```
#%% md
##### CDT Data Analysis 2022 - Coursework (100%)
# Analysing gravitational wave signals
## Deadline Jan 27th, 4pm.
#%% md
#### <div class = "tip">Instructions</div>
#%% md
Please ensure you have read the information in the **background_info** folder before starting this coursework.
#%% md
**These assessments are equivalent to an exam**:
- Submit your work (.ipynb file) via email to the module organiser. Note that you must make sure your notebook compiles fully and that all supporting files that may be included are added to your email.
- Don't worry about how your code looks - marks are not given for pretty code, but rather for the approach used in solving the problem, your reasoning, explanation and answer.
- Please also take note of your University’s policy on **plagiarism**.
#%% md
#### <div class = "tip">Tips</div>

#%% md
- Explain all your reasoning for each step. A *significant fraction* of the marks are given for explanations and discussion, as they evidence understanding of the analysis.
- Some of these steps will take a while to run and compile. It's a good idea to add in print statements to your code throughout eg `print('this step is done')` to make sure that your bit of code has finished.
- Add the import packages statements at the top of your Jupyter notebook. We will use the `pandas` package to read in the data, with eg `dataIn=pd.read_csv('filename.csv')`.
#%% md
***
#%% md
```

```
Gravitational waves are disturbances in the curvature of spacetime, generated by accelerated masses, that propagate as waves outward from their source at the speed of light.  They are predicted in General Relativity and other theories of gravity and since 2017, they have now been observed!

In this exercise we will analyse some mock gravitational wave data from two unknown astrophysical objects merging together and coelescing. We will use a Monte Carlo Markov Chain (MCMC) to compare a scaled model that predicts how the wave changes depending on the total mass of the merging objects and their distance from us to the observed waveform.  This will allow us to determine the nature of the orbiting objects that merged to form the gravitational wave using MCMC, whether for instance they could be originating from merging white dwarfs, neutron stars or black holes.

The mock or simulated waveforms measure the strain as two compact, dense astrophysical objects coalesce. The strain describes the amplitude of the wave. The system is parameterised by the masses of the merging objects, $M_1$ and $M_2$, and their distance from the observer $D$.

Other useful parameters and equations relevant for this assessment are given in the background information folder.
#%% md
***
#%% md
***
#%% md
## Part A - The data

1. Read in the datafile of the observed waveform `Observedwaveform.csv`. These files store the strain as a function of "GPS time" for the merger of two bodies.
```

```

2. The GPS time of the merger for your waveform is 1205951542.153363.  Your data will need to be shifted so that the merger occurs at time = 0 secs.  This is required as we will compare our data with a which have the merger at t=0s.

3. We need to estimate the average noise and its standard deviation in our data. This requires careful thought about where the noise can be seen in the waveform.
#%% md
**Answer:**
#%% md
*Your answer here*
#%% md
# Part A - Answers
## 1
As stated in Chapter 8, the frequency of the inspiral signal depends on the chirp mass of the two objects merging.
This chirp mass is given by:
$$M_{ch} = \dfrac{(M_1 M_2)^{3/5}}{(M_1 + M_2)^{1/5}}.$$

The observed signal's associated chirp mass required black holes to be involved rather than neutron stars.

This could still allow a BH-NS merger, however unbalanced masses, i.e. a mass ratio $q = M_2/M_1 \ll 1.0$, give a signal with a consistently oscillating amplitude as the objects orbit each other. This was not seen in 2015, suggesting a mass ratio closer to 1.

Combined with the chirp mass, this requires 2 Black Holes.
#%% md
## 2

#%%
```

```
import numpy as np
import pandas as pd
import scipy as sc
import matplotlib.pyplot as plt
#%%
data_events = pd.read_csv('Observedwaveform.csv')
data_events['time (s)'] = data_events['time (s)'] - 1205951542.153363
data_events.head()
#%% md
# 3
#%%
plt.figure(figsize=(20,4))
plt.plot(data_events['time (s)'], data_events['strain'], color='dodgerblue', label='Observed Waveform')
plt.legend(loc='upper right')
#%% md
Noise can be seen after the merger. Will take the noisy section as any data after t = 0.01s
#%%
# select noisy data
noise = data_events.loc[np.where(data_events['time (s)'] > 0.01)]
noise
#%%
noise_mean = np.mean(noise['strain'])
noise_std = np.std(noise['strain'])

print('noise mean =', noise_mean)
print('noise_std  =', noise_std)
#%% md
***
#%% md
## Part B - Using model waveforms to estimate the total mass and distance to the system "a by-eye estimate")

In this part of the question we will attempt to produce a waveform for any mass and distance values using a reference waveform with $M=40 M_{sun}$, $D=
```

```
1$Mpc and $q=M_2/M_1 = 1$ and scaling it by any new mass and/or distance.

The reference waveform/template we will use is``` reference_Mtot40Msun_Dist1Mpc.csv```.

You will need to follow the steps below when answering this question:

1. Open the reference/template file using the `pandas` package. Write a function in python to scale the time and strain of any waveform with $q=1$, total mass $M$ and distance $D$ from the reference waveform file ```reference_Mtot40Msun_Dist1Mpc.csv``` using the equations for how the waveform strain and time depends on mass and distance from the Background_info_mini_project.ipynb notebook.

2. Test your function works by substituting in $M=70\,M_{sun}$ and $D=5$Mpc, and compare your resulting waveform with the template in `reference_Mtot70Msun_Dist5Mpc.csv`.  Comment on your result.

3. Use your function to scale the template waveform ($M=40 M_{sun}$, $D=1$Mpc) to make an initial rough estimate "by eye" of the total mass and distance that "best" fits your data (e.g. to within +/- 5 Msun, +/- 100 Mpc).
#%% md
**Answer:**
#%% md
*Your answer here*
#%% md
1. Open the reference/template file using the `pandas` package. Write a function in python to scale the time and strain of any waveform with $q=1$, total mass $M$ and distance $D$ from the reference waveform file ```reference_Mtot40Msun_Dist1Mpc.csv``` using the equations for how the waveform strain and time depends on mass and distance from the
```

```
Background_info_mini_project.ipynb notebook.
#%%
reference_waveform = pd.read_csv('reference_Mtot40Msun_Dist1Mpc.csv')
reference_waveform.head()
#%%
plt.figure(figsize=(20,4))
plt.plot(reference_waveform['time (s)'], reference_waveform['strain'], color='dodgerblue', label='Reference Waveform')
plt.legend(loc='upper right')
#%% md
From the notes, this is the template equation:

$$
h(t,M,D) = \left(\dfrac{M}{M_{\rm {ref}}}\right) \left(\dfrac{D_{\rm {ref}}}{D}\right) h(t_{\rm {ref}})
$$

where:

$$ t_{\rm {ref}}=\left(\dfrac{M_{\rm {ref}}}{M}\right)t  $$
#%%
def h(t, M, D):
    M_ref = 40 # Msol
    D_ref = 1  # Mpc
    t_ref = (M_ref / M) * t
    ref = pd.read_csv('reference_Mtot40Msun_Dist1Mpc.csv')
    closest_ind = np.abs(ref['time (s)'].values - t_ref).argmin()
    h_tref = ref['strain'][closest_ind]
    return (M / M_ref) * (D_ref / D) * h_tref

#%% md

#%% md
2. Test your function works by substituting in $M=70\,M_{sun}$ and $D=5$Mpc, and compare your resulting
```

```
waveform with the template in `reference_Mtot70Msun_Dist5Mpc.csv`.  Comment on your result.
#%%
test_waveform = pd.read_csv('reference_Mtot70Msun_Dist5Mpc.csv')
plt.figure(figsize=(20,4))
plt.plot(test_waveform['time (s)'], test_waveform['strain'], color='dodgerblue', label='Test Waveform')
plt.legend(loc='upper right')
#%%
t_min = test_waveform['time (s)'].min()
t_max = test_waveform['time (s)'].max()
t = np.linspace(t_min,t_max,2000)
scaled_ref = [h(time, M=70, D=5) for time in t]
#%%
plt.figure(figsize=(20,4))
plt.plot(test_waveform['time (s)'], test_waveform['strain'], color='dodgerblue', label='Test Waveform')
plt.plot(t, scaled_ref, color='crimson', label='Scaled')
plt.legend(loc='upper left')
#%% md
3. Use your function to scale the template waveform ($M=40 M_{sun}$, $D=1$Mpc) to make an initial rough estimate "by eye" of the total mass and distance that "best" fits your data (e.g. to within +/- 5 Msun, +/- 100 Mpc).
#%%
t_min_data = data_events['time (s)'].min()
t_max_data = data_events['time (s)'].max()
t = np.linspace(t_min_data,t_max_data,2000)
scaled_data = [h(time, M=60, D=1000) for time in t]
scaled_df = pd.DataFrame(data=t, columns=['time (s)'])
scaled_df['strain'] = scaled_data

plt.figure(figsize=(20,4))
plt.plot(data_events['time (s)'], data_events['strain'], color='dodgerblue', label='Test Waveform')
plt.plot(scaled_df['time (s)'], scaled_df['strain'
```

```
], color='crimson', label='Scaled')
plt.legend(loc='upper left')
#%% md
A "by eye" estimate gets the chirp mass to be approx 60 Solar Masses and the merger to have occurred 1000 Mpc away
#%% md
***
#%% md
## Part C- Get data and model to have the same x values.

Now that we have our observed data, and can scale the template data to any mass and distance, we need to do one more fix.  Currently our data and our templates have different sampling on the $x$ axis - ie they have different values of $x$ (time(.  We need to try and match the $x$ times up so that for each value of $x$ we can compare the $y$ values (the observed strain with the strain from the scaled template).

We need to only consider the times when we have observed data, so we will trim our data set.

1.  Our data waveform starts at some time $t_{\rm min}$. Find out what this is.  Next, take your observed data waveform and output data for $t$ > $t_{\rm min}$ and $t$ < $0$ (ie only keep information for times $\le 0$ (before the merger), or for times where there is data).  Verify, by plotting, that your new observed waveform only has data in this restricted time range.

2. We now need to put both observed and template waveforms on the same time sampling, ie the same number of data points.  The model waveforms have approx 20,000+ time steps, yet the data has less than hundreds in the time range specified!

We need to interpolate between our observed data and
```

```
 the template. To do this use the following code:

(assuming `x[index]` and `y[index]` are the observed data from Part D.1 and scaled template time is your scaled reference template to your suggested values of $M$ and $D$ from Part C3.)

```
from scipy.interpolate import interp1d

# get interpolation object using data
interp_fn =interp1d(x[index],y[index],bounds_error=False)

# now get scaled template and get the strains for the same x axis as data
interp_strain = interp_fn(scaled_template_time)

#plot
plt.plot(scaled_template_time,interp_strain)
```

Briefly verify that this works.

Hints:
 * *One can use the following code example `index = np.where((data > 5)&(data < 10))[0]`. This type of statement returns a list of indices (`index`) where the conditions in the bracket have been met. `data_[index]` pulls out `data` that satisfy the conditions in the brackets above.*
#%% md
Answer
#%% md
Your answer here
#%%
# select waveform data for times after tmin and pre-merger
working_data = data_events.copy()
working_data = working_data.loc[t_min_data < working_data['time (s)']]
```

```
working_data = working_data.loc[working_data['time (s)'] < 0.0]

plt.figure(figsize=(20,4))
plt.plot(working_data['time (s)'], working_data['strain'], color='dodgerblue', label='Working Data')
plt.legend(loc='upper left')
#%%
from scipy.interpolate import interp1d

# get interpolation object using data
interp_fn =interp1d(working_data['time (s)'], working_data['strain'],bounds_error=False)

# now get scaled template and get the strains for the same x axis as data
interp_strain = interp_fn(scaled_df['time (s)'])

#plot
plt.figure(figsize=(20,4))

plt.plot(scaled_df['time (s)'],interp_strain, color='crimson')
plt.plot(working_data['time (s)'], working_data['strain'], color='dodgerblue', label='Working Data')
#%% md
***
#%% md
## Part D - Estimating the best fit total mass using MCMC

Now that we know how to make the scaled template (ie 40Msun,1Mpc template file) and the observed data have the same time sampling, we can use MCMC to find out the total mass of the system that made the data we see.

*If you run into any difficulties completing this component of the coursework, you can still attempt the following parts using your by-eye estimates for $M$ and $D$ from Part B.*
```

Think carefully about what the likelihood function will be in this case (see Chapters 6-9).

1. Use MCMC to sample the total mass, M to produce a best-fit value for your data.

2. Display the results in an appropriate manner and comment on your findings, as well as your results from the MCMC.

3. Report the median and 90% credible limits on your values.

You may assume that:
- the noise is described by a Gaussian distribution ,
- the total mass of the system is in the range [20, 100] M_{sun}.

Hints:

 * _Think very carefully about the form of your likelihood since here we are comparing observed data with a model_

 * _You should work with "log(Likelihood)" to avoid numerical errors - note this will affect both your posterior and the step in the MCMC chain where we usually write $p_{\rm proposed}/p_{\rm current}$_

 * _The step size between samples of the MCMC is quite important. A suggested value for the mass is $0.1\,M_{sun}$_

 * _The initial guess of your mass is also very important. You may find yourself getting into a local minimum rather than your code finding the true minimum._

 * _Test your MCMC on a small number of samples (e.g

```
. 10-100) before trying it with a larger number (e.g. $10^5$ or $10^6$)_

 * _At the end, ask yourself if you need to include every sample?_

 * _Depending on your step size, this part can take a long time to run. Suggest that you move all your plotting routines to a different code cell to save you re-running everything 10000s of times when you just want to change a plot command._

 * _To find out how long it will take for a Jupyter notebook to compile the MCMC code cell, add the following snippet to your code before you go into your MCMC loop (where Nsteps is the number of steps your MCMC is using):_

```def time_spent_waiting(n):
    from datetime import datetime,timedelta
    preddur=[n*0.01,n*0.02]
    print('predicted duration: {:.2f}-{:.2f} mins'.format(preddur[0]/60.,preddur[1]/60.))
    return```
#%% md
**Answer:**
#%% md
*Your answer here*


#%%
working_data
#%%
def h(t, M, D):
    M_ref = 40 # Msol
    D_ref = 1  # Mpc
    t_ref = (M_ref / M) * t
    ref = pd.read_csv('reference_Mtot40Msun_Dist1Mpc.csv')
    closest_ind = np.abs(ref['time (s)'].values - t_ref).argmin()
```
```

```
    h_tref = ref['strain'][closest_ind]
    return (M / M_ref) * (D_ref / D) * h_tref
#%%
from scipy.stats import norm

var = np.var(working_data['strain'])

def log_likelihood(M, data, variance):
    D = 1000

    loglike_val = 0.0
    for i in range(len(data['strain'])):
        h_obs = data['strain'].iloc[i]
        t = data['time (s)'].iloc[i]
        h_model = h(t, M, D)
        residual = h_obs - h_model
        chi_squared = np.sum((residual / variance) ** 2)
        loglike_val += -0.5 * chi_squared
    return loglike_val



def MCMC_Mass(N_mcmc, M_prior, sigma_mcmc):
    M_current = np.zeros(N_mcmc+1)
    M_current[0] = M_prior

    for i in range(N_mcmc):
        print(i)
        p_current = log_likelihood(M_current[i], working_data, var) # put current value in posterior equation
        dM = np.random.normal(0,sigma_mcmc) # randomly draw a value of theta to trial
        M_proposed = M_current[i] + dM #get new proposed theta (random theta + stepsize)
        p_proposed = log_likelihood(M_proposed, working_data, var) # calculate posterior p for proposed theta

        # #  keep this value if probability proposed
```

```
 theta greater than the current prob
        # if p_proposed > p_current:
        #     M_current[i+1] = M_proposed
        # else:
        # # if probability lower
        # # use the ratios of probability to define probability of whether we move to that value or not
        #     p_new_move = p_proposed - p_current
        #     # generate random number for probability
        #     u_random = np.random.uniform(0,1)
        #     # if u_random < p_new_move, then accept, if not, reject
        #     if u_random <= p_new_move:
        #         M_current[i+1] = M_proposed
        #     else:
        #         M_current[i+1] = M_current[i]

        p_acceptance = p_proposed - p_current

        # Accept or reject the proposal
        if p_acceptance >= 0 or np.log(np.random.rand()) < p_acceptance:
            M_current[i+1] = M_proposed
        else:
            M_current[i+1] = M_current[i]

        print(M_current[i])
    return M_current


#%%
# set up MCMC step paramaters
N_mcmc = 200
M_prior = 60.
# choose a value for width of normal distribution to get the step in height
# this is between the prior and the likelihood values
sigma_mcmc = 0.2

```

```
M_current = MCMC_Mass(N_mcmc, M_prior, sigma_mcmc)

# get mean + std from mcmc generated samples
mean_mcmc=np.mean(M_current)
std_mcmc=np.std(M_current)

print('The mean Mass from the MCMC is {:.2f} +/- {:.2f} Solar Masses'.format(mean_mcmc,std_mcmc))
#%% md
***
#%%
plt.plot(M_current)
plt.xlabel('Number of Runs')
plt.ylabel('Total Mass')
plt.grid()
#%% md
## Part E - Putting it all together

If you run into any difficulties completing Part E, you can still attempt this part using your by-eye estimates for $M$ and $D$ from Part B.

1. Calculate the chirp mass for your system and the individual masses of your merging bodies. Comment on  your individual masses.

2. Comment on what your analysis suggests are the best astrophysical candidates for the merging objects?  What information are you missing to rule out other astrophysical candidates?
#%% md
**Answer:**
#%% md
*Your answer here*


#%% md
***
#%%
def chirp_mass(M1, M2):
    return (M1 * M2)**(3/5) / (M1 + M2)**(1/5)
```

```
#%% md
Unsure how to get from total mass, M, to the chirp or component masses, however the signal does not appear to vary wildly in peak amplitude over the orbits suggesting that the mass ratio is somewhere closer to 1 than 0. This would imply that the two components are on the order of 30 Solar Masses.

Since the most massive known Neutron Star is estimated to be around 2 Solar Masses, this would imply that the components were stellar mass black holes.
```