Written Report – 6.419x Module 5

Evan Woods

Contents

Part 1. Ocean Flow

Problem 2

In this problem, we will try to identify areas in the Philippine Archipelago with long-range correlations. Your task is to identify two places on the map that are not immediately next to each other but still have some high correlation in their flows. Your response should be the map of the Archipelago with the two areas marked (e.g., circled). You claim that those two areas have correlated flows. Explain how you identified the correlation between the flows of those two areas.

Solution:

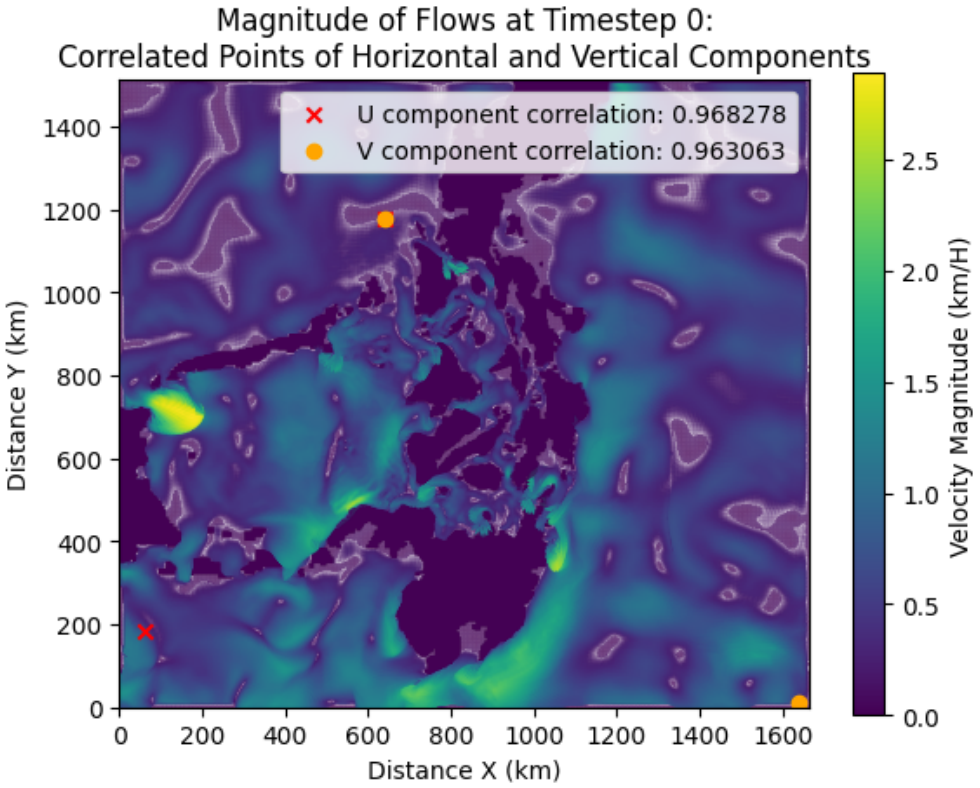

Figure 1: Correlated Points in the Phillipine Archipelago Region with Respect to the Horizontal (U-component) and Vertical (V-component) of the Flow Magnitude.

In Figure 1, correlated points with respect to the horizontal U-component of the flow magnitude are marked with a red 'X'. Correlated points with respect to the vertical V-component are marked with an orange circle. Of the correlated points along the horizontal component of the flow magnitude, the points are at locations (63 km, 183 km) and (642 km, 1179 km) labeling from left to right on the plot. These points have a correlation of 0.968278 with respect to the horizontal component of the magnitude of the flow at these locations. The vertical component contains two correlated points at locations (642 km, 1179 km) and (1641 km, 12 km) from left to right on the plot, respectively. These points have a correlation of 0.963063 with respect to the vertical component of the magnitude of the flow at these locations. These points are calculated to be

1152 km apart for the correlated points with respect to the horizontal component and 1536 km apart for the correlated points with respect to the vertical component. It is worth noting that these points seem to be near stationary with respect to the magnitude of the flow direction upon observation of the points in Figure 1. This is intuitive as stationary, calm water velocities are more likely to be correlated than flow velocities that are less laminar and more turbulent across time. These points are correlated with respect to their respective horizontal and vertical component of the flow magnitude for all time steps over the course of 300 hours.

To identify these pairs of points, I loaded each csv of the U and V components of the magnitude of flows into arrays with shape (100, 504, 555) corresponding to the (time, y-axis, and x-axis). I also read the mask and flipped the mask vertically to later identify only water-based flow locations as shown in Listing 1.

Listing 1: Loading matrices U, V, and mask.

```
# Load the first file to determine shape
sample_u = np.loadtxt(os.path.join(DATA_DIR, "1u.csv"), delimiter=",")
ny, nx = sample_u.shape

# Initialize 3D arrays
U = np.zeros((NUM_TIMESTEPS, ny, nx))
V = np.zeros((NUM_TIMESTEPS, ny, nx))

# Load each timestep into the 3D array
for t in range(1, NUM_TIMESTEPS + 1):
   u_path = os.path.join(DATA_DIR, f"{t}u.csv")
   v_path = os.path.join(DATA_DIR, f"{t}v.csv")
   U[t-1] = np.loadtxt(u_path, delimiter=",")
   V[t-1] = np.loadtxt(v_path, delimiter=",")

mask = pd.read_csv('OceanFlow/mask.csv', header=None).values # Shape: (504,
    555)
# The mask is upside-down
mask = np.flipud(mask)
```

For ease of use, I transposed and reshaped the U and V matrices such that each point was with respect to every 100 timesteps. This would later assist in identifying correlations between points. I flattened the mask into the same transposed and reshaped shape of the U and V matrices, and identified the locations where the indices of the mask contain values indicating water. I then used these indices to index into the reshaped U and V matrices to identify locations containing water only. I then used these indices to retain the corresponding x and y coordinates with respect to the original U, and V coordinates by dividing the water indices by the length of the y-axis of the U matrix for the corresponding y-coordinates and taking the modulus of the length of the U matrix of the water indices for the corresponding x-coordinates. I then removed all zero valued flow regions still present in the water-only regions so as to find meaningful correlated flows of moving water, and used this calculated index to index the U and V matrices of water only data and the coordinates corresponding to the original U and V matrices. This process is shown in Listing 2.

Listing 2: Reshaping U, V, and mask, & retaining original coordinates for the reshaped data.

```
# Step 1: Reshape U to (y*x, time)
U_reshape = U.transpose(1, 2, 0).reshape(504*555, 100)
V_reshape = V.transpose(1, 2, 0).reshape(504*555, 100)

# Step 2: Flatten the mask (y,x) -> (y*x,)
mask_flat = mask.flatten() # shape (504*555,)
```

```
# Step 3: Keep only rows where mask == 1 (i.e., water)
water_indices = np.where(mask_flat == 1)[0]

U_water_data = U_reshape[water_indices] # shape (n_water_points, 100)
V_water_data = V_reshape[water_indices]
# Step 4: Find corresponding (y, x) coordinates
y_coords = water_indices // 555
x_coords = water_indices % 555

# Step 5: Build the coordinate matrix
coords = list(zip(y_coords, x_coords))

# Check where all values are exactly zero across all timesteps
nonzero_mask = ~(np.all(U_water_data == 0, axis=1) | np.all(V_water_data ==
    0, axis=1))

# Apply mask to U, V, and coords
U_water_data = U_water_data[nonzero_mask]
V_water_data = V_water_data[nonzero_mask]
coords = [c for i, c in enumerate(coords) if nonzero_mask[i]]
```

I next identified a random sample of points that are a sufficient distance from each other. The algorithm details of this process are in Listing 3. The algorithm will accept a number of points to identify, the coordinates of the points, and the percent of the maximum distance that the points must be in order to qualify as being long-distance pairs of coordinates. The maximum distance is calculated as the Euclidean Distance between the lower-left hand coordinate and the upper right hand coordinate of the original U matrix (points (0,0) and (504, 555) respectively). The required minimum distance is a defined 80% of this euclidean distance maximum. Next, the first point is randomly selected from the list of coordinates, and candidate points that have not yet been selected are identified amongst the difference between the set of the range of the possible selected points and the set of the already selected indices. These candidate points are shuffled. For each index in the list of candidates, the euclidean distance is calculated between that pair of coordinates and the coordinates of all other already selected points. If the euclidean distance between that point and all other selected points are greater than or equal to the required minimum distance, then the point is appended to the selected indices. Otherwise, if a candidate cannot be identified, then the required minimum distance is slightly reduced to 95% of the original distance requirement to identify a candidate point. Using this algorithm, all 100 selected points are a required minimum distance from each other based upon a randomly selected seed point and all points are of water-based coordinates with a non-zero component of flow respective to the horizontal or vertical corresponding matrix.

Listing 3: Identifying long-range distances.

```
def select_distant_indices(coords, num_points=100, distance_percent=0.8):
"""Select `num_points` indices such that each is `distance_percent` away
    from the others."""
n = len(coords)
max_possible_distance = euclidean_distance((0,0), (504, 555))
required_min_distance = distance_percent * max_possible_distance

# Pick first point randomly
selected_indices = [random.randint(0, n - 1)]
```

```
    while len(selected_indices) < num_points:
        candidates = list(set(range(n)) - set(selected_indices))
        random.shuffle(candidates)
        for idx in tqdm(candidates, desc="Selecting␣Indices...", ascii=""):
            if all(euclidean_distance(coords[idx], coords[other]) >=
                required_min_distance for other in selected_indices):
                selected_indices.append(idx)
                break
        else:
            # If we cannot find a candidate (too strict requirement), relax the
                minimum distance slightly
            required_min_distance *= 0.95

    return selected_indices
```

The correlation matrix is next computed by indexing using the long-range-indices into the corresponding matrix of horizontal and vertical components and calculating the correlation of these coordinates with respect to all 100 time steps using a pearson correlation as shown in Listing 4.

Listing 4: Caluclating the correlation matrix for the horiztonal U-component of the flow magnitude.

```
# Compute correlation matrix between rows
n_rows = U_water_data[long_range_indices].shape[0]
correlation_matrix = np.zeros((n_rows, n_rows))

for i in range(n_rows):
    for j in range(n_rows):
        correlation_matrix[i, j], _ = stats.pearsonr(U_water_data[
            long_range_indices][i], U_water_data[long_range_indices][j])

# Compute correlation matrix between rows
n_rows = V_water_data[long_range_indices].shape[0]
v_correlation_matrix = np.zeros((n_rows, n_rows))

for i in range(n_rows):
    for j in range(n_rows):
        v_correlation_matrix[i, j], _ = stats.pearsonr(V_water_data[
            long_range_indices][i], V_water_data[long_range_indices][j])
```

The corresponding correlation matrices are then observable in Figures 2 and 3.

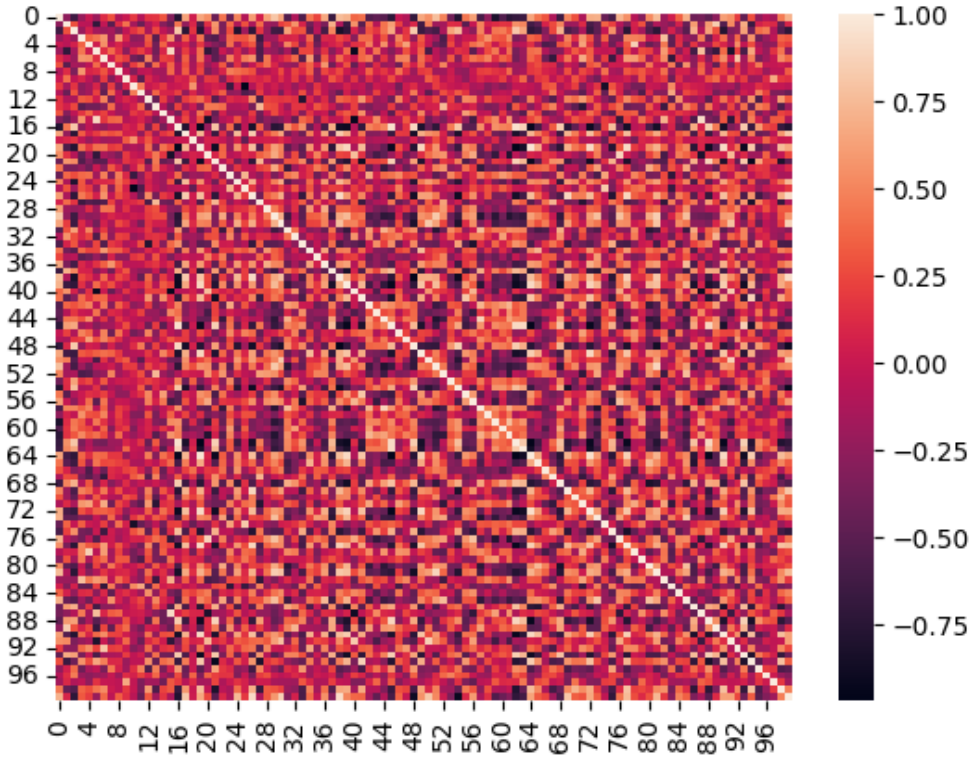

Figure 2: Correlated Matrix with Respect to 100 Randomly Selected Points from the Horizontal (U-component) of the Flow Magnitude.

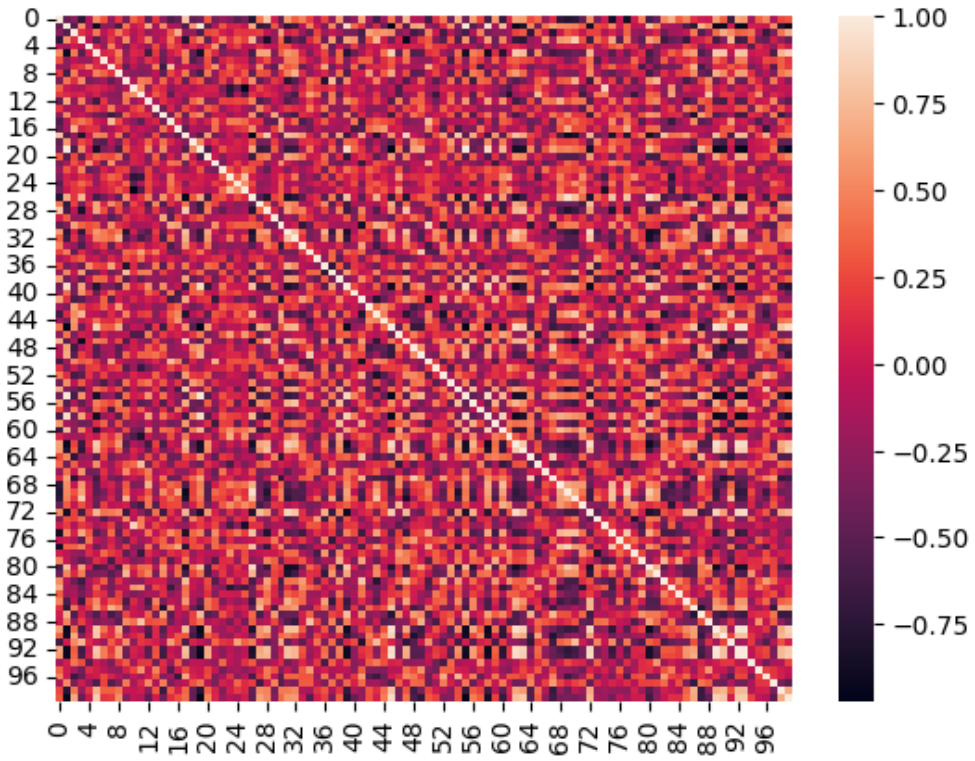

Figure 3: Correlation Matrix with Respect to 100 Randomly Selected Points from the Vertical (V-component) of the Flow Magnitude.

The indices of the identified points of correlated flows are then thresheld above an arbitrary value of 0.95 to extract points that are highly correlated. Indices that are perfectly correlated, such as correlations with the same points themselves, are removed and the remaining list of points are pruned of points that are considered duplicates, e.g. point (39, 40) is the same as point (40, 39). This is shown in Listing 5.

Listing 5: Removing duplicate points and perfect correlations.

```
v_indices = np.argwhere(v_correlation_matrix > 0.95)
indices = np.argwhere(correlation_matrix > 0.95)
v_indices = v_indices[v_indices[:, 0] != v_indices[:, 1]]
indices = indices[indices[:, 0] != indices[:, 1]]
```

The corresponding correlation values are identified for these indices in the correlation matrices and the first point pairs and corresponding correlations are plot as shown in Figure 1.

Problem 3a

We assume that the velocity of a particle in the ocean, with certain coordinates, will be determined by the corresponding water flow velocity at those coordinates. Implement a procedure to track the position and movement of multiple particles as caused by the time-varying flow given in the data set. Explain the procedure, and show that it works by providing examples and plots.

Draw particle locations uniformly at random across the entire map, do not worry if some of them are placed on land. Simulate the particle trajectories for 300 hours and provide a plot of the initial state, a plot of the final state, and two plots at intermediate states of the simulation. You may wish to draw colors at random for your particles in order to help distinguish them.

Solution:

In order to simulate particle trajectories, I first identified 10 random points from the Magnitude matrix at time zero which was comprised of the sum of squares of the U and V matrices. A list comprehension was used as well as the random library as shown in Listing 6. The particle coordinates were then zipped together into a tuple and placed into a pandas dataframe. I renamed the columns per the dimension and timestep of the particles. Then for each timestep, I indexed into the U component and V component of the flow magnitude with respect to the coordinates of the previous timestep. The horizontal component was added to the previous coordinate for the x dimension and the vertical component was added to the previous coordinate for the y dimension. I then created unique colors by using a color map from matplotlib and used a dictionary comprehension to create a unique color mapping for each particle.

Listing 6: Defining Random Particles and Simulating Movement.

```
# Draw particle directories uniformly at random across the entire map
x_particle_coordinates = [random.randint(0, MAG[0, :, :].shape[1]) for _ in
    range(10)]
y_particle_coordinates = [random.randint(0, MAG[0, :, :].shape[0]) for _ in
    range(10)]
particle_coordinates = list(zip(y_particle_coordinates,
   x_particle_coordinates))
particle_coordinate_df = pd.DataFrame(particle_coordinates)
particle_coordinate_df.rename(columns={0: "y", 1: 'x'}, inplace=True)
particle_coordinate_df.rename(columns={"y": "y_0", "x": 'x_0'}, inplace=
   True)
particle_coordinate_df["Particle"] = range(10)
particle_coordinate_df.set_index("Particle", inplace=True)
for t in range(1, 100):
   particle_coordinate_df[f"y_{t}"] = particle_coordinate_df[f"y_{t-1}"] +
      np.round(V[t, particle_coordinate_df[f"y_{t-1}"].values,
      particle_coordinate_df[f"x_{t-1}"].values]).astype(int)
   particle_coordinate_df[f"x_{t}"] = particle_coordinate_df[f"x_{t-1}"] +
      np.round(U[t, particle_coordinate_df[f"y_{t-1}"].values,
      particle_coordinate_df[f"x_{t-1}"].values]).astype(int)

num_particles = 10

cmap = plt.get_cmap('tab20') # tab20 has 20 distinct colors; better for
   categorical data
colors = [cmap(i % 20) for i in range(num_particles)] # wrap around if >20
   particles

# Create a mapping: particle index -> color
particle_colors = {particle_index: colors[i] for i, particle_index in
   enumerate(particle_coordinate_df.index)}
```

I then plot the magnitude of the flow for the Phillipine Archipellago and scaled each point by the grid spacing of 3 kilometers as observable in Figure 4. Each particle was assigned an arrow that was scaled by an arbitrary value of 100 in order to visibly see the velocity, and the velocity was determined by the current timestep of the particle with respect to the particle coordinate in either the U or V matrix for the horizontal and vertical components respectively. These flow vectors were scaled by the grid spacing of 3 kilometers and colored distinctly. I adjusted the timestep to visualize the particle movement and at timestep ¿ 0, I plot smaller points of every point within the dataframe up until that current timestep. This allows for the trace of the particle's history, the particles current position, and the particles future trajectory to be visibly observed in Figures 5, 6, and 7.

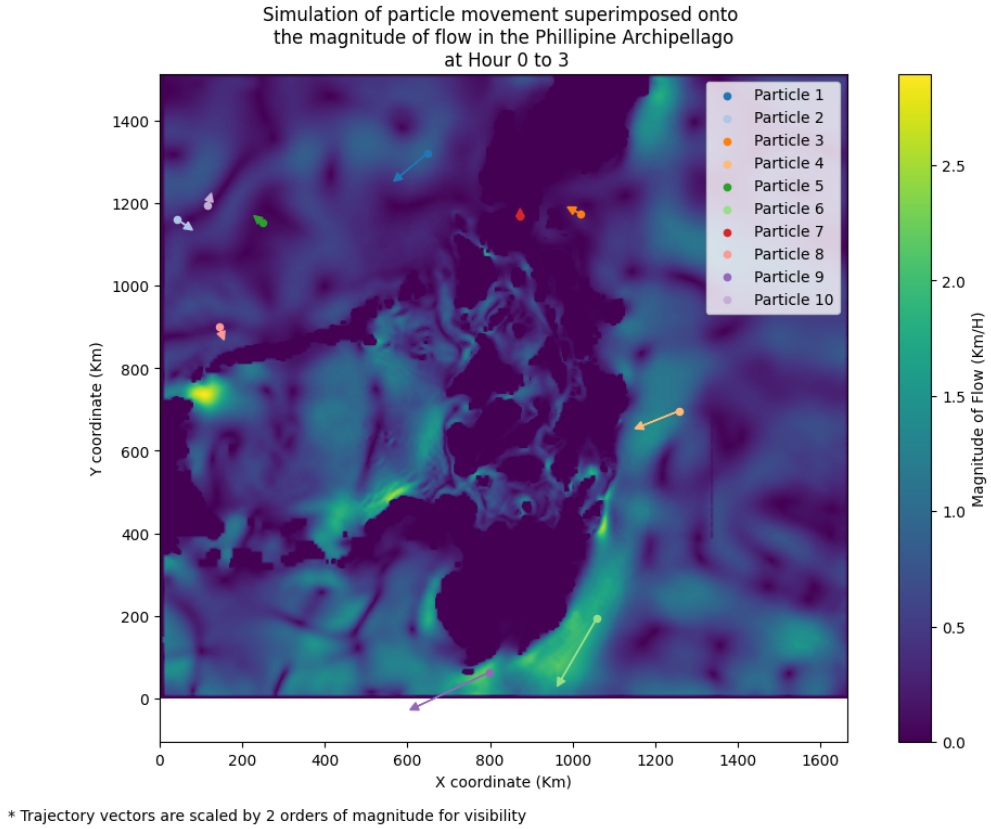

Figure 4: Simulation of Particle Movement at Timestep 0.

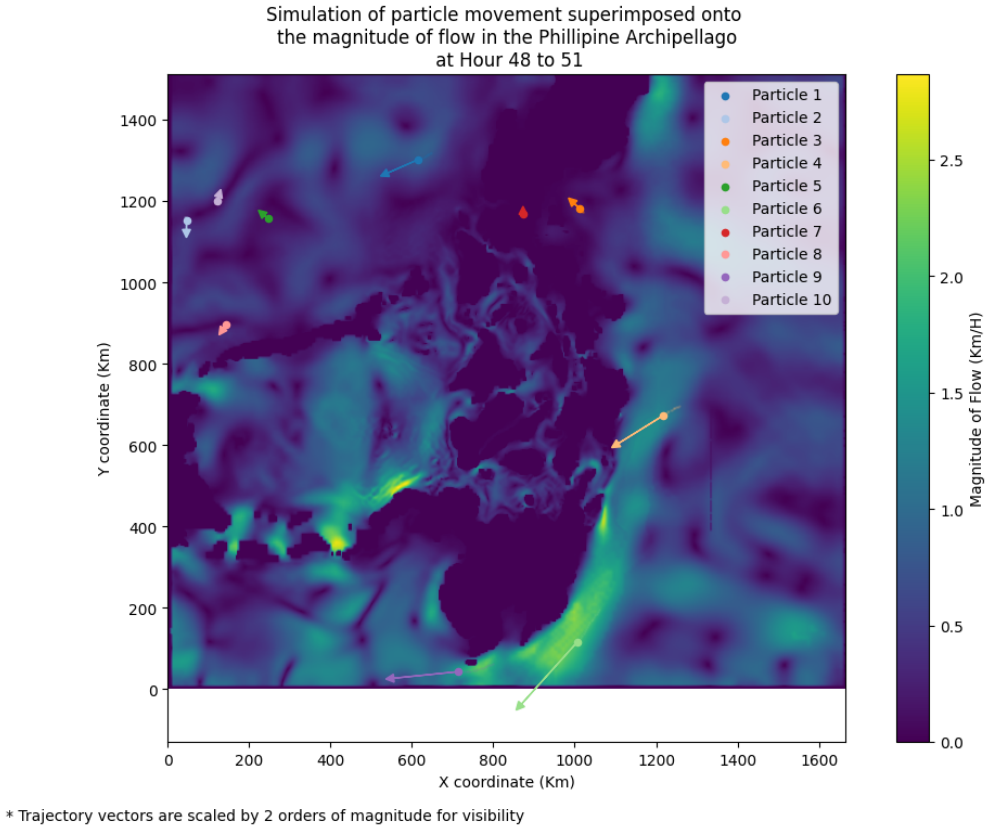

Figure 5: Simulation of Particle Movement at Timestep 48.

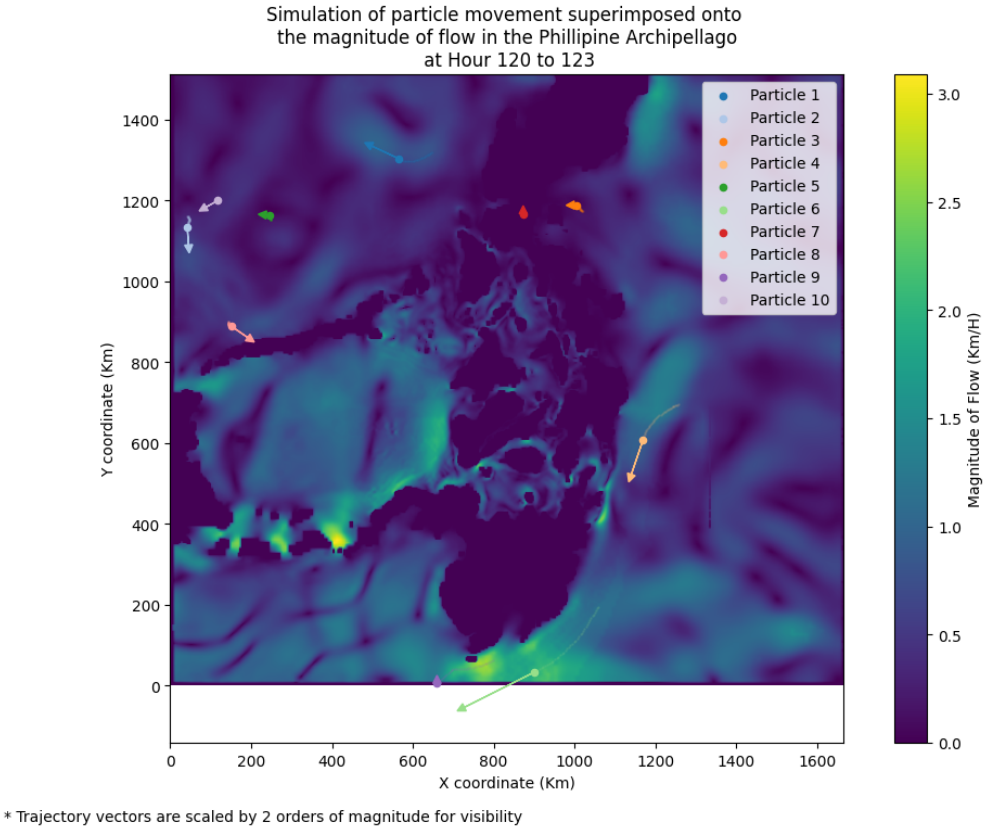

Figure 6: Simulation of Particle Movement at Timestep 120.

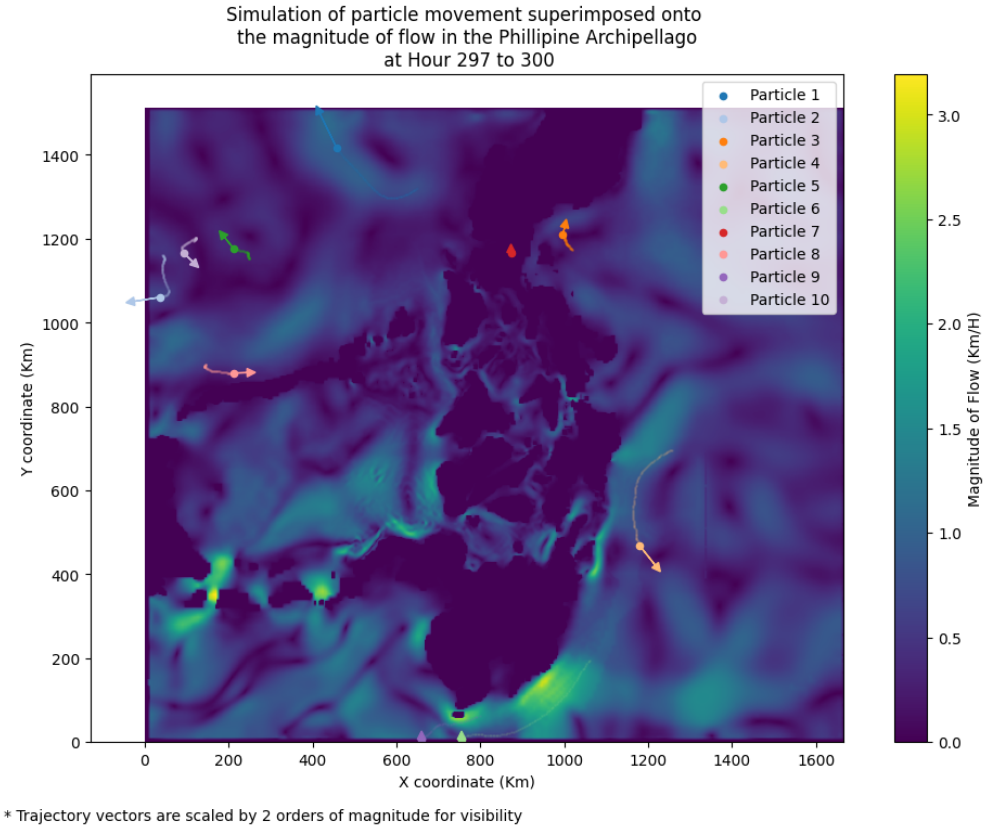

Figure 7: Simulation of Particle Movement at Final Timestep.

Problem 3b

A (toy) plane has crashed north of Palawan at T = 0. The exact location is unknown, but data suggests that the location of the crash follows a Gaussian distribution with mean (100, 350) (namely (300km, 1050km)) with variance σ^2. The debris from the plane has been carried away by the ocean flow. You are about to lead a search expedition for the debris. Where would you expect the parts to be at 48 hrs, 72 hrs, and 120 hrs? Study the problem by varying the variance of the Gaussian distribution. Either pick a few variance samples or sweep through the variances if desired. (Hint: Sample particles and track their evolution.)

Solution:

In Figures 8, 9, and 10 the movement of debris particles have been simulated. The particles are each sampled from a normal Gaussian distribution with the same mean of 100 for the x coordinate and 350 for the y coordinate. To justify a realistic variety of unknown possible variances, the maximum variance across time for the horizontal and vertical component of the magnitude in the Philippine Archipellago were computed. 10 random particles were sampled from a uniform distribution with minimum variance of 0 for both x and y and a maximum variance of 6 for x and 4 for y. This process is visible in Listing 7. The coordinates of each simulated debris piece for the beginning of the hour are listed in the legend and each particle is color-coded appropriately. Note that the coodinates are in kilometers and the plot has been zoomed-in to clearly indicate the trajectory of the simulated debris.

Listing 7: Simulating Random Debris Sampled from a Normal Gaussian Distribution with a Variety of Variances.

```
# Create unique colors for each particle
num_particles = 10

x_index = 100
y_index = 350

# Mean positions for each particle (could also be different)
mean_x = 100
mean_y = 350

# Different variances for each particle
variance_x = np.random.uniform(0, 6, size=num_particles)
variance_y = np.random.uniform(0, 4, size=num_particles)

# Standard deviations are sqrt(variance)
std_x = np.sqrt(variance_x)
std_y = np.sqrt(variance_y)

# Sample particle positions
sampled_x = np.random.normal(loc=mean_x, scale=std_x)
sampled_y = np.random.normal(loc=mean_y, scale=std_y)

# Object Structure: [Particle] [Time]
sampled_y = sampled_y.astype(np.float64).reshape(-1, 1) * GRID_SPACING_KM
sampled_x = sampled_x.astype(np.float64).reshape(-1, 1) * GRID_SPACING_KM
sampled_y = torch.from_numpy(sampled_y).to('cuda')
```

```
sampled_x = torch.from_numpy(sampled_x).to('cuda')


n_particles = sampled_y.shape[0]

for t in range(1, 100):
    new_sample_y = sampled_y[:, t-1] + V[t - 1, (sampled_y[:, t-1] / 3).int
        (), (sampled_x[:, t-1] / 3).int()] # First Hour
    new_sample_x = sampled_x[:, t-1] + U[t - 1, (sampled_y[:, t-1] / 3).int
        (), (sampled_x[:, t-1] / 3).int()]

    new_sample_y = new_sample_y + V[t - 1, (new_sample_y / 3).int(), (
        new_sample_x / 3).int()] # Second Hour
    new_sample_x = new_sample_x + U[t - 1, (new_sample_y / 3).int(), (
        new_sample_x / 3).int()] # Second Hour

    new_sample_y = new_sample_y + V[t - 1, (new_sample_y / 3).int(), (
        new_sample_x / 3).int()] # Third Hour
    new_sample_x = new_sample_x + U[t - 1, (new_sample_y / 3).int(), (
        new_sample_x / 3).int()] # Third Hour

    new_sample_y = new_sample_y.reshape(n_particles, 1)
    new_sample_x = new_sample_x.reshape(n_particles, 1)

    sampled_y = torch.cat([sampled_y, new_sample_y], axis=1)
    sampled_x = torch.cat([sampled_x, new_sample_x], axis=1)
```

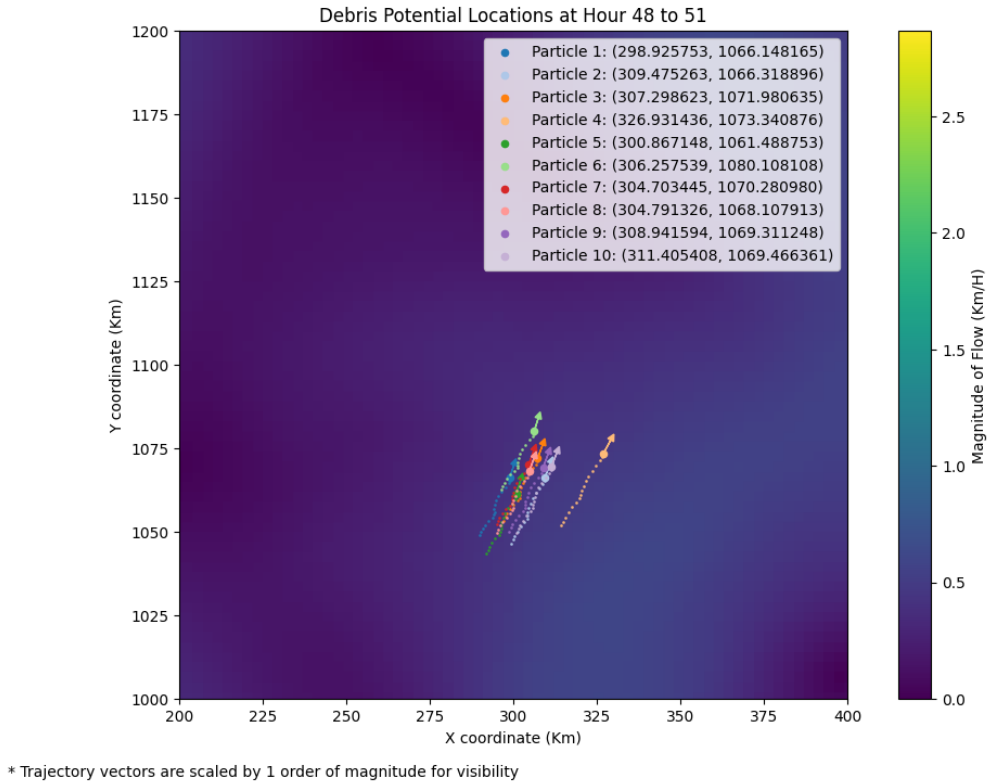

Figure 8: Simulation of Debris Movement at Hour 48.

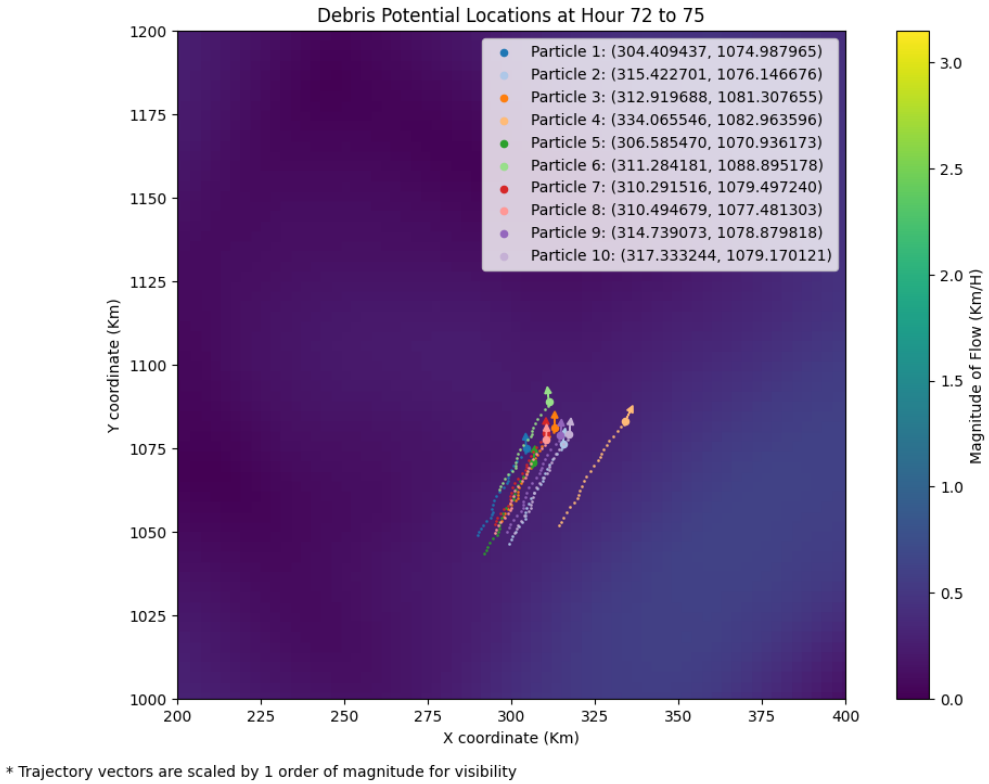

Figure 9: Simulation of Debris Movement at Hour 72.

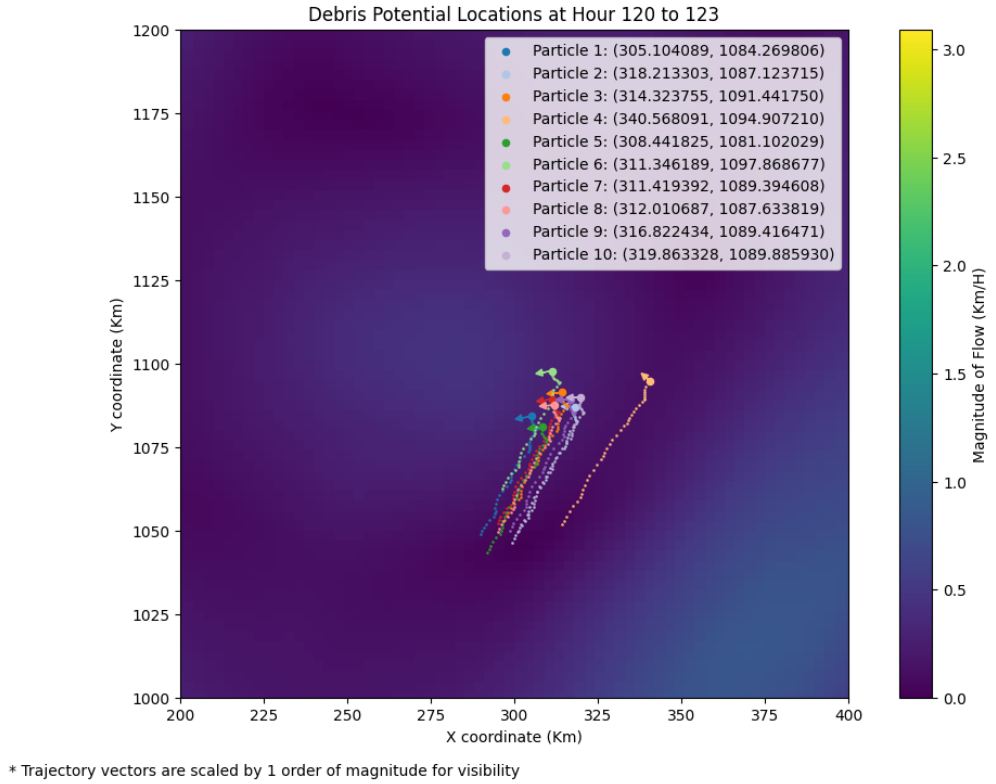

Figure 10: Simulation of Debris Movement at Hour 120.

Part 2. Estimating Flows with Gaussian Processes

Problem 4a

Pick a location of your liking from the map for which you are given flow data (ideally from a location on the ocean not in the land). Moreover, consider the two vectors containing the flow speed for each direction: you will end up with two vectors of dimension 100. You are asked to find the parameters of the kernel function that best describes the data independently for each direction.

Solution:

The location I have chosen is of an area from the first hour of the data that has the highest flow rate at coodinate (105 km, 735 km). It is located in a region where the flow direction is uncertain, and would therefore be an interesting exercise to examine the properties of this point given this uncertainty and high flow rate. The kernel function I have chosen to use to model the point is the squared exponential kernel. I have chosen this point because the function is smooth and predictions change gently. Given that I am modeling the ocean, I expect laminar flow even if the magnitude of the flow is 2.5 kilometers per hour or greater. The physical properties of water should entail a change in the current that is smooth and therefore I have chosen a model that best reflects these basic physical properties. The timesteps are considered for every three days rather than every three hours. In Figure 11 the point of interest is observable.

The best sigma and lengthscale for a parameter of tau of 0.1 at this location is 0.31623 and 2.0 for the horizontal component and 0.31628 and 2.0 for the vertical component respectively. The log likelihoods were -1.17019 and 0.10169. The code in Listing 8 shows the code that produced the results.

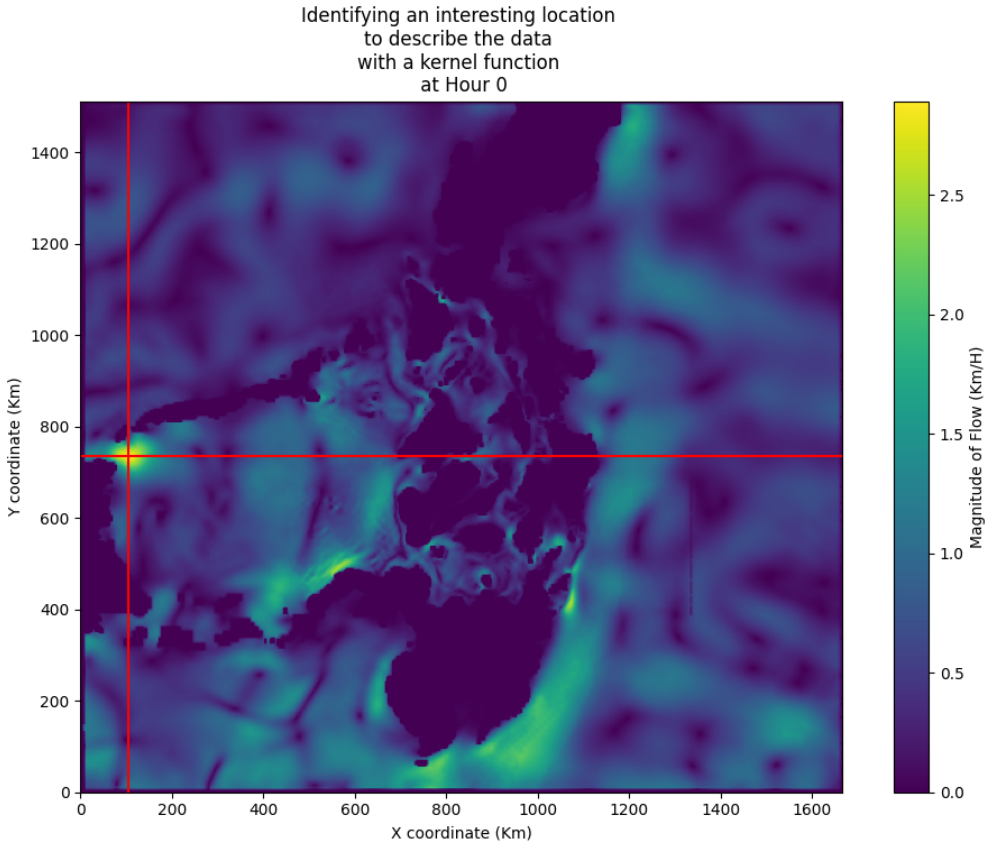

Figure 11: Location of Interest: X=105 Km; Y=735 Km

Listing 8: Estimating the kernel hyperparameters for the initial coordinate of 105 km, 735 km.

```
# Coordinates in km
x_point_init = 105
y_point_init = 735

# Coordinates in indices
x_coordinate = torch.tensor(x_point_init // 3)
y_coordinate = torch.tensor(y_point_init // 3)

# Coodinates
print(f"x_coordinate, y_coordinate: {x_coordinate, y_coordinate}")

# Kernel Parameters:
variance = np.array([0.1, 1, 2, 3, 4, 5, 6])
sigma_l = np.sqrt(variance)
tau = 0.1
ell = np.array([0.1, 0.5, 1, 2, 4])

(
```

```
    u_best_sigma,
    u_best_lengthscale,
    u_best_log_likelihood,
    v_best_sigma,
    v_best_lengthscale,
    v_best_log_likelihood
   ) = compute_ll_for_var_lengthscale_of_U_and_V_component(U, V, sigma_l, ell,
       tau, x_coordinate, y_coordinate, verbose_tau=False)
```

Problem 4b

Run the process described in the point (a) for at least three more points in the map, you free to choose more if you wish. What do you observe? Which of your kernel parameters show patterns? Which do not?

Solution:
The points I chose where points (775, 50), (415, 360), and (1075, 420) in units of kilometers. These points are shown in Figure 12. I calculated the best hyperparameters for the points of interest and determined that given the different points, the best set of sigma and lengthscale that maximize the log-likelihood of the predicted means vary with location, but the same measure of tau, 0.1, was used for each calculation. The calculations were computed using the manner as depicted in Listing 8

For coordinate at point 1 (775, 50), the best standard deviations for the horizontal and vertical components respectively are both 0.31628. The best lengthscales are 4.0 and 2.0 respectively. The reported log likelihoods of this location are 0.4956 and 1.92537.

For coordinate at point 2 (415, 360), the best standard deviations for the horizontal and vertical components respectively are 1.0 and 2.44949. The lengthscales were 4.0 and 0.5. The log-likelihoods were 1.77622 and -11.54037.

For the coordinate at point 3 (1075, 420), the best standard deviations for the horizontal and vertical components respectively are 1.0 and 0.31628. The lengthscales were both best set to 4.0, and the log-likelihoods were 2.06370 and 1.89869.

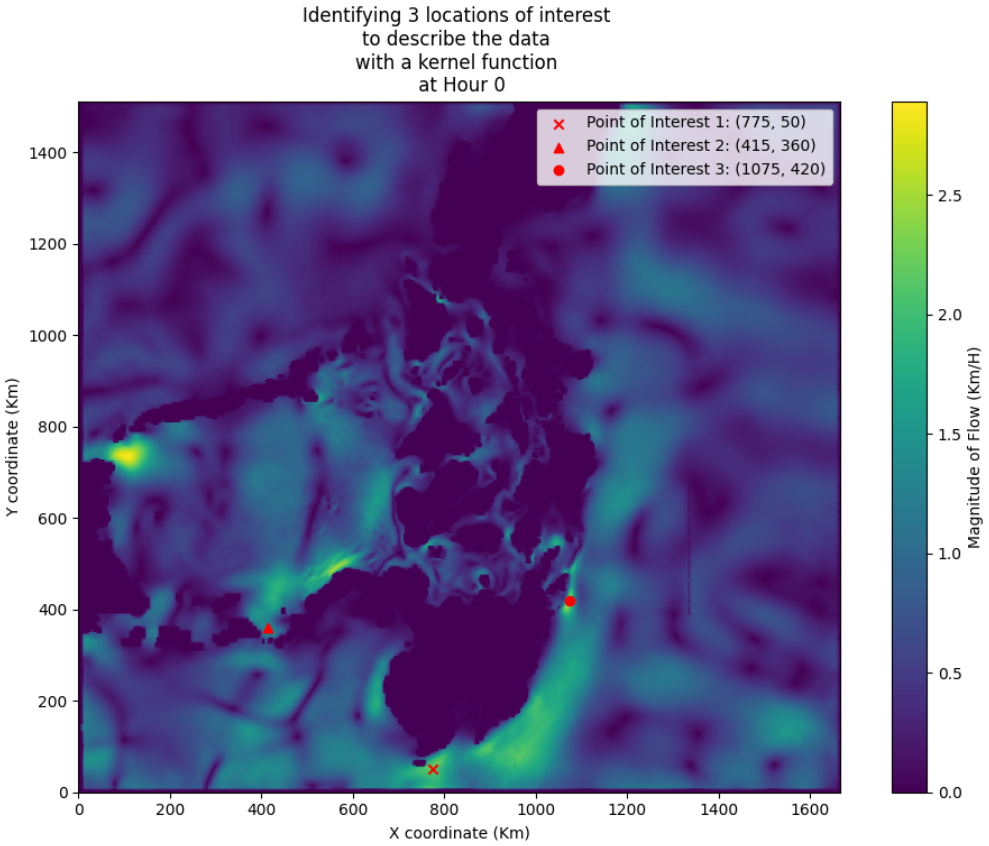

Figure 12: Three points of interest.

Problem 4c

We have suggested one particular value for τ. Consider other possible values and comment on the effects such parameter has on the estimated parameters and the estimation process's performance. Try at least two values different from that used in Problem 4.a.

Solution:
I increased the value of tau from 0.001 to 0.01, 0.1, and 1.0 and computed the optimal hyperparamters for the coordinate at 105 km, 735 km as depicted is Figure 11. This introduced more or less random noise into the squared-exponential kernel function and as a result, the future predictions of variance were regularized and the confidence interval of the predictions were modified. As a result of these findings, the hyperparamter of tau was established to produce the best results when set to a value of 0.1.

Tau set to 0.001 indicated there was nearly no detectable noise in the system. This resulted in horizontal and vertical components of sigma, lengthscale, and log-likelihoods to be detected as 2.44949, 0.1, and -218.51341 for the horizontal u-component and 2.44949, 0.1, and -166.13292 for the vertical v-component respectively. Later in the study, the amplitude of the resulting predicted future velocities of the horizontal and vertical components of flow magnitudes had noticably increased ranges in the amplitudes and very small confidence intervals for 3 standard deviations from the predicted means.

Tau set to 0.01 indicted a useable parameter after inspecting the resulting effect on the future prediction amplitude range and confidence interval spread three standard deviations above and below the predicted mean. The reported optimal hyperparameter values of sigma, lengthscale, and log-likelihood of the computed predictions were 2.44949, 0.1, and -15.03119 for the horizontal u-component and 2.44949, 0.1, and -8.45695 for the vertical v-component respectively.

Tau set to 0.1 resulted in an observable increase in regularization of the predicted velocities in the component magnitude of flow for the horizontal and vertical directions. The confidence interval was further restricted, and the range of amplitudes was reduced to present an unfolding laminar evolution of velocities with respect to time. The reported hyperparameters that optimized the log-likelihood of the predicted mean and variance were 0.31628 and 2.0 for the sigma and lengthscale that computed a log-likelihood of 01.17019 for the horizontal component and the same hyperparameter values of 0.31628 and 2.0 to calculate a log-likelihood of 0.10169 for the vertical component of the flow magnitude.

Tau set to 1.0 resulted in too much regularization to prove useful. The future predicted signal was diminished to a nearly flat line with wide-ranging confidence intervals 3 standard deviations above and below the predicted mean. The best sigma and lengthscales were 1.4121 and 4.0 for the horizontal component to compute -9.61526 for the maximum log-likelihood and 1.4121 and 4.0 for the vertical component to compute -9.49058 for the maximum log-likelihood.

Problem 4d

Currently, most of the commonly used languages like Python, R, Matlab, etc., have pre-installed libraries for Gaussian processes. Use one library of your choice, maybe the language or environment you like the most, and compare the obtained results. Did you get the same parameters as in problem 4.a? If not, why are they different? Elaborate on your answer.

Solution:

I used scikit-learn's "gaussian_process" module to define a kernel using the ConstantKernel and RBF (Radial Basis Function). I evaluated the fit model using the same module's GaussianProcessRegressor. I partitioned the data using 10 k-folds and split the data into train and test sets with a 90/10 split for both the horizontal u-component and vertical v-component. I performed a manual hyperparameter search with a parameter search space of 0.0001, 0.001, 0.01, 0.1, 0.5, 1, 2, and 4 for the lengthscale and the square-root of variances of 0.1, 1, 2, 3, 4, 5, and 6.

The lengthscales of 0.1, 0.5, 1, 2, and 4 represent correlations between flow velocities that would correspond to reasonable potential lengths of time between purported flow velocities of 18 minutes, 90 minutes, 3 hours, 6 hours, and 12 hours given 100 time steps and 3 hours between each sampled time. The range of variance was calculated for all points for all coordinates in both the horizontal and vertical components of the magnitude of flow velocities to discover that the maximum variance for the horizontal component was a value of approximately, but not greater than, 6, and the maximum variance for the vertical component was a value of approximately, but not greater than, 4. The lowest absolute value of the variance was no variance in the magnitude of flow for either component due to stationary water with no current. For these reasons, the variance parameter search space was defined.

Upon evaluation of the log-marginal-likelihood-value of the fit model for the horizontal and vertical components of the magnitude of flows, convergence warnings indicated that smaller hyperparameters would best fit the model and that the current search space was too small to reach model convergence. Therefore, smaller parameters for the lengthscale were included to normalize the variances with the intention of reaching model convergence. However, parameters of 0.0001, 0.001, and 0.01 indicate a correlation in flow velocities with respect to periods of 1.08 seconds, 10.8 seconds, and 1.8 minutes respectively and do not align with the first principal characterstics of the properties of the ocean within the larger context of the problem definition. That is to say it is unlikely that ocean currents are correlated every second, ten seconds, 2 minutes or periods of time with increasing levels of granularity for that matter. Furthermore, including these parameters into the search space did not allow the model to reach convergence. The model resulted in reporting optimal hyperparameters of 2.23607 and 0.31623 for the standard deviation (σ). Lengthscale (ℓ) normalization factors were reported to be 1.0 for the horizontal and vertical components of the flow magnitude, respectively. The maximum marginal log-likelihoods were reported to be -86.00327 and -39.21640 in alignment with the horizontal and vertical components of the flow velocity magnitude. Because the model did not reach convergence with reasonable hyperparameters for the context of the problem, because the packages require numpy data structures that cannot be computed efficiently with a GPU, and because the time to identify the reported hyperparameters (33.1 seconds) was greater than that of the custom algorithm defined in Listing 8, custom algorithms were implemented to define the optimal hyperparameters for each coordinate within a reasonable search space given the context of the problem definition. As a result, reduced compute times by means of efficient torch tensor data structures and powerful GPU compute were leveraged.

Problem 5

In the previous problem, we have found a good set of parameters to model the sequence of speeds at one location as a Gaussian process. Recall that we have assumed our 100 observations came at a rate of one every three days. We are going to assume that when we advance to our simulations, we will choose a smaller time step. Thus, we need to interpolate how the flow would look like at some unobserved points.

You are given flow information every three days. Pick some time stamps in-between each observation for which to estimate the flow. For example, you want flows every day, so there will be two unknown points between two observations. You could pick only one, or more than two. Make your choice and explain why.

Compute the conditional distribution (mean and covariance) at the time locations selected in part (a). Use the kernel parameters that you obtained in Problem 4.a, and use the same location as you did in Problem 4.a.

For the initial estimate of the mean at the unknown time locations, you can use zero, use the average of all the observations, or take the average of the two closest observed points.

Plot your predictions.

Solution:
I have chosen to create time locations for every single day with every three days being the true value of the flow. There will be a prediction of the flow for the two days surrounding the true values. This allows the data to have predictions before and after each true value in order to clearly see the predictive quality and effectiveness of the model around each true point.

The best standard deviation, σ, for the horizontal and vertical components of the flow velocity magnitude discovered in problem 4(a) were 0.31623 for both components. The best lengthscale, ℓ, for both components in the same problem were identified to be 2.0. These hyperparameters were implemented in defining the model that would compute the conditional distribution for coordinate 105 km, 735 km. Again, this point is visualized in Figure 11. The plots of the predicted means, observed training points, and confidence intervals are observable in Figures 13 and 14 for the horizontal and vertical components, respectively.

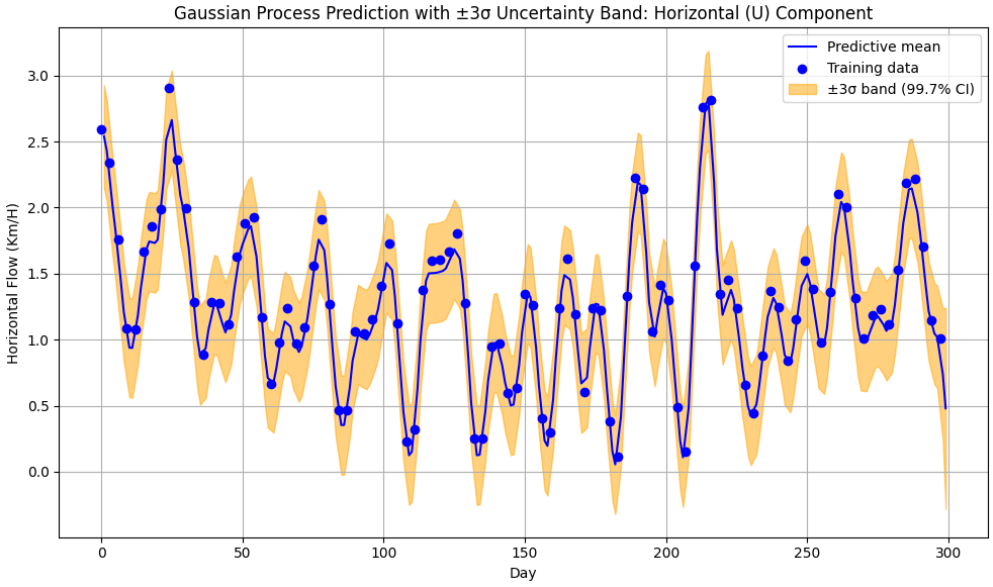

Figure 13: Future predictions of the horizontal flow velocities at coordinate 105 km, 735 km.

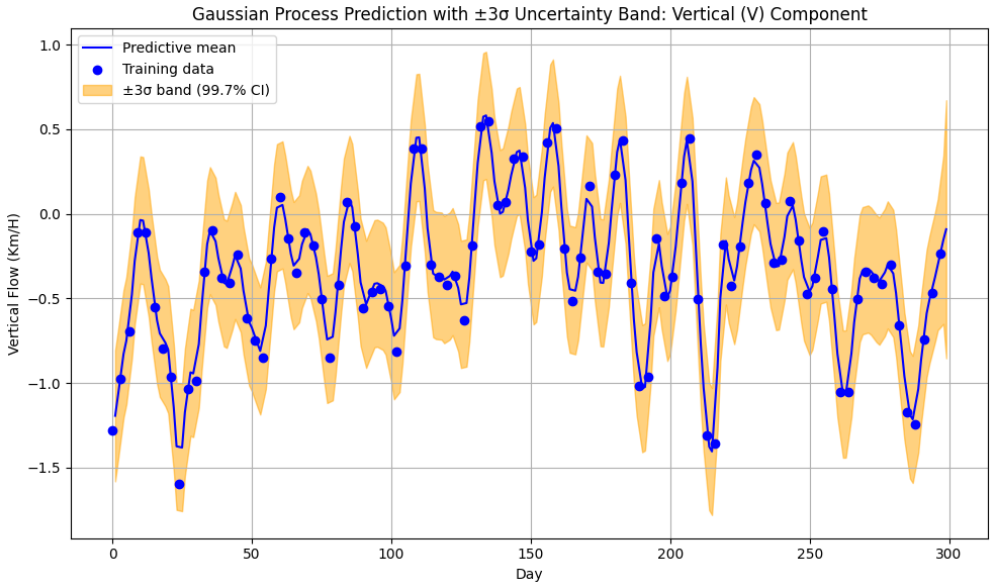

Figure 14: Future predictions of the vertical flow velocities at coordinate 105 km, 735 km.

Problem 6a

Modify the simulator that you built in Problem 3.3 to use this new flow estimated flow information. Note that with this new change, you will be able to simulate the flow of particles for 300 days! Regarding data, originally, we have 100 measurements per point, now with this approach, let us say you use the estimates for two extra points to get one flow data per day, so in total, you should have at your disposal 300 descriptions of flow per location.

Now repeat Problem 3 (b). This time you will be simulating flows for 300 days. This allows some debris to arrive on land. Where are some possible places along the coast where one could find debris? Again, to do this, pick some σ of your choice, and simulate the movement of particles with initial location sampled from the bivariate Gaussian. Evolve the location of the particles. Some times particles trajectories will terminate on the shore. Continue to keep track of such particles. These points are likely where you could find the debris.

Provide a plot that includes your initial, final, and at least one intermediate state of your simulation. For the final state, clearly mark one location on land where you would search for debris. Also mark one location over the ocean where you would search for debris. Provide a brief justification for both choices.

Try at least one other value for σ, and create the same three plots. Comment if your conclusions should change.

Solution:
I chose the mode standard deviation and lengthscale to represent the optimal generally applicable hyper-parameters to compute the conditional distribution. These values were equivalent for both horizontal and vertical components. The chosen standard deviation was computed as 0.31628 with 64-bit precision, and the chosen lengthscale was computed as 2.0.

In Table 1, the typical scattering radius for a general altitude is displayed. To compute the variety of variances for the debris scattering, I chose arbitrarily 100 debris sampled from a uniform distribution with sigma between 0 and 80 kilometers initially. Each of the 100 debris were then sampled randomly from a normal gaussian distribution about a mean of coordinate pair of 300 km and 1050 km for x and y respectively. 80 kilometers was chosen as the sigma for the debris scattering variance to simulate a low-altitude breakup debris scattering pattern. The initial day of impact, the intermediary 72nd day after impact, and the ultimate 300th day after impact plots are observable in Figures 15, 16, and 17.

Situation	**Typical Scattering Radius**	**Notes**
Controlled crash / intact	< 10 km	Debris remains near the impact site due to minimal breakup.
Mid-air breakup (low altitude)	20–80 km	Debris spreads over a moderate area; influenced by altitude and structural breakup.
Mid-air breakup (high altitude)	100–200 km	High-altitude disintegration leads to widespread debris dispersion.
Drift over time (currents)	1000–4000+ km	Ocean currents and winds can carry debris over vast distances over weeks or months.

Table 1: Typical debris scattering distances for aircraft crashes in oceanic conditions

Sources: OpenAI ChatGPT 2025, (BEA) 2012, Jansen, Coppini, and Pinardi 2016, Board 2015, Oceanic and (NOAA) 2024.

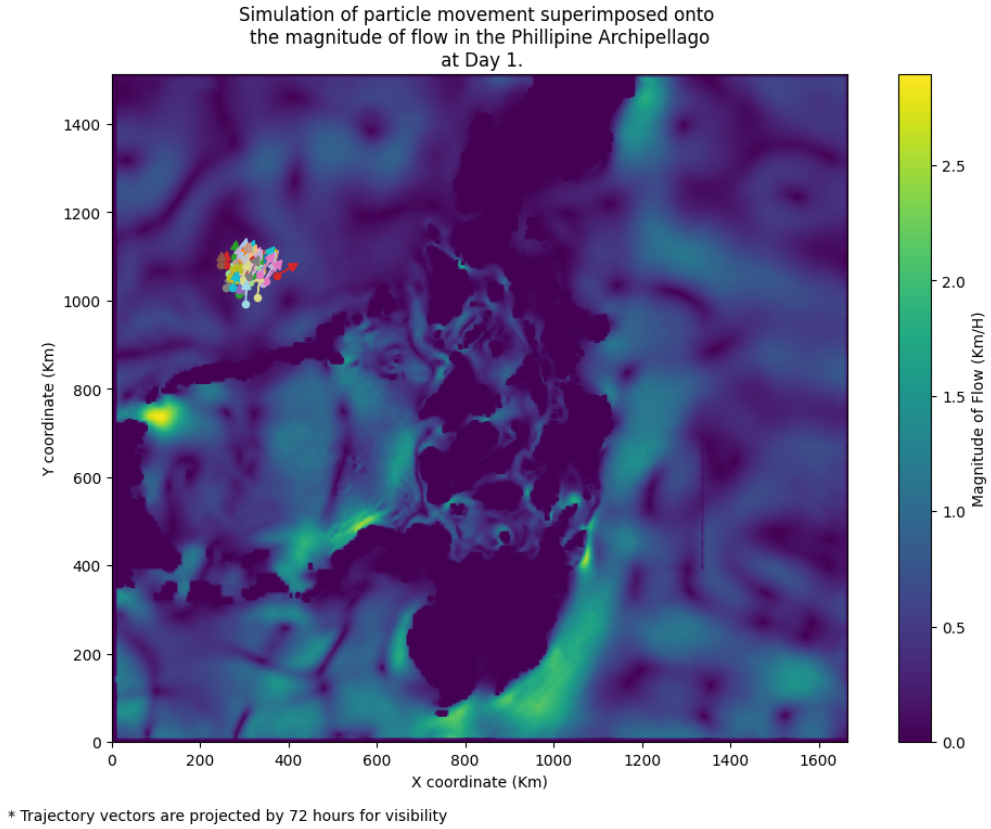

Figure 15: Initial Low-Altitude Breakup & Debris Scattering Pattern

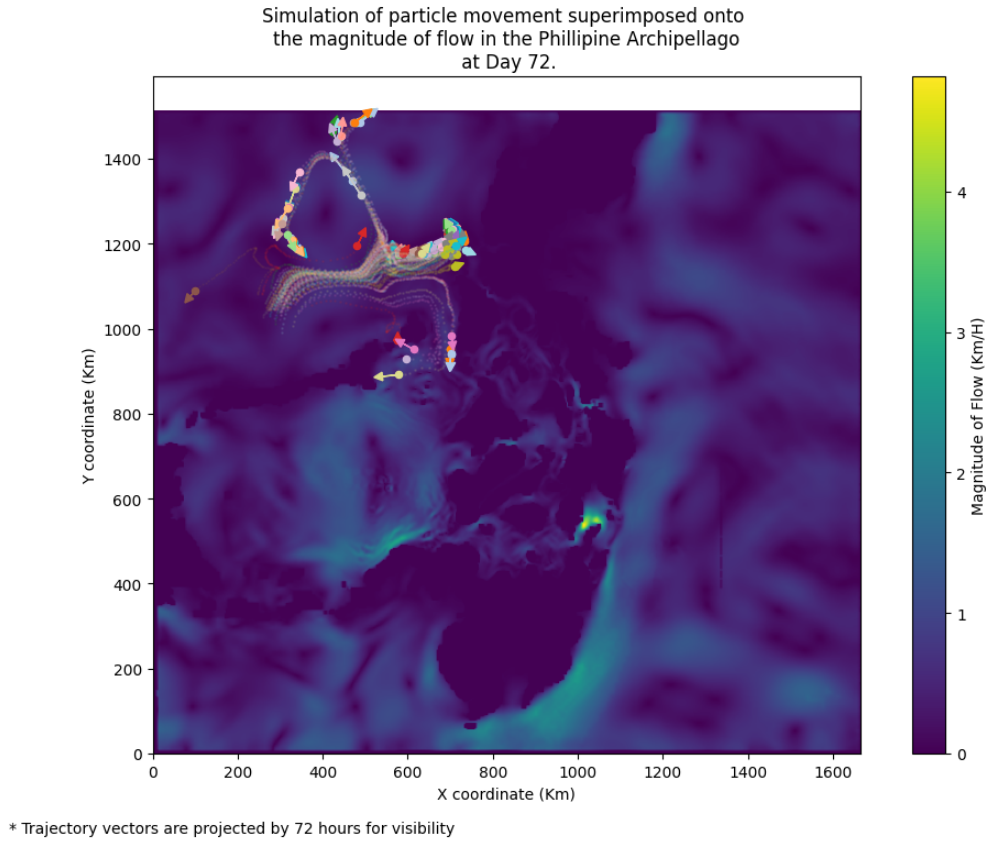

Figure 16: Low-Altitude Breakup & Debris Scattering Pattern 72 Days After Initial Impact

In Figure 17 I chose to mark a location on land to search for debris at coordinate 780 km, 1250 km. In Figure 16 it is clear that a number of debris drift toward this location and eventually beach or would be distantly visible from this location. After performing this simulation multiple times, there has always been an observable debris beached at this approximate location. I marked the coordinate (325 km, 1300 km) to indicate the ocean location to search for debris. This location was chosen because multiple simulated debris are located at this coordinate pair, and other simulated debris would be distantly observable from that location in the ocean.

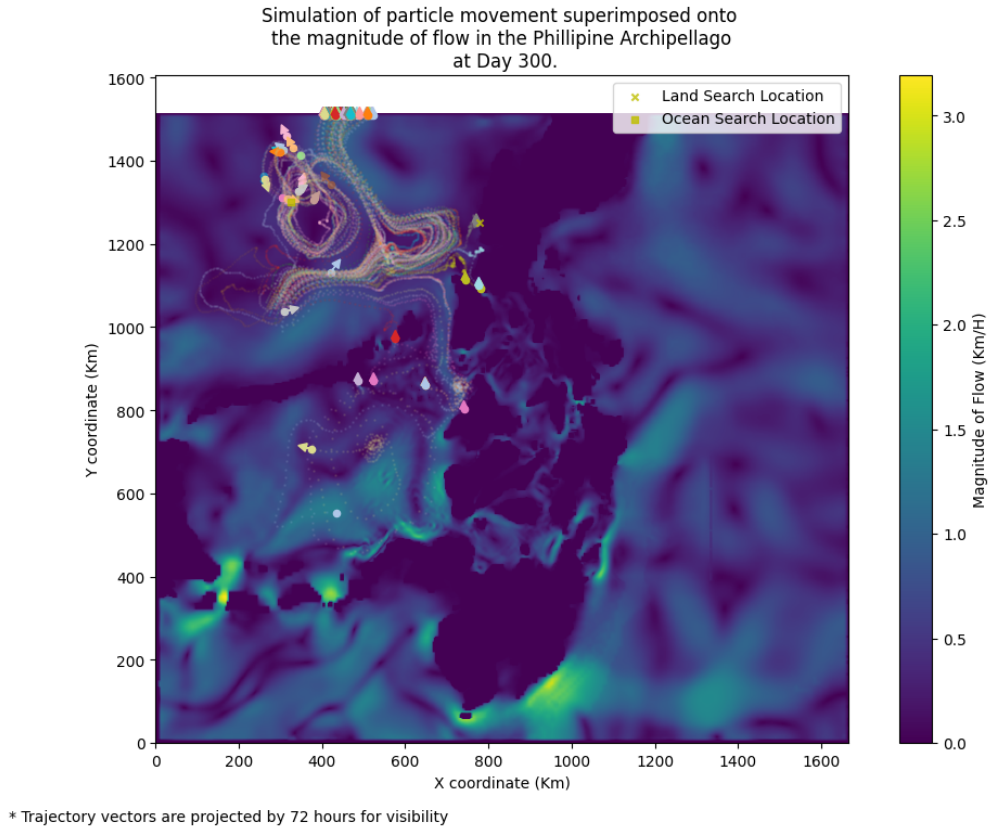

Figure 17: Low-Altitude Breakup & Debris Scattering Pattern 300 Days After Initial Impact with Labeled Land & Ocean Search Locations

To simulate a high-altitude breakup and debris scattering pattern, I again chose to simulate arbitrarily 100 debris particles. This time, however, I chose a standard deviation ceiling of 200 kilometers with a minimum bound of 0 kilometers for the range of potential variances that were sampled from a uniform distribution. These variances detailed the distribution of a normal guassian with mean coordinates of 300 kilometers and 1050 kilometers from which 100 debris were randomly sampled. This initial starting position of the resultant debris scattering is observable in Figure 18. An intermediary simulation of the 72nd day after the initial impact is observable in Figure 19.

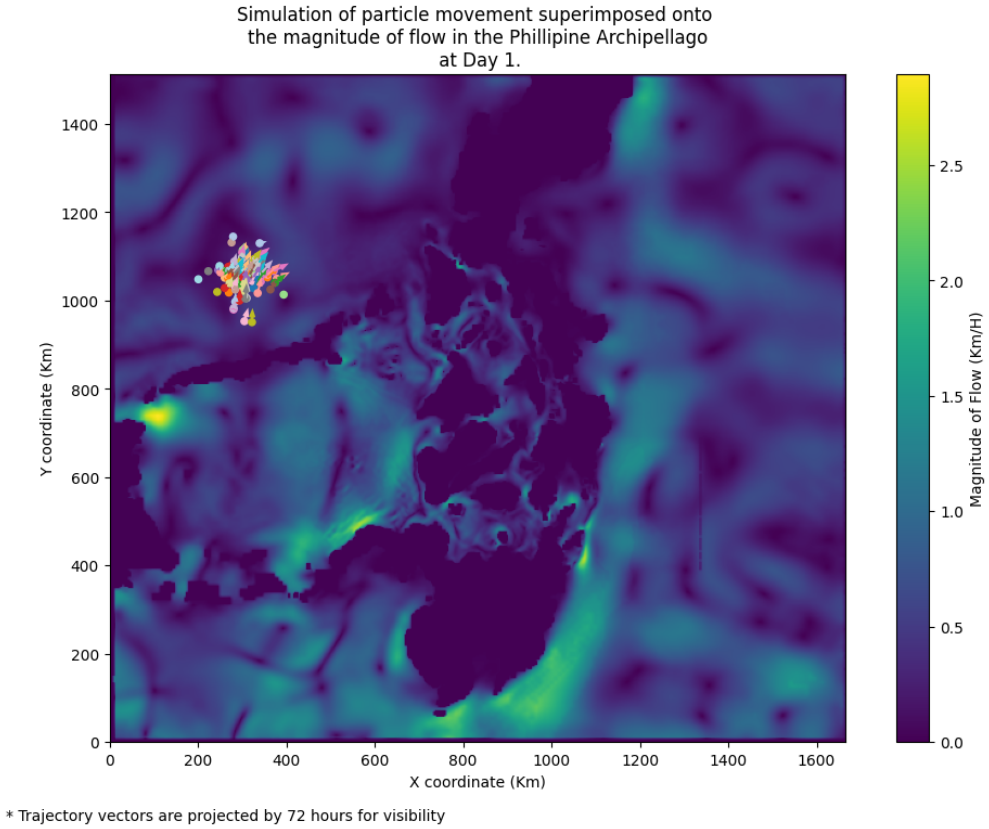

Figure 18: Initial High-Altitude Breakup & Debris Scattering Pattern

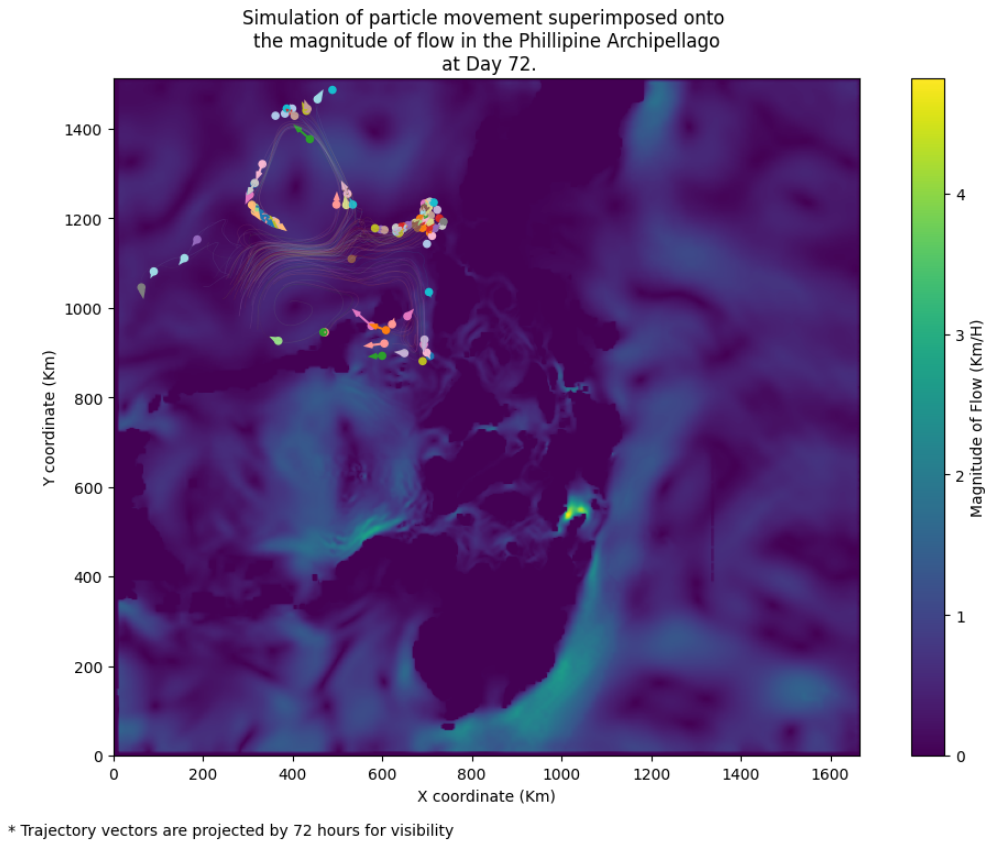

Figure 19: High-Altitude Breakup & Debris Scattering Pattern 72 Days After Initial Impact

I chose to mark locations to begin a search on land at coordinates 780 kilometers and 1250 kilometers because several simulations result in beached debris being located at this approximate coordinate pair, and Figure 19 indicates a significant number of debris floating toward this location three days after the initial impact. I chose to mark the location to begin a search in the ocean at coordinates 325 kilometers and 1300 kilometers because a large number of debris circle back around to this location due to the current in this area. There is an observable possibility of finding debris at or near this location or sighting distant debris from this coordinate pair.

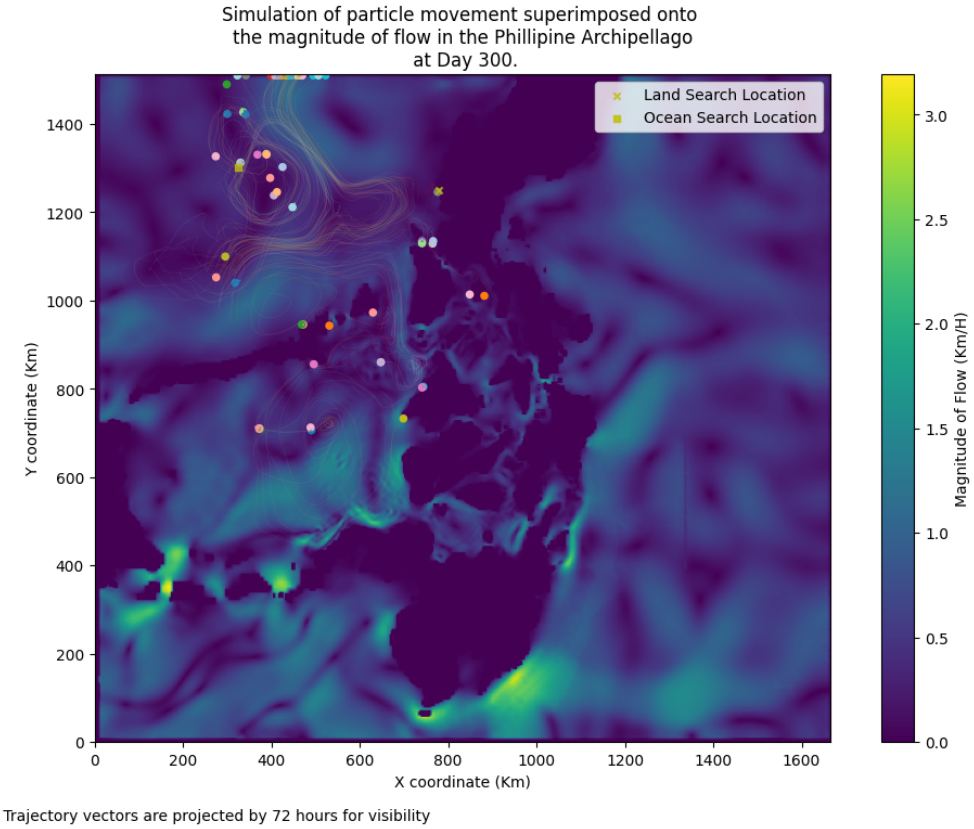

Figure 20: High-Altitude Breakup & Debris Scattering Pattern 300 Days After Initial Impact with Land & Ocean Search Locations Indicated.

Problem 6b

Thanks to your efforts, most parts of the (toy) plane were found, either inland or in the sea. As a final stage, you are tasked with locating three new monitoring stations on the coast. The purpose of these stations is to monitor general ocean debris. Using the tools you have build in this homework, propose the location of such three new stations. Simulate the trajectories of as many particles as you want, initialized at random locations uniformly distributed on the map. This is essentially a repeat of Problem 3 (a) with the new simulation; this time, remove particles that start on land so that they do not confuse your conclusions.

Many of the particles will end up on the coast. A good location for a monitoring station will be areas where many of such particles land on the coast.

Provide a plot that includes your initial, final, and at least one intermediate state of your simulation. For the final state, clearly mark where you would place your three monitoring stations. Provide a justification for why you chose these locations.

Solution:
I created a simulation to predict the unknown coordinates of 200 debris that were uniformly scattered throughout the Phillipine Archipelago. I attempted to augment this matrix by 1000 using expectation maximization practices. This allowed more particles to be simulated, and I successfully created a matrix that defined the movement of 1200 debris. In alignment with the instructions of this work, I am reporting the matrix using 200 debris points calculated using a conditional distribution. In Figure 21 the initial coordinates of the 200 debris are shown uniformly scattered, the colors are distinct, and future trajectories are plot. There is an evolution of debris by day 72 as shown in Figure 22, and by the final 300th day, a number of debris have becomed beached. I chose to place monitoring stations as shown in Figure 23 at the following coordinates (440 km, 320 km) for Station 1 (marked with a yellow "X") in Figure 21. Station 2 is located at (670 km, 230 km) and is marked with an upward tripod. Station 3 is located at (1100 km, 680 km) and is marked with a yellow triangle. These stations are located at places where there are multiple simulated debris impacts, where there is a high magnitude of current to deliver debris with strong probability and often, and where there is high visibility to identify other debris that may become beached within sight.

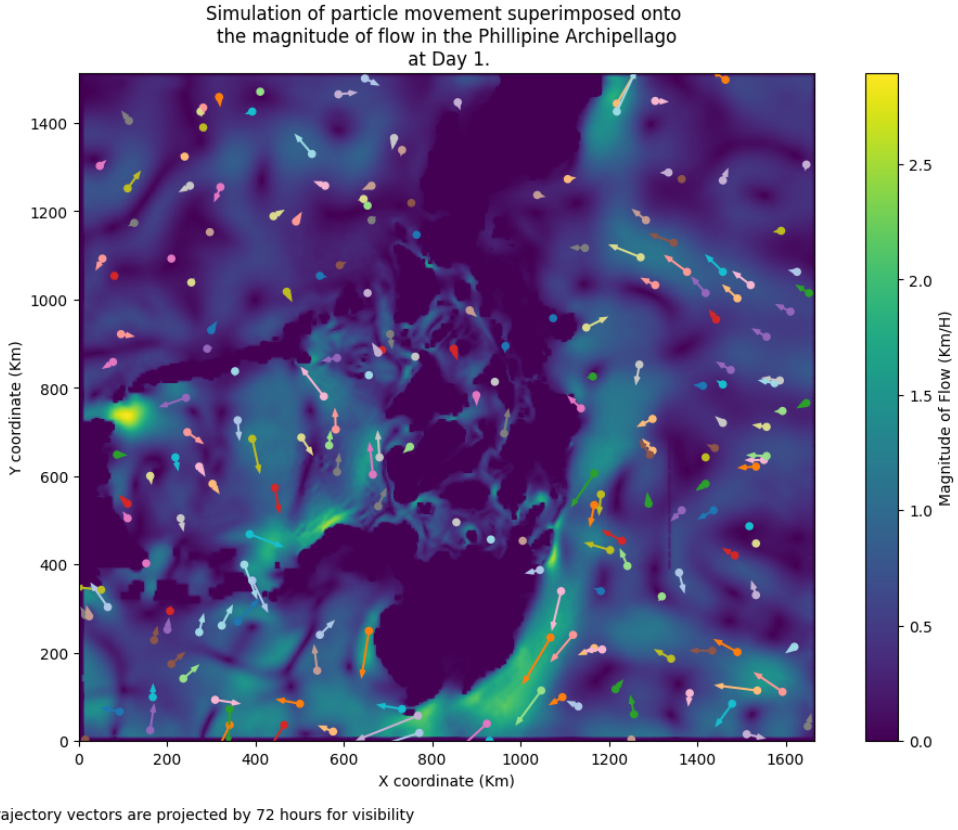

Figure 21: Initial Debris Scattering of 200 Random Locations

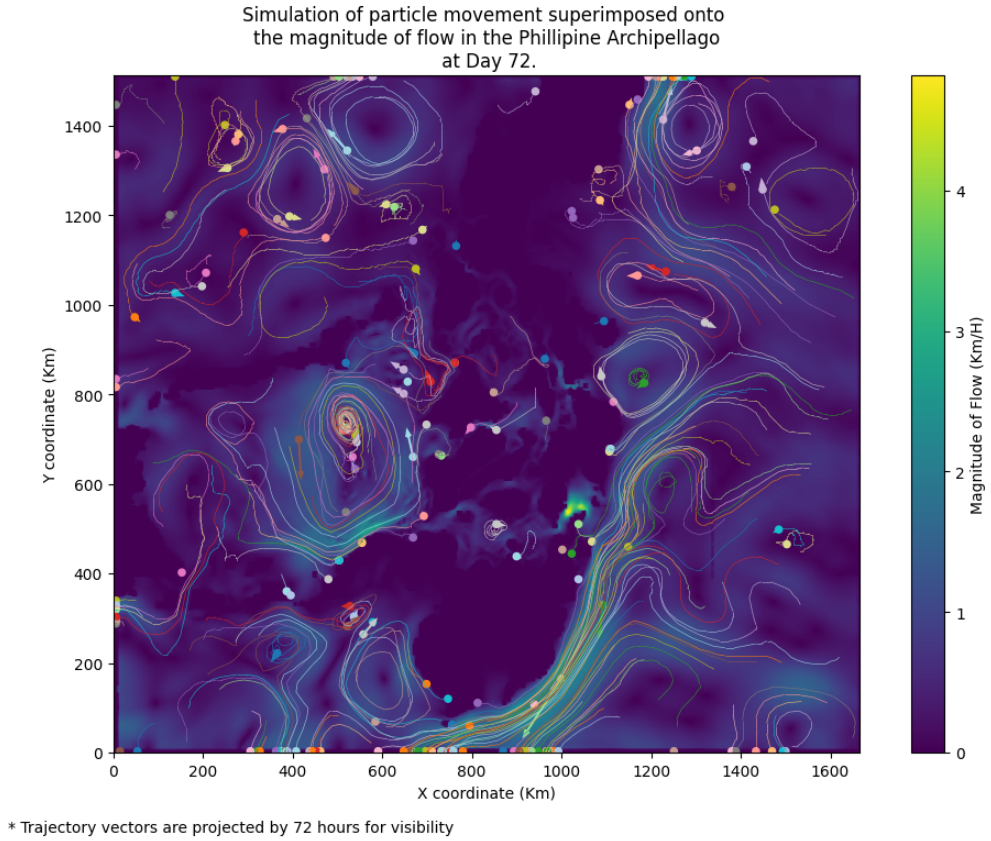

Figure 22: Evolution of Simulated Debris Movement 72 Days After Initial Impact.

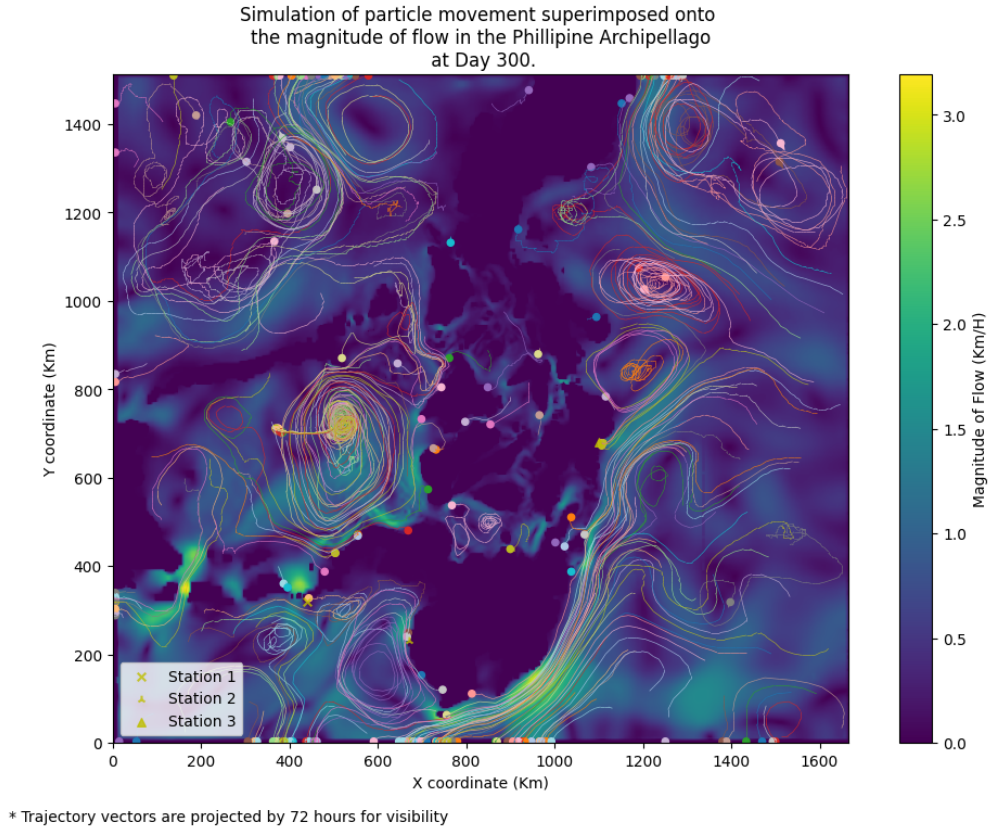

Figure 23: 300 Days After Initial Impact & Indicated Suggested Monitoring Station Placement

References

(BEA), Bureau d'Enquêtes et d'Analyses (2012). *Final Report on the accident on 1st June 2009 to the Airbus A330-203 registered F-GZCP operated by Air France flight AF 447 Rio de Janeiro – Paris*. Tech. rep. Bureau d'Enquêtes et d'Analyses pour la sécurité de l'aviation civile. URL: https://www.faa.gov/sites/faa.gov/files/AirFrance447_BEA.pdf.

Board, Dutch Safety (2015). *Crash of Malaysia Airlines flight MH17: Final Report*. Tech. rep. Dutch Safety Board. URL: https://onderzoeksraad.nl/wp-content/uploads/2023/11/debcd724fe7breport_mh17_crash.pdf.

Jansen, E., G. Coppini, and N. Pinardi (2016). "Drift simulation of MH370 debris using superensemble techniques". In: *Natural Hazards and Earth System Sciences* 16.7, pp. 1623–1628. DOI: 10.5194/nhess-16-1623-2016. URL: https://nhess.copernicus.org/articles/16/1623/2016/.

Oceanic, National and Atmospheric Administration (NOAA) (2024). "Modeling Oceanic Transport of Floating Marine Debris". In: *NOAA Marine Debris Program*. URL: https://marinedebris.noaa.gov/modeling-and-monitoring/modeling-oceanic-transport-floating-marine-debris.

OpenAI ChatGPT (2025). *Assistance in deriving the mathematical equations, functions, and plots*. https://chat.openai.com. Accessed via OpenAI ChatGPT. URL: https://chat.openai.com.