Problem Set 5 Part I

By: Kali Benavides

Collaborators: Les Armstrong, Maja Svanberg, Serena Patel, Lan Ha

Problem 1

Part A)

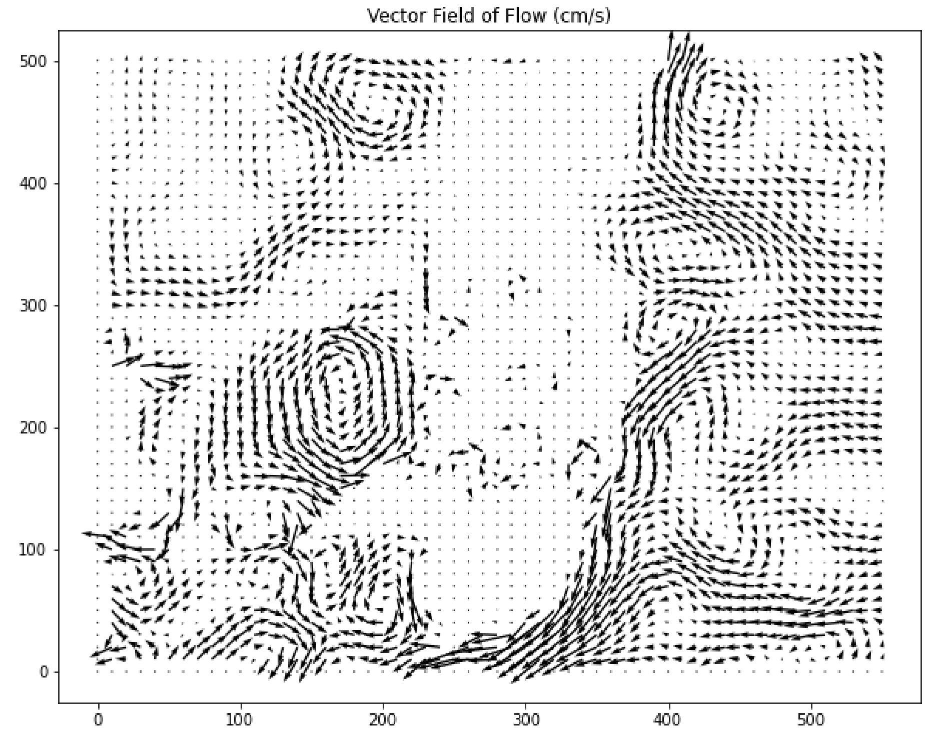
Average Flow over all times T was found by looping through each u vector and v vector file, and converting the flow values to cm/s by multiplying by 25/0.9. And then adding each array of u together and all the v arrays together to create an array that had values for the sum of all u vectors and another array that contained the sum of all v vectors.

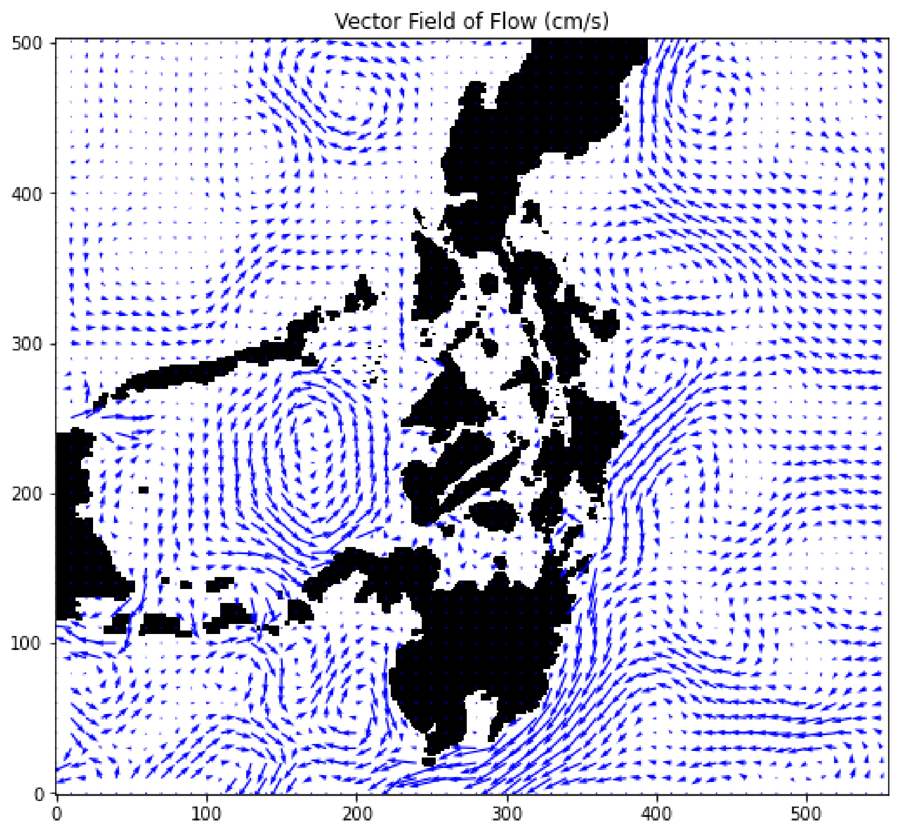
The average was then calculated by dividing each array by the 100 (Representing the number of time stamps)

The maximum horizontal average flow is 51.8 cm/s

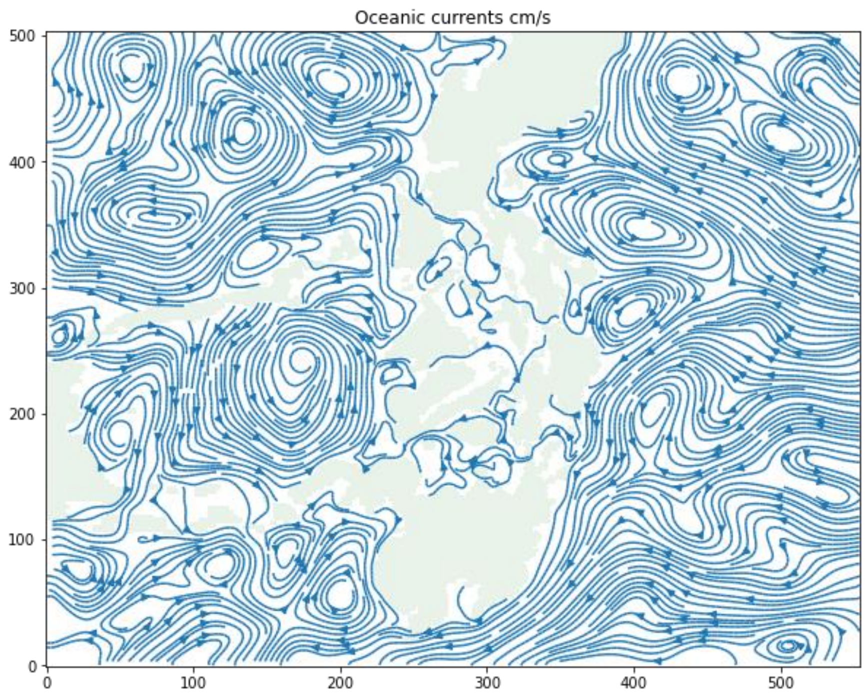
The maximum vertical average flow is 43.4 cm/s

Below quiver plots represent average flow as vector fields:





And represented with a stream plot:



Code for Part A

```python
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import LogNorm
import pandas as pd
import os
import re
import time

import seaborn as sns
from math import floor
```

```python
#Problem 5.1
#A Visualize the average flow (i.e., averaged over all times T, but not over locations) as a 2-D vector field.
#Multiply the flow values by 25/0.9 to get a unit of cm/second (cmps)

#The time interval between any two data snapshots is 3 hours. The grid spacing used is 3 kilometers.
#In this problem, the matrix index (0, 0) will correspond to the coordinate (0km, 0km),or the *bottom, left* of the plot
```

```python
u = '1u.csv'
v = '1v.csv'
mask = 'mask.csv'

U1 = pd.read_csv(u, header = None).to_numpy()
V1 = pd.read_csv(v, header = None).to_numpy()
mask = pd.read_csv(mask,header=None).to_numpy()
cmps = (25/0.9)
#U1 = np.multiply(U1, cmps)
#V1= np.multiply(V1, cmps)

row = U1.shape[0]
col = U1.shape[1]
```

```python
#Location of both folders
path1 = r"C:\Users\kalib\Documents\IDS.131\Pset5\Data"
#List of all text files in the folders
filelist1 = os.listdir(path1)
```

```python
def loopfile(filelist, path):

    #Create empty array to save the sum of the u and v matrices
    All_sum_u = np.zeros((U1.shape))
    All_sum_v = np.zeros((U1.shape))
    cmps = (25/0.9)

    for filename in filelist:
        if filename.endswith("u.csv"):
        #Look first only at u files
            T1_u = pd.read_csv(f'{path}/{filename}', header =None).to_numpy()
            #Convert each Time matrix to cm/s
            T1_u = np.multiply(T1_u, cmps)
            #Save the sum for that file in the array
            All_sum_u = np.add(All_sum_u, T1_u)
        elif filename.endswith("v.csv"):
        #Look first only at u files
            T1_v = pd.read_csv(f'{path}/{filename}', header =None).to_numpy()
            #Convert each Time matrix to cm/s
            T1_v = np.multiply(T1_v, cmps)
            #Save the sum for that file in the array
            All_sum_v = np.add(All_sum_v, T1_v)

    return All_sum_u, All_sum_v
```

```python
#Find sum from files in path 1
#time_start = time.time()

All_sum_u, All_sum_v = loopfile(filelist1, path1)

Avg_u = All_sum_u/100
Avg_v = All_sum_v/100

#print('Time elapsed {}'.format(time.time()-time_start))
```

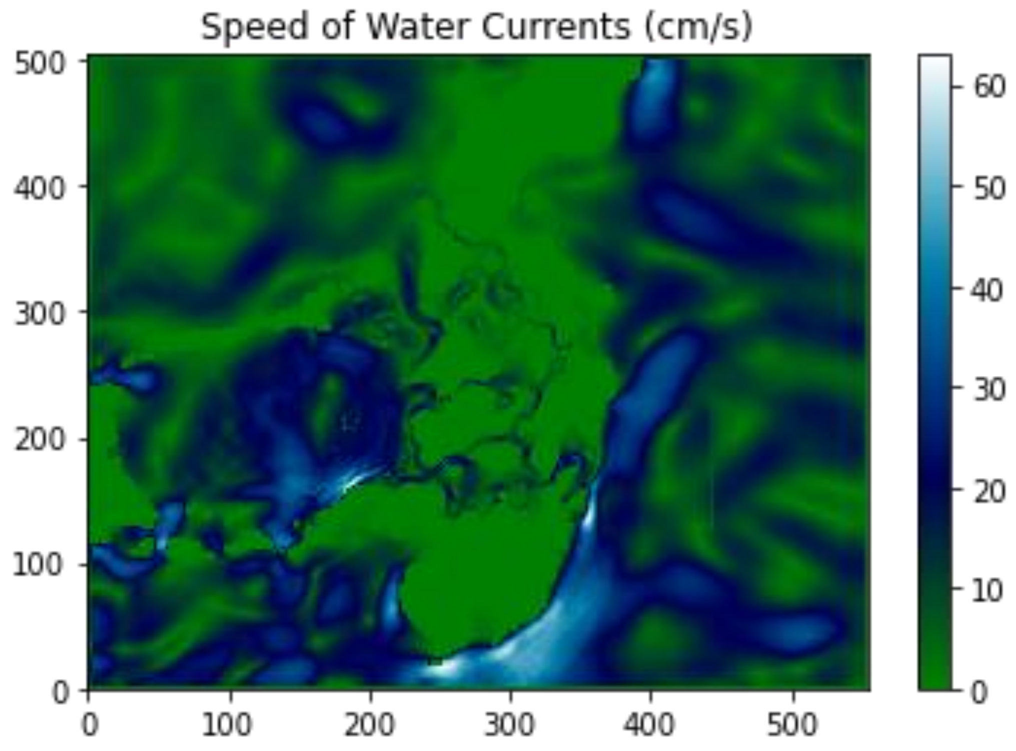
```python
print('Maximum avg U flow is ', np.max(Avg_u))
print('Maximum avg V flow is ', np.max(Avg_v))
```

```
Maximum avg U flow is  51.80300500000002
Maximum avg V flow is  43.4080388888889
```

Part B)

The average flow vectors calculated in part A were used to calculate the average speeds by the following calculation.

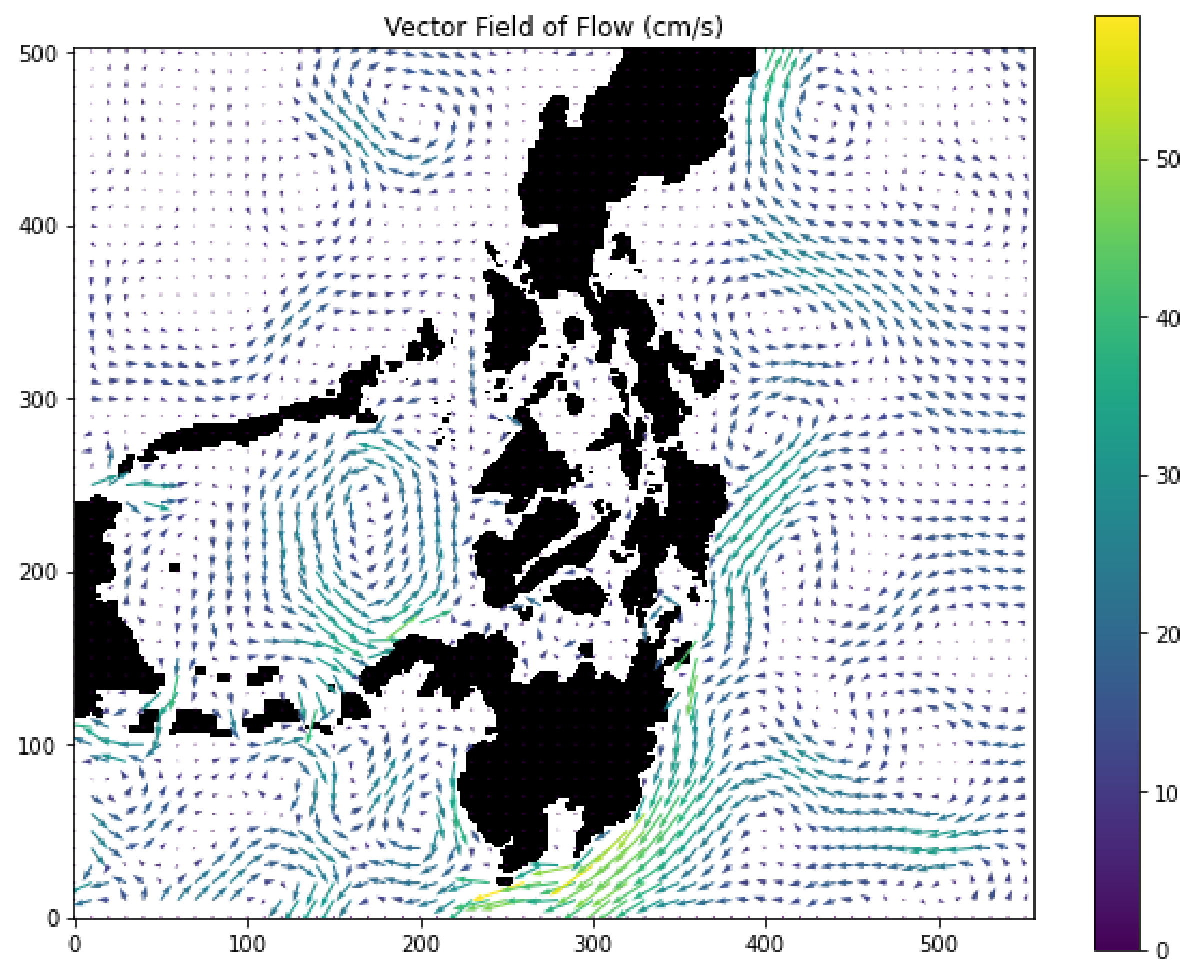
$$\boldsymbol{Speed} = \sqrt{\boldsymbol{u}^2 + \boldsymbol{v}^2}$$



Do you notice any areas with high average speed but low average flow, or vice versa? Why might that be?

Below is the vector field plot of flow represented by arrows with the speed represented by the color of the arrow. Generally there is high correlation between average speed and average flow.

Areas of low average speed but high average flow could represent areas where the water flow is turbulent and so the horizontal or vertical flow vectors are high but perhaps the other vector direction is very low so then the overall speed of the water is low.



Code for part B

```
SumOfVectors = np.add(Avg_u**2, Avg_v**2)
```

```
#Calculating speed from U(Horizontal) and V (vertical) vector components
Speed = np.sqrt(SumOfVectors)
Speed.shape
#Speed_re = Speed[::10,:][:,::10]
```

```
(504, 555)
```

Code for Plotting Quiver

```python
#Grid of coordinates for Full data set points
X,Y = np.meshgrid(np.arange(0, col), np.arange(0, row))
#X_km,Y_km = np.meshgrid(np.arange(0, (col)*3,3), np.arange(0, (row)*3,3))
```

```python
fig1, ax1 = plt.subplots(figsize=(10, 8))

ax1.set_title('Vector Field of Flow (cm/s)')
#matplotlib.pyplot.quiver(x_coordinate, y_coordinate, x_direction, y_direction)
#Resample to plot only every 10 vector points
color = Speed[::10,:][:,::10]
Q = ax1.quiver(X[::10,:][:,::10] , Y[::10,:][:,::10] , (Avg_u[::10,:][:,::10]) , (Avg_v[::10,:][:,::10]), color)
#Q = ax1.quiver(X[::10,:][:,::10] , Y[::10,:][:,::10] , np.flip(Avg_u[::10,:][:,::10],0) , np.flip(Avg_v[::10,:][:,::10],0) )

#strm = ax1.streamplot(x,y, np.flip(u, 0), np.flip(v, 0), density=1.8, linewidth=0.8, maxlength=0.5)
ph = ax1.imshow(np.flip(mask,0), cmap = 'gray', origin='upper', interpolation='nearest')

#ax1.set_xlabel('km')
#ax1.set_ylabel('km')
ax1.invert_yaxis()
fig1.colorbar(Q)
plt.show()
```

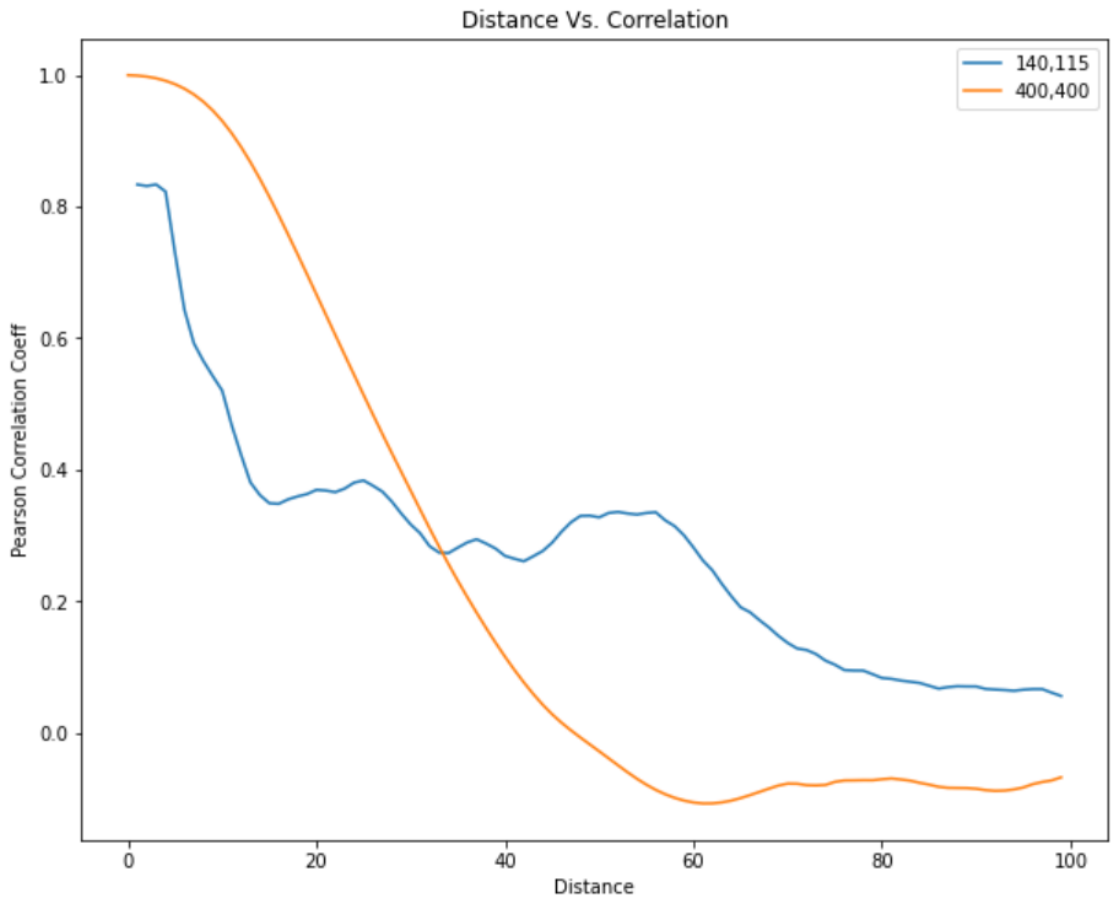
Code for Streamplot

```python
fig1, ax1 = plt.subplots(figsize=(10, 8))
ax1.set_title('Oceanic currents cm/s')

#matplotlib.pyplot.quiver(x_coordinate, y_coordinate, x_direction, y_direction)
S = ax1.streamplot(X, Y, Avg_u, Avg_v, density=5)
ph = ax1.imshow(np.flip(mask,0), origin='upper', alpha=0.09, interpolation='nearest',  aspect='auto', cmap ='ocean')

#ax1.imshow(np.flip(mask,0), cmap = 'gray', origin='upper', interpolation='nearest')
ax1.invert_yaxis()
plt.show()
```

Code for Heat Map

```python
fig, ax = plt.subplots()
im = ax.pcolor(Speed, cmap='ocean')
#ax1.imshow(np.flip(mask,0), origin='upper', alpha=0.09, interpolation='nearest',  aspect='auto', cmap ='gray')
plt.title('Speed of Water Currents (cm/s)')
fig.colorbar(im)
plt.show()
```

Part C)



How would you describe the spatial correlations in ocean flow?

There are very high correlations for ocean flow speeds that are located very close together. As the distance grows the correlation between speed in one location versus another is very low. Eventually the correlation between the speeds at two different locations is zero as they are far apart and independent of one another.

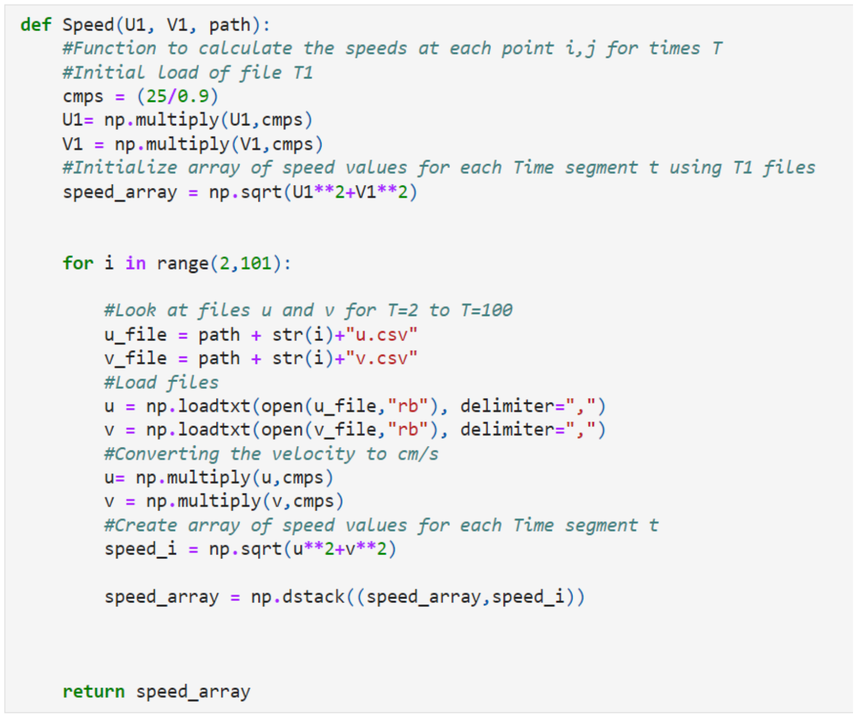
Would Gaussian processes be a good model for spatial correlations in ocean flow? Explain why or why not. Yes because the ocean flow is affected by many random variables (Temperature, pressure, wind). The distribution of many of these variables could be taken as a gaussian distribution. Therefore, the overall ocean flow could also be interpreted as having a gaussian distribution.

Procedure:

Compute Matrix of Speeds:

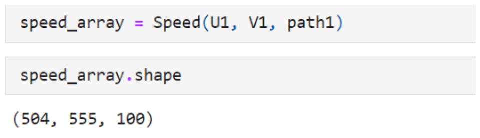
Created a function to compute the an array of the speeds for every time stamp from T1 to T2.

Output is a 3 dimensional matrix, Of size (504, 555, 100)



```python
def Speed(U1, V1, path):
    #Function to calculate the speeds at each point i,j for times T
    #Initial load of file T1
    cmps = (25/0.9)
    U1= np.multiply(U1,cmps)
    V1 = np.multiply(V1,cmps)
    #Initialize array of speed values for each Time segment t using T1 files
    speed_array = np.sqrt(U1**2+V1**2)


    for i in range(2,101):

        #Look at files u and v for T=2 to T=100
        u_file = path + str(i)+"u.csv"
        v_file = path + str(i)+"v.csv"
        #Load files
        u = np.loadtxt(open(u_file,"rb"), delimiter=",")
        v = np.loadtxt(open(v_file,"rb"), delimiter=",")
        #Converting the velocity to cm/s
        u= np.multiply(u,cmps)
        v = np.multiply(v,cmps)
        #Create array of speed values for each Time segment t
        speed_i = np.sqrt(u**2+v**2)

        speed_array = np.dstack((speed_array,speed_i))


    return speed_array
```

Output is a three dimensional array of all the speeds in the 504,500 spatial coordinate grid for all 100 time periods



```python
speed_array = Speed(U1, V1, path1)
```

```python
speed_array.shape
```

```
(504, 555, 100)
```

Compute Distance from i,j to x,y

A function to compute the distances between the (x,y) coordinate of interest.

(140, 115) and (400, 400) respectively.

This outputs a dataframe with columns for the x-coordinate, the y-coordinate and the distance from the specified x,y coordinate of interest.

Filtered this dataframe to only look at coordinates that were less than 100 index points away from x,y

```python
#Function to calculate L1 distance
#Reference: https://www.geeksforgeeks.org/python-calculate-city-bloc
def cityblock_distance(A, B):

    result = np.sum([abs(a - b) for (a, b) in zip(A, B)])
    return result
```

Reference: https://www.geeksforgeeks.org/python-calculate-city-block-distance/

```python
def distance(array, x, y):
    #Input array is array of speeds at one time T, just to get a grid of coord.
    #Input x and y are the coordinates for the point we are interested in
    #Distance is calculated using L1 distance,
    #Will save to an array coordinates and distances
    #And then identify an array of those coordinates whose distance is less than 100 (i.e. very close to the x,y coordinate)
    distance = []
    x_coord = []
    y_coord = []
    #Iterate through every row and column of the array
    row = np.arange(0,array.shape[0]+1)
    col = np.arange(0, array.shape[1]+1)

    for i in col:
        for j in row:
            #Calculate L1 distance
            dist = cityblock_distance([i,j], [x,y])
            #Save distances to a list
            distance.append(dist)
            #Save coordinates to a list
            x_coord.append(i)
            y_coord.append(j)
    #Save the lists from the for loop to a dataframe
    df = pd.DataFrame()
    df['x'] = x_coord
    df['y'] = y_coord
    df['Distance'] = distance
    #Filter the dataframe to save only those distances less than 100
    df_closedistance = df[df['Distance'] < 100]

    return df_closedistance
```

Output:

```python
close_distance = distance(U1, 115, 140)
close_distance.head()
```

	x	y	Distance
8220	16	140	99
8724	17	139	99
8725	17	140	98
8726	17	141	99
9228	18	138	99

```python
close_distance.shape
```

```
(19801, 3)
```

```python
close_distance2 = distance(U1, 400, 400)
```

```python
close_distance2.head()
```

	x	y	Distance
152405	301	400	99
152909	302	399	99
152910	302	400	98
152911	302	401	99
153413	303	398	99

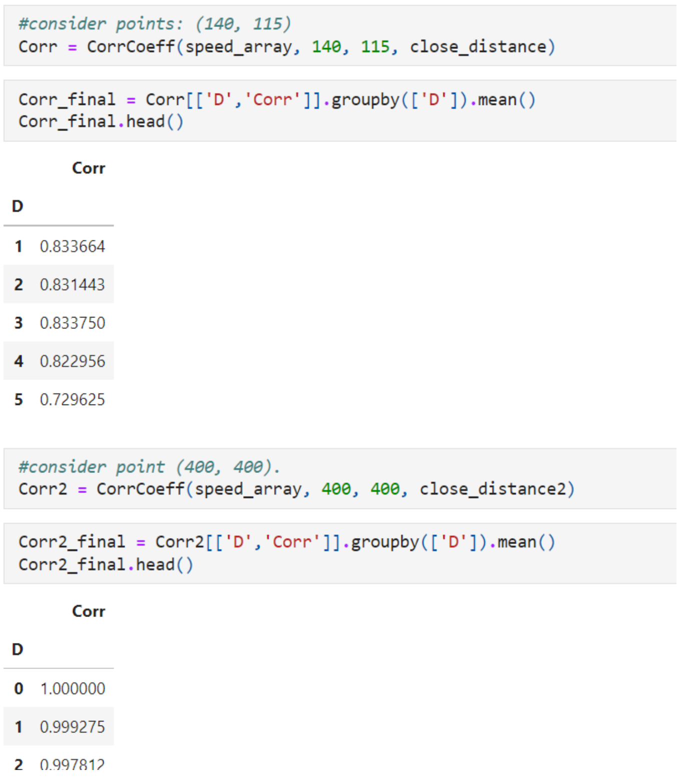
Compute Pearson Correlation Coefficient between x,y and all i,j

Iterate through the dataframe of coordinates and distances between them. (Only looking at distances less than 100 units)

Use the coordinates to access the corresponding speed in the matrix of all speeds that was computed above. The speed is then used to compute a Pearson correlation coefficient between this point i,j and the coordinate of interest x,y. The dataframe of distances and corresponding Pearson correlation coefficient between each coordinate is returned. This dataframe is then grouped by the distance value and a mean of the Pearson Correlation Coeff. for each distance is computed. This is then graphed as function of distance.

```python
def CorrCoeff(speed_array, x, y, close_distance):
    #Inputs are the 3D array of the speeds; row/col of the coordinate of interest; dataframe of col/row and distance
    #Below calculates the Pearson coeff between the speeds for coordinate x,y and all other coordinates
    a = speed_array[y,x,:]
    #a = speed_array[x,y,:] #Access column of speed values located at the x,y coordinates of interest
    Correlation = [] #Initiate an array to save correlation values
    Dis = [] #Initiate an array to save the distances between the x,y of interest and all other coordinates (D<100)
    for row in range(len(close_distance)):
        #Retrieves row value saved in Distance dataframe
        i = close_distance.iloc[row, 0]
        #Retrieves col value saved in Distance dataframe
        j = close_distance.iloc[row, 1]
        #Retrieves distance value saved in Distance dataframe
        distance = close_distance.iloc[row, 2]
        #Retrieves speed value from 3D array of all speeds
        b = speed_array[j,i,:]
        #If all values in B are zero then do not compute Pearson Corr Coeff. This point is a land coordinate
        if not b.any():
            continue
        else:
         #Looks at each column and only calculates correlation for those where there are no zero values
            PC = stats.pearsonr(a, b)
            #Saves the correlation value to a list
            Correlation.append(PC[0])
            #Saves the distance value to a list
            Dis.append(distance)
    #Saves both distance and correlation lists to a dataframe
    df = pd.DataFrame()
    df['D'] = Dis
    df['Corr'] = Correlation
    return df #Returns the dataframe
```

Output:



```
#consider points: (140, 115)
Corr = CorrCoeff(speed_array, 140, 115, close_distance)
```

```
Corr_final = Corr[['D','Corr']].groupby(['D']).mean()
Corr_final.head()
```

```
       Corr
D
1  0.833664
2  0.831443
3  0.833750
4  0.822956
5  0.729625
```

```
#consider point (400, 400).
Corr2 = CorrCoeff(speed_array, 400, 400, close_distance2)
```

```
Corr2_final = Corr2[['D','Corr']].groupby(['D']).mean()
Corr2_final.head()
```

```
       Corr
D
0  1.000000
1  0.999275
2  0.997812
```

Problem 2

Part A)

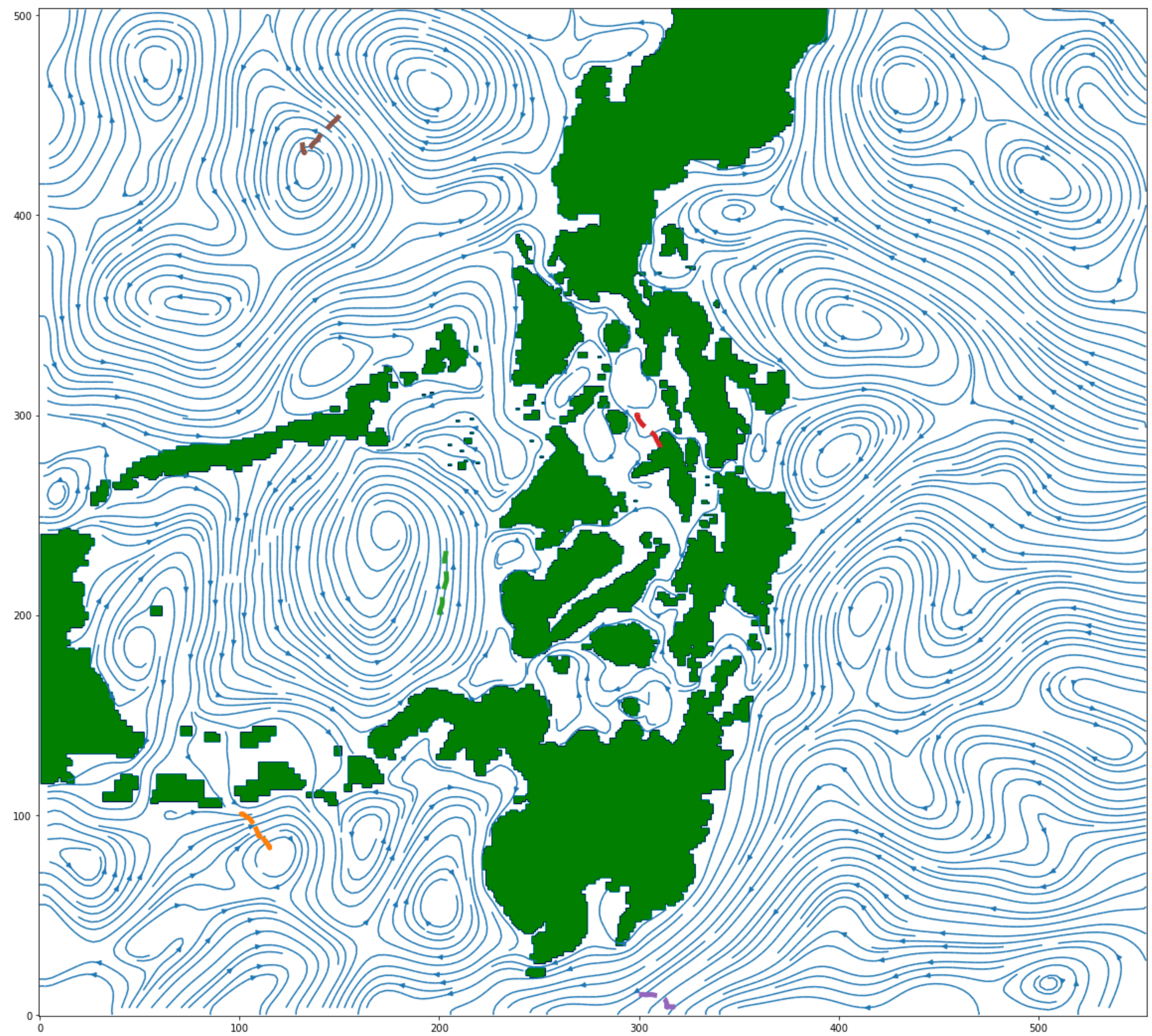
Procedure to track the position and the movement of the particle, caused by the time-varying flow: Created two functions one which determined the velocity at a given position by converting the initial position to an index (Dividing by 3 which is the grid spacing on this 3km by 3 km sampling grid). The index value returned is rounded down to find the corresponding index point on the array of both the horizontal and vertical velocity flows. The velocities at that given point are then inputted into another function which updates the position of the object based on that given velocity and then returns a new position.

$$x_{new} = x + u[i, j] * dt$$

Where dt = 3 hours and x is the current position and $u[i, j]$ is the corresponding horizontal velocity component. And likewise for the vertical velocity component.
This process is then looped through all the times recorded (T1 to T100) and outputting a list of x,y positions over time.

The procedure was tested by initiating with random geographical locations on the map and plotting their path from time T=1 to T=100 (i.e. After 300 hours)

Below are the paths of the different initializations shown as dotted lines in different colors. Plotted over the average flow vectors of the ocean current.


0
100
200
300
400
500
0
100
200
300
400
500

Code for Part A

```python
def update_position(x,y,u,v):
    #x and y should be in km

    #Flow is in units km/h (Converted in function below)
    #time interval between any two data snapshots is 3 hours
    T =100 #Number of time stamps
    dt = 3 #Hours between time intervals
    i = int(np.floor(x/3))
    j = int(np.floor(y/3))
    u_x = u[i,j]
    v_y = v[i,j]
    #Converting position from index location to cm location
    #Multiplying by change in time dt and the flow vector at that position and time stamp
    x_new = (x) + dt*(u_x)
    y_new = (y) + dt*(v_y)


    return x_new, y_new
```

```python
def TimeStamp(x,y):
    #Function to track changing position of the object over time
    x_pos = []
    y_pos = []
    time = 0
    T =40 #Number of time stamps
    dt = 3 #Hours between time intervals
    for i in range(T):
        x_pos.append(x)
        y_pos.append(y)
        #Time (hours)
        time = (i + 1)*dt

        path = r"C:\Users\kalib\Documents\IDS.131\Pset5\Data/"
        u_file = path + str(i+1)+"u.csv"
        v_file = path +str(i+1)+"v.csv"

        u = np.loadtxt(open(u_file,"rb"), delimiter=",")
        v = np.loadtxt(open(v_file,"rb"), delimiter=",")

        #Converting the velocity to cm/s
        cmps = (25/0.9)
        u= np.multiply(u,cmps)
        v = np.multiply(v,cmps)
        #Converting the velocity then to km/h
        kmh = (3600/1e5)
        u= np.multiply(u,kmh)
        v = np.multiply(v,kmh)

        x, y  = update_position(x,y,u,v)

    return x_pos, y_pos, time
```

Testing procedure on random points and saving output to a dataframe

```python
def path_df(x_list, y_list):

    df = pd.DataFrame()

    for i in range(len(x_list)):
        x0 = x_list[i]
        y0 = y_list[i]

        x_pos, y_pos, time = TimeStamp(x0, y0)

        xname = 'x pos'+str(i)
        yname = 'y pos'+str(i)

        df[xname] = x_pos
        df[yname] = y_pos

    return df
```

Plotting resulting path

```python
fig1, ax1 = plt.subplots(figsize=(20, 20))
ph = ax1.imshow(np.flip(mask,0), cmap ='ocean')
S = ax1.streamplot(X, Y, Avg_u, Avg_v, density=5)

ax1.invert_yaxis()

df_km = df/3
num = int(df.shape[1]/2)
for i in range(num):
    x_col = 'x pos'+str(i)
    y_col = 'y pos'+str(i)
    plt.plot(df_km[x_col], df_km[y_col],'--', label='line with marker',linewidth=5)
```

Part B)

Generated a set of x,y coordinate points sampled from a multivariate gaussian distribution with mean (300km,1050km) to serve as the initial x,y coordinates. (10 different initial x,y coordinates chosen for each variance value)

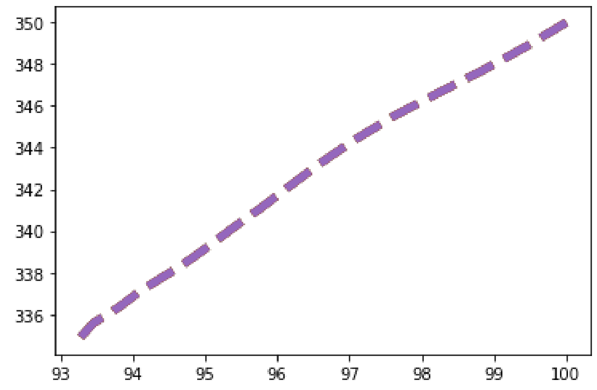
Looked at how the path of the objects change with varying variance from Var = 0, 25, 50, 75, 100

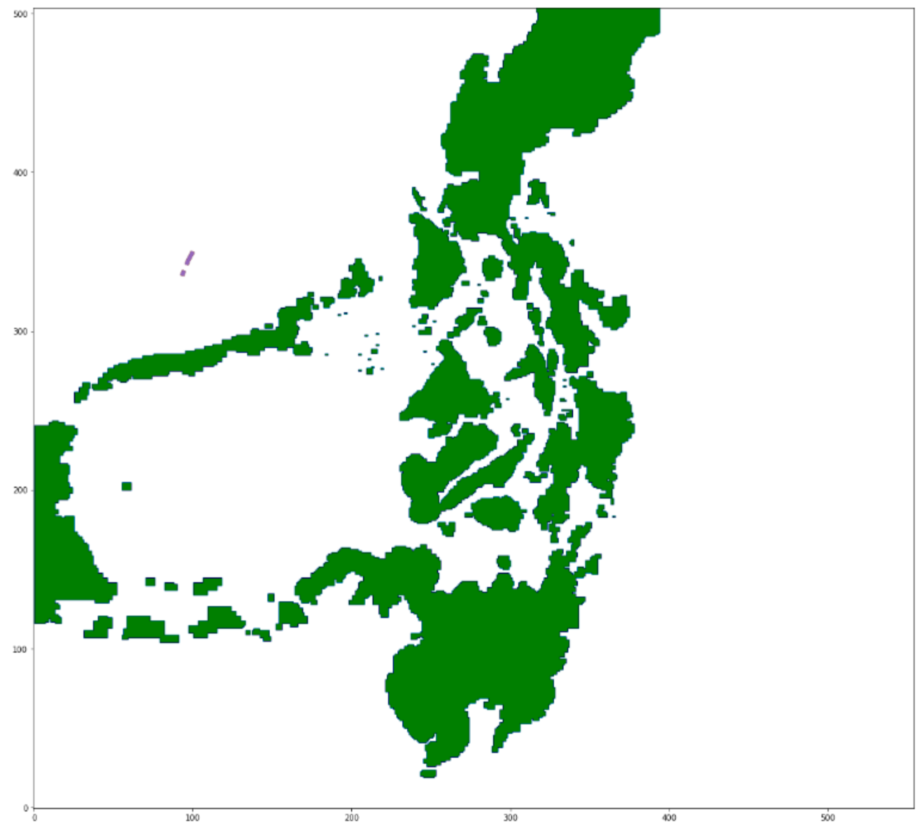
And corresponding Covariance matrix (Var*Identity matrix). This is an approximation of the covariance based on the assumption that X and Y are independent variables.

Variance	Time (hours)	Sample Mean of X Position (km)	Sample Mean of Y Position (km)
0	48	281.655	1009.62
25	48	283.67	1010.52
50	48	278.877	1009.52
75	48	279.77	1009.03
100	48	288.716	1012.14
0	72	279.79	1004.53
25	72	281.244	1004.7
50	72	276.136	1002.39
75	72	277.56	1003.59
100	72	285.176	1003.94
0	120	279.792	1004.53
25	120	280.899	1004.11
50	120	275.78	1001.25
75	120	277.32	1003.28
100	120	284.56	1002.36

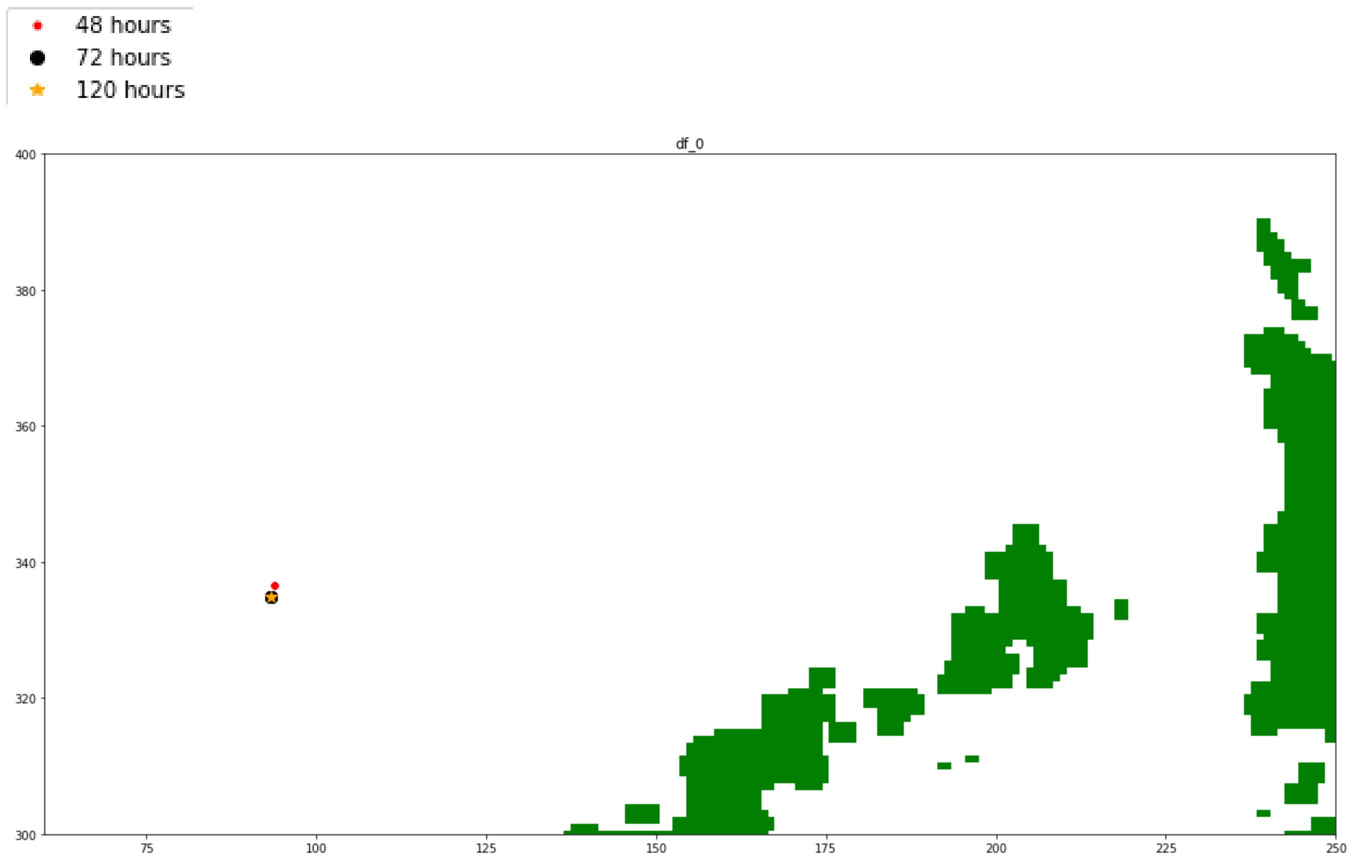
Variance = 0

All start at the same location and end at the same location at t=120 hours

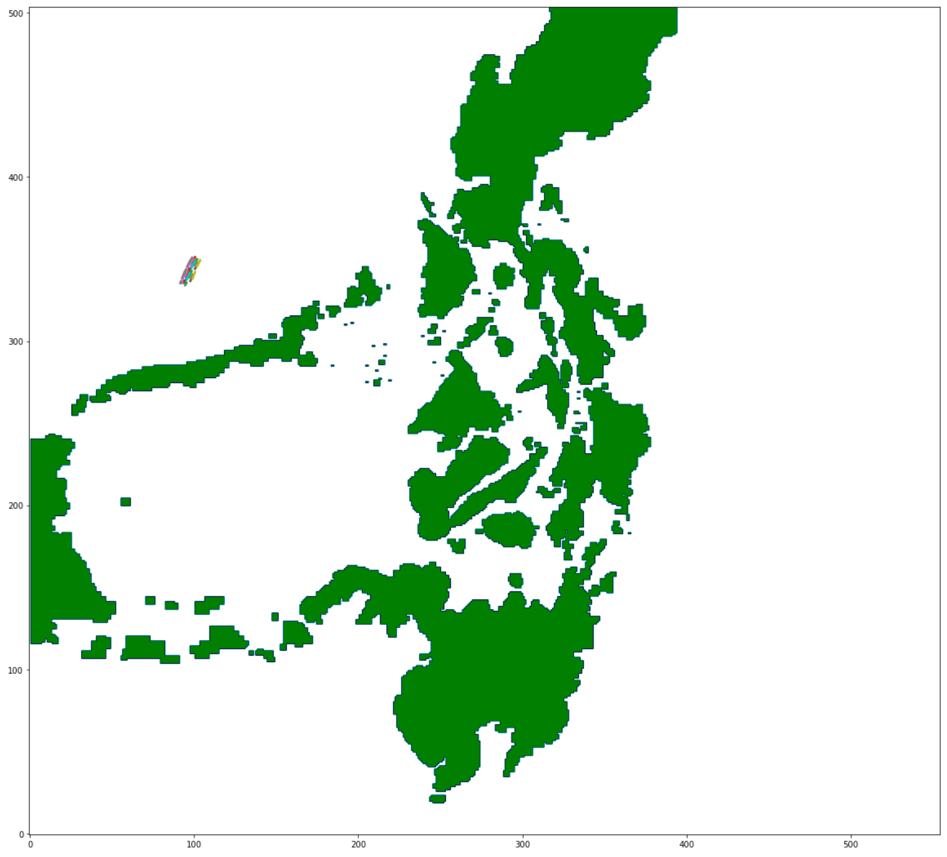




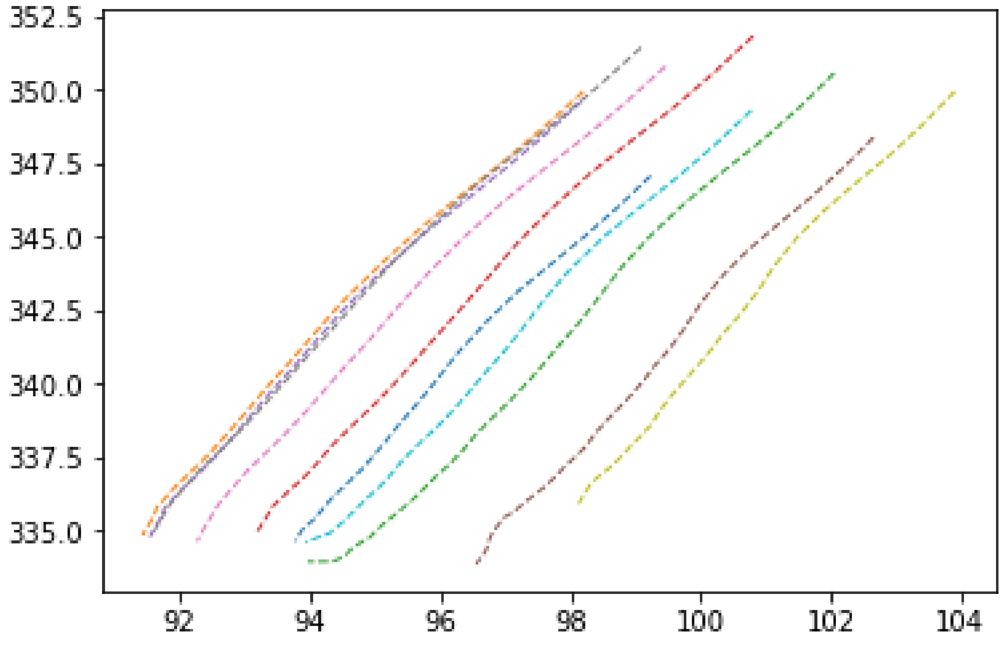
Variance = 0


48 hours
72 hours
120 hours
df_0
400
380
360
340
320
300
75
100
125
150
175
200
225
250

Variance = 25



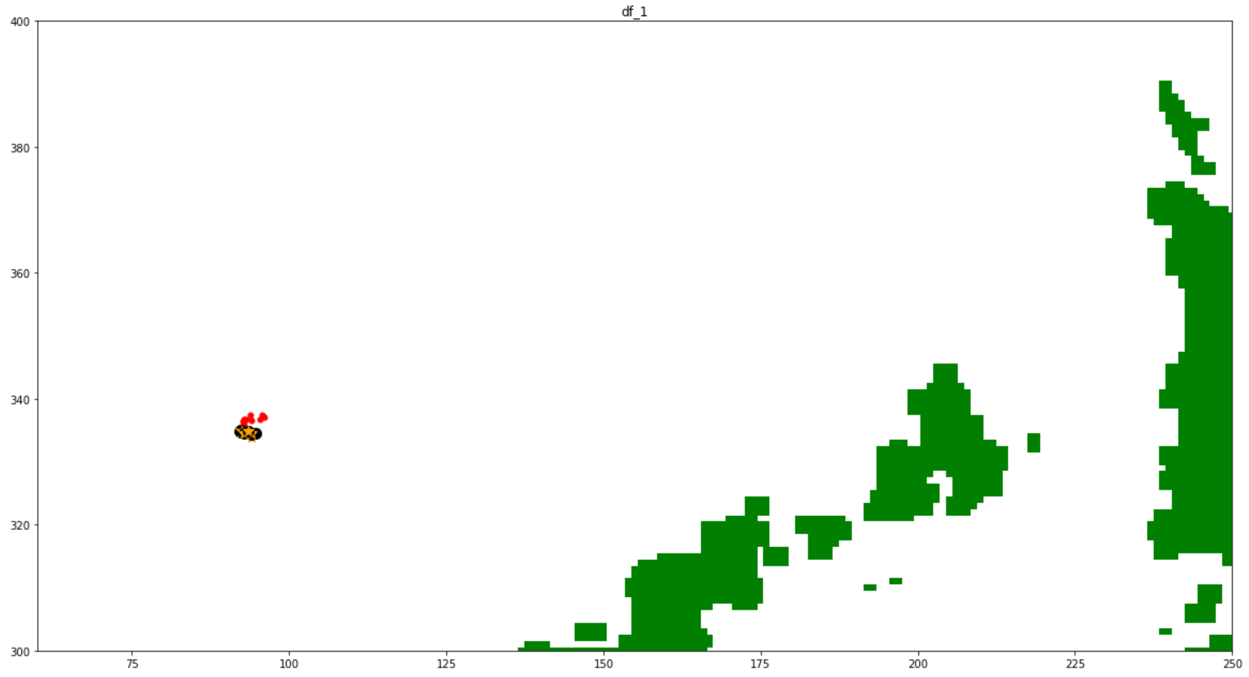
500
400
300
200
100
0
0
100
200
300
400
500



352.5
350.0
347.5
345.0
342.5
340.0
337.5
335.0
92
94
96
98
100
102
104

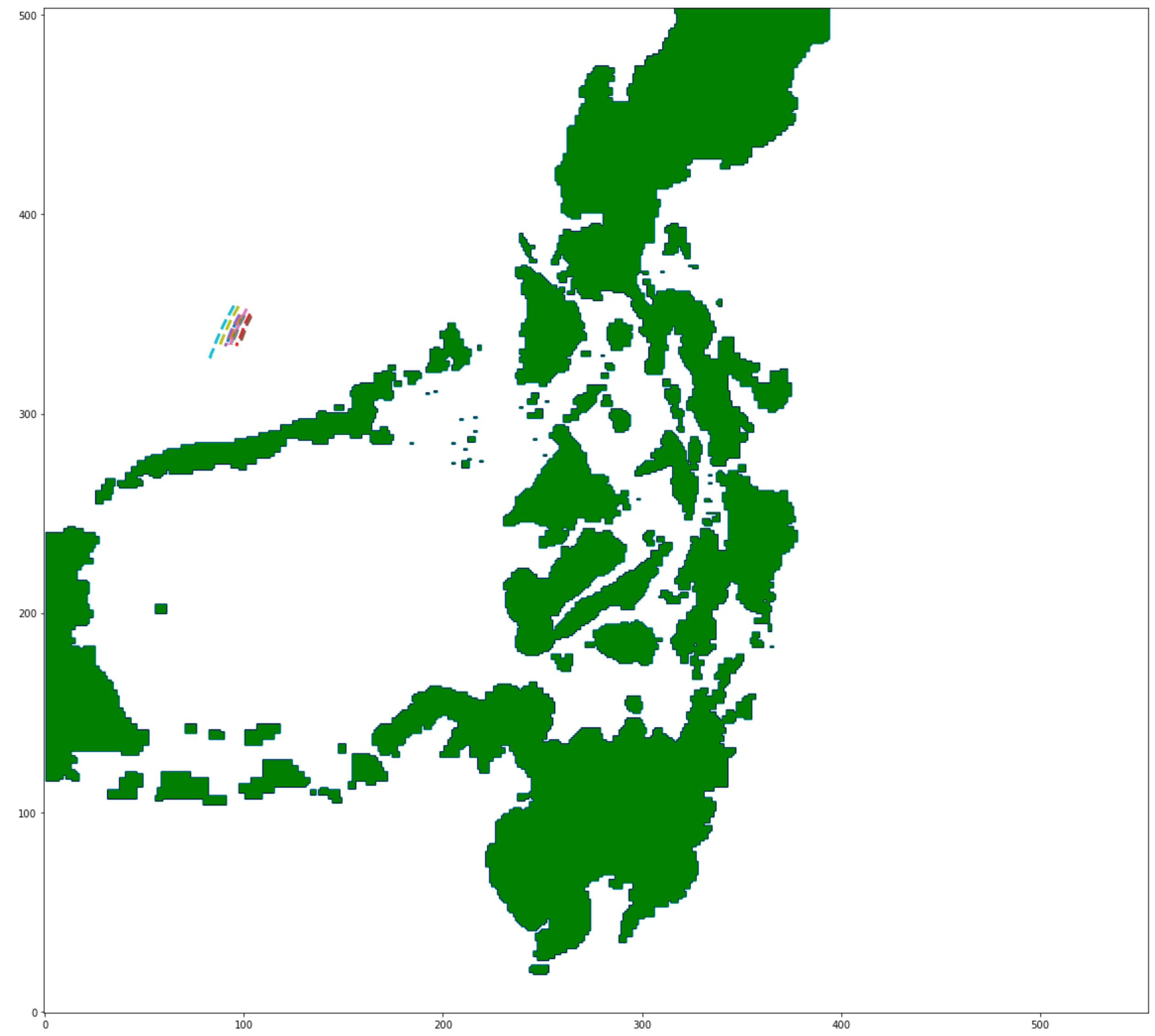
Variance = 25

● 48 hours
● 72 hours
★ 120 hours

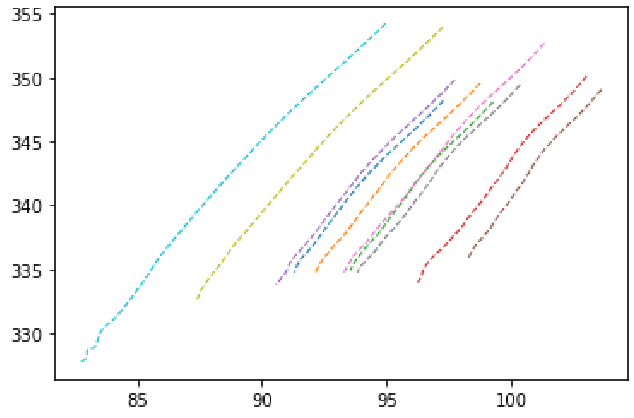


df_1
400
380
360
340
320
300
75
100
125
150
175
200
225
250

Variance = 50

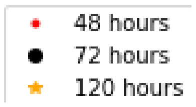


500
400
300
200
100
0
0
100
200
300
400
500

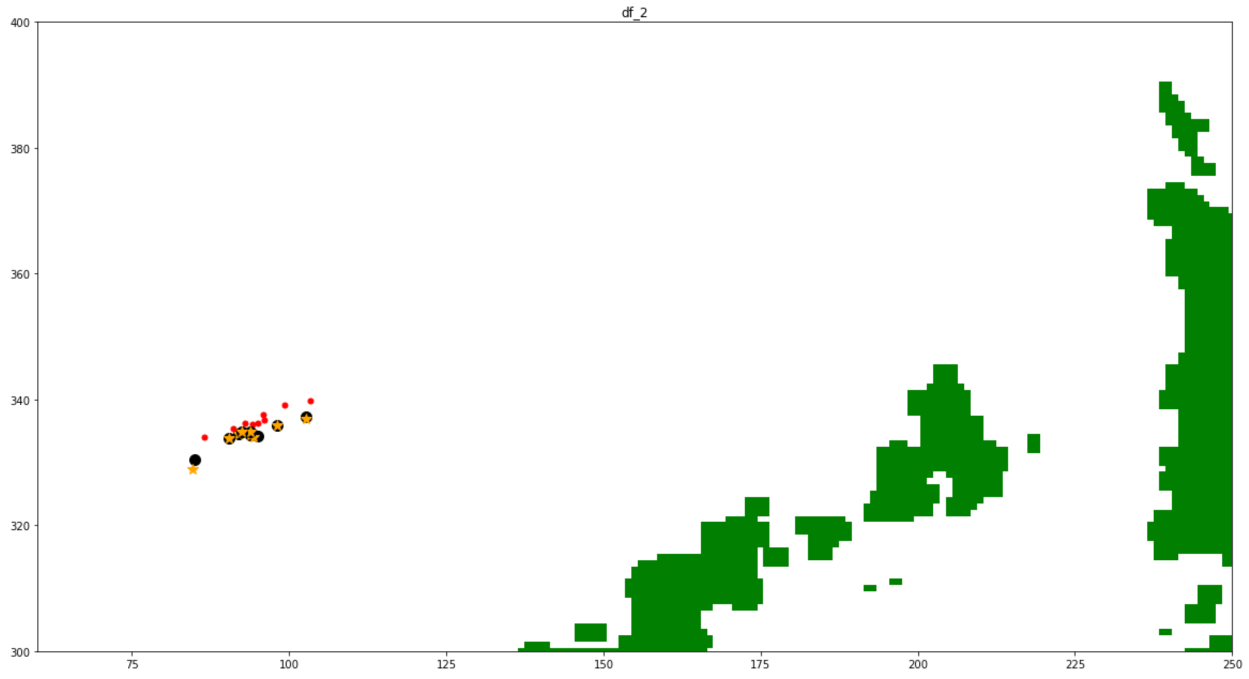


355
350
345
340
335
330
85
90
95
100

Variance = 50

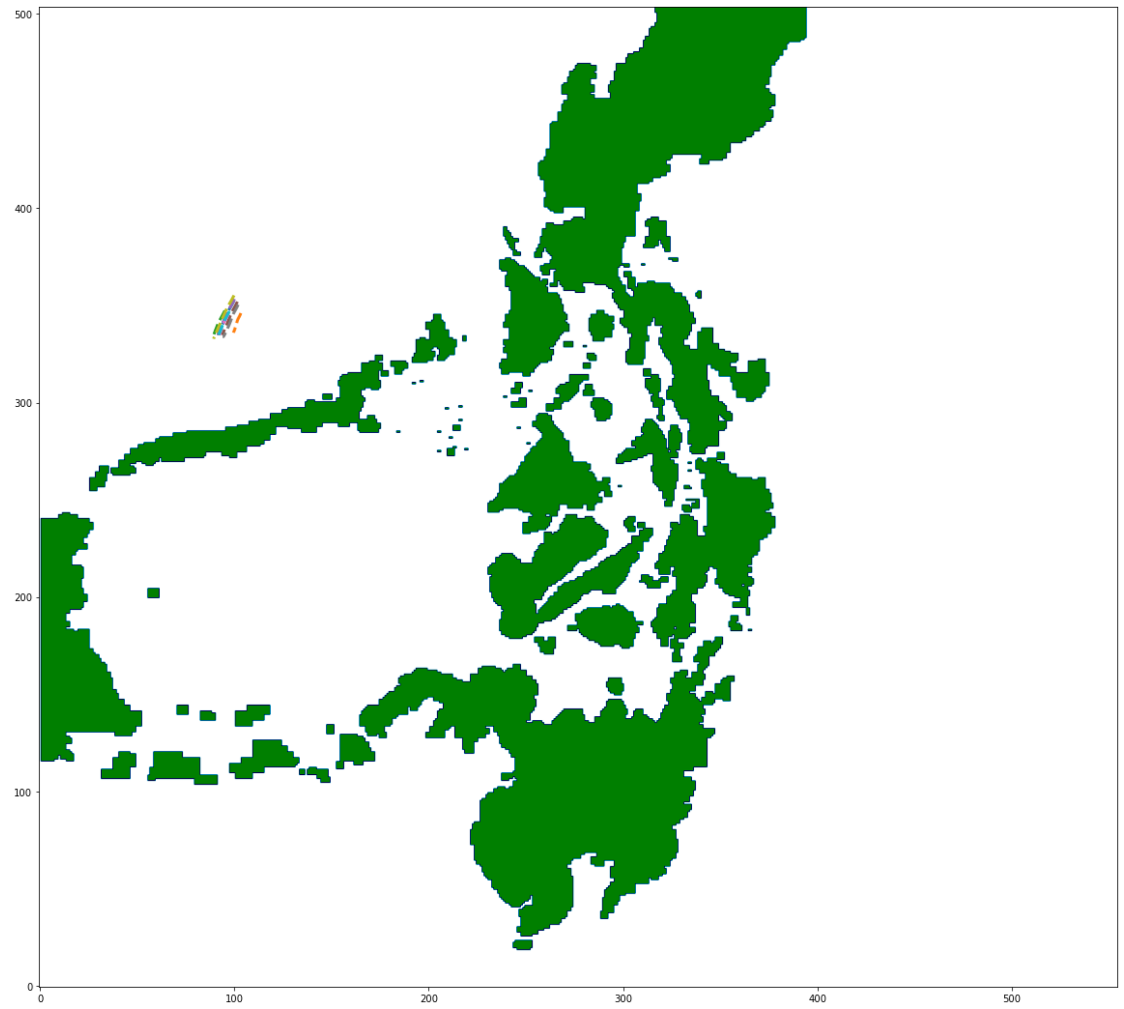


48 hours
72 hours
120 hours

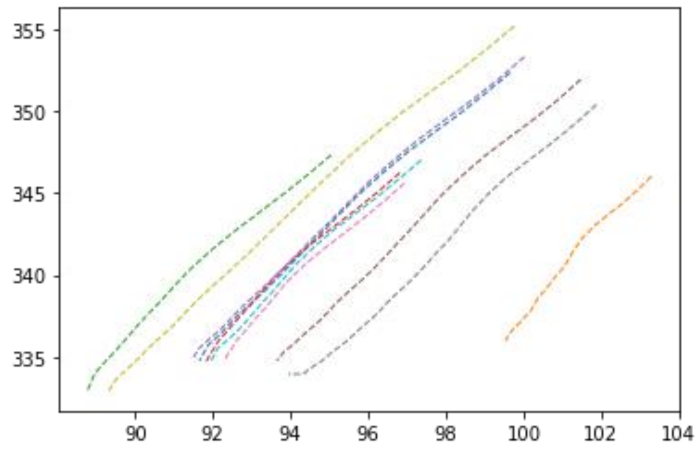


df_2
400
380
360
340
320
300
75
100
125
150
175
200
225
250

Variance = 75

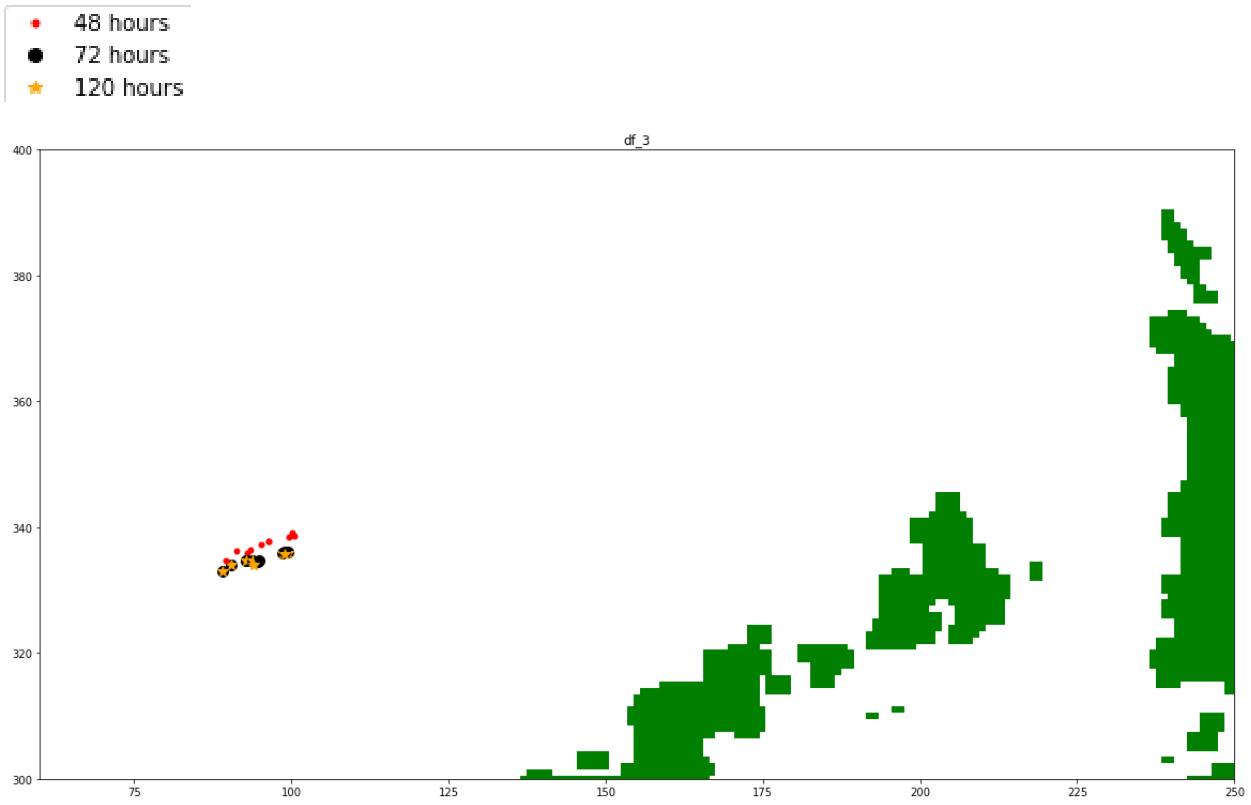


500
400
300
200
100
0
0
100
200
300
400
500

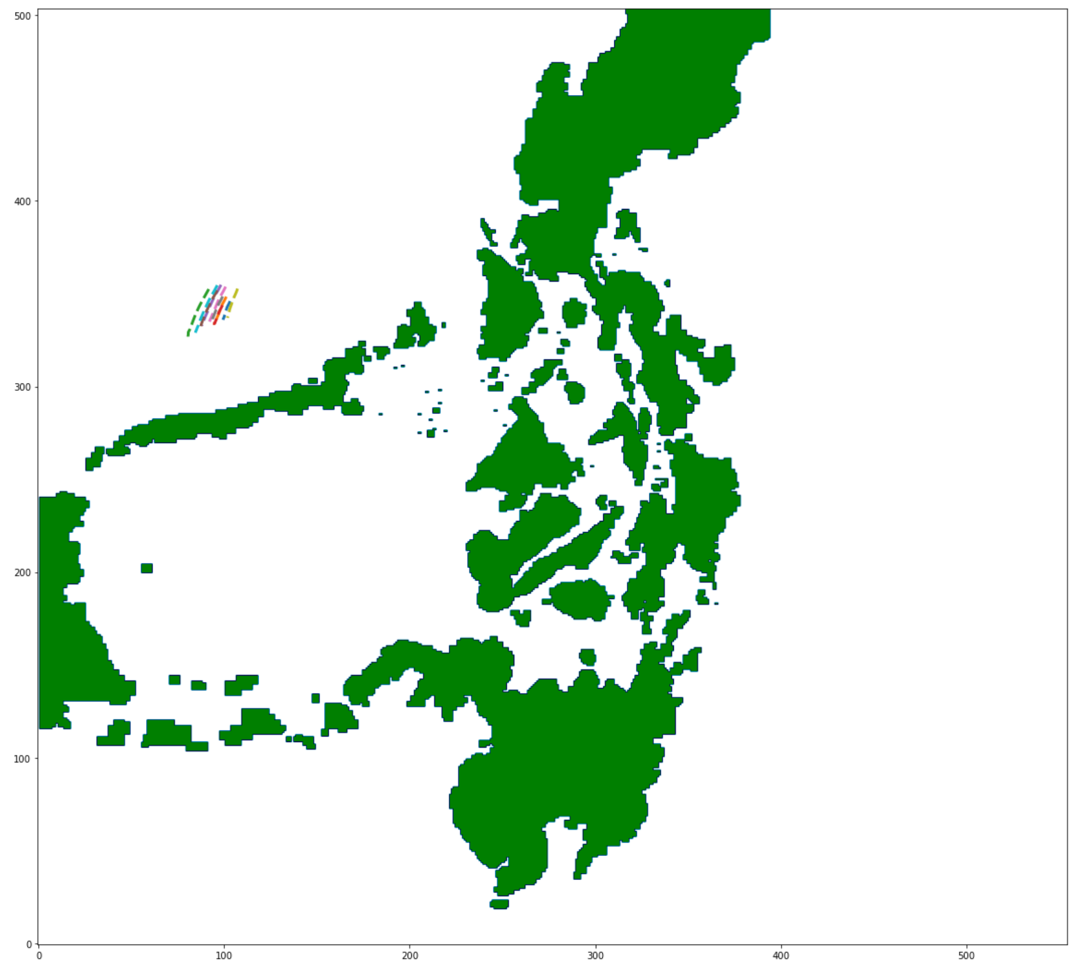


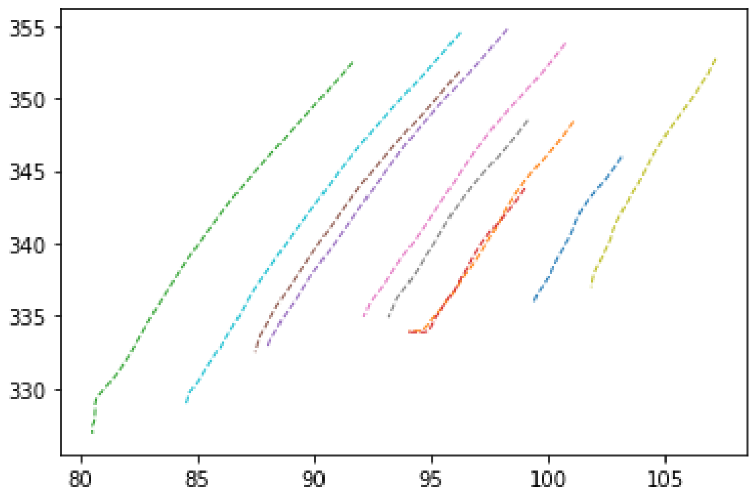
355
350
345
340
335
90
92
94
96
98
100
102
104

Variance = 75

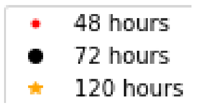

● 48 hours
● 72 hours
★ 120 hours
df_3
400
380
360
340
320
300
75
100
125
150
175
200
225
250

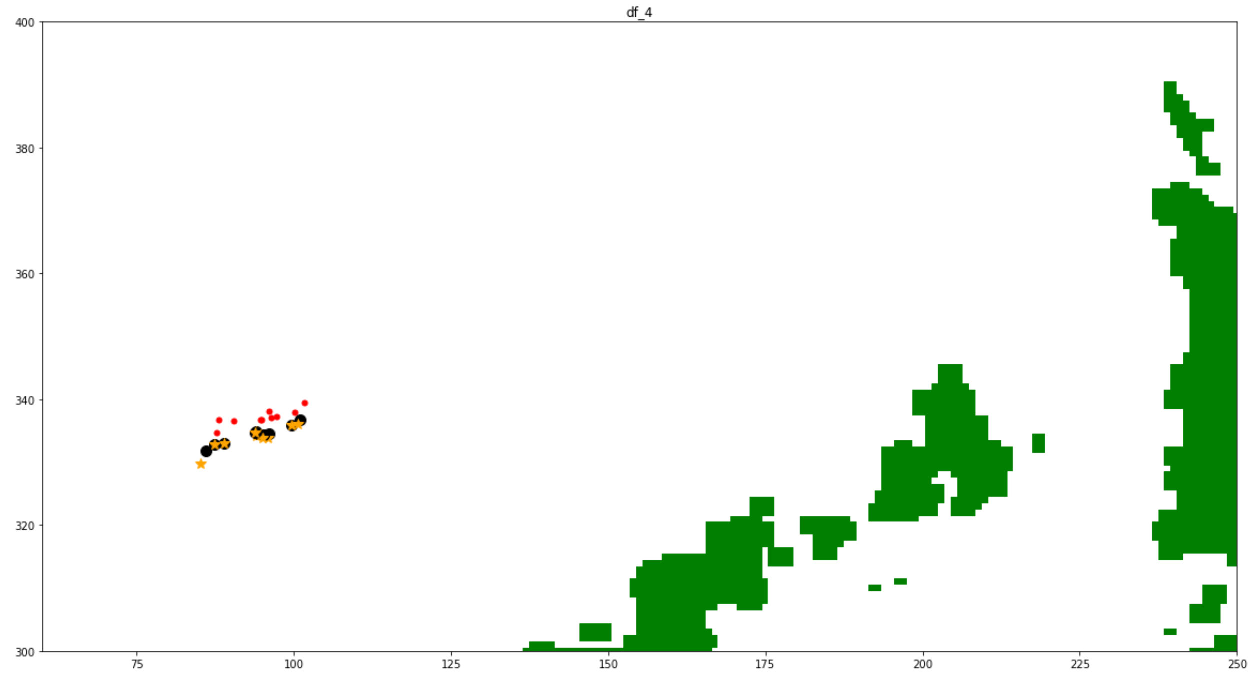
Variance = 100





Variance = 100





Code for Part B

****Utilized procedure established in Part A**

Testing for different variances (var) and sampling from a multivariate normal distribution.

```
def loopvariance(var):
    mean = (300, 1050)
    size = 10
    cov = np.multiply(var[i],np.identity(2))
    x_sample, y_sample = np.random.multivariate_normal(mean, cov, size).T
    df_name = path_df(x_sample,y_sample)
    return df_name
```

#Loop through variance values of 0, 25, 50, 75, 100

Selecting 10 x,y initial coordinates from a multivariate gaussian distribution and calculating the path each object would take over 120 hours.

```
start_time = datetime.now()

var = np.arange(start=0,stop=125,step=25 )

dict_of_df = {} # initialize empty dictionary

for i in range(len(var)):
    dict_of_df["df_{}".format(i)] =loopvariance(var)

end_time = datetime.now()
print('Duration: {}'.format(end_time - start_time))
```

```
Duration: 0:31:03.016012
```

```
#Plot results of all paths over time plotted with mask of land

for key in dict_of_df.keys():
    fig1, ax1 = plt.subplots(figsize=(20, 20))
    ph = ax1.imshow(np.flip(mask,0), cmap ='ocean')
    ax1.invert_yaxis()
    plt.xlim(60, 250)
    plt.ylim(300, 400)
    plt.title(key)
    #Converting back to index
    df_1_nonkm = (dict_of_df[key].iloc[15,:])/3
    df_2_nonkm = (dict_of_df[key].iloc[23,:])/3
    df_3_nonkm = (dict_of_df[key].iloc[39,:])/3

    num = int((dict_of_df[key].shape[1])/2)
    for i in range(num):
        x_col = 'x pos'+str(i)
        y_col = 'y pos'+str(i)
        plt.plot(df_1_nonkm[x_col],df_1_nonkm[y_col], '.', color='red', label ='48 hours',ms=10)
        plt.plot(df_2_nonkm[x_col],df_2_nonkm[y_col], 'o', color='black',label ='72 hours',ms=10)
        plt.plot(df_3_nonkm[x_col],df_3_nonkm[y_col], '*', color='orange', label ='120 hours',ms=10)
        #plt.legend()
```

```
#At time t =120 hours x coordinates
Var_0_time120_x = df_1.iloc[39,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_25_time120_x = df_2.iloc[39,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_50_time120_x = df_3.iloc[39,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_75_time120_x = df_4.iloc[39,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_100_time120_x = df_5.iloc[39,[0,2,4,6,8,10,12,14,16,18]].mean()
```

```
print(Var_0_time120_x, Var_25_time120_x, Var_50_time120_x, Var_75_time120_x, Var_100_time120_x)
```

```
279.792468 280.8991958191755 275.7837658921412 277.3190619758878 284.55656827089865
```

```
#At time t =48 hours x coordinates
Var_0_time48_x = df_1.iloc[15,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_25_time48_x = df_2.iloc[15,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_50_time48_x = df_3.iloc[15,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_75_time48_x = df_4.iloc[15,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_100_time48_x = df_5.iloc[15,[0,2,4,6,8,10,12,14,16,18]].mean()
print(Var_0_time48_x, Var_25_time48_x, Var_50_time48_x, Var_75_time48_x, Var_100_time48_x)
```

```
281.65506 283.67096931917547 278.8771115521413 279.7710693758878 288.7161176058986
```

```
#At time t = 72 hours x coordinates
Var_0_time72_x = df_1.iloc[23,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_25_time72_x = df_2.iloc[23,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_50_time72_x = df_3.iloc[23,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_75_time72_x = df_4.iloc[23,[0,2,4,6,8,10,12,14,16,18]].mean()
Var_100_time72_x = df_5.iloc[23,[0,2,4,6,8,10,12,14,16,18]].mean()
print(Var_0_time72_x, Var_25_time72_x, Var_50_time72_x, Var_75_time72_x, Var_100_time72_x)
```

```
279.792468 281.2442285191754 276.1363217521412 277.5639489758878 285.1758233058987
```