br$_{py}DocsDocumentation$

Release 1.0

Juan Martinez-Sykora

Mar 20, 2019

Contents

Contents:

CHAPTER 1

br_fft

This library is a superclass of Helita. Helita documentation can be found http://helita.readthedocs.io/en/latest/index.html

1.1 BifrostData class

bifrost.py includes the BifrostData class (among others) which is needed for br_fft.

1.2 FFTData class

This class can be found within bifrost_fft.py. It performs operations on Bifrost simulation data in its native format. After creating a class for a specific snap root name and directory (much like with BifrostData), one can get a dictionary of the frequency and amplitude of the Fourier Transform for a certain quantity over a range of snapshots.

We have defined 3 variables that allow us to decompose the velocity in Alfvenic, fast mode and longitudinal component ('alf', 'fast', and 'long'). Here, we show the transformation of 'alf',

[8]:
```
from br_fft import bifrost_fft as brft
```

[9]:
```
dd = brft.FFTData(file_root = 'cb10f', fdir = '/net/opal/Volumes/Amnesia/mpi3drun/
↪Granflux')
```

[10]:
```
transformed = dd.get_fft('alf', snap = [430, 431, 432], iix = 5, iiy = 20)
```

```
WARNING: cstagger use has been turned off, turn it back on with "dd.cstagop = True"
```

[11]:
```
transformed.keys()
```

[11]:
```
dict_keys(['freq', 'ftCube'])
```

Depending on the number of snaps and the size of the cube, using cuda or python multiprocessing may speed up the calculation.

1.2.1 1. Using cuda

If pycuda is available, the code imports reikna (a python library that contains fft functions using pycuda). In order to make use of the GPU, use the function run_gpu(). The default is to **not** use the GPU, even if there is one available.

[12]:
```
dd.run_gpu() # to use GPU
dd.run_gpu(False) # to stop use of GPU
```

When get_fft() is called, the GPU will be used in accordance with the last call to run_gpu(). If the GPU has limited memory, the user can specify numBlocks in the call to get_fft(). This will send the calculation over to the GPU in several blocks as opposed to all at once. To use 5 different blocks, a call would look like this:

[13]:
```
usingBlocks = dd.get_fft('bx', snap = [400, 401, 402], numBlocks = 5)
```

1.2.2 2. Using python multiprocessing

This can be used whether or not pycuda is available, as multiprocessing is a library that comes with python. It makes use of threading on the CPU. In order to use a multiprocessing threadpool when calculating the Fourier Transform, specify numThreads with a number greater than 1, when calling get_fft():

[14]:
```
usingThreads = dd.get_fft('bx', snap = [400, 401, 402], numThreads = 10)
```

1.2.3 FFT demo #1

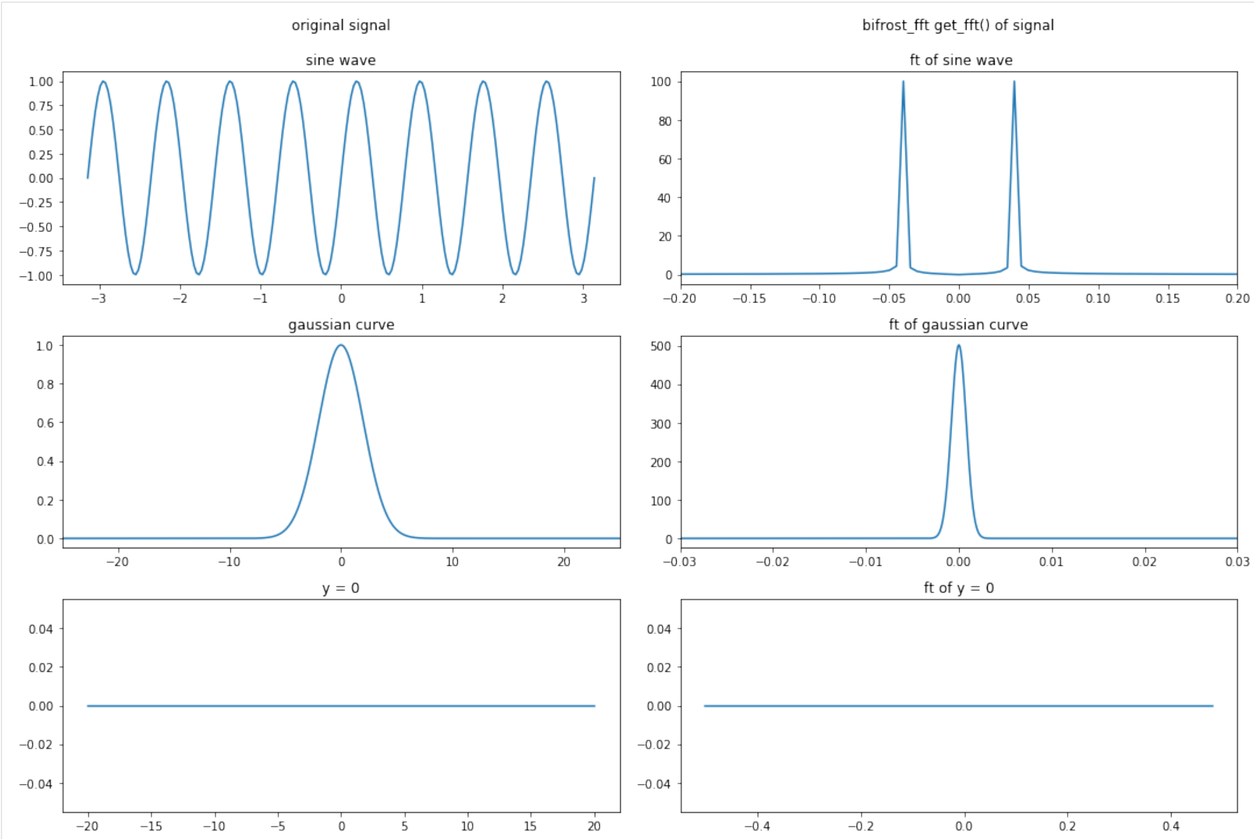
This first demo tests the get_fft() method with standard functions: a sine wave, a gaussian curve, and y = 0. It pre-sets dd.preTransform and

[15]:
```
import numpy as np
import helita.sim.cstagger
from helita.sim.bifrost import BifrostData, Rhoeetab, read_idl_ascii
from helita.sim.bifrost_fft import FFTData
import matplotlib.pyplot as plt

# note: this calls bifrost_fft from user, not /sanhome

dd = FFTData(file_root='cb10f',
             fdir='/net/opal/Volumes/Amnesia/mpi3drun/Granflux')

# test 1: ft of y = sin(8x)
x = np.linspace(-np.pi, np.pi, 201)
dd.preTransform = np.sin(8 * x)
dd.freq = np.fft.fftshift(np.fft.fftfreq(np.size(x)))
dd.run_gpu(False)
# preTransform is already set
tester = dd.get_fft('not a real var', snap='test')
fig = plt.figure(figsize=(15,10))

numC = 3
numR = 2

# plotting original sin signal
ax0 = fig.add_subplot(numC, numR, 1)
ax0.plot(x, dd.preTransform)
ax0.set_title('original signal' + '\n\nsine wave')
```

(continues on next page)

(continued from previous page)

```
# plotting transformation sin signal
ax1 = fig.add_subplot(numC, numR, 2)
ax1.plot(tester['freq'], tester['ftCube'])
ax1.set_title('bifrost_fft get_fft() of signal' + '\n\n ft of sine wave')
ax1.set_xlim(-.2, .2)

# test 2: ft of gaussian curve
n = 30000  # Number of data points
dx = .01  # Sampling period (in meters)
x = dx*np.linspace(-n/2, n/2, n)  # x coordinates

stanD = 2  # standard deviation
dd.preTransform = np.exp(-0.5 * (x/stanD)**2)

# plotting original gaussian signal
ax2 = fig.add_subplot(numC, numR, 3)
ax2.plot(x, dd.preTransform)
ax2.set_xlim(-25, 25)
ax2.set_title('gaussian curve')

# plotting transformation of gaussian signal
dd.freq = np.fft.fftshift(np.fft.fftfreq(np.size(x)))
ft = dd.get_fft('not a real var', snap='test')  # preTransform is already set
ax3 = fig.add_subplot(numC, numR, 4)
ax3.plot(ft['freq'], ft['ftCube'])
ax3.set_xlim(-.03, .03)
ax3.set_title('ft of gaussian curve')

# test 3: ft of y = 0
# plotting original horizontal line
x = np.linspace(-20, 20, 50)
dd.preTransform = [0] * 50
ax4 = fig.add_subplot(numC, numR, 5)
ax4.plot(x, dd.preTransform)
ax4.set_title('y = 0')

# plotting transformed signal
dd.freq = np.fft.fftshift(np.fft.fftfreq(np.size(x)))
ft = dd.get_fft('not a real var', snap='test')  # preTransform is already set
ax5 = fig.add_subplot(numC, numR, 6)
ax5.plot(ft['freq'], ft['ftCube'])
ax5.set_title('ft of y = 0')

plt.tight_layout()
plt.show()
```



1.2.4 FFT demo #2

Here, we use get_fft() to find the transformation result for bx at each z position (from a local network containing 2d simulations).

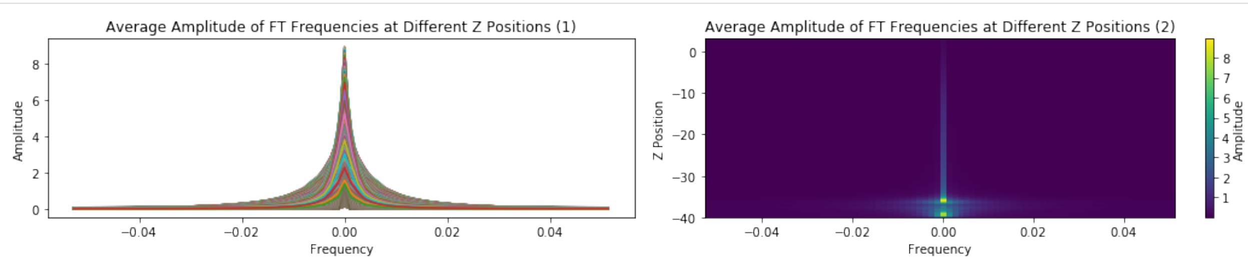
[22]:
```
import numpy as np
import helita.sim.cstagger
from helita.sim.bifrost import BifrostData, Rhoeetab, read_idl_ascii
from helita.sim.bifrost_fft import FFTData
import matplotlib.pyplot as plt

snaps = np.arange(280, 360)
v = 'bx'

dd = FFTData(file_root='l2d90x40r_it',
             fdir='/net/opal/Volumes/Amnesia/mpi3drun/2Druns/genohm/rain/l2d90x40r/')

# getting ft
transformed = dd.get_fft(v, snaps)
ft = transformed['ftCube']
freq = transformed['freq']
zaxis = dd.z

# making empty array to later contain the avergaes for each z position
zstack = np.empty([np.size(freq), np.shape(ft)[1]])
# filling ztack with average ft for each (x,y) in each z level
```

(continues on next page)

(continued from previous page)

```
for k in range(0, np.shape(ft)[1]):
    avg = np.average(ft[:, k, :], axis=(0))
    zstack[:, k] = avg

# preparing plots
fig = plt.figure(figsize = (15, 3))
numC = 1
numR = 2

# ploting freq vs amp with multiple lines (1 for each z position)
ax0 = fig.add_subplot(numC, numR, 1)
ax0.plot(freq, zstack)
ax0.set_xlabel('Frequency')
ax0.set_ylabel('Amplitude')
ax0.set_title(
    'Average Amplitude of FT Frequencies at Different Z Positions (1)')
ax0.set_aspect('auto')

# plotting amp at different freq & z with image
ax1 = fig.add_subplot(numC, numR, 2)
im1 = ax1.imshow(zstack.transpose(), extent=[freq[0], freq[-1], zaxis[0], zaxis[-1]])
ax1.set_xlabel('Frequency')
ax1.set_ylabel('Z Position')
ax1.set_title(
    'Average Amplitude of FT Frequencies at Different Z Positions (2)')
ax1.set_aspect('auto')
c1 = fig.colorbar(im1, ax = ax1)
c1.set_label('Amplitude')
plt.tight_layout()
plt.show()
```

```
WARNING: cstagger use has been turned off, turn it back on with "dd.cstagop = True"
```



CHAPTER 2

br_uvotrt

This library is a superclass of Helita. Helita documentation can be found http://helita.readthedocs.io/en/latest/index.html

2.1 BifrostData class

bifrost.py includes the BifrostData class (among others) which is needed for br_uvotrt.

This library has two main sub-libraries: bifrost_uvotrt and br_dem

2.2 UVOTRTData class

This class can be found within bifrost_uvotrt.py. It performs operations on Bifrost simulation data in its native format. After creating a class for a specific snap root name and directory (much like with BifrostData), one can get, intensity, spectral profiles, VDEM cubes and other useful analysis on synthetic spectral for a range of snapshots.

In order to create VDEM cubes using cuda this UVOTRTData depends on br_dem library.

[ ]:
```
from br_uvotrt import bifrost_uvotrt as br_uvt

# loading the class
brv=br_uvt.UVOTRTData('en024031_emer3.0str',snap=260)

# saving a VDEM cube using cuda code and saving the data in an npz file
brv.vdem_cuda(save_vdem='test',tg_axis=np.linspace(4.7,7.5,15),vel_axis=np.linspace(-
↪40,40,41),zcut=0.0)

# calculating spectral prifles using the cuda code (depends on br_cuda and br_intcu)
synprof =  brv.get_intny('fe_8_108.073')
```

CHAPTER 3

br_topo

This library is a superclass of Helita. Helita documentation can be found http://helita.readthedocs.io/en/latest/index.html

3.1 BifrostData class

bifrost.py includes the BifrostData class (among others) which is needed for br_topo.

This library has one main sub-libraries: bifrost_topology

3.2 TopologyData class

This class can be found within bifrost_topology.py. It performs magnetic field topoloty operations on Bifrost simulation data in its native format. After creating a class for a specific snap root name and directory (much like with BifrostData), one can get factor q or integrations along magnetic field lines for a range of snapshots.

In order to get factor q:

[ ]:
```
from br_topo import bifrost_topology as bt

# loading the class
brv=bt.TopologyData('en024031_emer3.0str',snap=260)

# calculating factor q (depends on br_topocu)
var=brv.get_topology('qfac')
```