WaveformCompression

December 12, 2022

[ ]:
```python
# Import lal packages
import lal, lalsimulation

# Import pycbc packages
from pycbc import psd, waveform, detector
from pycbc.filter import match, matched_filter
import pycbc.vetoes

# Import my ROM package
from ReducedModel import *

# Import other packages
import matplotlib.pyplot as plt
import scipy.constants as constant
```

[ ]:
```python
# Define a few aesthetic colours using Hex codes
myblue   = '#0F56B5'
myred    = '#EF4647'
mygrey   = '#666666'
mygreen  = '#2CA02C'
mypurple = '#9467bd'

rc_params = {
    'backend': 'pdf',
    'axes.labelsize': 24,
    'axes.titlesize': 32,
    'font.size': 24,
    'legend.fontsize': 18,
    'xtick.labelsize': 18,
    'ytick.labelsize': 18,
    'font.family': 'serif',
    'font.sans-serif': ['Bitstream Vera Sans'],
    'font.serif': ['Times New Roman'],
    'text.latex.preamble': r'\usepackage{amsmath} \usepackage{amssymb}␣
↪\usepackage{amsfonts}',
    'text.usetex':True,
    'axes.linewidth':1.75,
```

```
    'patch.force_edgecolor':True
}
plt.rcParams.update(rc_params);
```

The above 'settings' are taken directly from Geraint's tutorial code.

1 IMRPhenomTHM

[ ]:
```
# This function takes the masses of the two binary elements and returns the␣
↪modes.

def mode_22(m_1, m_2):
    f_min        = 20.      # Starting frequency -- this can change to find an␣
↪appropriate range of waveforms
    f_ref        = 20.      # Reference frequency
    delta_t      = 1/2048   # Sampling time
    phase        = 0.0      # Phase at reference frequency
    mass_1       = m_1      # In solar units
    mass_2       = m_2

    chi_1x      = 0
    chi_1y      = 0
    chi_1z      = 0
    chi_2x      = 0
    chi_2y      = 0
    chi_2z      = 0

    distance    = 100 * 1e6 * lal.PC_SI # Luminosity distance in SI
    laldict     = lal.CreateDict()
    l_max   = 2

    hlm     = lalsimulation.SimInspiralChooseTDModes(phase,delta_t,mass_1,
                                                 mass_2,chi_1x, chi_1y,␣
↪chi_1z,
                                                 chi_2x, chi_2y, chi_2z,
                                                 f_min,f_ref, distance,␣
↪laldict, l_max,
                                                 lalsimulation.IMRPhenomTHM
                                                 )
    #epoch = h_plus.epoch.gpsSeconds + h_plus.epoch.gpsNanoSeconds/1e9
    times_modes    = lalsimulation.SphHarmTimeSeriesGetMode(hlm, 2, 2).deltaT *␣
↪np.arange(len(lalsimulation.SphHarmTimeSeriesGetMode(hlm, 2, 2).data.data))␣
↪#+ epoch
    mode_22 = lalsimulation.SphHarmTimeSeriesGetMode(hlm, 2, 2).data.data

    return times_modes, np.real(mode_22)
```

We define the mass ratio q as:

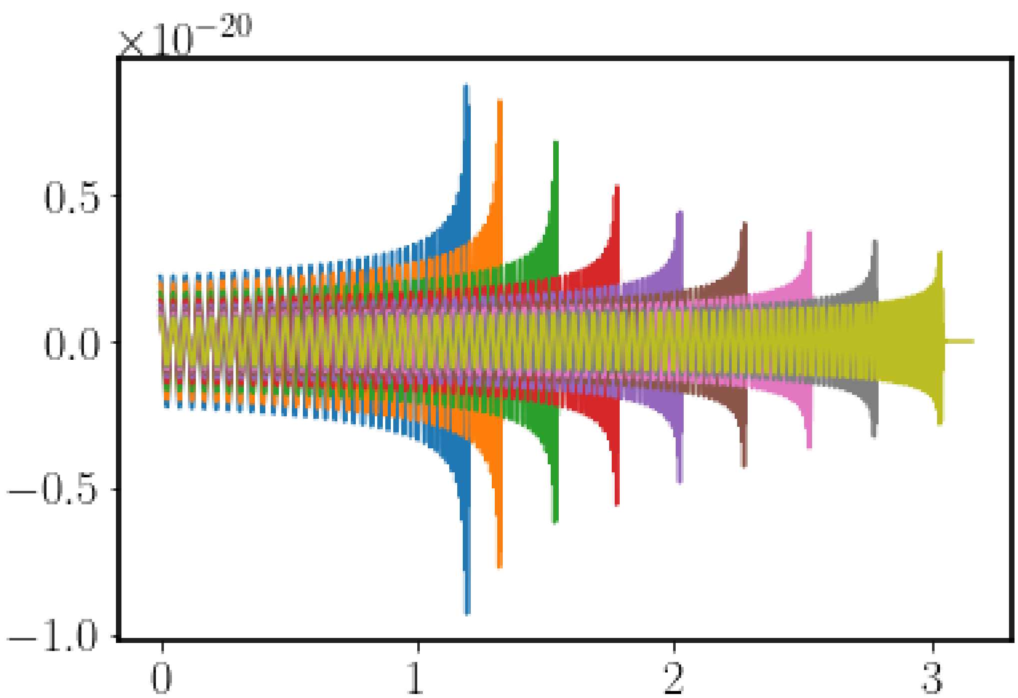
$$q = \frac{m_1}{m_2}$$

[ ]:
```
#Defınding the total mass, a range of mass ratios q and thus generating the
 ↪associated waveforms.

M = 50 * lal.MSUN_SI
q = np.arange(1, 10, 1)
mass_1 = (M*q)/(q+1)
mass_2 = M/(q+1)
time_modes = []
waveforms = []

for i in range(len(q)):
    t, mode = mode_22(mass_1[i], mass_2[i])
    time_modes.append(t)
    waveforms.append(mode)

for i in range(len(q)):
    plt.plot(time_modes[i], waveforms[i])
plt.show()
```



It is clear that the binary systems merge at different points in time. Since the point in time that a merge occurs in arbitrary for the sake of building a reduced basis and empirical interpolant, we should shift these waveforms such that the merge points line up. We can then truncate the inspiral part so that the domain is consistent.

[ ]:
```
# Credit to Lewis Bradley fro this code. I have taken it for now

def truncateWaveforms(waveforms, times, time_geometric, mass_total, delta_t, verbose=False): #Given a list of arrays, a time in geometric units, the total mass of the system (in kg), and a sampling time we can truncate the arrays to a certain time and return them as a 2D array.
    verboseprint = print if verbose else lambda *a: None
    #Firstly convert the geometric time into SI units.
    M = mass_total * constant.G/(constant.c)**3
    time = time_geometric * M
    verboseprint("{t_geometric}M in SI units is {t_SI}s".format(t_geometric=time_geometric, t_SI=round(time, 2)))

    #Now we find the number of elements to truncate the array by.
    truncateLength = int(np.floor(time/delta_t)) #Converts to integer and rounds-down so we only get less than the maximum duration rather than over.
    verboseprint("New array size is {len} by {truncateLength} elements".format(len=len(waveforms), truncateLength=truncateLength))

    #Now loop through each array and truncate it to the truncation length
    truncatedWaveforms = np.empty([len(waveforms), truncateLength])
    truncatedTimes = np.empty([len(times), truncateLength])
    for i in range(len(waveforms)):
        truncatedWaveforms[i] = waveforms[i][:truncateLength]
        truncatedTimes[i] = times[i][:truncateLength] #This could throw an error as haven't checked that the times and waveform lists have same length (although they should)


    return truncatedWaveforms, truncatedTimes
```

Currently, this code does not achieve what is required above and only truncates the upper part; this is a fix that will occur soon.

[ ]:
```
time_cgs = 5000
delta_t = 1/2048

truncatedWaveforms, truncatedTimes = truncateWaveforms(waveforms, time_modes, time_cgs, M, delta_t, verbose=True)
reducedBasis, errors = greedy(truncatedWaveforms, truncatedTimes[0], error=1e-45, verbose=False)
print('The RB shape is: ', reducedBasis.shape)
```

```python
print(reducedBasis)
B, nodes = empirical_interpolation(reducedBasis)

wave_choice = 30
h = truncatedWaveforms[wave_choice]
t = truncatedTimes[wave_choice]
interpolant  = np.dot(h[nodes], B)

plt.plot(t, h)
plt.plot(t, interpolant, linestyle='--')
plt.show()

for i in range(len(q)):
    h = truncatedWaveforms[i]
    t = truncatedTimes[i]
    interpolant  = np.dot(h[nodes], B)
    plt.plot(t, h)
    plt.plot(t, interpolant, linestyle='--')
plt.show()
```

```
5000M in SI units is 1.23s
New array size is 9 by 2521 elements
The RB shape is:  (2521, 9)
[[ 9.48928021e-01  8.54257498e-01  8.25876292e-01 …  6.03665762e-01
   3.87709055e-02  4.59016238e-01]
 [ 9.47231309e-01  8.52719473e-01  8.24375884e-01 …  6.02538276e-01
   3.87003866e-02  4.58158359e-01]
 [ 9.41967251e-01  8.47970296e-01  8.19771380e-01 …  5.99142751e-01
   3.84841469e-02  4.55575923e-01]
 …
 [-2.28769963e-05 -4.93998432e-01 -1.52592709e+00 …  1.26354161e+00
  -1.31228490e-01 -3.58109004e+00]
 [-1.36852502e-05 -7.73443540e-01 -1.45656313e+00 …  1.35782285e+00
  -4.32990683e-01 -3.80734644e+00]
 [ 5.76854936e-06 -1.03606966e+00 -1.37246108e+00 …  1.44388431e+00
  -7.33873569e-01 -4.01486392e+00]]
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
```

```
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
ERROR_repeat_node
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
/var/folders/99/p9l8gcl141vf76_vh206t0d00000gn/T/ipykernel_3679/3772813024.py in
 ↪<module>
      6 print('The RB shape is: ', reducedBasis.shape)
      7 print(reducedBasis)
----> 8 B, nodes = empirical_interpolation(reducedBasis)
      9
     10 wave_choice = 30

~/Documents/GitHub/Y4_2022-2023_ROQ/joseph/ReducedModel.py in
 ↪empirical_interpolation(basis, verbose)
    129
    130         V_select = V[:i, :i]
--> 131         V_inv    = np.linalg.inv(V_select.T)
    132         B = np.dot(V_inv, basis[:i])
    133         I = np.dot(basis[i, nodes], B)

<__array_function__ internals> in inv(*args, **kwargs)

~/opt/anaconda3/lib/python3.9/site-packages/numpy/linalg/linalg.py in inv(a)
    543     signature = 'D->D' if isComplexType(t) else 'd->d'
    544     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 545     ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    546     return wrap(ainv.astype(result_t, copy=False))
    547

~/opt/anaconda3/lib/python3.9/site-packages/numpy/linalg/linalg.py in
 ↪_raise_linalgerror_singular(err, flag)
     86
     87 def _raise_linalgerror_singular(err, flag):
---> 88     raise LinAlgError("Singular matrix")
     89
     90 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

The waveform has not been successfully modelled by my empirical interpolation algorithm. I think this is because of the truncation issues descirbes above. If the greedy algorithm is spitting out singular matrices then there must be an issue with the training space. Upon printing the RB, there are clearly repeat entries. I will have to perform some extra troubleshooting.