MCEN 7221/ASEN 5027 Turbulence Project 2

Duncan McGough

May 10, 2019

This project can be found at: `https://github.com/duncanam/5037project`

Contents

List of Figures

1 Problem 1

In this project report, Direct Numerical Simulation (DNS) data was taken and analyzed. Two different kinds of simulations were analyzed, one being Homogeneous Isotropic Turbulence (HIT) and the other being Homogeneous Shear Turbulence (HST).

1.1 Max and Min Values

The DNS data for HIT and HST cases was put into a min/max function to find the minimum and maximum values for each velocity component in the HIT and HST fields. Noting that the data is nondimensionalized, the results are:

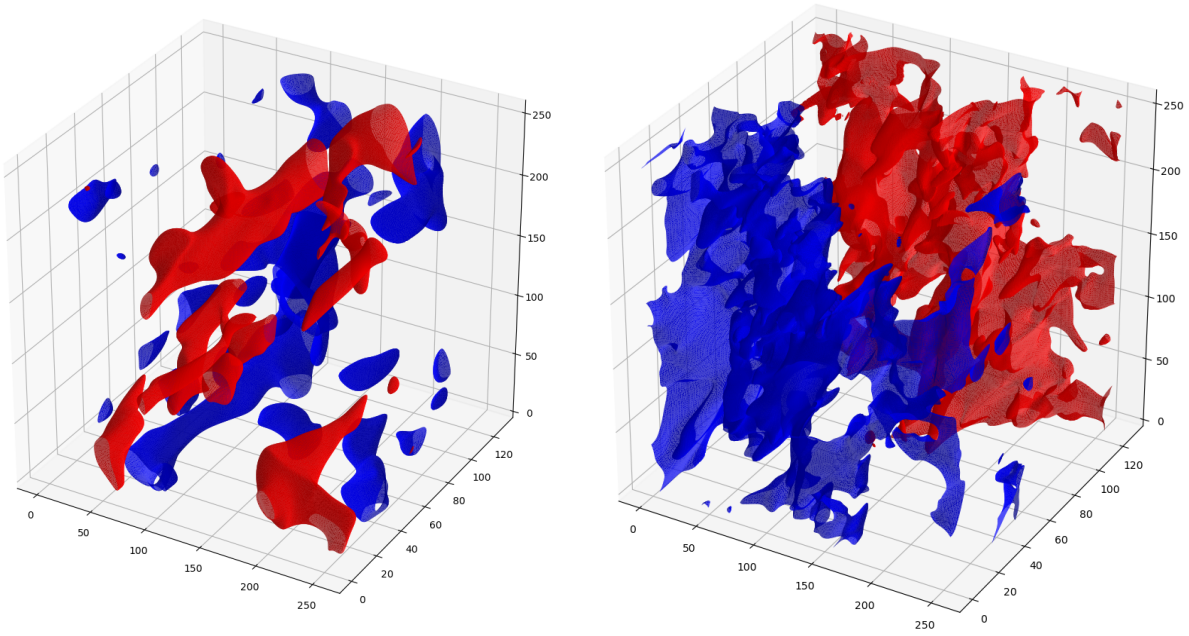
```
Max HIT u: 0.8576899766921997
Max HIT v: 0.5416799783706665
Max HIT w: 0.7064499855041504
Max HST u: 5.862400054931641
Max HST v: 3.033799886703491
Max HST w: 3.131500005722046

Min HIT u: -0.7590799927711487
Min HIT v: -0.6005899906158447
Min HIT w: -0.7692000269889832
Min HST u: -5.138700008392334
Min HST v: -3.3306000232696533
Min HST w: -2.963900089263916
```

One can notice that the extreme values are in the HST case are of greater magnitude than those of the HIT case. In the HIT case, the values are generally smaller and closer together in magnitude, whereas in the HST case it is evident that the u-component has a standout larger velocity whereas the v- and w-components are similar in magnitude. This makes sense due to the nature of the simulation, as the shear has a certain imposed velocity (at least initially) that drives the flow.

1.2 3D Isosurfaces

Three-dimensional isosurfaces can be created and plotted for the HIT and HST cases. When visualized, they appear as in Figure 1.

The two colors in Figure 1 represent the maximum and minimum velocities divided by two, respective to the colors blue and red. It is immediately noticeable that the HST case has a distinct division between the maximum and minimum isosurfaces, whereas the HIT case is more uniform. Another distinct feature between the cases is that HIT case has more frequent separate blob entities in a homogeneous formation of isosurfaces whereas the HST case are distinct in their shear layers.



(a) HIT

(b) HST

Figure 1: 3D Isosurfaces

1.3 Image Plane Slices

The data can be sliced at certain planes, leaving 2D slices that can be plotted. In Figures 2 and 3, the u, v, and w velocity component fields are sliced at k=[1, 128, 256] z-locations. Note that colorbar here is scaled to HST for easy comparison. Increasing slice locations are located in the columns of images whereas different velocity components are the rows of images. It is noted that u-component is the most active and polar with the velocity field in the HST data, as expected. The HIT data is more uniform and homogeneous. An interesting feature of the data is that the k=1 and k=256 planes appear identical. This is due to a repeating boundary condition that is applied to the domain of the simulation.

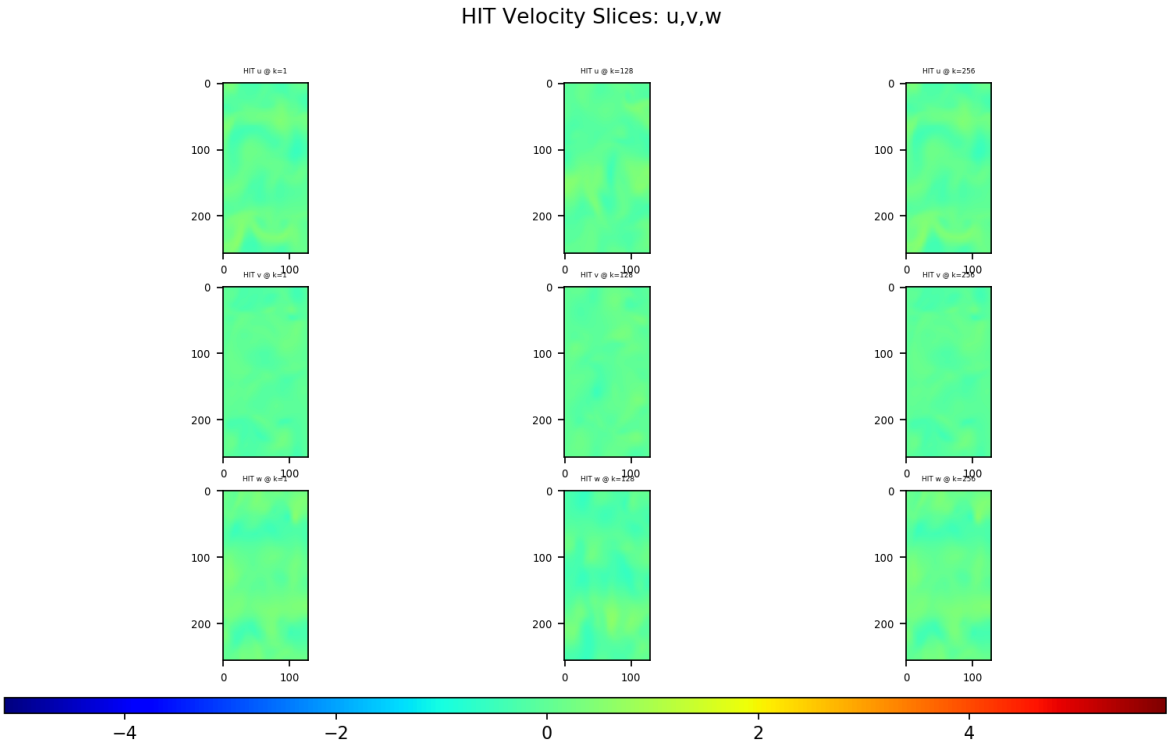


Figure 2: HIT Velocity Component Slices

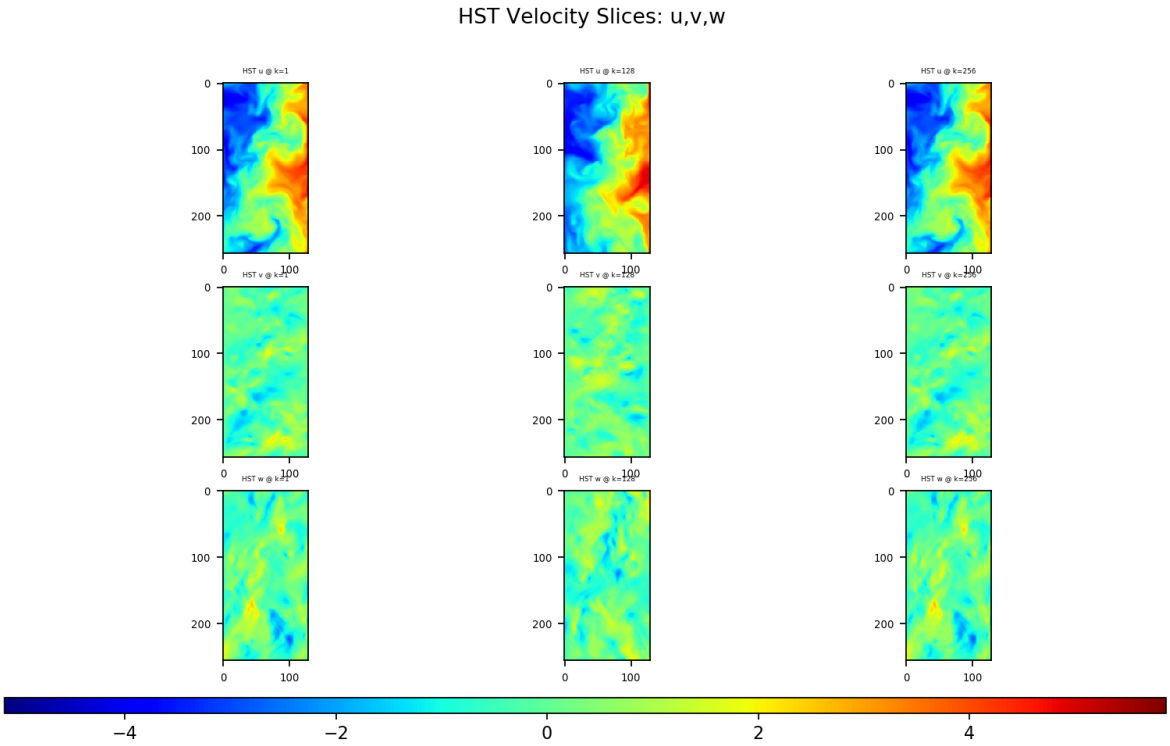


Figure 3: HST Velocity Component Slices

1.4 Spatial Directions

For the HIT case, one can see from Figure 2 that the fields are spatially homogeneous and the statistics for these fields are expected to be invariant in all directions.

For the HST case, one sees from Figure 3 that some fields are homogeneous and some are not. The field has strong variation in the y-direction, across the thin axis of the slice. This is due to the shear, where one would expect to see the two opposing layers of velocity. The other directions x and z are homogeneous and their statistics are expected to be invariant.

1.5 x-z Averages

For both the HIT and HST data, x-y averages can be calculated using:

$$\langle u \rangle_{xz}(j) = \frac{1}{N_x N_z} \sum_{k=1}^{N_z} \sum_{i=1}^{N_x} u(i,j,k)$$

These values can then be plotted, and seen in Figure 4. It is interesting to see that the fluctuations in the HIT case are more or less "random", whereas there is an incredibly strong linear behavior in the HST x-z average of u. This is displaying the shear that is occurring in the domain. Another interesting feature is that the u and w components in the HIT x-z averages appear to be mirrored across some positive-valued line in velocity. Lastly, the v-components of both HIT and HST are quite small, and the w-component in HST is also quite small. The w-component of HST almost has a sinusoidal shape, with the change in concavity being at the location where the x-y u-average changes sign. This could indicate that u-velocity shear is driving flow in the w-direction.

1.6 HST Fluctuating Velocity

The fluctuating velocity can be computed for the HST case utilizing:

$$u_i'(x,y,z) = u(x,y,z) - \langle u_i \rangle_{xz}(y)$$

Slices can be taken at k=[1, 128, 256] to create two-dimensional visualizations, seen in Figure 5. Something really interesting happens when the fluctuating HST velocities are compared with the standard slice velocities in Figure 3. The v and w components have similar structures in both normal slices and the fluctuation slices, but the u components are significantly different in structure.

1.7 HIT Fluctuating Velocity

The fluctuating velocity can be computed for the HIT case utilizing:

$$u_i'(x,y,z) = u(x,y,z) - \langle u_i \rangle_{xyz}$$

Note that there is a new average definition here that encompasses the entire volume:

$$\langle u \rangle_{xyz} = \frac{1}{N_x N_y N_z} \sum_{k=1}^{N_z} \sum_{j=1}^{N_y} \sum_{i=1}^{N_x} u(i,j,k)$$

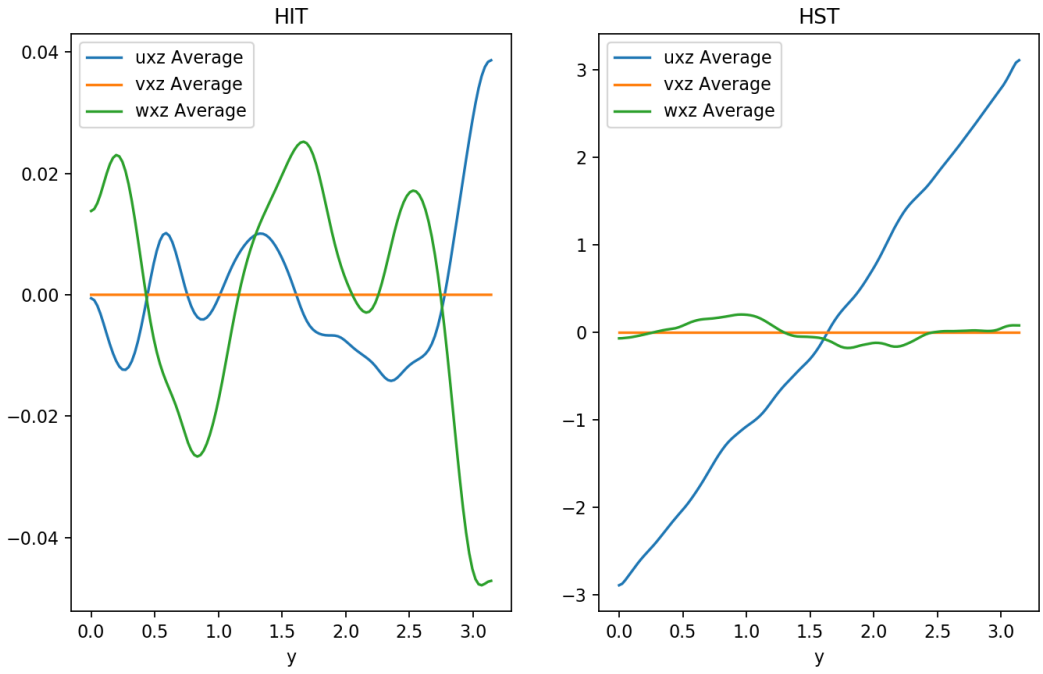


Figure 4: HIT and HST $u_{i,xz}$ Averages

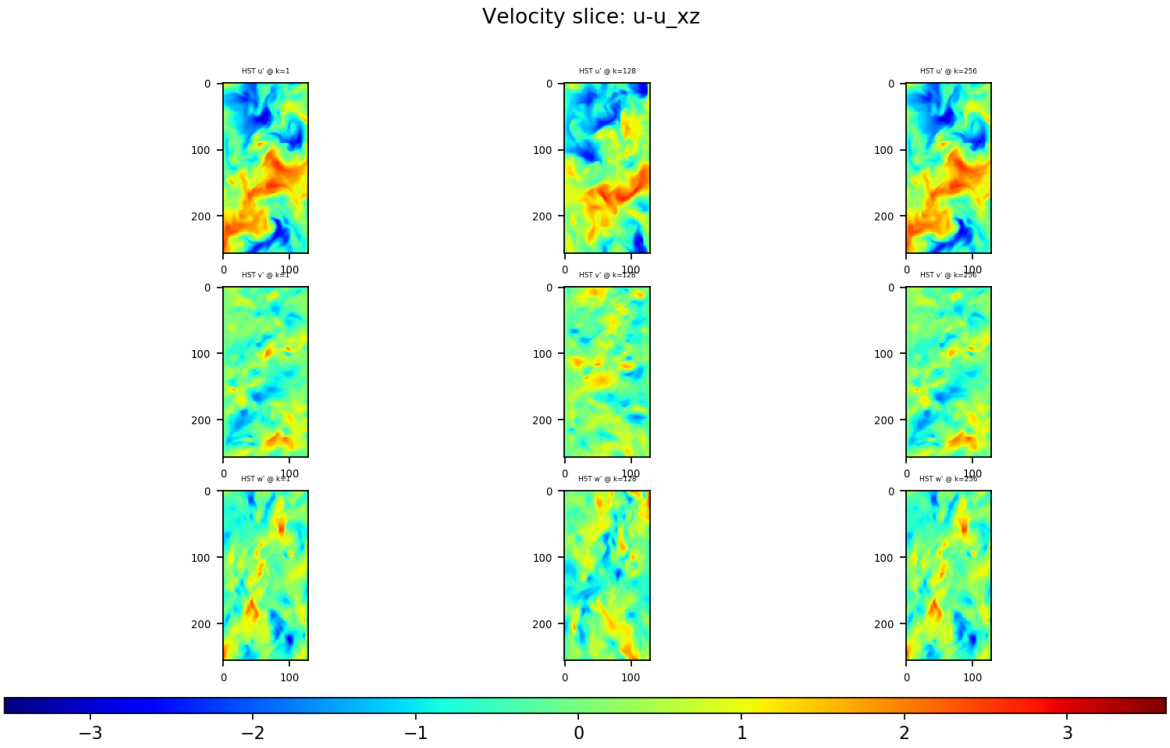


Figure 5: HST Fluctuating Velocity Slices

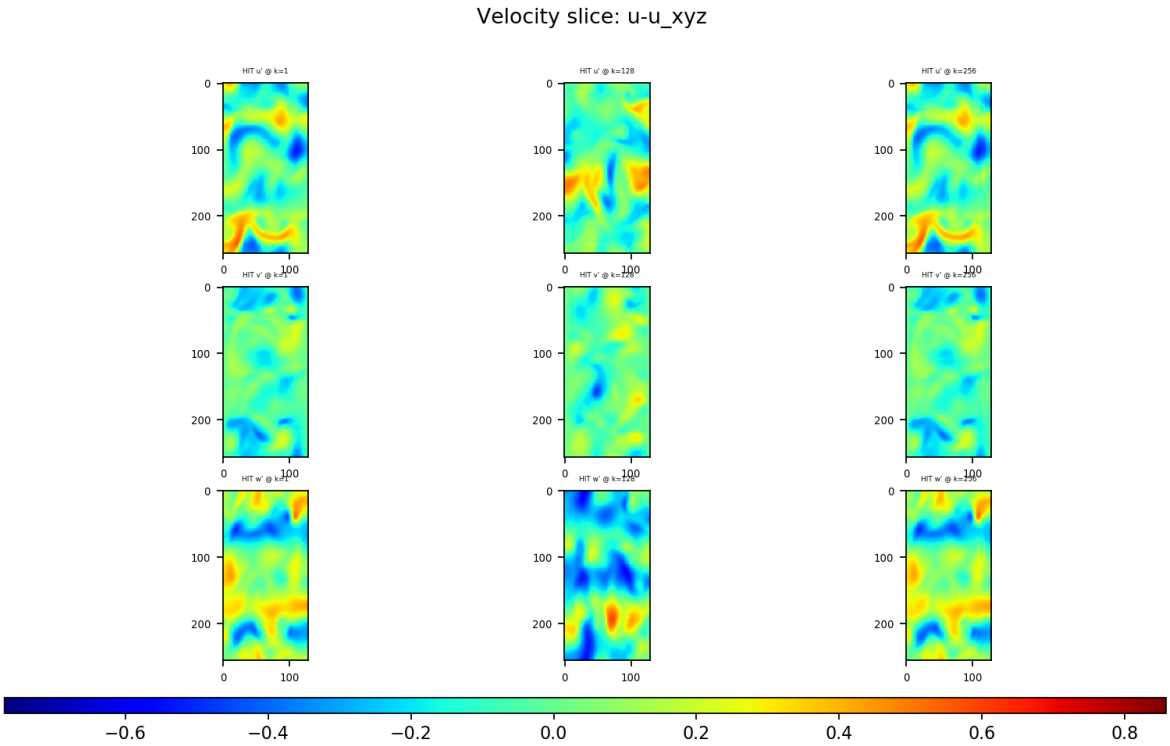


Figure 6: HIT Fluctuating Velocity Slices

When this is done, it obtains the figure seen in Figure 6. When compared to Figure 2 one can see that the structure and magnitude of the fields themselves don't change all that much. This is likely because the average itself is fairly small due to the nature of the HIT field and therefore the average won't change the fluctuating field significantly.

1.8 Reynolds Stresses

Now one can calculate the HST Reynolds stresses which are defined as $\langle u_i' u_j' \rangle_{xz}(y)$ since the fluctuating velocities were previously calculated. Additionally, a comparison can be done with the HIT full-volume Reynolds stresses, where instead the full volume average is utilized such that the Reynolds stresses take the form $\langle u_i' u_j' \rangle_{xyz}$. When plotted against each other, the result takes the form in Figure 7.

Since the average definitions are different, it can be seen that the HST data has fluctuating Reynolds stresses and the HIT data has constant Reynolds stresses. For some components, the HIT Reynolds stress is significantly lower than the HST, such as in in $u'u'$ and $w'w'$. Others, such as $v'u'$ (and it's identical twin as the matrix is symmetric) have the HIT Reynolds stress as the peak value. It is noted that the diagonal components tend to have lower Reynolds stresses whereas the off-diagonal components besides $w'u'$ and $u'w'$ have higher Reynolds stresses for HIT when compared to HST. This makes sense, as stresses should be higher in directions of opposing velocity. Recall that it was mentioned that in the HST case, statistics should be homogeneous except in the y-direction. One can see that in the $w'u'$ and $u'w'$ cases as well as the diagonal the effects of the shear layer are present.

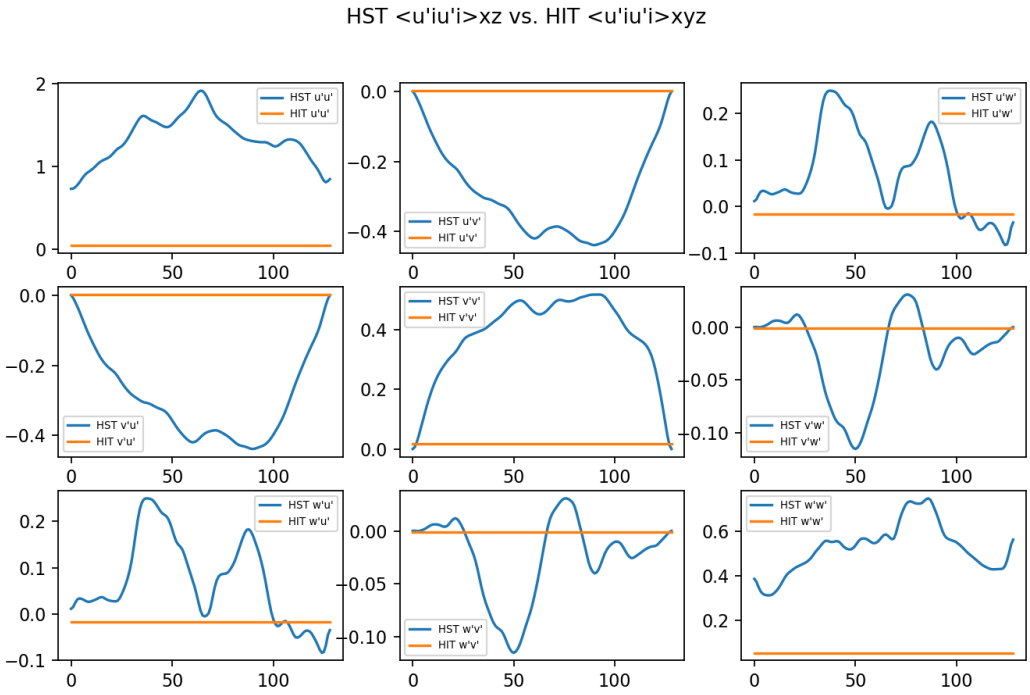


Figure 7: HIT and HST Reynolds Stresses

1.9 Moments, Skewness, and Kurtosis of u'

The 2nd moment of u' was calculated using the appropriately-defined mean of the nth-moment's nth power of u' for each data set, either HIT or HST. Then σ was calculated for each set and skewness was obtained by dividing the third moment by the cube of σ and kurtosis was calculated by dividing the fourth moment by σ^4. The skewness and kurtosis was then plotted for HST and HIT and is seen in Figure 8, where S represents skewness and K represents kurtosis. In terms of a Gaussian plot, changing the skewness will alter the "tails" of the PDF plot by changing the lengths of these tails and how far the PDF plot is skewed to one side or another. The larger the skewness, the more the PDF will be "leaning" to one side, with positive and negative skewness denoting direction. The kurtosis gives effective thickness of these tails, with high kurtosis giving a much spikier, thin PDF whereas a low kurtosis gives a smoother PDF with a rounder shape. The HIT data has relatively low skewness and higher kurtosis, so this means that the resulting PDF won't be leaning much but will be thin. The HST data ranges across many values. Values on the edges should be skewed with higher kurtosis whereas values towards the middle of the data should be more Gaussian. This will be explored next.

1.10 HST and HIT PDFs

The probability density functions (PDFs) for the HIT and HST data are plotted in Figures 9 and 10. For the HST case, the u-velocity components have slightly less kurtosis that the rest of the velocity components. One can also note that the HST v-velocity components

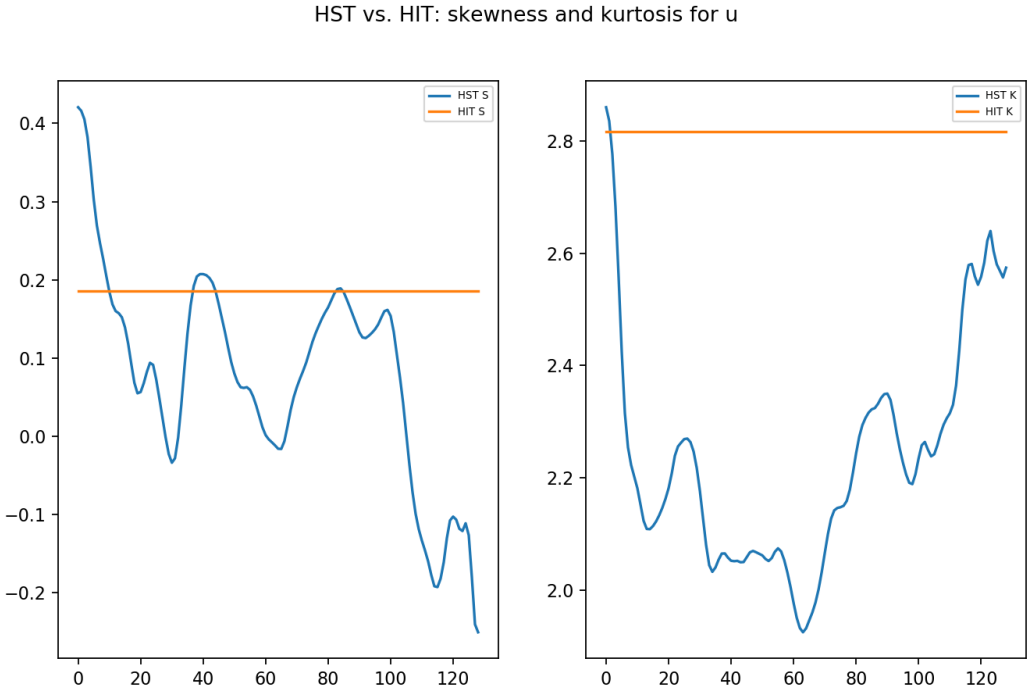


Figure 8: Skewness and Kurtosis for HST and HIT

have some noticeable skew. The HIT data shows that the u and w components have very Gaussian distributions whereas the v component has considerably more kurtosis. This shows that the v-component field in HIT has some bias towards a very consistent velocity, or likely stillness in the HIT case. The HST PDFs show that there is more variation in velocity across the u-component due to the shear layer(s).

2 Problem 2

This next section will only consider the HIT data.

2.1 Velocity Gradient Tensor

The velocity gradient tensor,

$$A_{ij}(x, y, z) = \frac{\partial u_i(x, y, z)}{\partial x_j},$$

can be calculated for the HIT data for each point in the volume. When volume-averaged, the result is:

$$\langle A_{ij} \rangle_{xyz} = \begin{bmatrix} 2.74741721e(-08) & 1.24782263e(-02) & 7.31944450e(-06) \\ 5.15238234e(-06) & -3.22052239e(-11) & 1.00500736e(-06) \\ -1.06929170e(-05) & -1.93251442e(-02) & -1.42071821e(-08) \end{bmatrix}$$

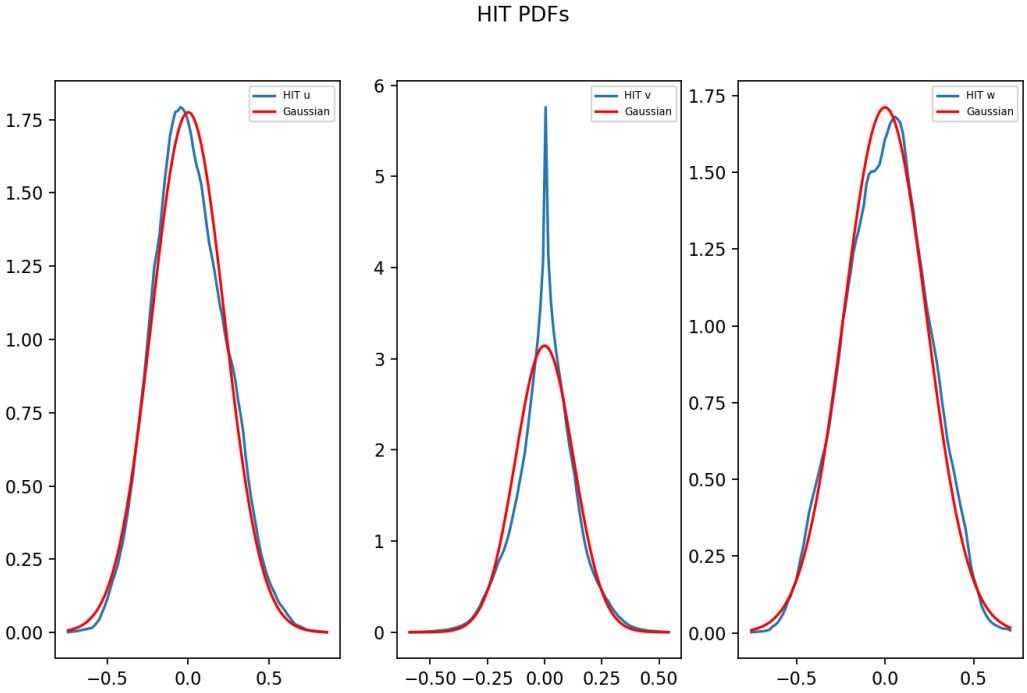


Figure 9: HIT Data Slice PDFs

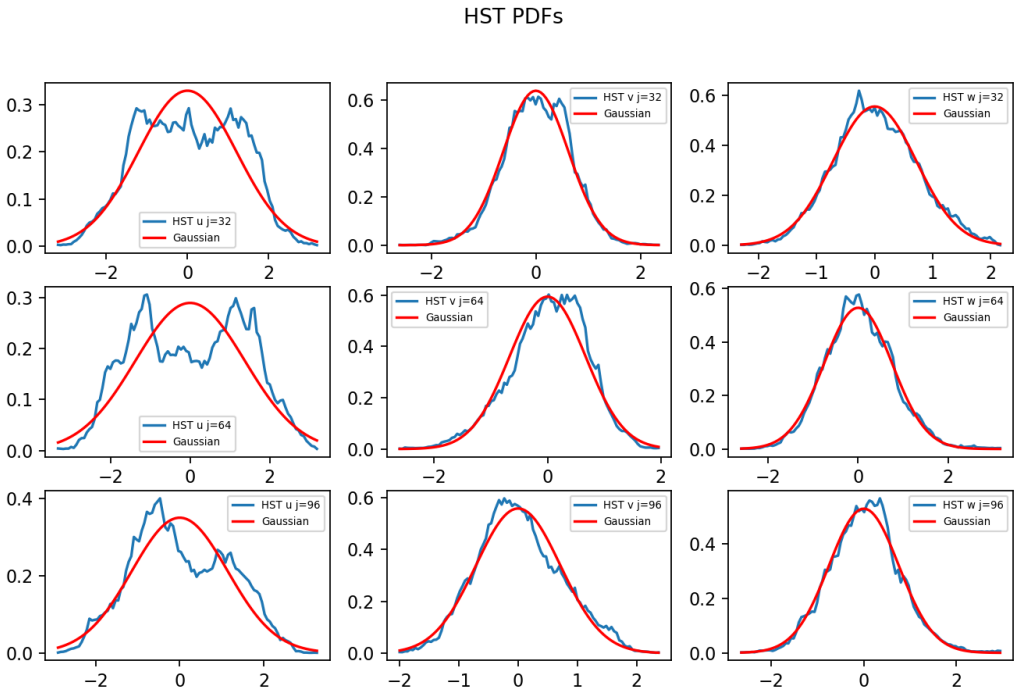


Figure 10: HST Data Slice PDFs

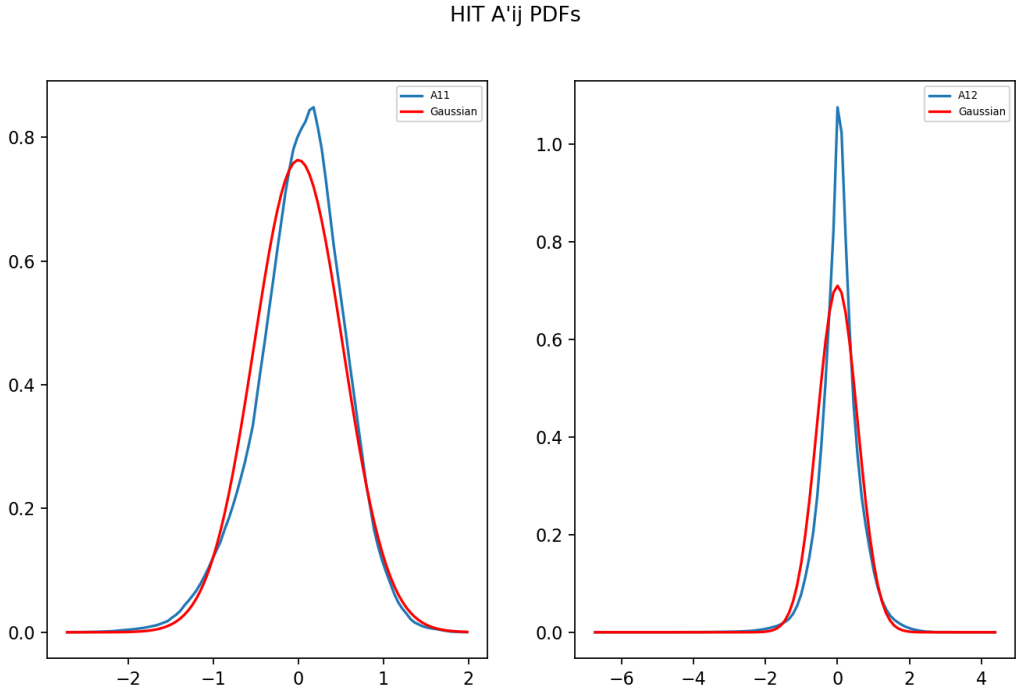


Figure 11: HIT A'_{ij} PDFs for A'_{11} and A'_{12}

Considering HIT, this result seems to make sense as all gradients should be small due to the isotropy.

2.2 Fluctuating Velocity Gradient Tensor Field

The fluctuating velocity gradient tensor field,

$$A'_{ij}(x, y, z) = A_{ij}(x, y, z) - \langle A_{ij} \rangle_{xyz},$$

can be computed and the PDFs of the components A'_{11} and A'_{12} are plotted and seen in Figure 11.

The two PDFs are fairly different in shape. A'_{12} has significantly more kurtosis than A'_{11}. A'_{11} has similar shape to Gaussian, with close agreement towards the tails and rising edges. A'_{12} also agrees with Gaussian near the tails and rising edges but drastically overshoots the peak. One can see a similarity to the full volume HIT PDFs in Figure 9 for the v-component in velocity and A'_{12} with the strong central peak. The slight stair-step near the peak seen in the w-component velocity HIT PDF lingers in the A'_{11} PDF.

2.3 A'_{11} Skewness and Kurtosis

The skewness and kurtosis for A'_{11} can be computed with:

$$S = \frac{\langle A'_{11} \rangle^3_{xyz}}{\sigma^3}$$

and

$$K = \frac{\langle A'_{11} \rangle^4_{xyz}}{\sigma^4}.$$

When this is done to the HIT data, it results in:

$$A'_{11}\ S = -0.3915937503182623$$

$$A'_{11}\ K = 3.560396769244712$$

These values can be compared to a Gaussian PDF, which has a skewness of 0 and a kurtosis of 3. This means that A'_{11} is fairly close to being Gaussian, but not perfect. This is confirmed by Figure 11, where it can be seen that the plot is slightly skewed to the right and slightly taller than Gaussian.

This can be extrapolated to form the basis that turbulent flows can be modeled approximately with Gaussian distributions. This is not a perfect approximation as seen in this example, but often it is "close enough".

2.4 Fluctuating Strain Rate and Pseudo Energy Dissipation

The fluctuating strain rate is defined as

$$S'_{ij}(x, y, z) = \frac{1}{2}\left[A'_{ij}(x, y, z) + A'_{ji}(x, y, z)\right],$$

and is defined everywhere inside the volume. This can be calculated and used to define the pseudo energy dissipation rate

$$\frac{\varepsilon}{\nu} = 2S'_{ij}S'_{ij}.$$

Notice that the repeated indices here indicate summation. This leads to the calculation of the time-averaged value of this quantity, which is denoted as $\frac{\langle \varepsilon \rangle_{xyz}}{\varepsilon}$. For the HIT data, this is:

$$\frac{\langle \varepsilon \rangle_{xyz}}{\varepsilon} = 2.861789663199326$$

2.5 2D $\frac{\langle \varepsilon \rangle_{xyz}}{\varepsilon}$ Plot

$\frac{\langle \varepsilon \rangle_{xyz}}{\varepsilon}$ can be sliced and plotted using the slice index $k = 128$. The result is seen in Figure 12. Comparing to Figure 6, it is noticed that some areas of low velocity are high-intensity regions of the pseudo energy dissipation rate field. Some structures are seen in both fields, but some are not.

2.6 Fluctuating Vorticity and Enstrophy

The fluctuating vorticity is calculated using

$$\omega'_i = (x, y, z) = \varepsilon_{ijk} A'_{kj},$$

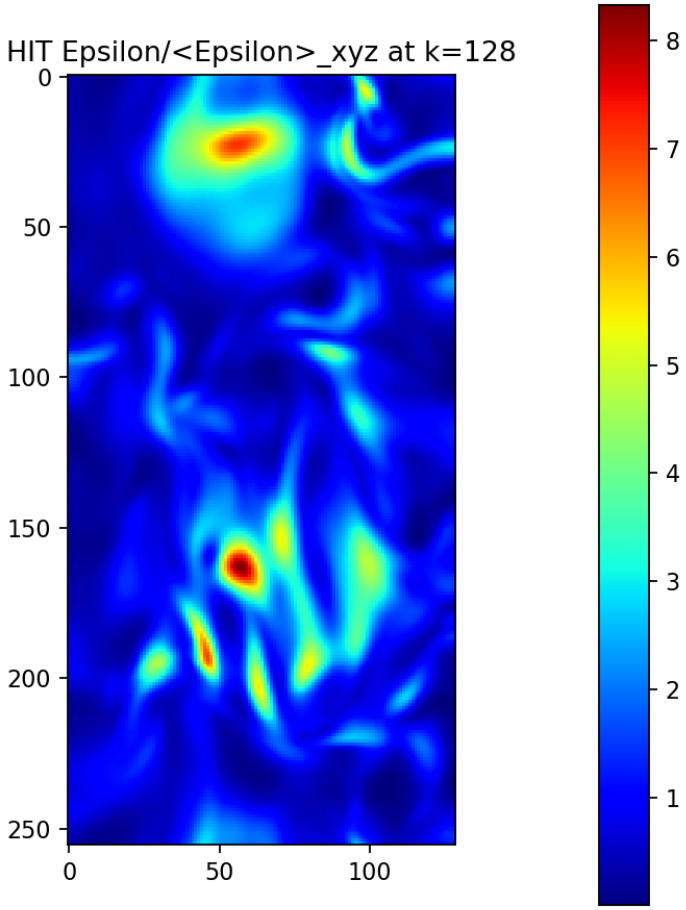


Figure 12: $\frac{\langle \varepsilon \rangle_{xyz}}{\varepsilon}$ Plotted at $k = 128$

where ε_{ijk} is the Levi-Civita or alternating tensor. The enstrophy can be calculated from this using

$$\Omega = \frac{1}{2}\omega_i'\omega_i'.$$

When this quantity is time-averaged across the entire domain, the HIT data gives a result of:

$$\langle\Omega\rangle_{xyz} = 2.1634319241507$$

2.7 Relationships

It appears that $\varepsilon \sim \Omega^2$. This can be seen in Hamlington and Dahm Figure 4.32. The value found for enstrophy in this problem does not agree with this relation. This could be due to numerical and calculation error.

2.8 Normalized Enstrophy Plot

Enstrophy is normalized as $\frac{\Omega}{\langle\Omega\rangle_{xyz}}$ and plotted at the $k = 128$ plane. This is seen in Figure 13. When compared to Figurefig:17, some similar structure can be seen, with a small spot of negative velocity contributing to the enstrophy. It is also noted that other isolated regions have increased enstrophy. Compared to Figure 12, two dominant high-enstrophy structures remain in the middle near the top and near the bottom third between the two plots. Generic structure also remains.

2.9 PDFs

The PDFs of the normalized pseudo energy dissipation rate and enstrophy can be plotted for the entire domain. These are seen in Figure 14. The PDFs are both log-normal and fairly small due to the increased multiplicative nature of the quantities that generated them. The profiles are similar, and one can see the similarity between the dissipation and enstrophy fields that were plotted in previous sections. The slightly increased span of the enstrophy PDF may indicate the increased number of distinct structures in the enstrophy field.

3 Problem 3

3.1 Autocorrelation and Scales

For HIT, the autocorrelation of u' is

$$\rho(r) = \frac{\langle u'(x,y,z)u'(x+r,y,z)\rangle_{xyz}}{\langle(u')^2\rangle_{xyz}}.$$

From this autocorrelation, the spatial integral scale is created as

$$\Lambda = \int_0^\infty \rho(r)dr.$$

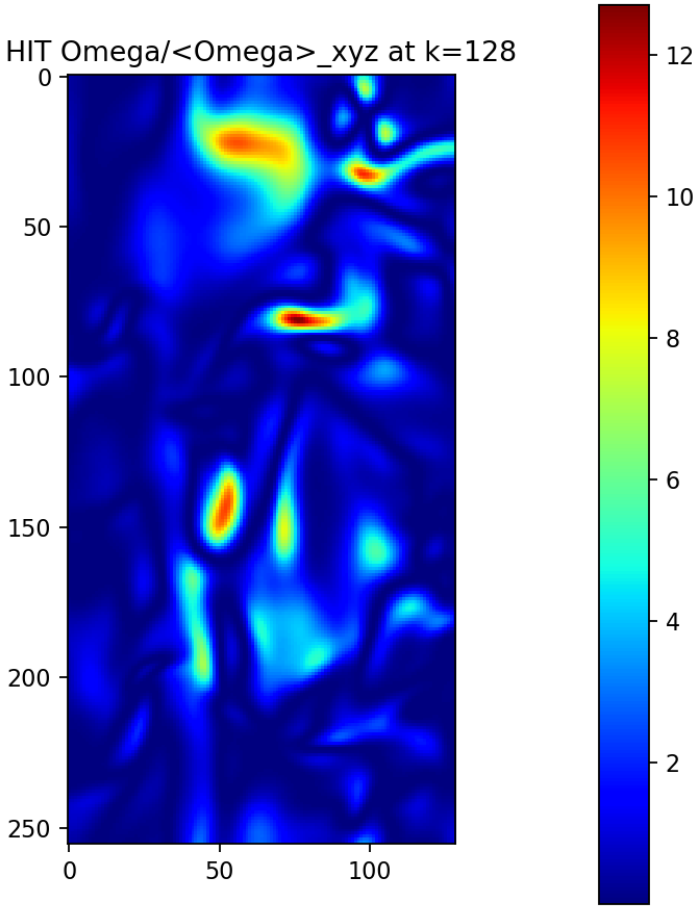


Figure 13: Enstrophy 2D Slice at $k = 128$

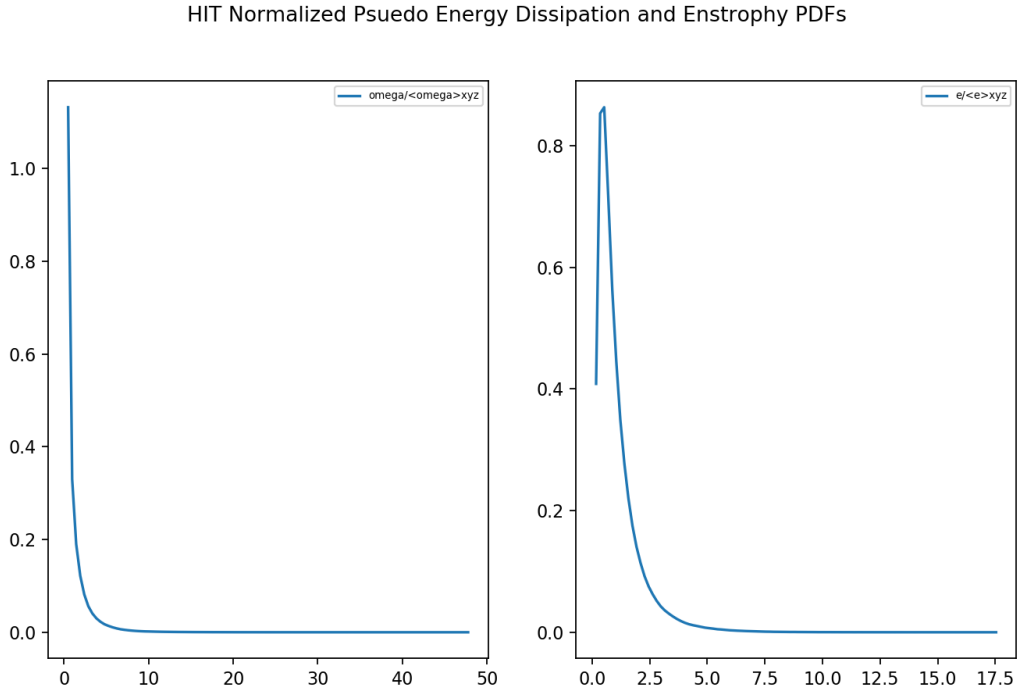


Figure 14: PDFs of Normalized Pseudo Energy Dissipation Rate and Enstrophy

For the HIT data, this is:

$$\Lambda = 10.537147171140688$$

Next the spatial Taylor scale is said to be

$$\lambda^2 = -2\left(\frac{d^2\rho}{dr^2}|_{r=0}\right)^{-1}$$

When the HIT data is considered, it results in:

$$\lambda^2 = 675.9795656051165$$

The resulting autocorrelation $\rho(r)$ is compared with exponential and Gaussian forms in Figure 15. Looking at this figure, the Reynolds number can be assumed to be large since the autocorrelation fluctuates so much. Since the autocorrelation describes how well a quantity is correlated with itself, large values and fluctuations in values indicate that the fluctuating velocity here is not correlated well with itself and is changing frequently, indicating high Reynolds number turbulent flow.

3.2 Joint PDF

The joint PDF of the normalized pseudo energy dissipation rate and enstrophy field can be calculated using:

$$I(\varepsilon/\langle\varepsilon\rangle_{xyz}, \Omega/\langle\Omega\rangle_{xyz}) = \frac{\beta(\Omega/\langle\Omega\rangle_{xyz}, \varepsilon/\langle\varepsilon\rangle_{xyz})}{\beta(\varepsilon/\langle\varepsilon\rangle_{xyz})\beta(\Omega/\langle\Omega\rangle_{xyz})}$$

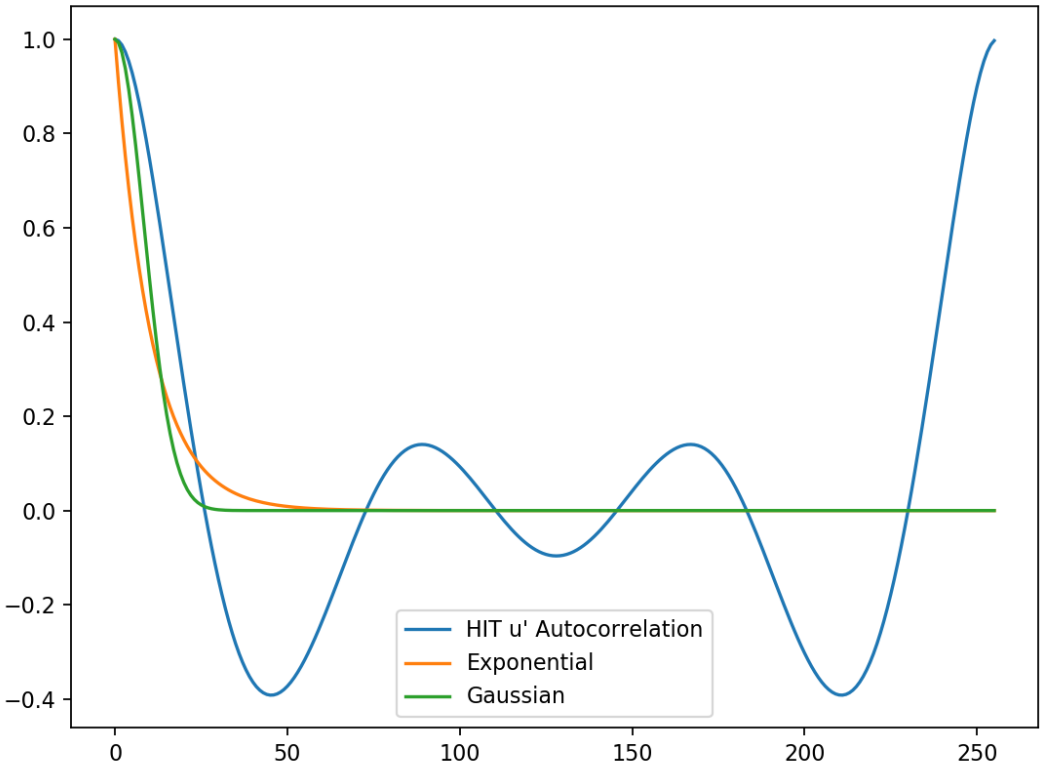


Figure 15: Autocorrelation Comparison

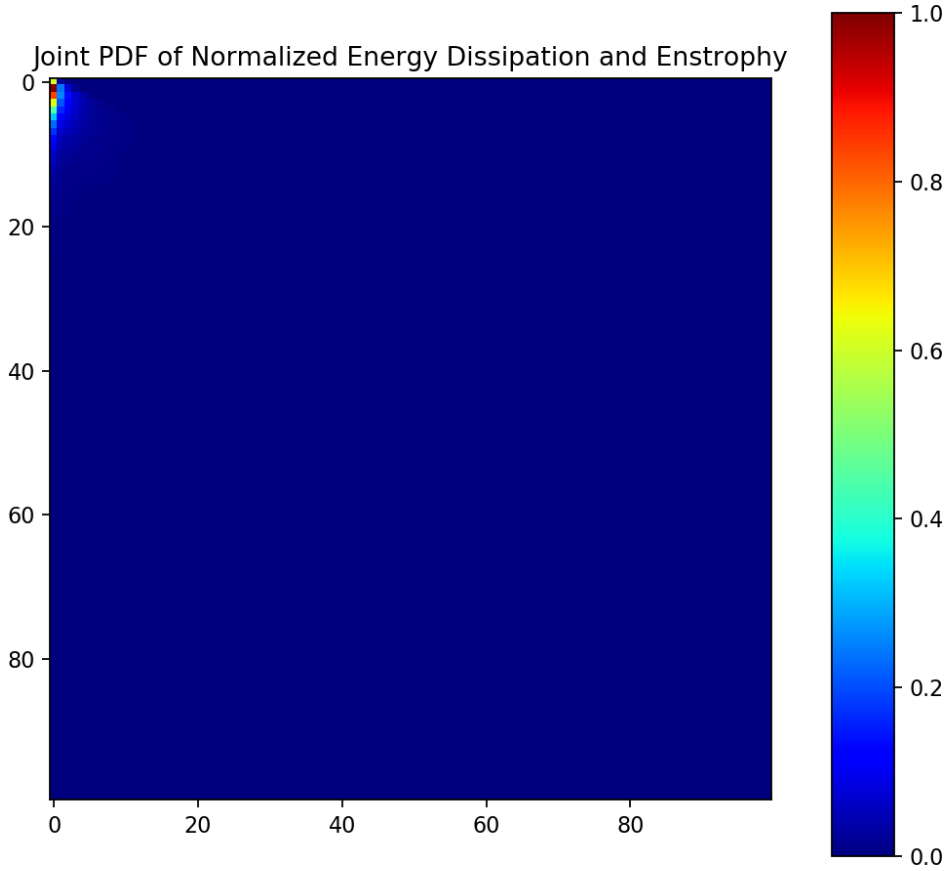


Figure 16: Joint PDF of Normalized Pseudo Energy Dissipation Rate and Enstrophy Field

When this is calculated and plotted, it results in Figure 16. Around the (0,0) coordinate, the PDF is at it's peak. This sharply falls off and backs down. This means that at both small values of energy dissipation and enstrophy the quantities are well correlated. In relation to the Burger's vortex, the vorticity is concentrated in a column, so here one can see that the correlation is also extremely concentrated in one region.

3.3 Agreement with Kolmogorov

The kinetic energy spectrum is described by

$$E(k) = k^3 \frac{d}{dk} \left(\frac{1}{k} \frac{d}{dk} E_f(k) \right).$$

Since the turbulence is homogeneous and isotropic and our previous calculated autocorrelation and PDFs have been consistent with HIT assumptions, we can say that the data agrees with Kolmogorov scaling.

A Code: Problem 1

```
#########################################################################
#     ____                 _                 _      ____    ___   _
#    |  _ \ _ __ ___ (_) ___   ___| |_  |___ \ / _ \/ |
#    | |_) | '__/ _ \| |/ _ \ / __| __|   __) | | | | | |
#    |  __/| | | (_) | |  __/ (__| |_   / __/| |_| | | |
#    |_|   |_|  \___// |\___|\___|\__| |_____|\__\_\_|
#                  |__/
#
# Purpose: Main script for ASEN 5037 Project 2 Question 1
#
# Note: for best results, use an iPython interpreter.
#
# Author: Duncan McGough
#
# Date Created:      4/11/19
# Date Edited:       5/7/19
#
#########################################################################
# IMPORT PACKAGES
import numpy as np
import matplotlib.pyplot as plt
#import cProfile
#from numba import jit
from scipy.stats import norm
from mpl_toolkits.mplot3d import Axes3D
from skimage.measure import marching_cubes_lewiner as mcl
import sys
sys.path.insert(0, './generic/') # path to generic functions

# Import generic functions:
from read_data import read_dat
from importdata import dataimport

#########################################################################
# PRINT WELCOME MESSAGE
print('\n'*100)
print('-'*60)
print('TURBULENCE PROJECT 2 Q1 SCRIPT')
print('Author: Duncan McGough')
print('-'*60)
print('\n')

#########################################################################
# DEFINE VARIABLES
pi = np.pi
nx = np.array([256,129,256])
lx = np.array([2*pi, pi, 2*pi])
dx = lx/nx
datafolder = './project2data/'
filenames = [datafolder+'HIT_u.bin', datafolder+'HIT_v.bin',
             datafolder+'HIT_w.bin', datafolder+'HST_u.bin',
```

```
            datafolder+'HST_v.bin', datafolder+'HST_w.bin']
uvw = ['u','v','w']

#########################################################################
# IMPORT THE DATA
[hit_u, hit_v, hit_w,
 hst_u, hst_v, hst_w] = dataimport(filenames, nx)

#########################################################################
# FIND MAX AND MIN VALUES FOR EACH COMPONENT AND PRINT THEM [1.1]
print('-'*60)
print('MIN AND MAX VALUES:\n')

max_hit_u = np.max(hit_u)
max_hit_v = np.max(hit_v)
max_hit_w = np.max(hit_w)
max_hst_u = np.max(hst_u)
max_hst_v = np.max(hst_v)
max_hst_w = np.max(hst_w)

print('Max HIT u:', max_hit_u)
print('Max HIT v:', max_hit_v)
print('Max HIT w:', max_hit_w)
print('Max HST u:', max_hst_u)
print('Max HST v:', max_hst_v)
print('Max HST w:', max_hst_w)
print('')

min_hit_u = np.min(hit_u)
min_hit_v = np.min(hit_v)
min_hit_w = np.min(hit_w)
min_hst_u = np.min(hst_u)
min_hst_v = np.min(hst_v)
min_hst_w = np.min(hst_w)

print('Min HIT u:', min_hit_u)
print('Min HIT v:', min_hit_v)
print('Min HIT w:', min_hit_w)
print('Min HST u:', min_hst_u)
print('Min HST v:', min_hst_v)
print('Min HST w:', min_hst_w)
print('')

#########################################################################
# PLOT ISOSURFACES OF TURBULENCE DATA [1.2]
### Do HIT:
## Calculate triangular-based isosurfaces:
#verts1, faces1, _, _ = mcl(hit_u, max_hit_u/2)
#verts2, faces2, _, _ = mcl(hit_u, min_hit_u/2)
#
## Start a new figure and 3D subplot:
#fig1 = plt.figure(figsize=(12,12))
#ax1 = fig1.add_subplot(111,projection='3d')
#
```

```
## Plot the surfaces (LARGE CALCULATION):
#ax1.plot_trisurf(verts1[:, 0], verts1[:,1], faces1,
#        verts1[:, 2], color=(1,0,0,1), lw=1)
#ax1.plot_trisurf(verts2[:, 0], verts2[:,1], faces2,
#        verts2[:, 2], color=(0,0,1,1), lw=1)
#
### Now do HST:
## Calculate triangular-based isosurfaces:
#verts3, faces3, _, _ = mcl(hst_u, max_hst_u/2)
#verts4, faces4, _, _ = mcl(hst_u, min_hst_u/2)
#
## Start a new figure and 3D subplot:
#fig2 = plt.figure(figsize=(12,12))
#ax2 = fig2.add_subplot(111,projection='3d')
#
## Plot the surfaces (LARGE CALCULATION):
#ax2.plot_trisurf(verts3[:, 0], verts3[:,1], faces3,
#        verts3[:, 2], color=(1,0,0,1), lw=1)
#ax2.plot_trisurf(verts4[:, 0], verts4[:,1], faces4,
#        verts4[:, 2], color=(0,0,1,1), lw=1)

#########################################################################
# SLICE DATA AND PLOT AS IMAGE [1.3]

# Define the colormap max and mins:
slcmin = np.min([min_hit_u, min_hit_v, min_hit_w, min_hst_u,
                 min_hst_v, min_hst_w])
slcmax = np.max([max_hit_u, max_hit_v, max_hit_w, max_hst_u,
                 max_hst_v, max_hst_w])

slccmap = 'jet' # set the colormap
fsize = 4 # fontsize
tsize = 6 # ticksize

## Define the subplot figure:
hitslc, hitaxs = plt.subplots(3,3, figsize=(10,6), dpi=160)
hitslc.suptitle('HIT Velocity Slices: u,v,w')

# Plot HIT k=1:
hitaxs[0,0].imshow(hit_u[:,:,0], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hitaxs[0,0].set_title('HIT u @ k=1',fontsize=fsize)
hitaxs[1,0].imshow(hit_v[:,:,0], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hitaxs[1,0].set_title('HIT v @ k=1',fontsize=fsize)
hitaxs[2,0].imshow(hit_w[:,:,0], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hitaxs[2,0].set_title('HIT w @ k=1',fontsize=fsize)

# Plot HIT k=128:
hitaxs[0,1].imshow(hit_u[:,:,127], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hitaxs[0,1].set_title('HIT u @ k=128',fontsize=fsize)
hitaxs[1,1].imshow(hit_v[:,:,127], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hitaxs[1,1].set_title('HIT v @ k=128',fontsize=fsize)
hitaxs[2,1].imshow(hit_w[:,:,127], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hitaxs[2,1].set_title('HIT w @ k=128',fontsize=fsize)
```

```
# Plot HIT k=256:
hitaxs[0,2].imshow(hit_u[:,:,255], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hitaxs[0,2].set_title('HIT u @ k=256',fontsize=fsize)
hitaxs[1,2].imshow(hit_v[:,:,255], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hitaxs[1,2].set_title('HIT v @ k=256',fontsize=fsize)
im = hitaxs[2,2].imshow(hit_w[:,:,255], vmin=slcmin, vmax=slcmax, cmap=
                                        slccmap)
hitaxs[2,2].set_title('HIT w @ k=256',fontsize=fsize)

# Set the colorbar:
cax = hitslc.add_axes([0.03, 0.04, 0.93, 0.02])
hitslc.colorbar(im,cax=cax, orientation='horizontal')

# Define the subplot figure:
hstslc, hstaxs = plt.subplots(3,3, figsize=(10,6), dpi=160)
hstslc.suptitle('HST Velocity Slices: u,v,w')

# Plot HST k=1:
hstaxs[0,0].imshow(hst_u[:,:,0], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hstaxs[0,0].set_title('HST u @ k=1',fontsize=fsize)
hstaxs[1,0].imshow(hst_v[:,:,0], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hstaxs[1,0].set_title('HST v @ k=1',fontsize=fsize)
hstaxs[2,0].imshow(hst_w[:,:,0], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hstaxs[2,0].set_title('HST w @ k=1',fontsize=fsize)

# Plot HST k=128:
hstaxs[0,1].imshow(hst_u[:,:,127], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hstaxs[0,1].set_title('HST u @ k=128',fontsize=fsize)
hstaxs[1,1].imshow(hst_v[:,:,127], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hstaxs[1,1].set_title('HST v @ k=128',fontsize=fsize)
hstaxs[2,1].imshow(hst_w[:,:,127], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hstaxs[2,1].set_title('HST w @ k=128',fontsize=fsize)

# Plot HST k=256:
hstaxs[0,2].imshow(hst_u[:,:,255], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hstaxs[0,2].set_title('HST u @ k=256',fontsize=fsize)
hstaxs[1,2].imshow(hst_v[:,:,255], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hstaxs[1,2].set_title('HST v @ k=256',fontsize=fsize)
hstaxs[2,2].imshow(hst_w[:,:,255], vmin=slcmin, vmax=slcmax, cmap=slccmap)
hstaxs[2,2].set_title('HST w @ k=256',fontsize=fsize)

# Set the colorbar:
cax = hstslc.add_axes([0.03, 0.04, 0.93, 0.02])
hstslc.colorbar(im,cax=cax, orientation='horizontal')

# Set tick font size for all:
for i in range(3):
    for j in range(3):
        plt.sca(hitaxs[i,j])
        plt.xticks(fontsize=tsize)
        plt.yticks(fontsize=tsize)

        plt.sca(hstaxs[i,j])
        plt.xticks(fontsize=tsize)
```

```
        plt.yticks(fontsize=tsize)

#########################################################################
# CALCULATE X-Z AVERAGES OF U V W [1.5]
# Average HIT:
hit_uxz = 1/(nx[0]*nx[2])*np.sum(hit_u, axis=(0,2))
hit_vxz = 1/(nx[0]*nx[2])*np.sum(hit_v, axis=(0,2))
hit_wxz = 1/(nx[0]*nx[2])*np.sum(hit_w, axis=(0,2))

# Average HST:
hst_uxz = 1/(nx[0]*nx[2])*np.sum(hst_u, axis=(0,2))
hst_vxz = 1/(nx[0]*nx[2])*np.sum(hst_v, axis=(0,2))
hst_wxz = 1/(nx[0]*nx[2])*np.sum(hst_w, axis=(0,2))

# Plot as a function of y:
plt.figure(figsize=(10,6), dpi=160)

plt.subplot(121)
plt.plot(np.linspace(0,lx[1],nx[1]), hit_uxz, label='uxz Average')
plt.plot(np.linspace(0,lx[1],nx[1]), hit_vxz, label='vxz Average')
plt.plot(np.linspace(0,lx[1],nx[1]), hit_wxz, label='wxz Average')
plt.title('HIT')
plt.xlabel('y')
plt.legend()

plt.subplot(122)
plt.plot(np.linspace(0,lx[1],nx[1]), hst_uxz, label='uxz Average')
plt.plot(np.linspace(0,lx[1],nx[1]), hst_vxz, label='vxz Average')
plt.plot(np.linspace(0,lx[1],nx[1]), hst_wxz, label='wxz Average')
plt.title('HST')
plt.xlabel('y')
plt.legend()

#########################################################################
# CALCULATE AND PLOT XY FIELDS OF FLUCTUATING COMPONENTS [1.6]
# Allocate for memory:
hst_up = np.zeros((nx[0],nx[1],nx[2]))
hst_vp = np.zeros((nx[0],nx[1],nx[2]))
hst_wp = np.zeros((nx[0],nx[1],nx[2]))

# Calculate each quantity:
for j in range(nx[1]):
    hst_up[:,j,:] = hst_u[:,j,:] - hst_uxz[j]
    hst_vp[:,j,:] = hst_v[:,j,:] - hst_vxz[j]
    hst_wp[:,j,:] = hst_w[:,j,:] - hst_wxz[j]

# Set some imshow settings:
slccmap = 'jet' # set the colormap
fsize = 4 # fontsize
tsize = 6 # ticksize

# Find the min and max of these values:
hst_up_min = np.min(hst_up)
hst_vp_min = np.min(hst_vp)
```

```
hst_wp_min = np.min(hst_wp)
hstp_min = np.min([hst_up_min, hst_vp_min, hst_wp_min])

hst_up_max = np.max(hst_up)
hst_vp_max = np.max(hst_vp)
hst_wp_max = np.max(hst_wp)
hstp_max = np.max([hst_up_max, hst_vp_max, hst_wp_max])

# Now we plot the results:
# Define the subplot figure:
hstpslc, hstpaxs = plt.subplots(3,3, figsize=(10,6), dpi=160)
hstpslc.suptitle('Velocity slice: u-u_xz')

hstpaxs[0,0].imshow(hst_up[:,:,0], vmin=hstp_min, vmax=hstp_max, cmap=
                                        slccmap)
hstpaxs[0,0].set_title('HST u\' @ k=1',fontsize=fsize)
hstpaxs[1,0].imshow(hst_vp[:,:,0], vmin=hstp_min, vmax=hstp_max, cmap=
                                        slccmap)
hstpaxs[1,0].set_title('HST v\' @ k=1',fontsize=fsize)
hstpaxs[2,0].imshow(hst_wp[:,:,0], vmin=hstp_min, vmax=hstp_max, cmap=
                                        slccmap)
hstpaxs[2,0].set_title('HST w\' @ k=1',fontsize=fsize)

# Plot HST k=128:
hstpaxs[0,1].imshow(hst_up[:,:,127], vmin=hstp_min, vmax=hstp_max, cmap=
                                        slccmap)
hstpaxs[0,1].set_title('HST u\' @ k=128',fontsize=fsize)
hstpaxs[1,1].imshow(hst_vp[:,:,127], vmin=hstp_min, vmax=hstp_max, cmap=
                                        slccmap)
hstpaxs[1,1].set_title('HST v\' @ k=128',fontsize=fsize)
hstpaxs[2,1].imshow(hst_wp[:,:,127], vmin=hstp_min, vmax=hstp_max, cmap=
                                        slccmap)
hstpaxs[2,1].set_title('HST w\' @ k=128',fontsize=fsize)

# Plot HST k=256:
hstpaxs[0,2].imshow(hst_up[:,:,255], vmin=hstp_min, vmax=hstp_max, cmap=
                                        slccmap)
hstpaxs[0,2].set_title('HST u\' @ k=256',fontsize=fsize)
hstpaxs[1,2].imshow(hst_vp[:,:,255], vmin=hstp_min, vmax=hstp_max, cmap=
                                        slccmap)
hstpaxs[1,2].set_title('HST v\' @ k=256',fontsize=fsize)
im = hstpaxs[2,2].imshow(hst_wp[:,:,255], vmin=hstp_min, vmax=hstp_max,
                                        cmap=slccmap)
hstpaxs[2,2].set_title('HST w\' @ k=256',fontsize=fsize)

# Set the colorbar:
#cax = hstpslc.add_axes([0.85, 0.05, 0.01, 0.9])
cax = hstpslc.add_axes([0.03, 0.04, 0.93, 0.02])
hstpslc.colorbar(im,cax=cax, orientation='horizontal')

for i in range(3):
    for j in range(3):
        plt.sca(hstpaxs[i,j])
        plt.xticks(fontsize=tsize)
```

```
        plt.yticks(fontsize=tsize)

#########################################################################
# PLOT SLICES OF FLUCTUATIONS MINUS FULL VOLUME AVERAGES [1.7]
 # Calculate Full Volume Averages:
hit_u_xyz = 1/(nx[0]*nx[1]*nx[2])*np.sum(hit_u)
hit_v_xyz = 1/(nx[0]*nx[1]*nx[2])*np.sum(hit_v)
hit_w_xyz = 1/(nx[0]*nx[1]*nx[2])*np.sum(hit_w)

# Find the fluctuations now:
hit_upf = hit_u - hit_u_xyz
hit_vpf = hit_v - hit_v_xyz
hit_wpf = hit_w - hit_w_xyz

np.save('hit_up',hit_upf)

# Set some imshow settings:
slccmap = 'jet' # set the colormap
fsize = 4 # fontsize
tsize = 6 # ticksize

# Find the min and max of these values:
hit_upa_min = np.min(hit_upf)
hit_vpa_min = np.min(hit_vpf)
hit_wpa_min = np.min(hit_wpf)
hitpf_min = np.min([hit_upa_min, hit_vpa_min, hit_wpa_min])

hit_upa_max = np.max(hit_upf)
hit_vpa_max = np.max(hit_vpf)
hit_wpa_max = np.max(hit_wpf)
hitpf_max = np.max([hit_upa_max, hit_vpa_max, hit_wpa_max])

# Now we plot the results:
# Define the subplot figure:
hitpslc, hitpaxs = plt.subplots(3,3, figsize=(10,6), dpi=160)
hitpslc.suptitle('Velocity slice: u-u_xyz')

hitpaxs[0,0].imshow(hit_upf[:,:,0], vmin=hitpf_min, vmax=hitpf_max, cmap=
                                    slccmap)
hitpaxs[0,0].set_title('HIT u\' @ k=1',fontsize=fsize)
hitpaxs[1,0].imshow(hit_vpf[:,:,0], vmin=hitpf_min, vmax=hitpf_max, cmap=
                                    slccmap)
hitpaxs[1,0].set_title('HIT v\' @ k=1',fontsize=fsize)
hitpaxs[2,0].imshow(hit_wpf[:,:,0], vmin=hitpf_min, vmax=hitpf_max, cmap=
                                    slccmap)
hitpaxs[2,0].set_title('HIT w\' @ k=1',fontsize=fsize)

# Plot HIT k=128:
hitpaxs[0,1].imshow(hit_upf[:,:,127], vmin=hitpf_min, vmax=hitpf_max, cmap
                                    =slccmap)
hitpaxs[0,1].set_title('HIT u\' @ k=128',fontsize=fsize)
hitpaxs[1,1].imshow(hit_vpf[:,:,127], vmin=hitpf_min, vmax=hitpf_max, cmap
                                    =slccmap)
hitpaxs[1,1].set_title('HIT v\' @ k=128',fontsize=fsize)
```

```
hitpaxs[2,1].imshow(hit_wpf[:,:,127], vmin=hitpf_min, vmax=hitpf_max, cmap
                                        =slccmap)
hitpaxs[2,1].set_title('HIT w\' @ k=128',fontsize=fsize)

# Plot HIT k=256:
hitpaxs[0,2].imshow(hit_upf[:,:,255], vmin=hitpf_min, vmax=hitpf_max, cmap
                                        =slccmap)
hitpaxs[0,2].set_title('HIT u\' @ k=256',fontsize=fsize)
hitpaxs[1,2].imshow(hit_vpf[:,:,255], vmin=hitpf_min, vmax=hitpf_max, cmap
                                        =slccmap)
hitpaxs[1,2].set_title('HIT v\' @ k=256',fontsize=fsize)
im = hitpaxs[2,2].imshow(hit_wpf[:,:,255], vmin=hitpf_min, vmax=hitpf_max,
                                         cmap=slccmap)
hitpaxs[2,2].set_title('HIT w\' @ k=256',fontsize=fsize)

# Set the colorbar:
#cax = hitpslc.add_axes([0.85, 0.05, 0.01, 0.9])
cax = hitpslc.add_axes([0.03, 0.04, 0.93, 0.02])
hitpslc.colorbar(im,cax=cax, orientation='horizontal')

for i in range(3):
    for j in range(3):
        plt.sca(hitpaxs[i,j])
        plt.xticks(fontsize=tsize)
        plt.yticks(fontsize=tsize)

#########################################################################
# CALCULATE THE REYNOLDS STRESSES [1.8]
# Combine the HST/HIT u'_i data into one data structure
hst_np = [hst_up, hst_vp, hst_wp]
hit_npf = [hit_upf, hit_vpf, hit_wpf]

# Allocate for Reynolds Stresses
hst_uipuip_xz = np.zeros((3,3,nx[1]))
hit_uipuip_xyz = np.zeros((3,3))

# Calculate Reynolds Stresses
for i in range(3):
    for j in range(3):
        # Multiply the u'_i together at every location:
        hst_uipuip = hst_np[i]*hst_np[j]
        hit_uipuip = hit_npf[i]*hit_npf[j]

        # Calculate <>xz of the HST ui'ui', or Reynolds Stress
        hst_uipuip_xz[i,j,:] = 1/(nx[0]*nx[2])*np.sum(hst_uipuip, axis=(0,
                                                2))

        # Calculate <>xyz of the HIT ui'ui', or Reynolds Stress
        hit_uipuip_xyz[i,j] = 1/(nx[0]*nx[1]*nx[2])*np.sum(hit_uipuip)

# Plot the data
plt.figure(figsize=(10,6), dpi=160)
plt.suptitle('HST <u\'iu\'i>xz vs. HIT <u\'iu\'i>xyz')
legsize=6
```

```
plt.subplot(331)
plt.plot(np.arange(nx[1]), hst_uipuip_xz[0,0,:], label='HST u\'u\'')
plt.plot(np.arange(nx[1]), hit_uipuip_xyz[0,0]*np.ones(nx[1]), label='HIT
                                    u\'u\'')
plt.legend(prop={'size': legsize})

plt.subplot(332)
plt.plot(np.arange(nx[1]), hst_uipuip_xz[0,1,:], label='HST u\'v\'')
plt.plot(np.arange(nx[1]), hit_uipuip_xyz[0,1]*np.ones(nx[1]), label='HIT
                                    u\'v\'')
plt.legend(prop={'size': legsize})

plt.subplot(333)
plt.plot(np.arange(nx[1]), hst_uipuip_xz[0,2,:], label='HST u\'w\'')
plt.plot(np.arange(nx[1]), hit_uipuip_xyz[0,2]*np.ones(nx[1]), label='HIT
                                    u\'w\'')
plt.legend(prop={'size': legsize})

plt.subplot(334)
plt.plot(np.arange(nx[1]), hst_uipuip_xz[1,0,:], label='HST v\'u\'')
plt.plot(np.arange(nx[1]), hit_uipuip_xyz[1,0]*np.ones(nx[1]), label='HIT
                                    v\'u\'')
plt.legend(prop={'size': legsize})

plt.subplot(335)
plt.plot(np.arange(nx[1]), hst_uipuip_xz[1,1,:], label='HST v\'v\'')
plt.plot(np.arange(nx[1]), hit_uipuip_xyz[1,1]*np.ones(nx[1]), label='HIT
                                    v\'v\'')
plt.legend(prop={'size': legsize})

plt.subplot(336)
plt.plot(np.arange(nx[1]), hst_uipuip_xz[1,2,:], label='HST v\'w\'')
plt.plot(np.arange(nx[1]), hit_uipuip_xyz[1,2]*np.ones(nx[1]), label='HIT
                                    v\'w\'')
plt.legend(prop={'size': legsize})

plt.subplot(337)
plt.plot(np.arange(nx[1]), hst_uipuip_xz[2,0,:], label='HST w\'u\'')
plt.plot(np.arange(nx[1]), hit_uipuip_xyz[2,0]*np.ones(nx[1]), label='HIT
                                    w\'u\'')
plt.legend(prop={'size': legsize})

plt.subplot(338)
plt.plot(np.arange(nx[1]), hst_uipuip_xz[2,1,:], label='HST w\'v\'')
plt.plot(np.arange(nx[1]), hit_uipuip_xyz[2,1]*np.ones(nx[1]), label='HIT
                                    w\'v\'')
plt.legend(prop={'size': legsize})

plt.subplot(339)
plt.plot(np.arange(nx[1]), hst_uipuip_xz[2,2,:], label='HST w\'w\'')
plt.plot(np.arange(nx[1]), hit_uipuip_xyz[2,2]*np.ones(nx[1]), label='HIT
                                    w\'w\'')
plt.legend(prop={'size': legsize})
```

```
#########################################################################
# CALCULATE 2,3,4TH MOMENTS [1.9]
# HIT moments (page 84 of text):
hit_2m = np.mean(hit_upf**2)
hit_sigma = np.sqrt(hit_2m)
hit_3m = np.mean(hit_upf**3)
hit_4m = np.mean(hit_upf**4)
hit_s = hit_3m/hit_sigma**3
hit_k = hit_4m/hit_sigma**4

# HST moments:
hst_2m = 1/(nx[0]*nx[2])*np.sum(hst_up**2, axis=(0,2))
hst_sigma = np.sqrt(hst_2m)
hst_3m = 1/(nx[0]*nx[2])*np.sum(hst_up**3, axis=(0,2))
hst_4m = 1/(nx[0]*nx[2])*np.sum(hst_up**4, axis=(0,2))
hst_s = hst_3m/hst_sigma**3
hst_k = hst_4m/hst_sigma**4

# Plot the data:
plt.figure(figsize=(10,6), dpi=160)
plt.suptitle('HST vs. HIT: skewness and kurtosis for u')
legsize = 6

plt.subplot(121)
plt.plot(np.arange(nx[1]), hst_s, label='HST S')
plt.plot(np.arange(nx[1]), hit_s*np.ones(nx[1]), label='HIT S')
plt.legend(prop={'size': legsize})

plt.subplot(122)
plt.plot(np.arange(nx[1]), hst_k, label='HST K')
plt.plot(np.arange(nx[1]), hit_k*np.ones(nx[1]), label='HIT K')
plt.legend(prop={'size': legsize})

#########################################################################
# PDFs OF THE DATA WITH GAUSSIAN OVERLAY [1.10]
# Number of histogram bins:
nbins = 100

# Reshape data:
hst_up_32_vec = np.reshape(hst_up[:,31,:],(nx[0]*nx[2]))
hst_vp_32_vec = np.reshape(hst_vp[:,31,:],(nx[0]*nx[2]))
hst_wp_32_vec = np.reshape(hst_wp[:,31,:],(nx[0]*nx[2]))
hst_up_64_vec = np.reshape(hst_up[:,63,:],(nx[0]*nx[2]))
hst_vp_64_vec = np.reshape(hst_vp[:,63,:],(nx[0]*nx[2]))
hst_wp_64_vec = np.reshape(hst_wp[:,63,:],(nx[0]*nx[2]))
hst_up_96_vec = np.reshape(hst_up[:,95,:],(nx[0]*nx[2]))
hst_vp_96_vec = np.reshape(hst_vp[:,95,:],(nx[0]*nx[2]))
hst_wp_96_vec = np.reshape(hst_wp[:,95,:],(nx[0]*nx[2]))

hit_up_vec = np.reshape(hit_upf,(nx[0]*nx[1]*nx[2]))
hit_vp_vec = np.reshape(hit_vpf,(nx[0]*nx[1]*nx[2]))
hit_wp_vec = np.reshape(hit_wpf,(nx[0]*nx[1]*nx[2]))
```

```
# Create a histogram of this data (density true for PDF):
hst_hist_32_u = np.histogram(hst_up_32_vec, bins=nbins, density=True)
hst_hist_32_v = np.histogram(hst_vp_32_vec, bins=nbins, density=True)
hst_hist_32_w = np.histogram(hst_wp_32_vec, bins=nbins, density=True)
hst_hist_64_u = np.histogram(hst_up_64_vec, bins=nbins, density=True)
hst_hist_64_v = np.histogram(hst_vp_64_vec, bins=nbins, density=True)
hst_hist_64_w = np.histogram(hst_wp_64_vec, bins=nbins, density=True)
hst_hist_96_u = np.histogram(hst_up_96_vec, bins=nbins, density=True)
hst_hist_96_v = np.histogram(hst_vp_96_vec, bins=nbins, density=True)
hst_hist_96_w = np.histogram(hst_wp_96_vec, bins=nbins, density=True)

hit_hist_u = np.histogram(hit_up_vec, bins=nbins, density=True)
hit_hist_v = np.histogram(hit_vp_vec, bins=nbins, density=True)
hit_hist_w = np.histogram(hit_wp_vec, bins=nbins, density=True)

# Find means of all:
hst_mu_32_u = np.mean(hst_up_32_vec)
hst_mu_32_v = np.mean(hst_vp_32_vec)
hst_mu_32_w = np.mean(hst_wp_32_vec)
hst_mu_64_u = np.mean(hst_up_64_vec)
hst_mu_64_v = np.mean(hst_vp_64_vec)
hst_mu_64_w = np.mean(hst_wp_64_vec)
hst_mu_96_u = np.mean(hst_up_96_vec)
hst_mu_96_v = np.mean(hst_vp_96_vec)
hst_mu_96_w = np.mean(hst_wp_96_vec)

hit_mu_u = np.mean(hit_up_vec)
hit_mu_v = np.mean(hit_vp_vec)
hit_mu_w = np.mean(hit_wp_vec)

# Find sigmas of all:
hst_sigma_32_u = np.std(hst_up_32_vec)
hst_sigma_32_v = np.std(hst_vp_32_vec)
hst_sigma_32_w = np.std(hst_wp_32_vec)
hst_sigma_64_u = np.std(hst_up_64_vec)
hst_sigma_64_v = np.std(hst_vp_64_vec)
hst_sigma_64_w = np.std(hst_wp_64_vec)
hst_sigma_96_u = np.std(hst_up_96_vec)
hst_sigma_96_v = np.std(hst_vp_96_vec)
hst_sigma_96_w = np.std(hst_wp_96_vec)

hit_sigma_u = np.std(hit_up_vec)
hit_sigma_v = np.std(hit_vp_vec)
hit_sigma_w = np.std(hit_wp_vec)

# Define Gaussian function:
def pdf(x,mu,sigma):
    return 1/np.sqrt(2*pi)*1/sigma*np.exp(-(x)**2/(2*sigma**2))

# Get Gaussian profile
hst_gauss_32_u = pdf(hst_hist_32_u[1][1:], hst_mu_32_u, hst_sigma_32_u)
hst_gauss_32_v = pdf(hst_hist_32_v[1][1:], hst_mu_32_v, hst_sigma_32_v)
hst_gauss_32_w = pdf(hst_hist_32_w[1][1:], hst_mu_32_w, hst_sigma_32_w)
hst_gauss_64_u = pdf(hst_hist_64_u[1][1:], hst_mu_64_u, hst_sigma_64_u)
```

```
hst_gauss_64_v = pdf(hst_hist_64_v[1][1:], hst_mu_64_v, hst_sigma_64_v)
hst_gauss_64_w = pdf(hst_hist_64_w[1][1:], hst_mu_64_w, hst_sigma_64_w)
hst_gauss_96_u = pdf(hst_hist_96_u[1][1:], hst_mu_96_u, hst_sigma_96_u)
hst_gauss_96_v = pdf(hst_hist_96_v[1][1:], hst_mu_96_v, hst_sigma_96_v)
hst_gauss_96_w = pdf(hst_hist_96_w[1][1:], hst_mu_96_w, hst_sigma_96_w)

hit_gauss_u = pdf(hit_hist_u[1][1:], hit_mu_u, hit_sigma_u)
hit_gauss_v = pdf(hit_hist_v[1][1:], hit_mu_v, hit_sigma_v)
hit_gauss_w = pdf(hit_hist_w[1][1:], hit_mu_w, hit_sigma_w)

# Plot the data:
plt.figure(figsize=(10,6), dpi=160)
plt.suptitle('HST PDFs')
legsize=6

plt.subplot(331)
plt.plot(hst_hist_32_u[1][1:], hst_hist_32_u[0],label='HST u j=32')
plt.plot(hst_hist_32_u[1][1:],hst_gauss_32_u, color='red',label='Gaussian'
                                        )
plt.legend(prop={'size': legsize})

plt.subplot(332)
plt.plot(hst_hist_32_v[1][1:], hst_hist_32_v[0],label='HST v j=32')
plt.plot(hst_hist_32_v[1][1:],hst_gauss_32_v, color='red',label='Gaussian'
                                        )
plt.legend(prop={'size': legsize})

plt.subplot(333)
plt.plot(hst_hist_32_w[1][1:], hst_hist_32_w[0],label='HST w j=32')
plt.plot(hst_hist_32_w[1][1:],hst_gauss_32_w, color='red',label='Gaussian'
                                        )
plt.legend(prop={'size': legsize})

plt.subplot(334)
plt.plot(hst_hist_64_u[1][1:], hst_hist_64_u[0],label='HST u j=64')
plt.plot(hst_hist_64_u[1][1:],hst_gauss_64_u, color='red',label='Gaussian'
                                        )
plt.legend(prop={'size': legsize})

plt.subplot(335)
plt.plot(hst_hist_64_v[1][1:], hst_hist_64_v[0],label='HST v j=64')
plt.plot(hst_hist_64_v[1][1:],hst_gauss_64_v, color='red',label='Gaussian'
                                        )
plt.legend(prop={'size': legsize})

plt.subplot(336)
plt.plot(hst_hist_64_w[1][1:], hst_hist_64_w[0],label='HST w j=64')
plt.plot(hst_hist_64_w[1][1:],hst_gauss_64_w, color='red',label='Gaussian'
                                        )
plt.legend(prop={'size': legsize})

plt.subplot(337)
plt.plot(hst_hist_96_u[1][1:], hst_hist_96_u[0],label='HST u j=96')
```

```
plt.plot(hst_hist_96_u[1][1:],hst_gauss_96_u, color='red',label='Gaussian'
                                        )
plt.legend(prop={'size': legsize})

plt.subplot(338)
plt.plot(hst_hist_96_v[1][1:], hst_hist_96_v[0],label='HST v j=96')
plt.plot(hst_hist_96_v[1][1:],hst_gauss_96_v, color='red',label='Gaussian'
                                        )
plt.legend(prop={'size': legsize})

plt.subplot(339)
plt.plot(hst_hist_96_w[1][1:], hst_hist_96_w[0],label='HST w j=96')
plt.plot(hst_hist_96_w[1][1:],hst_gauss_96_w, color='red',label='Gaussian'
                                        )
plt.legend(prop={'size': legsize})

# HIT:
plt.figure(figsize=(10,6), dpi=160)
plt.suptitle('HIT PDFs')
legsize=6

plt.subplot(131)
plt.plot(hit_hist_u[1][1:], hit_hist_u[0], label='HIT u')
plt.plot(hit_hist_u[1][1:], hit_gauss_u, color='red',label='Gaussian')
plt.legend(prop={'size': legsize})

plt.subplot(132)
plt.plot(hit_hist_v[1][1:], hit_hist_v[0], label='HIT v')
plt.plot(hit_hist_v[1][1:], hit_gauss_v, color='red',label='Gaussian')
plt.legend(prop={'size': legsize})

plt.subplot(133)
plt.plot(hit_hist_w[1][1:], hit_hist_w[0], label='HIT w')
plt.plot(hit_hist_w[1][1:], hit_gauss_w, color='red',label='Gaussian')
plt.legend(prop={'size': legsize})


#########################################################################
# PRINT LINE END
print('\n')
print('-'*60)
print('\n')

#########################################################################
# SHOW FIGURES
plt.close('all')
#plt.show()
```

B Code: Problem 2

```
#########################################################################
```

```
#     ____                 _              _      ____    ___   ____
#    |  _ \ _ __ ___ (_) ___   ___| |_   |___ \ / _ \|___ \
#    | |_) | '__/ _ \| |/ _ \ / __| __|    __) | | | |   __) |
#    |  __/| | | (_) | |  __/ (__| |_    / __/| |_|  / __/
#    |_|   |_|  \___// |\___|\___|\__|  |_____|\__\_|_____|
#                  |__/
#
# Purpose: Main script for ASEN 5037 Project 2 Question 2
#
# Note: for best results, use an iPython interpreter.
#
# Author: Duncan McGough
#
# Date Created:     5/7/19
# Date Edited:      5/10/19
#
###############################################################################
# IMPORT PACKAGES
import numpy as np
import matplotlib.pyplot as plt
#import cProfile
#from numba import jit
from scipy.stats import norm
import sys
sys.path.insert(0, './generic/') # path to generic functions

# Import generic functions:
from read_data import read_dat
from importdata import dataimport

###############################################################################
# PRINT WELCOME MESSAGE
print('\n'*100)
print('-'*60)
print('TURBULENCE PROJECT 2 Q2 SCRIPT')
print('Author: Duncan McGough')
print('-'*60)
print('\n')

###############################################################################
# DEFINE VARIABLES
pi = np.pi
nx = np.array([256,129,256])
lx = np.array([2*pi, pi, 2*pi])
dx = lx/nx
datafolder = './project2data/'
filenames = [datafolder+'HIT_u.bin', datafolder+'HIT_v.bin',
             datafolder+'HIT_w.bin', datafolder+'HST_u.bin',
             datafolder+'HST_v.bin', datafolder+'HST_w.bin']
uvw = ['u','v','w']

###############################################################################
# IMPORT THE DATA
[hit_u, hit_v, hit_w,
```

```
 hst_u, hst_v, hst_w] = dataimport(filenames, nx)

# Pack for iterating:
hit_n = [hit_u, hit_v, hit_w]

########################################################################
# CALCULATE Aij AND <Aij>xyz: [2.1]
# Preallocate:
A = np.ones((3,3,nx[0],nx[1],nx[2]))
Ap = np.ones((3,3,nx[0],nx[1],nx[2]))
Abar = np.ones((3,3))

# NOTE: numpy's gradient() function has axis definitions that are
#       counterinuitive. Axis 0 is z, axis 1 is y, and
#       axis 2 is x.

for i in range(3): # iterate u_i
    for j in range(3): # iterate x_i
        A[i,j,:,:,:] = np.gradient(hit_n[i], dx[j], axis=2-j)
        Abar[i,j] = np.mean(A[i,j,:,:,:])
        Ap[i,j] = A[i,j,:,:,:] - Abar[i,j] # for 2.2

print('<A>xyz = \n')
print(Abar)
print('\n')

########################################################################
# FIND A' AND PLOT PDFs [2.2]
# Ap = A' and found in section above.

# Reshape data:
A11p_vec = np.reshape(Ap[0,0,:,:,:], (nx[0]*nx[1]*nx[2]))
A12p_vec = np.reshape(Ap[0,1,:,:,:], (nx[0]*nx[1]*nx[2]))

# Define bins:
nbins=100

# Take histogram of this (density True for PDF):
A11p_hist = np.histogram(A11p_vec, bins=nbins, density=True)
A12p_hist = np.histogram(A12p_vec, bins=nbins, density=True)

# Define Gaussian function:
def pdf(x,mu,sigma):
    return 1/np.sqrt(2*pi)*1/sigma*np.exp(-(x)**2/(2*sigma**2))

# Find Standard Deviation:
A11p_sigma = np.std(A11p_vec)
A12p_sigma = np.std(A12p_vec)

# Find mean:
A11p_mean = np.mean(A11p_vec)
A12p_mean = np.mean(A12p_vec)

# Get the Gaussian profile (these are normalized by the hist already):
```

```
A11p_g = pdf(A11p_hist[1][1:], A11p_mean, A11p_sigma)
A12p_g = pdf(A12p_hist[1][1:], A12p_mean, A12p_sigma)

# Plot the data:
plt.figure(figsize=(10,6), dpi=160)
plt.suptitle('HIT A\'ij PDFs')
legsize=6

plt.subplot(121)
plt.plot(A11p_hist[1][1:], A11p_hist[0],label='A11')
plt.plot(A11p_hist[1][1:], A11p_g, color='red',label='Gaussian')
plt.legend(prop={'size': legsize})

plt.subplot(122)
plt.plot(A12p_hist[1][1:], A12p_hist[0],label='A12')
plt.plot(A12p_hist[1][1:], A12p_g, color='red',label='Gaussian')
plt.legend(prop={'size': legsize})

#########################################################################
# FIND SKEWNESS AND KURTOSIS OF A'11 [2.3]
# Define for clarity:
A11p = Ap[0,0,:,:,:]

# Find skew and kurtosis:
A11p_s = np.mean(A11p**3)/A11p_sigma**3
A11p_k = np.mean(A11p**4)/A11p_sigma**4

# print data:
print('A\'11 S: \n')
print(A11p_s)
print('\n')
print('A\'11 K: \n')
print(A11p_k)
print('\n')

# Now do the same for the Gaussian data:
# s=0 and k=3 for gaussian.

#########################################################################
# FLUCTUATING STRAIN RATE FIELD AND PSEUDO ENERGY DISSIPATION [2.4]
# Allocate:
Sp = np.zeros((3,3,nx[0],nx[1],nx[2]))
enu = np.zeros((nx[0],nx[1],nx[2]))

for i in range(3):
    for j in range(3):
        # Calculate S'ij=0.5(A'ij+A'ji):
        Sp[i,j,:,:,:] = 0.5*(Ap[i,j,:,:,:]+Ap[j,i,:,:,:])

        # Pseudo energy dissipation field:
        enu[:,:,:] = enu + 2*Sp[i,j,:,:,:]*Sp[i,j,:,:,:]

# Now volume-averaged:
enu_mean = np.mean(enu)
```

```
# Print results:
print('Volume-Averaged Epsilon/Nu: \n')
print(enu_mean)
print('\n')

######################################################################
# 2D X-Y PLOT OF  eps/<eps>xyz [2.5]
# Colormap max and min:
slcmin = np.min(enu[:,:,127]/enu_mean)
slcmax = np.max(enu[:,:,127]/enu_mean)
slcmap = 'jet'

# Plot:
fig1=plt.figure(figsize=(6,6), dpi=160)
plt.title('HIT Epsilon/<Epsilon>_xyz at k=128')
im=plt.imshow(enu[:,:,127]/enu_mean, vmin=slcmin, vmax=slcmax, cmap=slcmap
                                    )
cax = fig1.add_axes([0.85, 0.05, 0.05, 0.9])
fig1.colorbar(im,cax=cax, orientation='vertical')


######################################################################
# FLUCTUATING VORTICITY FIELD AND ENSTROPHY [2.6]
# Allocate memory:
wp = np.zeros((3,nx[0],nx[1],nx[2]))

# Alternating tensor function:
def levicivita(a,b,c):
    if ((a==1)&(b==2)&(c==3))|((a==2)&(b==3)&(c==1))|((a==3)&(b==1)&(c==2)
                                                      ):
        return 1.0
    elif ((a==1)&(b==3)&(c==2))|((a==2)&(b==1)&(c==3))|((a==3)&(b==2)&(c==
                                                      1)):
        return -1.0
    else:
        return 0.0

# Calculate vorticity:
for i in range(3): # free index
    for j in range(3): # summation
        for k in range(3): # summation
            wp[i,:,:,:] = wp[i,:,:,:] + levicivita(i,j,k) + Ap[k,j,:,:,:]

# Allocate memory:
omega = np.zeros((nx[0],nx[1],nx[2]))

# Now calculate the enstrophy field:
for i in range(3):
    omega[:,:,:] = omega + 0.5*wp[i,:,:,:]*wp[i,:,:,:]

# Find the mean:
omega_mean = np.mean(omega)
```

```
# Print result:
print('<Omega>xyz: \n')
print(omega_mean)
print('\n')

#########################################################################
# 2D PLOT OF NORMALIZED ENSTROPHY [2.8]
# Colormap max and min:
slcmin = np.min(omega[:,:,127]/omega_mean)
slcmax = np.max(omega[:,:,127]/omega_mean)
slcmap = 'jet'

# Plot:
fig2=plt.figure(figsize=(6,6), dpi=160)
plt.title('HIT Omega/<Omega>_xyz at k=128')
im=plt.imshow(omega[:,:,127]/omega_mean, vmin=slcmin, vmax=slcmax, cmap=
                                        slcmap)
cax = fig2.add_axes([0.85, 0.05, 0.05, 0.9])
fig2.colorbar(im,cax=cax, orientation='vertical')

#########################################################################
# CALCULATE PDFS OF THE TWO PREVIOUS QUANTITIES [2.9]
# Create histograms:
nbins = 100
obar_o_hist = np.histogram(omega/omega_mean, bins=nbins, density=True)
ebar_e_hist = np.histogram(enu/enu_mean, bins=nbins, density=True)

# Save data:
np.save('obar_o_hist',obar_o_hist)
np.save('ebar_e_hist',ebar_e_hist)
np.save('obar_o', omega/omega_mean)
np.save('ebar_e', enu/enu_mean)

# Plot the data:
plt.figure(figsize=(10,6), dpi=160)
plt.suptitle('HIT Normalized Psuedo Energy Dissipation and Enstrophy PDFs'
                                        )
legsize=6

plt.subplot(121)
plt.plot(obar_o_hist[1][1:], obar_o_hist[0],label='omega/<omega>xyz')
plt.legend(prop={'size': legsize})

plt.subplot(122)
plt.plot(ebar_e_hist[1][1:], ebar_e_hist[0],label='e/<e>xyz')
plt.legend(prop={'size': legsize})

#########################################################################
# PRINT LINE END
print('\n')
print('-'*60)
print('\n')

#########################################################################
```

```
# SHOW FIGURES
#plt.close('all')
plt.show()
```

C Code: Problem 3

```
##########################################################################
#     ____                 _              _      ____    ___  _____
#    |  _ \ _ __ ___ (_) ___   ___| |_  |___ \ / _ \|___ /
#    | |_) | '__/ _ \| |/ _ \ / __| __|   __) | | | | |_ \
#    |  __/| | | (_) | |  __/ (__| |_   / __/| |_| |___) |
#    |_|   |_|  \___// |\___|\___|\__| |_____|\__\_\____/
#                  |__/
#
# Purpose: Main script for ASEN 5037 Project 2 Question 3
#
# Note: for best results, use an iPython interpreter.
#
# Author: Duncan McGough
#
# Date Created:      5/9/19
# Date Edited:       5/10/19
#
##########################################################################
# IMPORT PACKAGES
import numpy as np
import matplotlib.pyplot as plt
#import cProfile
#from numba import jit
#from scipy.stats import norm
#from mpl_toolkits.mplot3d import Axes3D
#from skimage.measure import marching_cubes_lewiner as mcl
import sys
sys.path.insert(0, './generic/') # path to generic functions

# Import generic functions:
from read_data import read_dat
from importdata import dataimport

##########################################################################
# PRINT WELCOME MESSAGE
print('\n'*100)
print('-'*60)
print('TURBULENCE PROJECT 2 Q3 SCRIPT')
print('Author: Duncan McGough')
print('-'*60)
print('\n')

##########################################################################
# DEFINE VARIABLES
pi = np.pi
```

```
nx = np.array([256,129,256])
lx = np.array([2*pi, pi, 2*pi])
dx = lx/nx
datafolder = './project2data/'
filenames = [datafolder+'HIT_u.bin', datafolder+'HIT_v.bin',
             datafolder+'HIT_w.bin', datafolder+'HST_u.bin',
             datafolder+'HST_v.bin', datafolder+'HST_w.bin']
uvw = ['u','v','w']

#########################################################################
# IMPORT THE DATA
[hit_u, hit_v, hit_w,
 hst_u, hst_v, hst_w] = dataimport(filenames, nx)

#########################################################################
# CALCULATE U-VELOCITY AUTOCORRECTION AND SPATIAL INTEGRAL SCALE
#   AND SPATIAL TAYLOR SCALE [3.1]
# Define autocorrelation function:
def rho(r,up):
    # Make a u' array that is wrapped based on r:
    up_p1 = np.concatenate((up[r:,:,:],up[0:r,:,:]))
    # Return the autocorrelation:
    return np.mean(up*up_p1)/np.mean(up**2)

# Load the HIT u' data from Question 1:
hit_p = np.load('hit_up.npy')

# Housekeeping for next step:
inf = nx[0] # define infinity
r_inf = np.arange(inf) # Define vector
rho_i = np.zeros((inf)) # allocate memory

# Now loop through infinity:
for i in range(inf):
    # Store rho() in vector for numeric integration:
    rho_i[i] = rho(i,hit_p)

# Calculate the spatial integral scale:
llam = np.trapz(rho_i,r_inf)

# Calculate and evaluate derivatives of rho:
d2pdr2_0 = np.gradient(np.gradient(rho_i))[0]

# Calculate the spatial taylor scale:
lam2 = -2*d2pdr2_0**(-1)

# Print results:
print('Spatial Integral Scale: \n')
print(llam)
print('\n')

print('Spatial Taylor Scale: \n')
print(lam2)
print('\n')
```

```
# Plot results:
plt.figure(figsize=(8,6), dpi=160)
plt.plot(r_inf,rho_i,label='HIT u\' Autocorrelation')
plt.plot(r_inf, np.exp(-r_inf/llam),label='Exponential')
plt.plot(r_inf, np.exp(-(np.pi/4)*(r_inf/llam)**2),label='Gaussian')
plt.legend()

#########################################################################
# JOINT PDFS [3.2]
# Load data:
ebar_e_hist = np.load('ebar_e_hist.npy', allow_pickle=True)
obar_o_hist = np.load('obar_o_hist.npy', allow_pickle=True)
ebar_e = np.load('ebar_e.npy', allow_pickle=True)
obar_o = np.load('obar_o.npy', allow_pickle=True)

# Define bins
nbins = 100

# Make histogram for numerator:
eo_hist, ebar_edge, obar_edge = np.histogram2d(np.reshape(ebar_e,(nx[0]*nx
                                    [1]*nx[2])), np.reshape(obar_o,(nx[0]
                                    *nx[1]*nx[2])), bins=nbins)

# Combine into joint histogram:
joint_hist = eo_hist/np.max(eo_hist)

# Define colormap
slcmap = 'jet'
slcmax = np.max(joint_hist)
slcmin = np.min(joint_hist)

# Plot
fig = plt.figure(figsize=(8,6), dpi=160)
im=plt.imshow(joint_hist, vmin=slcmin, vmax=slcmax, cmap=slcmap)
plt.title('Joint PDF of Normalized Energy Dissipation and Enstrophy')
cax = fig.add_axes([0.85, 0.05, 0.05, 0.9])
fig.colorbar(im,cax=cax, orientation='vertical')


#########################################################################
# PRINT LINE END
print('\n')
print('-'*60)
print('\n')

#########################################################################
# SHOW FIGURES
#plt.close('all')
plt.show()
```

D Code: Import Data

```
##########################################################################
#        ___                                  _      ____          _
#       |_ _|_ __ ___   _ __    ___  _ __| |_  |   _ \  __ _| |_ __ _
#        | || '_ ` _ \| '_ \ / _ \| '__| __| | | | |/ _` | __/ _` |
#        | || | | | | | |_) | (_) | |  | |_  | |_| | (_| | || (_| |
#       |___|_| |_| |_| .__/ \___/|_|   \__| |____/ \__,_|\__\__,_|
#                     |_|
#
# Purpose: parse and import all data from HIT and HST data
#
# Author: Duncan McGough
#
# Date Created:      4/11/19
# Date Edited:       4/11/19
#
##########################################################################
# IMPORT PACKAGES
import numpy as np
from read_data import read_dat

##########################################################################
# READ IN DATA AND PARSE
def dataimport(filenames, nx):
    # Allocate memory:
    u = np.zeros((nx[0],nx[1],nx[2],len(filenames)))

    # Step through each file:
    for i in range(len(filenames)):
        # Read in the file:
        tmp = read_dat(filenames[i])

        # Shift the file such that it can be reshaped:
        tmp0 = tmp[1:(nx[0]*nx[1]*nx[2]+1)]

        # Reshape into a block of data:
        u[:,:,:,i] = np.reshape(tmp0, (nx[0],nx[1],nx[2]))

    # Return each u,v,w for HIT and HST respectively:
    return [u[:,:,:,0], u[:,:,:,1], u[:,:,:,2],
            u[:,:,:,3], u[:,:,:,4], u[:,:,:,5]]
```

E Code: Read Data

```
##########################################################################
#        ____                  _    ____          _
#       |  _ \ ___  __ _  __| | |  _ \  __ _| |_ __ _
#       | |_) / _ \/ _` |/ _` | | | | |/ _` | __/ _` |
#       |  _ <  __/ (_| | (_| | | |_| | (_| | || (_| |
#       |_| \_\___|\__,_|\__,_| |____/ \__,_|\__\__,_|
```

```
#
# Purpose: Read in binary HIT and HST data for Project 2
#
# Author: Duncan McGough, adapted from Peter Hamlington's script
#
# Creation Date:    4/11/19
# Edited Date:      4/11/19
#
#########################################################################
# IMPORT PACKAGES
import numpy as np

#########################################################################
def read_dat(fn):
    # Open the file:
    fid = open(fn, 'rb')

    # Read in the data:
    out = np.fromfile(fid, np.single)

    # Close the file:
    fid.close()

    # Return the data:
    return out
```