Written Report – 6.419x Module 5

Name: Sandipan Dey

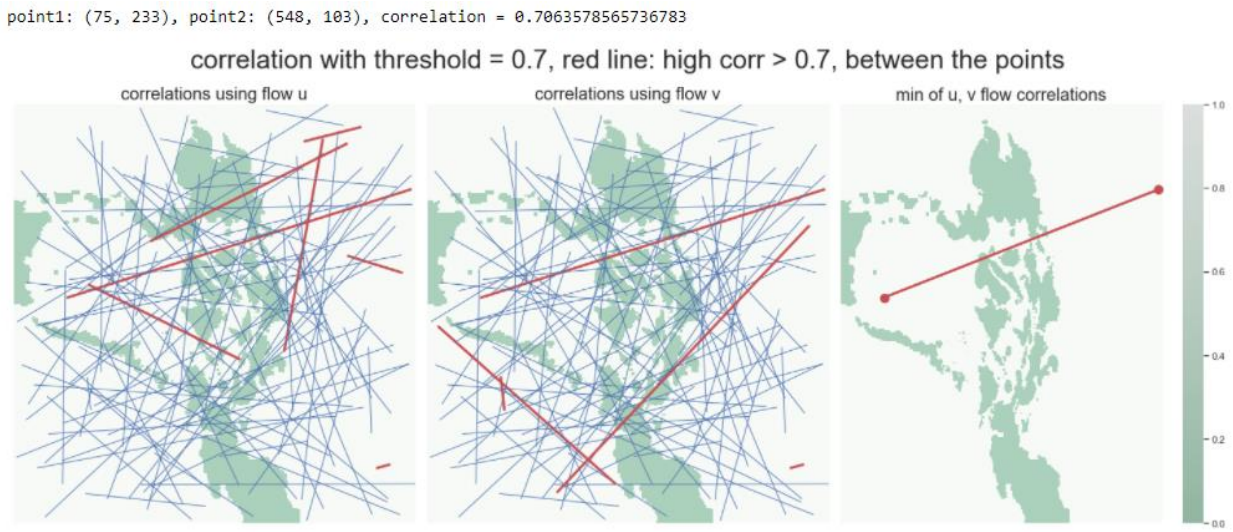
Problem 2: [10 bonus pts] Identifying long-range correlations.

Include your answer to all parts of this problem (2) into your written report.

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 did you found those two areas have correlated flows.

Solution

The next figure shows 2 locations that are far apart but still have high positive correlation > 0.7 (threshold). In the next figure the blue lines correspond to low correlation, where the red ones correspond to high correlations (> threshold).



How the correlation was computed:

- First 100 random pairs were chosen from all possible pair of 555x504 points.

- Correlations between each of the pair of points, for both flow u and v were computed (between 100 dim vectors across the time axis).
- The minimum of correlations of u and v flows were chosen to compute the final correlation.
- The following code snippet shows how it was computed:

```
np.random.seed(12)
n = 100
thresh = 0.8
cor_u, cor_v = {}, {}
i1s, j1s = np.random.choice(504, n, replace=True), np.random.choice(504, n, replace=True)
i2s, j2s = np.random.choice(504, n, replace=True), np.random.choice(504, n, replace=True)
plt.figure(figsize=(20,8))
plt.subplot(131), sns.heatmap(u.mean(axis=0), cmap="YlGnBu", cbar=None)
for k in range(n):
    cor_u[(i1s[k],j1s[k]), (i2s[k],j2s[k])] = np.corrcoef(u[:,i1s[k],j1s[k]], u[:,i2s[k],j2s[k]])[0,1]
    cor_v[(i1s[k],j1s[k]), (i2s[k],j2s[k])] = np.corrcoef(v[:,i1s[k],j1s[k]], v[:,i2s[k],j2s[k]])[0,1]
    ## plot
dist_thresh = 100
for k in range(n):
    cor = min(cor_u[(i1s[k],j1s[k]), (i2s[k],j2s[k])], cor_v[(i1s[k],j1s[k]), (i2s[k],j2s[k])])
    cor_high = cor > thresh
    far_enough = np.sqrt((i1s[k]-i2s[k])**2+(j1s[k]-j2s[k])**2) > dist_thresh
    if cor_high and far_enough:
        print((i1s[k],j1s[k]), (i2s[k],j2s[k]), cor)
        ## plot
```

Why the two marked locations could be correlated:

As can be seen from the below figure, comparing the u-flows and v-flows of the two locations, they have similar trends over time and that's why they are highly correlated (e.g., for v flows, the blue and green lines decrease till 75*3 hrs and then increase).

Problem 3: [20 pts] Simulating particle movement in flows.

In this problem, you are asked to build a simulator that can track a particle's movement on a time-varying flow.

Problem 3.a: **[10 points]** **Include your answer to all parts of this problem (3.a) into your written report.**

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 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

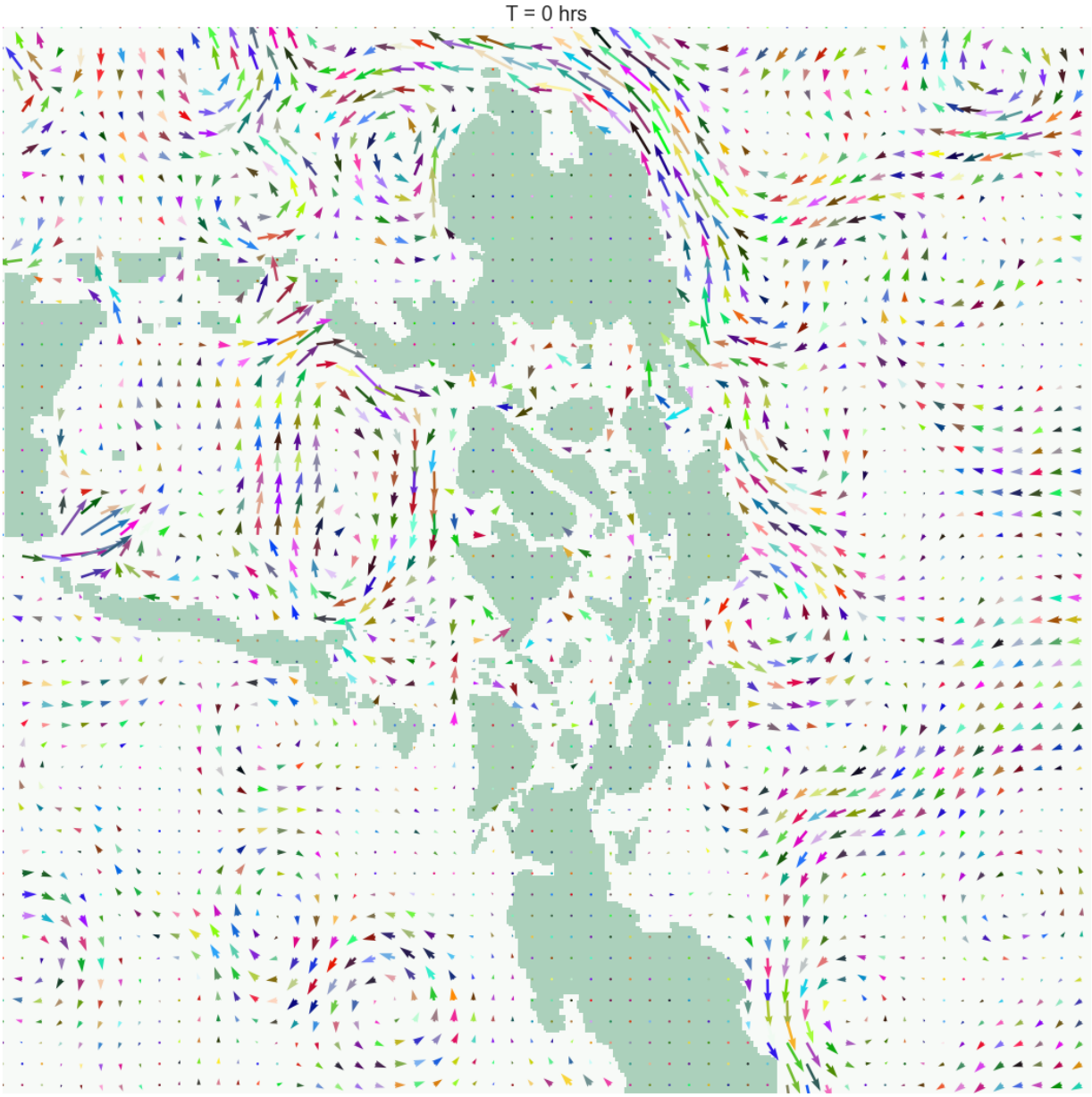
Explanation

The following code shows the function that implements the simulation for the evolution of flow velocities. At a point (x,y), the velocity vector at time t is given by the tuple (u(t,y,x), v(t,y,x)), since in u and v matrices, x axis represents the columns and y axis represents the rows (x, y being integers, so that velocities are discretized). We can draw quivers with the position of a location and the directions from the flow matrices and evolve over time t.

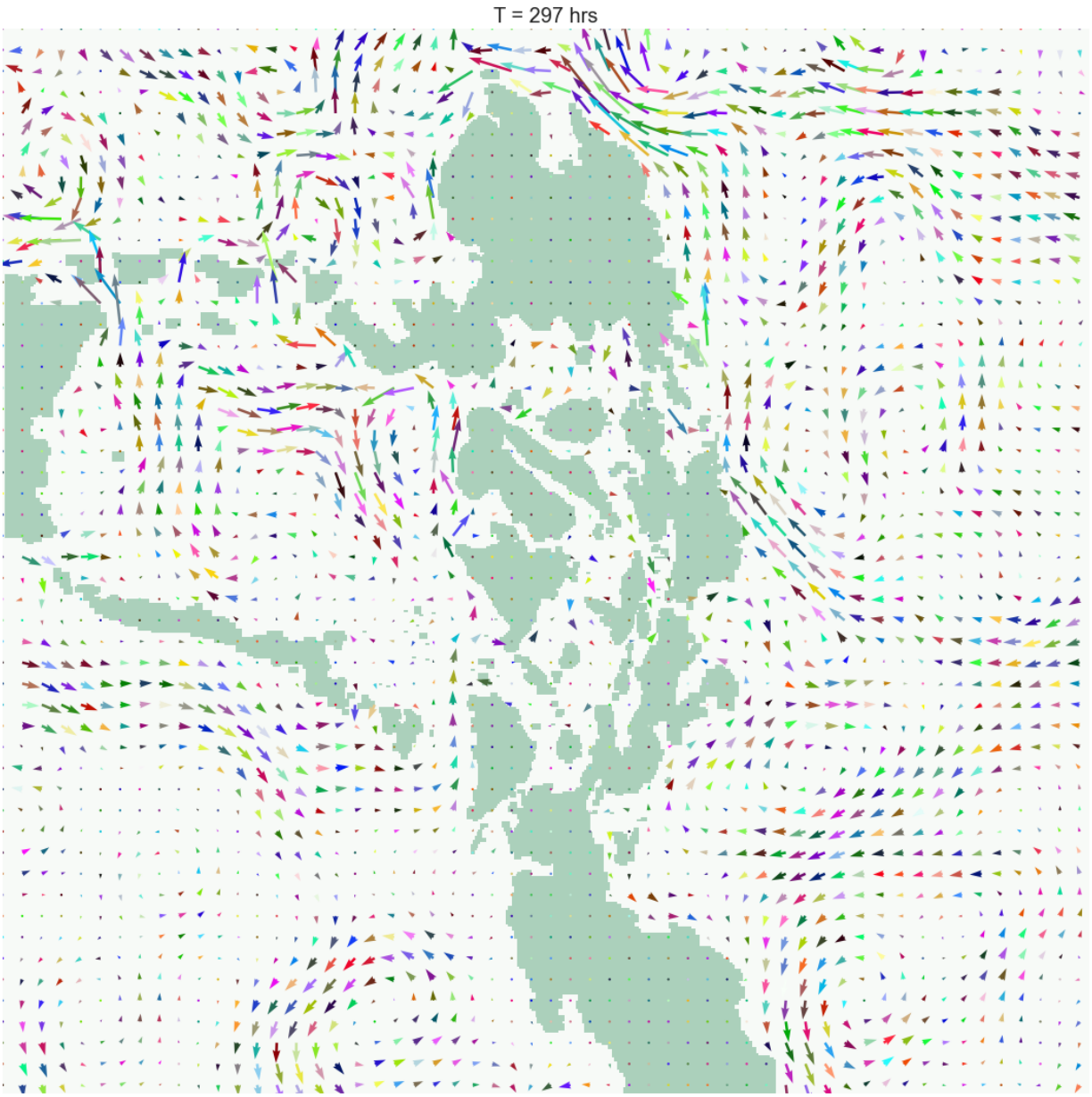
```
def plot_flows_over_time(u, v, mask):
    xx, yy = np.meshgrid(range(0, mask.shape[1], 10), range(0, mask.shape[0], 10))
    for t in range(100):
        plt.figure(figsize=(15,15))
        sns.heatmap(mask, cmap=pal, alpha=0.4, cbar=None)
        plt.quiver(xx, yy, u[t,yy,xx],v[t,yy,xx], scale=0.1, units='xy',
                   color=[(np.random.random(),np.random.random(),np.random.random()) for _ in range(np.prod(u[t,yy,xx].shape))])
        plt.title('T = {} hrs'.format(3*t), size=20)
        plt.axis('off')
        plt.tight_layout()
        plt.show()
```

The next figures show the initial, final and the intermediate flows.

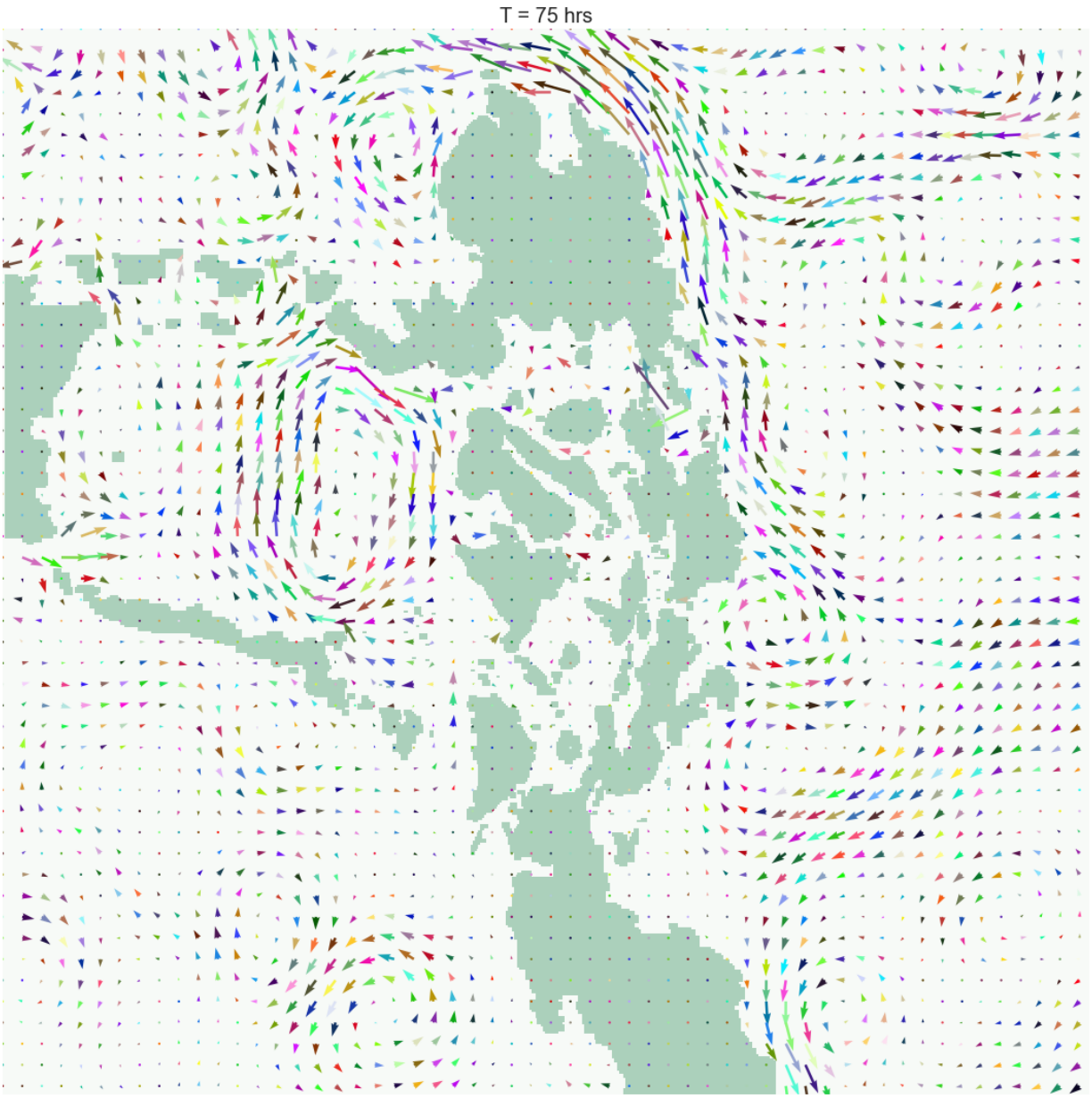
Initial Flow

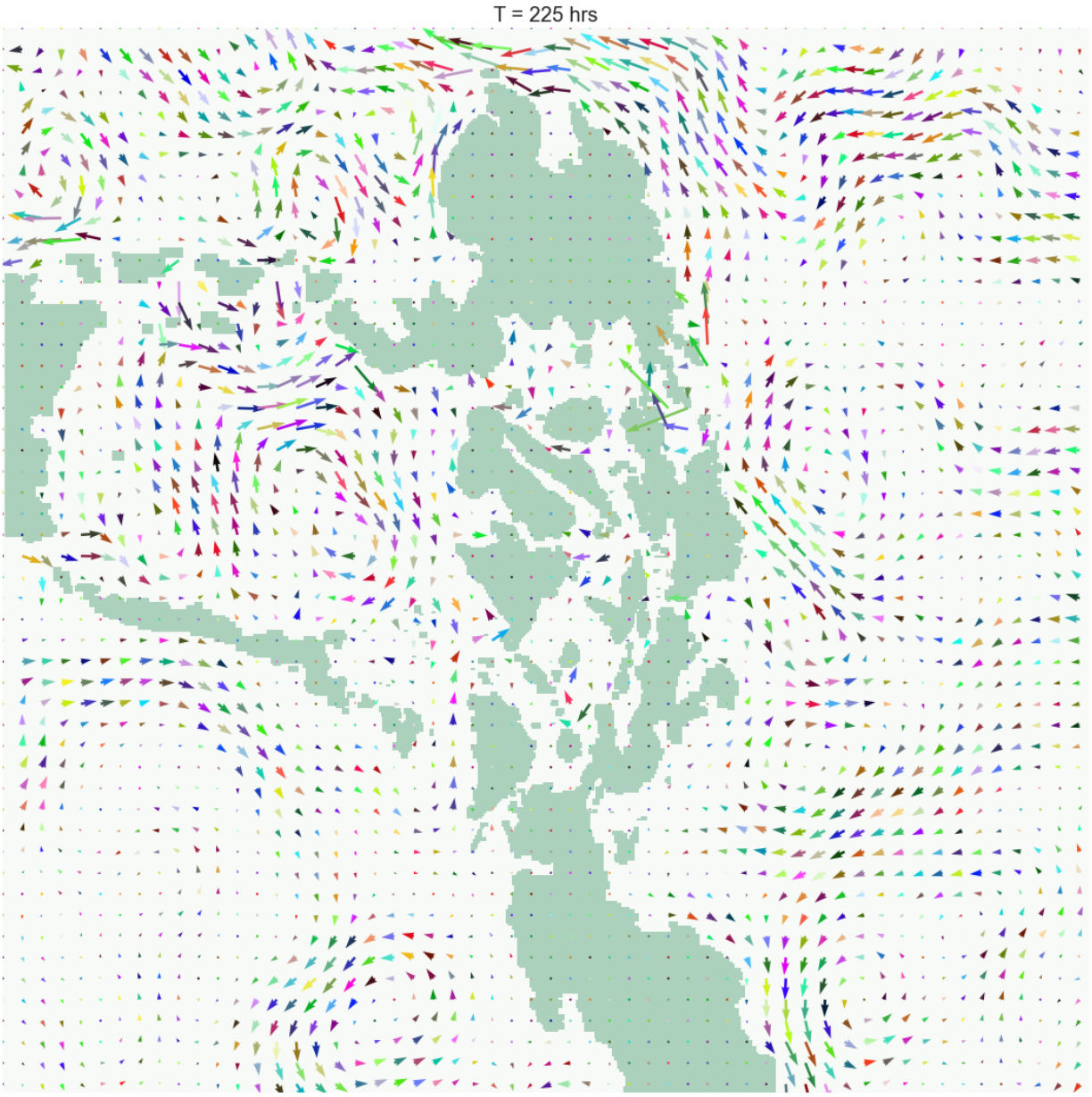


Final Flow



Intermediate Flows





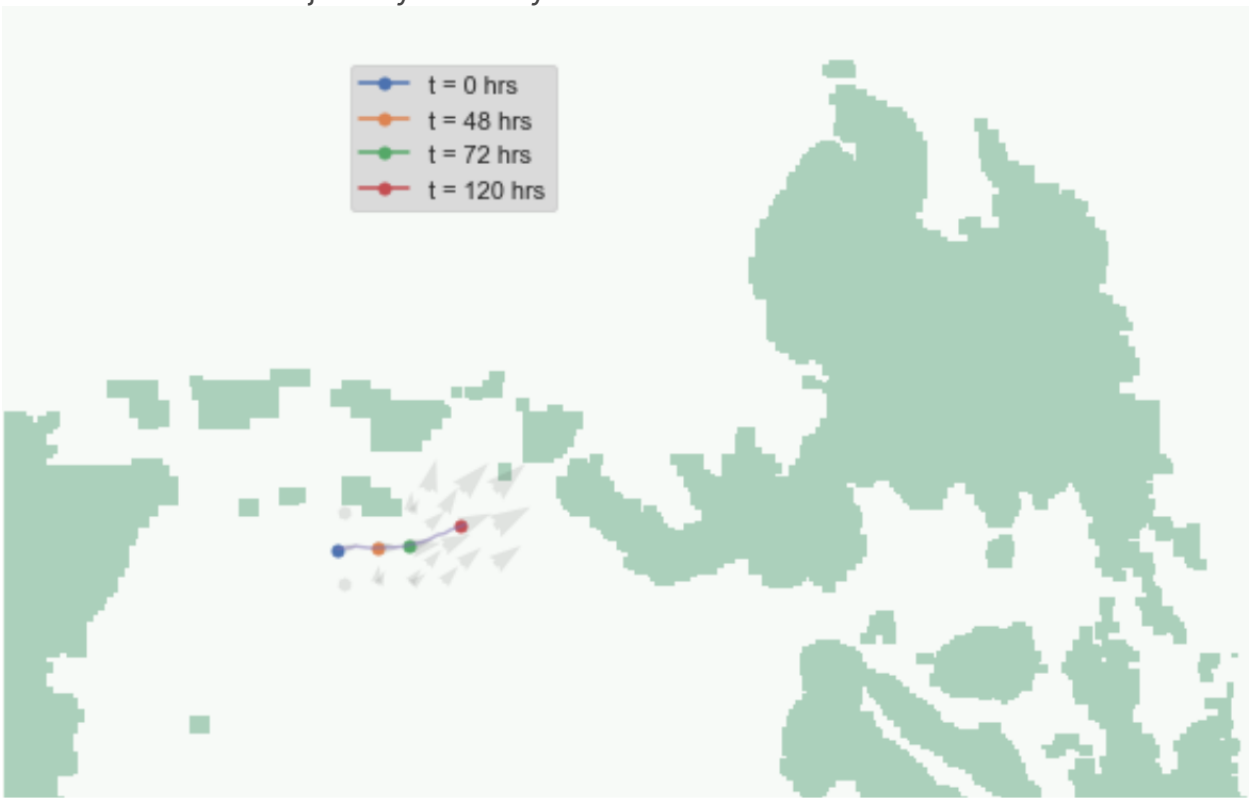
Problem 3.b: **[10 points]** **Include your answer to all parts of this problem (3.b) into your written report.**

A (toy) plane has crashed north of Palawan. The exact location is unknown, but data suggests that the location of the crash follows a Gaussian distribution with mean (namely) with variance . 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 hrs, hrs, 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

First let's visualize how the **mean** location drifts with time, I have deliberately created one single plot instead of three to show the location of the plane at different three times (shown as legends), since I think it's more insightful and easier to find the trajectory this way.



Now, let's sample 100 particles around the mean for 3 different variance values from the Gaussian distribution and visualize where the plane could have been at different time points.


t=48 hrs, σ=1
t=72 hrs, σ=1
t=120 hrs, σ=1
t=48 hrs, σ=10
t=72 hrs, σ=10
t=120 hrs, σ=10
t=48 hrs, σ=25
t=72 hrs, σ=25
t=120 hrs, σ=25

Comments on where one should concentrate search activities based on the observed results

It's quite natural to assume a normal distribution of locations around which debris could have been placed. Since this approach provides us a controlled way to increase the search space (just increase σ), it can be efficient, since we have to explore less locations till finding the debris.

However, because we only have data for 100-time instances, separated 3 hours each, technically, we can only run such simulation for a total of 300 hours. Hence, simulation can't extend beyond the time for which we don't have any (flow) data (i.e., larger time scales).

We can see from the previous exercise that in the debris simulation based on this time scale, the points did not move too much or too far.

Problem 4 [20 pts] Creating a Gaussian process model for the flow.

In this problem, we will create a Gaussian process model for the flow given the information we have.

Problem 4.a: **[10 points]** **Include your answer to all parts of this problem (4.a) into your written report.**

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. That means you will do the following for each direction, and you will obtain a model for each direction at that particular location. Find the instructions below. For each step, please clearly state your selections, your thought process, and design choices.

Recall the steps to estimate the parameters of the kernel function.

- Pick a kernel function. Here you are free to chose the kernel function you find adequate. Please clearly explain you selection.
- Identify the parameters of the kernel function. E.g. the the squared exponential kernel function: where the parameters will be .

- Find a suitable search space for each of the parameters. For you may wish to consider a range such as hours to hours (to time indices).
  You will find a good set of parameters via cross-validation. Pick a number and split the data -wise defining some training and some testing points. Each vector is of dimension , so let us say you picked . So you will have ten different partitions, where there are points for training and points for testing.

  For each possible set of parameters and each data partition.

- Build an estimate for the mean at each possible time instance. For example, you can estimate the mean as all zeros, or take averages – moving or otherwise – over your training data.

- Construct the covariance matrix for the selected kernel functions and the selected set of parameters.

- Compute the conditional mean and variance of the testing data points, given the mean and variance of the training data points and the data itself. Recall that:

- In this case, are the velocities for the testing data-points, are the velocities for the training data-points. are the mean velocities for the testing data-points. are the mean velocities for the training data-points. is the covariance of the training data-points. is the covariance of the testing data-points. is the cross-covariance. are the observed velocities for the training data-points. Finally, is the parameter indicating the variance of the noise in the observations. You can pick .
- Compute the log-likelihood performance for the selected parameters. You are free to select to compute it only on the testing data, or on the complete vector.

  For each possible set of parameters, you will then have the performance for each of the partitions of your data. Find the parameters that maximize performance. Save the computed cost/performance metric for each choice of parameters, then create a plot over your search space that shows this metric.

  Note: You may run into numerical issues during this step. If you do, consider some of the following:

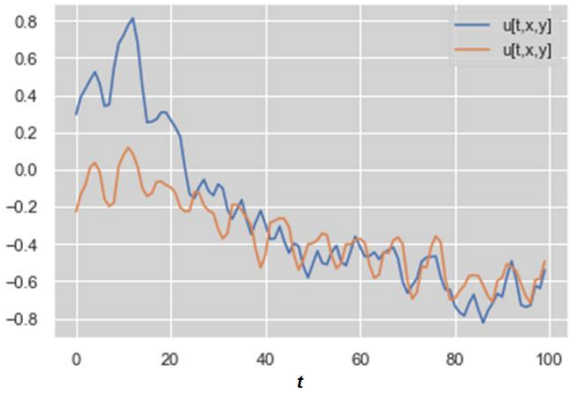
- Ensure you are working with -bit (double precision) floating point representations. Some linear algebra libraries default, or even silently enforce (such as `JAX`) -bit representations.

- Use routines for the solution for linear systems of equations to invert your matrices, avoid computing inverses directly. In Matlab, you can use the `\` operator, `numpy`'s `linalg.solve` routine can take two matrices as arguments.

- Large length-scales can cause your covariance matrix to be near-singular. This is caused by tight correlations between neighboring predictions. If you run into this issue you may wish to try: sparsifying your predictions by skipping every second element, or increasing the number of folds in your cross-validation; whitening your data; choosing a different kernel.

Solution

Here using the same point (100, 350) where the toy plane crashed.



Choice and the parameters of kernel function

Squared exponential (RBF) kernel was chosen as kernel function.

$$K(z_i, z_j) = \sigma^2 \exp\left(-\frac{\|z_i - z_j\|^2}{\ell^2}\right).$$

where the parameters will be $\theta = \{\sigma, \ell\}$.

I think it makes sense to assume that the similarity between the flow velocities decrease exponentially with the square of the time difference they were recorded at.

Search space for each kernel parameter

The search space is the Cartesian product of the following values of the hyper-parameters l and σ (each of size 10, chosen at uniform gaps in between 0.1 and 5),

as shown below, with 100 possible combinations of hyper-parameter values in the search space.

```
l
```

```
array([0.1       , 0.64444444, 1.18888889, 1.73333333, 2.27777778,
       2.82222222, 3.36666667, 3.91111111, 4.45555556, 5.        ])
```

```
σ
```

```
array([0.1       , 0.64444444, 1.18888889, 1.73333333, 2.27777778,
       2.82222222, 3.36666667, 3.91111111, 4.45555556, 5.        ])
```

The number of folds (k) for the cross-validation

10-fold cross-validation was used (k=10). The following code shows how the average marginal log-likelihood over k folds was computed on the held-out test dataset. The prior mean (μ1) is set to mean of all the 100 data points.

```
z = u[:,x,y]
#z = v[:,x,y]
k = 10
l = np.linspace(0.1, 5, 10)
σ = np.linspace(0.1, 5, 10)
ll_mean_k_folds = np.zeros((len(l), len(σ)))
for i in range(1, k+1):
    x2, y2, x1, y1 = get_train_test_fold(z, i, k)
    ll = np.zeros((len(l), len(σ)))
    for i in range(len(l)):
        for j in range(len(σ)):
            Σ22 = compute_sigma(x2, x2, σ[j], l[i])
            Σ12 = compute_sigma(x1, x2, σ[j], l[i])
            Σ21 = compute_sigma(x2, x1, σ[j], l[i])
            Σ11 = compute_sigma(x1, x1, σ[j], l[i])
            Σ1_2 = Σ11 - Σ12@(np.linalg.solve(Σ22 + τ*np.eye(len(y2)), Σ21))
            ll[i,j] = compute_log_marginal_likelihood(y1, Σ1_2)
    ll_mean_k_folds += ll
ll_mean_k_folds /= k
```

Optimal kernel parameters from the search

For the GP model for the u-flow direction:

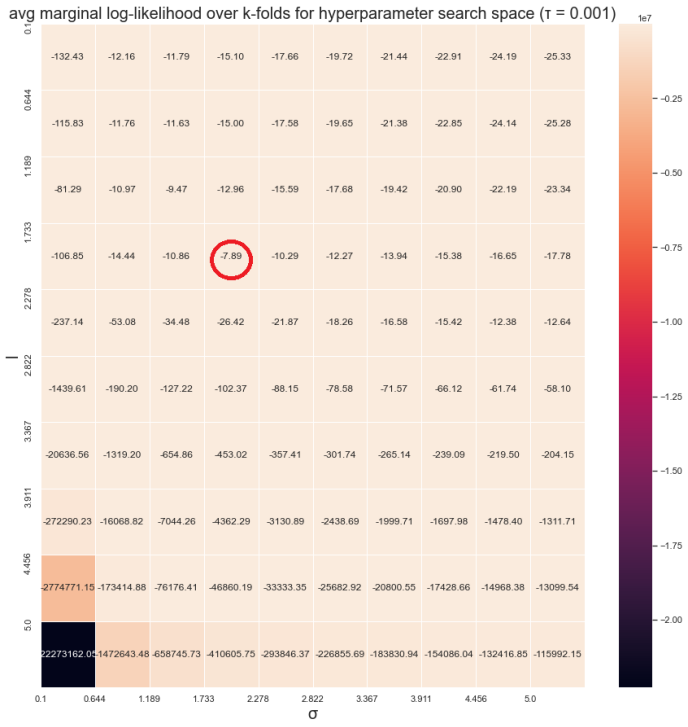
$l = 1.733$, $\sigma = 1.733$, with corresponding maximum marginal-log-likelihood = -7.89, as shown in the next heatmap.

For the GP model for the v-flow direction:

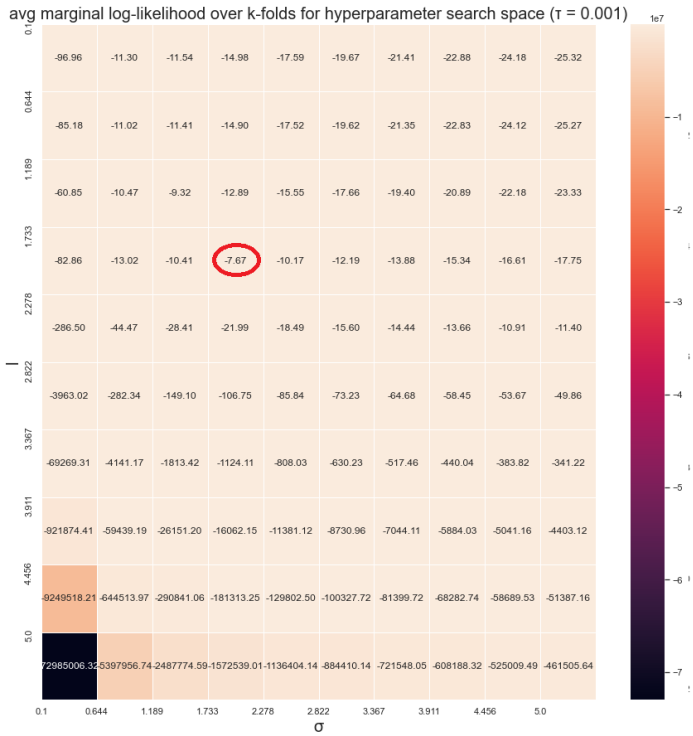
$l = 1.733$, $\sigma = 1.733$, with corresponding maximum marginal-log-likelihood = -7.67, as shown in the next heatmap.

Plot of the computed cost/performance metric over the search space for the kernel parameters

For the GP model for the u-flow direction:



For the GP model for the v-flow direction:



Problem 4.b:[5 points]Include your answer to all parts of this problem (4.b) into your written report.

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

Optimal kernel values for three new locations that are different from the last location

(407,350): optimal GP hyper-parameters obtained with 10-fold CV: $l = 1.733$, $\sigma = 1.733$ for u-flow and $l = 1.733$, $\sigma = 1.733$ for v-flow direction
(100,33): optimal GP hyper-parameters obtained with 10-fold CV: $l = 1.733$, $\sigma = 1.733$ for u-flow and $l = 1.733$, $\sigma = 1.733$ for v-flow direction
(8,450): optimal GP hyper-parameters obtained with 10-fold CV: $l = 1.733$, $\sigma = 1.733$ for u-flow and $l = 1.733$, $\sigma = 1.733$ for v-flow direction

For both the hyper-parameters, a pattern is observed:

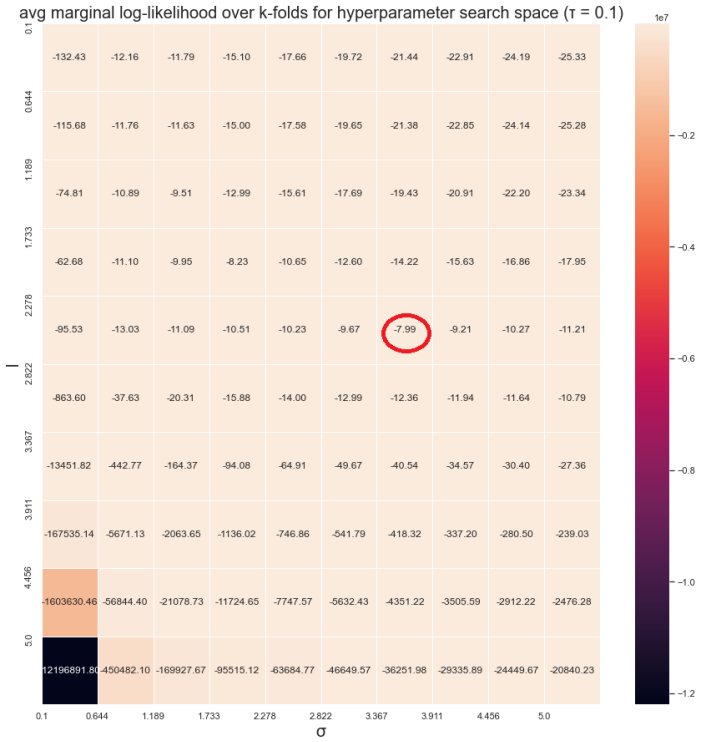
we get maximum average log-likelihood on held-out folds with the same values of l *(length-scale)* and σ parameters, typically at $l = 1.733$, $\sigma = 1.733$.

Problem 4.c:[5 points]Include your answer to all parts of this problem (4.c) into your written report.

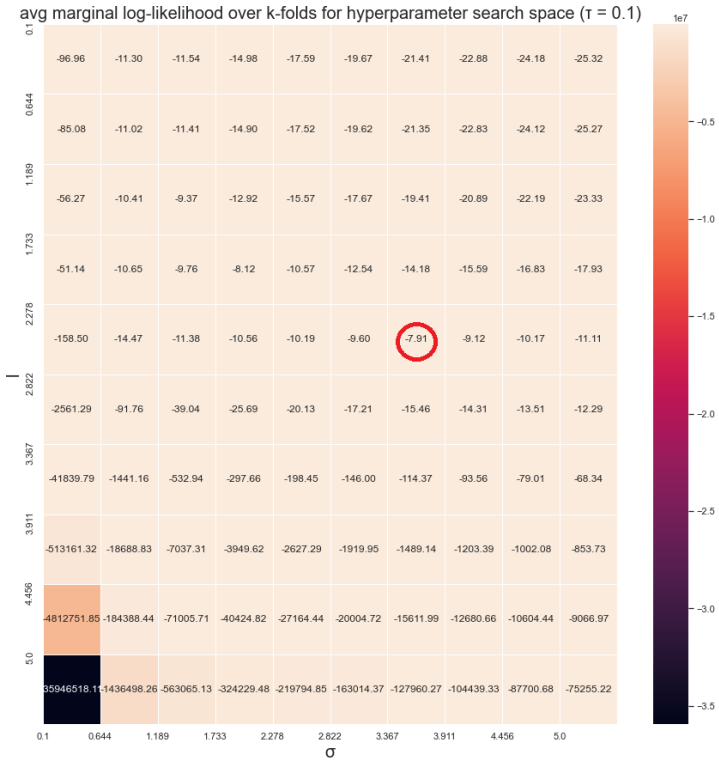
We have suggested one particular value for τ. Consider other possible values and comment on the effects such parameter has on the estimated parameters and the estimation process's performance. Try at least two values different from that used in Problem 4.a.

(407,350), **τ=0.1**: optimal GP hyper-parameters obtained with 10-fold CV: l = 2.278 , σ = 3.367 for u-flow and l = 2.278, σ = 3.367 for v-flow direction.

Heatmap for u flow

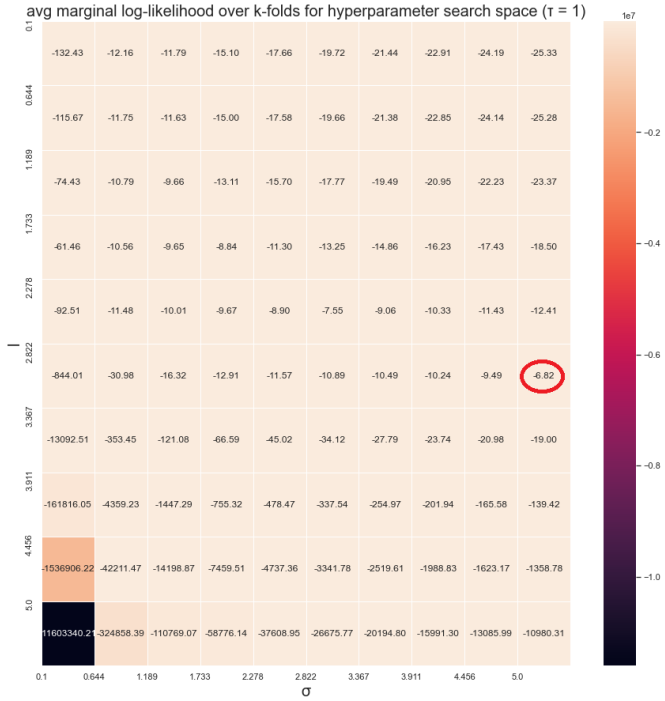


Heatmap for v flow

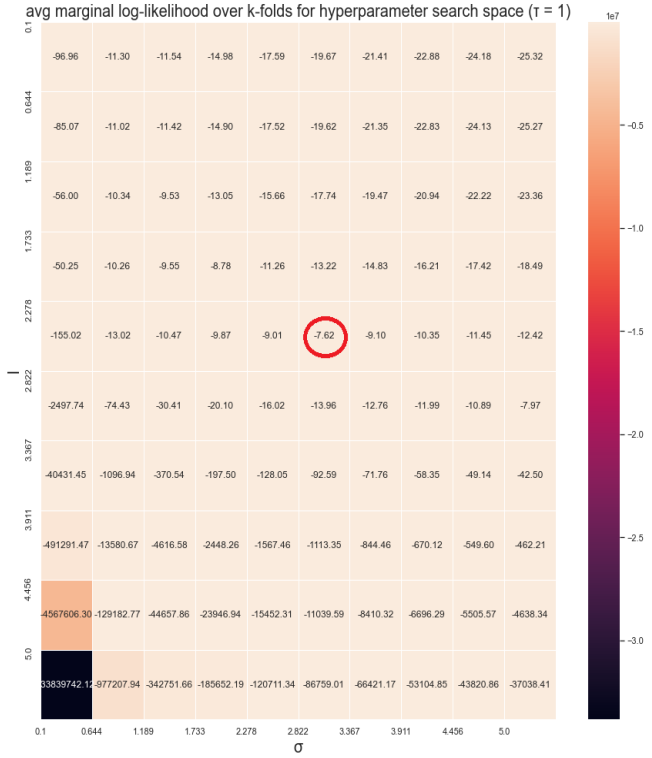


(407,350), **τ=1**: optimal GP hyper-parameters obtained with 10-fold CV: l = 2.822 , σ = 5.0 for u-flow and l = 2.278, σ = 2.822 for v-flow direction.

Heatmap for u flow



Heatmap for v flow



As can be seen from above, the optimal values for **τ=0.1** and **τ=1** differ from those found in with **τ= 0.001**, and also they differ from each other too.

Problem 4.d:[10 bonus points]Include your answer to all parts of this problem (4.d) into your written report.

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

- **Optimal kernel parameters as found through the software library**

  (length scale) $l = 3.465$, $\sigma = 0.367$ (as shown in the next figure)

- **Details on the library used:**

  Gpy - A Gaussian Process (GP) framework in Python

  https://gpy.readthedocs.io/

- **Whether these results differ from those found in Problem 4.a / explanation.**

  The results seem to be little different from the ones obtained in 4.a. This may be due to the grid search values used, and / or different initialization values used for the prior mean and difference in the noise variances used. The next figures show the comparison of the predictions generated (the predictions for the u-flow direction is shown).

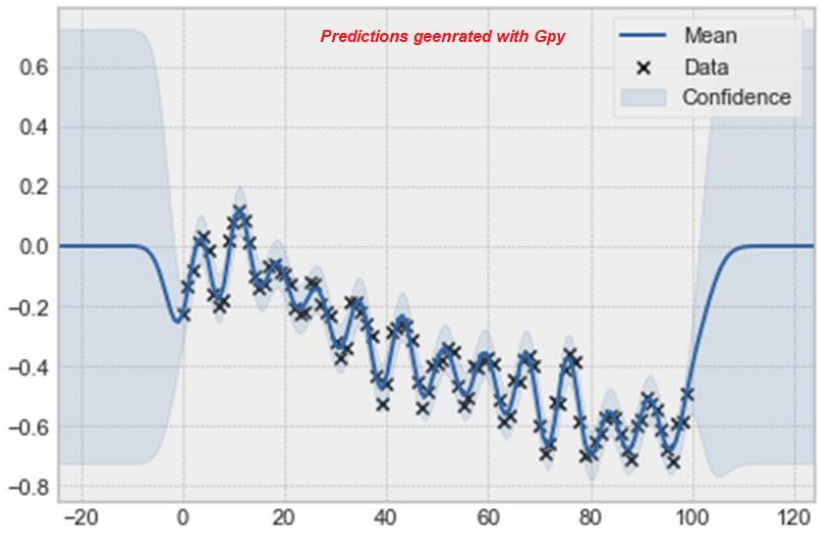
```
import GPy
kernel = GPy.kern.RBF(input_dim=1, variance=1., lengthscale=1.)
model = GPy.models.GPRegression(np.arange(100).reshape(-1,1), z.reshape(-1,1), kernel)
model.constrain_positive("noise")
model.randomize()
model.noise_variance = .001
model.optimize(messages=True, max_iters=1e5)
model.plot(levels=10)
```

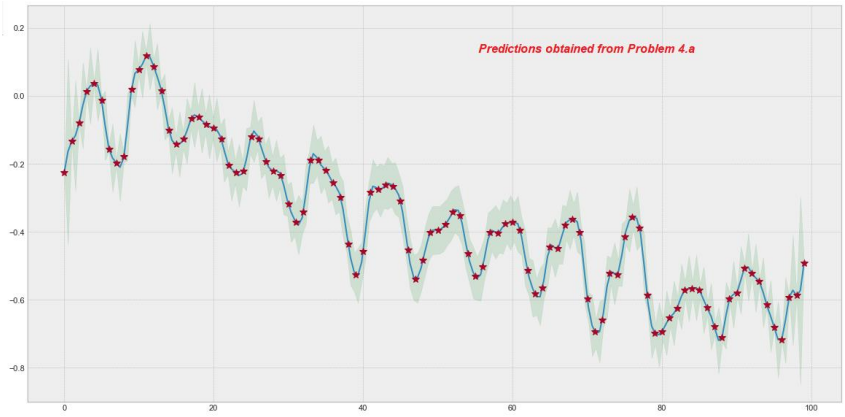
```
WARNING:GP regression:reconstraining parameters GP_regression
```

optimizer	L-BFGS-B (Scipy implementation)
runtime	00s18
evaluation	000018
objective	-1.107E+02
||gradient||	+2.851E-12
status	Converged

```
Model: GP regression
Objective: -110.65842465710392
Number of Parameters: 3
Number of Optimization Parameters: 3
Updates: True
```

GP_regression.	value	constraints	priors
rbf.variance	0.13529554482249784	+ve	
rbf.lengthscale	3.4648205958989213	+ve	
Gaussian_noise.variance	0.0012448713812574885	+ve	





Problem 5: Estimating unobserved flow data. **[15 points]** **Include your answer to all parts of this problem (5) into your written report.**

Problem 5 [15 pts] Estimating unobserved flow data.

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, clearly showing:

- The predicted means.
- The predicted standard deviation as a band (three standard deviations above and below the mean).
- The observed data points.

Solution

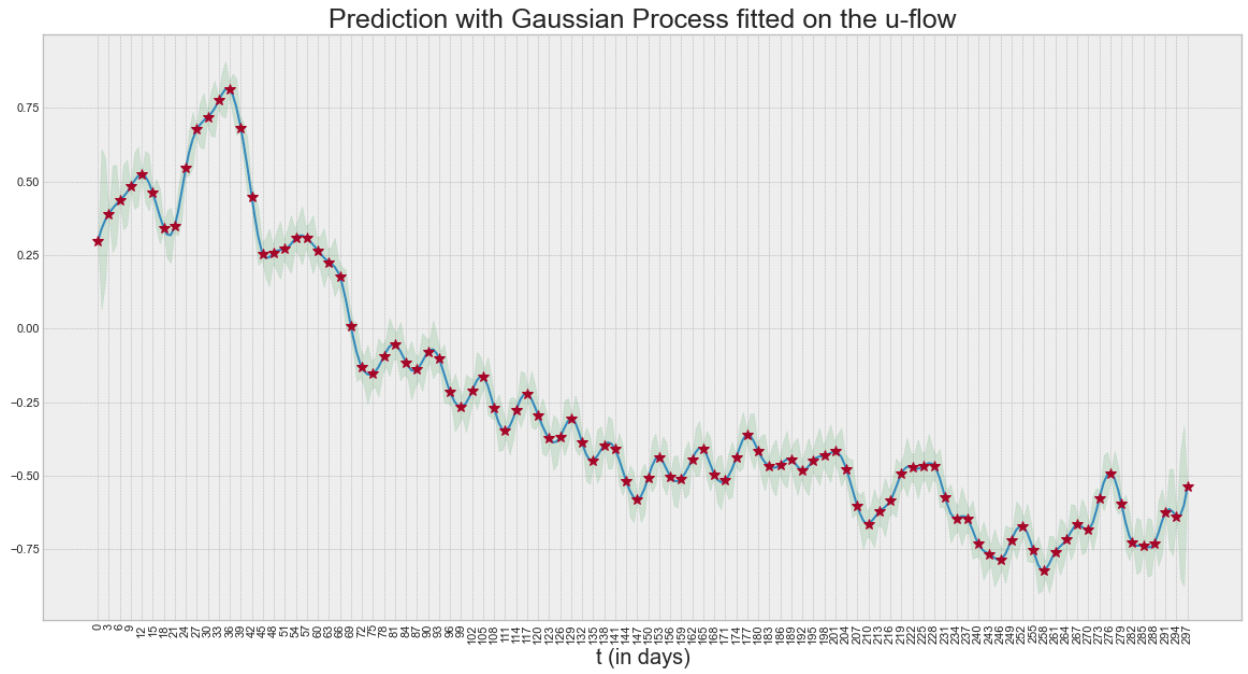
The **predicted means** are shown in as **blue continuous line** and the observed data points are shown as **red stars** in the next figure.

- **Choice of time-stamps at which to create predictions**

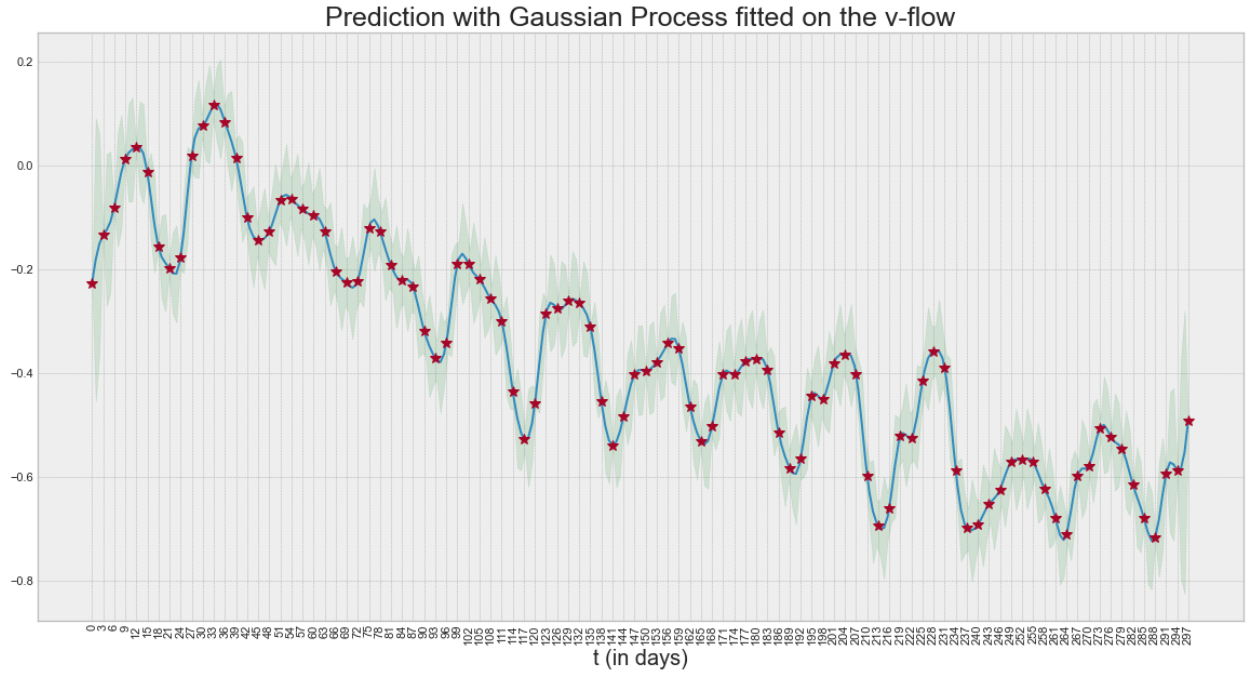
  Wanted flows every day, so two unknown points were chosen between every two observations, so in total we have 300 datapoints to predict, corresponding to 300 days.

- **The prior means** were chosen as the mean of the entire (observed) dataset (i.e., u and v velocities corresponding to the 100 points, respectively).

- The **horizontal** velocity component at the chosen location (100, 350).



- The **vertical** velocity component at the chosen location.



The code used to compute the prediction:

```
z = u[:,x,y] #v[:,x,y]
μ1 = np.mean(z)
x = np.arange(0, 100)
x_ = np.linspace(0, 99, 300)
#μ1 = np.zeros(len(x_))
Σ22 = compute_sigma(x, x, σ, l)
Σ12 = compute_sigma(x_, x, σ, l)
Σ21 = compute_sigma(x, x_, σ, l)
Σ11 = compute_sigma(x_, x_, σ, l)
μ2 = np.mean(z)
μ1_2 = μ1 + Σ12@(np.linalg.solve(Σ22 + τ**2*np.eye(len(z)), z - μ2))
Σ1_2 = Σ11 - Σ12@(np.linalg.solve(Σ22 + τ**2*np.eye(len(z)), Σ21))
```

Problem 6 [28 pts] A longer time-scale simulation.

In the previous problems, we learned to model the flow at one location as a Gaussian process. Thus, we can extend this to estimate the flow at any point in time at that particular location using the kernel function parameters. At a certain point in time, the flow can be computed as the realization of a multivariate Gaussian random variable with parameters given by the conditional distributions given the flow data. At this point, you are asked to simulate a particle moving according to the flow data and using the estimates for times between the original timestamps of the problem.

Ideally, one would have to estimate the parameters of the flow at every point in the map. However, having to run parameter selection models seems like much computational work. So, here we take a more straightforward approach: use your results from Problem 4.b to choose a value of your kernel parameters that is generally representative of the points that you tested.

Problem 6.a: **[15 points]** **Include your answer to all parts of this problem (6.a) into your written report.**

Solution

- Plot with the initial / intermediate / final state of the simulation with GP (used 10 sample points from the Gaussian distribution with mean location (100,350) and s.d. 5):

- Marking a location **on the coast** and another one **over the ocean** of the final state of the simulation where one should search for debris **and** provides a justification.

As can be seen from above, the debris is most likely to land at (112, 380), keeping in view of the smoothness introduced by GP while getting carried away.

- Plots (initial, intermediate, final) for one other choice of σ.

As can be seen the conclusion changes with higher standard deviation, the debris are more dispersed and can potentially land at any of the above two locations.

Problem 6.b: **[14 points]** **Include your answer to all parts of this problem (6.b) into your written report.**

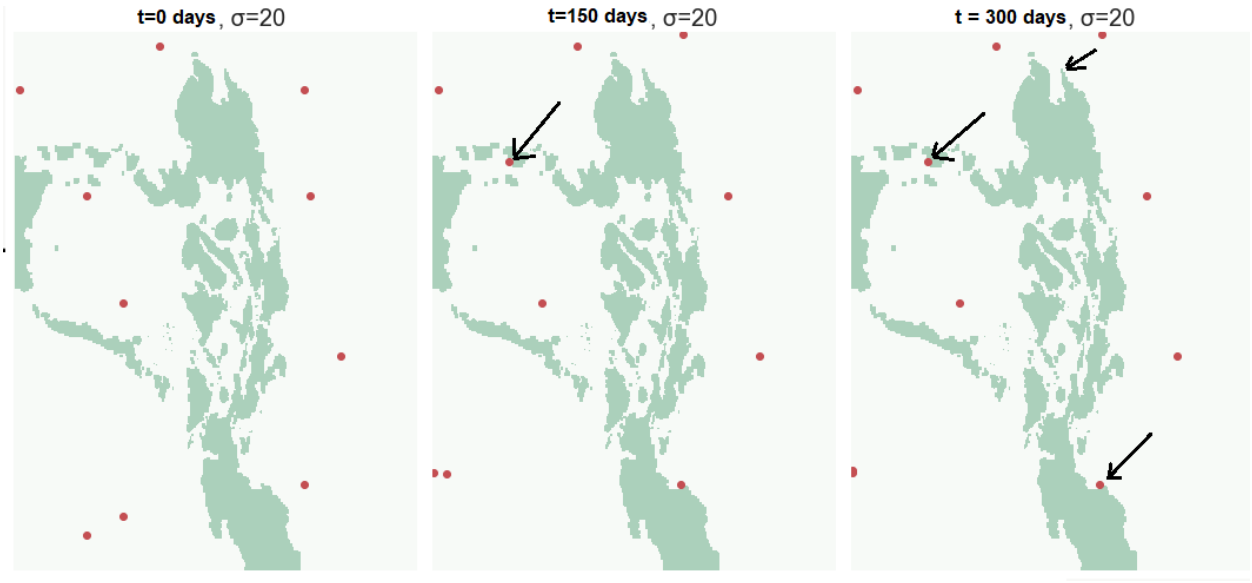
Thanks to your efforts, most parts of the (toy) plane were found, either inland or in the sea. As a final stage, you are tasked with locating three new monitoring stations on the coast. The purpose of these stations is to monitor general ocean debris. Using the tools you have build in this homework, 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

- **Plot with the initial / intermediate / final state of the simulation**
- **Three locations on the final state of the simulation where monitoring stations should be placed.**



- **Explanation for choosing these locations.**

  Combined with the above simulation and looking at the following flow velocities obtained earlier, we can see that the debris at certain parts in the sea are likely to end up to those locations of the shore, following the flow directions.