of 25 solar mass units each for both objects were inputted and when calculated, produced a correlation coefficient of about 0.231, which is lower than 0.977 as expected. To further test the program, values were inputted that were closer to the observed event, but not the same, in order to see if it produced a correlation coefficient that was between the two previous ones. When mass values of 33 and 29 solar mass units were used, the coefficient was 0.933, much closer to when the observed and simulated masses were the same. This helps support the validity of the simulation since even as the mass values get closer to being correct, the correlation coefficient is still lower than when the simulation is designed exactly like the observed event. The results of this project showed that the simulation does an effective job of predicting gravitational waves, at least for the inspiral portion.

In future projects, students could develop code to automate the process that calculates the stiff time within this program. This would keep the user from having to go into a separate program and calculate the stiff time whenever they change the input masses. There may also be a way to automate these “guess and check” variables that are used to slide waveforms around to match better. This would probably involve finding a correlation coefficient between the overlapping models and then looping through with different factors. Then the program could stick with the factor that produces the best coefficient.

Another improvement for this project would be to include the merger phase when comparing simulated data to observations. The inspiral portion in this project was calculated using a similar method to how professionals simulate the data. However, for the merger portion, professionals use Numerical Relativity (NR) simulations to simulate the waveforms. These calculations are extremely computationally intensive so instead of using NR, this program uses a simpler analytical model. Because of this, the model can be run on most computers, but there is a drawback in that it will not be completely accurate. It would be interesting to see work done that incorporates the merger portion into this comparison with observed data to see how the simulation matches up.

References:

[1] California Institute of Technology. (n.d.). Learn More. Retrieved April 5, 2022, from https://www.ligo.caltech.edu/page/learn-more

[2] Sahely, Bichara. (2012, April 29). Euler's Formula is the Key to Unlocking the Secrets of Quantum Physics. Retrieved April 5, 2022, from https://bsahely.com/wp-content/uploads/2015/09/1204-0102v1.pdf

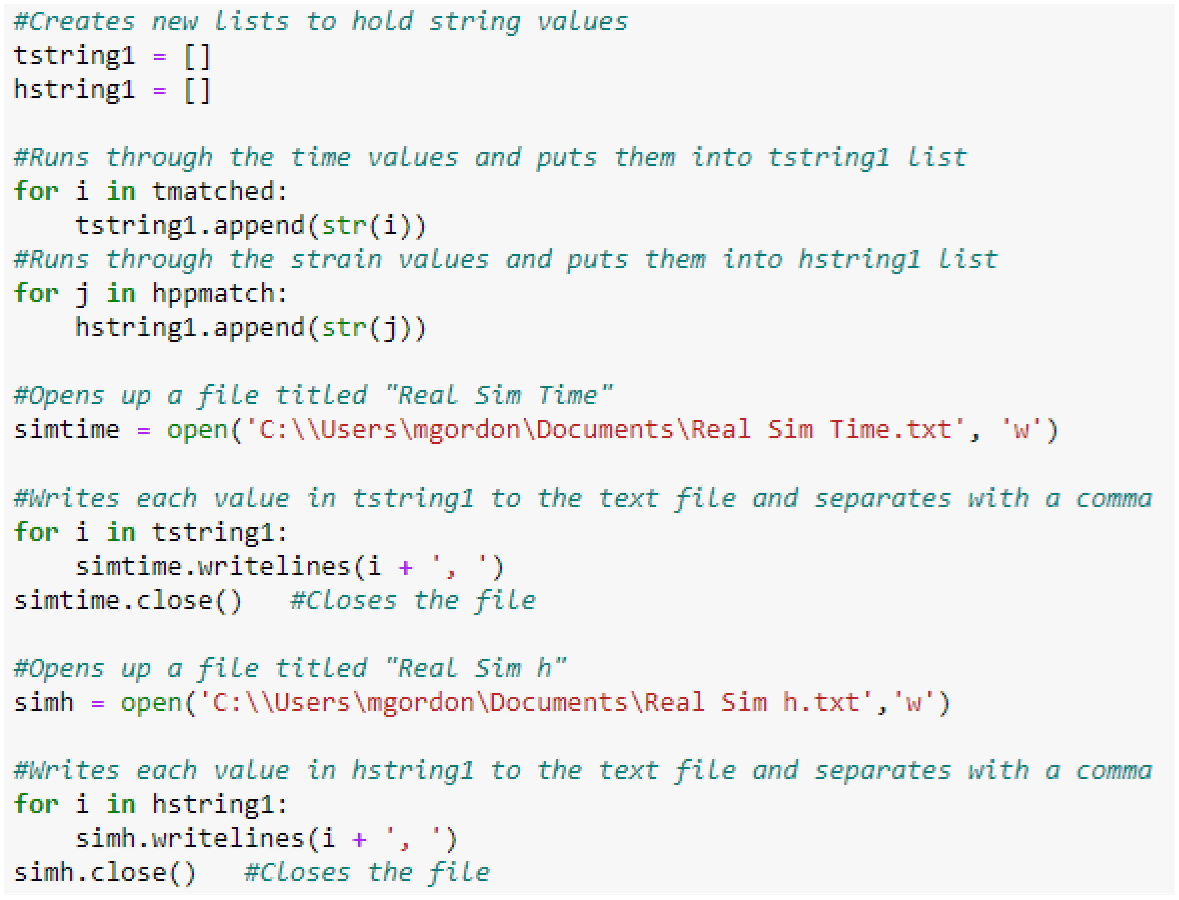
[3] Hancock, Daniel. Analytical Approximations of Gravitational Waves (2021). GitHub Repository. https://github.com/devaoneil/AnalyticGravWave/blob/master/AnalyticGravitationalWave.pdf

[4] Babiuc, Maria and Buskirk, Dillon. MathScripts (2018). GitHub Repository. mbabiuc / October 2018 (github.com)

[5] Gravitational Wave Open Science Center (Updated 2019, May 15). Data release for event GW150914. Retrieved February 9, 2022, from GWOSC (gw-openscience.org)

"This research has made use of data, software and/or web tools obtained from the Gravitational Wave Open Science Center (https://www.gw-openscience.org/), a service of LIGO Laboratory, the LIGO Scientific Collaboration and the Virgo Collaboration. LIGO Laboratory and Advanced LIGO are funded by the United States National Science Foundation (NSF) as well as the Science and Technology Facilities Council (STFC) of the United Kingdom, the Max-Planck-Society (MPS), and the State of Niedersachsen/Germany for support of the construction of Advanced LIGO and construction and operation of the GEO600 detector. Additional support for Advanced LIGO was provided by the Australian Research Council. Virgo is funded, through the European Gravitational Observatory (EGO), by the French Centre National de Recherche Scientifique (CNRS), the Italian Istituto Nazionale di Fisica Nucleare (INFN) and the Dutch Nikhef, with contributions by institutions from Belgium, Germany, Greece, Hungary, Ireland, Japan, Monaco, Poland, Portugal, Spain."

Appendix A - Code Added to Daniel Hancock's Program



```
#Creates new lists to hold string values
tstring1 = []
hstring1 = []

#Runs through the time values and puts them into tstring1 list
for i in tmatched:
    tstring1.append(str(i))
#Runs through the strain values and puts them into hstring1 list
for j in hppmatch:
    hstring1.append(str(j))

#Opens up a file titled "Real Sim Time"
simtime = open('C:\\Users\mgordon\Documents\Real Sim Time.txt', 'w')

#Writes each value in tstring1 to the text file and separates with a comma
for i in tstring1:
    simtime.writelines(i + ', ')
simtime.close()   #Closes the file

#Opens up a file titled "Real Sim h"
simh = open('C:\\Users\mgordon\Documents\Real Sim h.txt','w')

#Writes each value in hstring1 to the text file and separates with a comma
for i in hstring1:
    simh.writelines(i + ', ')
simh.close()   #Closes the file
```

Appendix B - My Comparison Program

```
import matplotlib.pyplot as plt

import numpy as np


tlist_obs = [...] (VALUES OMITTED TO SAVE SPACE)

hlist_obs = [...] (VALUES OMITTED TO SAVE SPACE)

tlist_sim = [...] (VALUES OMITTED TO SAVE SPACE)

hlist_sim = [...] (VALUES OMITTED TO SAVE SPACE)


max_obs = max(hlist_obs)   #Maximum strain value of observed data

ind_obs = hlist_obs.index(max_obs)   #Index of that max in the strain list

diff_obs = tlist_obs[ind_obs]   #Time value for max of observed strain


new_tlist_obs = []   #Creates new list

for i in tlist_obs:

    if(diff_obs < 0):   #If the time value for max strain is less than one,
add that time to get it to 0

        new_tlist_obs.append(i + abs(diff_obs))

    else:

        new_tlist_obs.append(i - abs(diff_obs) + 0.0075)   #0.0075 = Magic
number to line up graphs


max_sim = max(hlist_sim)   #Maximum strain value of simulated data

ind_sim = hlist_sim.index(max_sim)   #Index of that max in the strain list

diff_sim = tlist_sim[ind_sim]    #Time value for max of sim strain
```

```
new_tlist_sim = []   #Creates new list

for i in tlist_sim:

    if(diff_sim < 0):   #If the time value for max strain is less than one
add that time to get it to 0

        new_tlist_sim.append(i + abs(diff_sim))

    else:

        new_tlist_sim.append(i - abs(diff_sim))



plt.xlabel('t')

plt.ylabel('h')

plt.plot(tlist_obs, hlist_obs, label='Observed Waveform')

plt.plot(tlist_sim, hlist_sim, label='Simulated Waveform')

plt.title('Observed and Simulated Waveforms')

plt.legend()

plt.xlim(-.5,.5)



plt.xlabel('t')

plt.ylabel('h')

plt.plot(new_tlist_obs, hlist_obs, label='Observed Waveform')

plt.plot(new_tlist_sim, hlist_sim, label='Simulated Waveform')

plt.title('Time-Adjusted Observed and Simulated Waveforms')

plt.legend()

plt.xlim(-0.075,0.02)



desired_t_start = -0.05   #Establishes desired start time



#For every time value, checks to see if (when rounded 2 decimal places)

# it is equal to the desired start time
```

```
for i in new_tlist_obs:

    if(round(i,2) == desired_t_start):

        t_start_obs = i   #If it is, sets variable equal to that time value




desired_t_end = 0.00   #Establishes desired end time



#For every time value, checks to see if (when rounded 4 decimal places)

# it is equal to the desired end time

for i in new_tlist_obs:

    if(round(i,4) == desired_t_end):

        t_end_obs = i   #If it is, sets variable equal to that time value



#Creates new list that is a subset of the original time list

#This new list goes from the desired start time to desired end time

#A new strain list is created from these indeces as well

cut_t_obs =
new_tlist_obs[new_tlist_obs.index(t_start_obs):new_tlist_obs.index(t_end_obs)
]

cut_h_obs =
hlist_obs[new_tlist_obs.index(t_start_obs):new_tlist_obs.index(t_end_obs)]

plt.plot(cut_t_obs,cut_h_obs, label='Observed Strain')



#Same processes, but for the simulated time lists

for j in new_tlist_sim:

    if(round(j,2) == desired_t_start):

        t_start_sim = j

for j in new_tlist_sim:

    if(round(j,4) == desired_t_end):
        t_end_sim = j



#Creates new list that is a subset of the original time list

#This new list goes from the desired start time to desired end time

#A new strain list is created from these indeces as well

#Factor is subtracted to make sure the simulated and observed lists are of
equal length

cut_t_sim =
new_tlist_sim[new_tlist_sim.index(t_start_sim):new_tlist_sim.index(new_tlist_
sim[-1])-25]

cut_h_sim =
hlist_sim[new_tlist_sim.index(t_start_sim):new_tlist_sim.index(new_tlist_sim[
-1])-25]

#Prints out the length of the observed and simulated lists to make sure they
match

print(len(cut_t_obs))

print(len(cut_t_sim))



plt.plot(cut_t_sim,cut_h_sim, label='Simulated Strain')

plt.xlabel('t')

plt.ylabel('h')

plt.title('Interval of Observed vs Simulated Strain')

plt.legend()



plt.plot(cut_h_obs,cut_h_sim)

plt.title('Correlation Graph')

plt.xlabel('Observed h')

plt.ylabel('Simulated h')

plt.xlim(-1,1)

plt.ylim(-1,1)
```

```
print(np.corrcoef(cut_h_obs,cut_h_sim))   #Print out correlation coefficient
```