Homework Report #5: Fluid Simulation

Author: Isaac Han

Email: cogitoergosum01001@gmail.com

Class: Data Analysis: Statistical Modeling and Computation in application

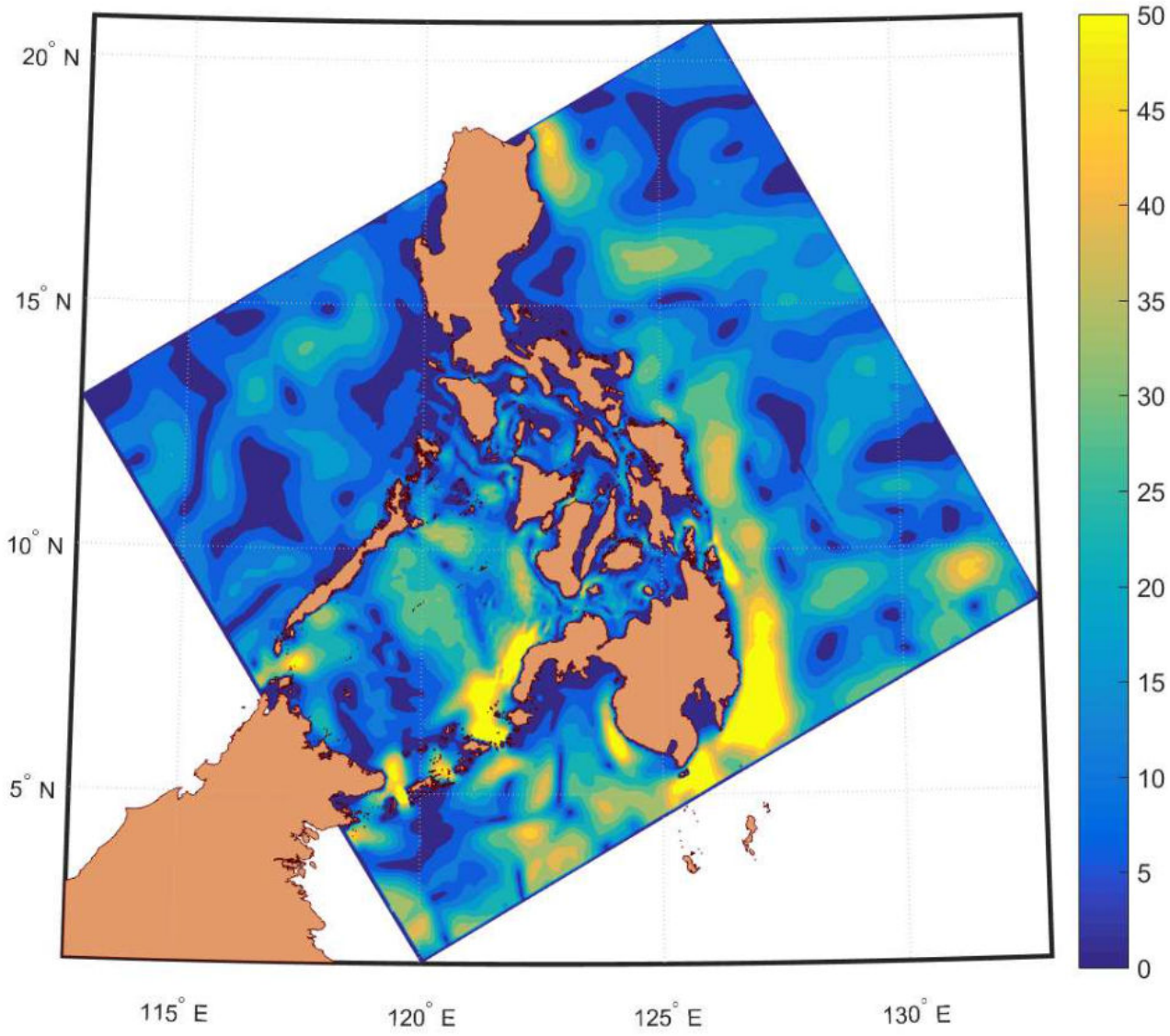
Professors: Prof. Uhler, Prof. Jegelka

Table of Contents

Abstract

Introduction

This paper concerns with the ocean current flow analysis of the Phillippine Archipelago. This region is an interesting multiscale ocean region, with mmultiple straits, islands, steep shelf-breaks.



Data Set

The data set that this paper would analyze is called OceanFlow. This data set contains the ocean flow of the Phillippine Archipelago. The numerical data represents an averaged flow from the surface to either near the bottom or 400m of depth, whichever is shallower. The data is measure with 2-D of coordinate data and time vector, resulting in 3-dimensional data representing an ocean flow.

Additional Info: Units and Specifics

The data were collected in Jan 2019. Flows are a velocity measured in km/h. The sampling data frequency is 3hrs. Thus 100 data observation represents 300 hrs of observing. 0hrs represent an arbitrary starting point in time. The grid spacing used is 3km. Matrix index with (0,0) represent the **bottom, left** of the plot, which would be consistant with plotting functions with MATLAB and Python3.

Autograded Part

Data Extraction & Q1-a

```
% untar('release_gp_homework_data.tar.gz')
addpath('OceanFlow')
% answer_Q1 = variance_of_data;
% min_ans = min(answer_Q1(answer_Q1~=0))
% lin_ind = find(answer_Q1 == min_ans)
% [row,col] = ind2sub([504,555], [lin_ind])
```

Q1-b: Find Max_x velocity

```
% [rows,columns] = deal(504,555);
% [ans_Q2_value,ans_Q2_idx] = max_x
% [dim1,dim2,dim3] = ind2sub([rows,columns,100],ans_Q2_idx)
```

Q1-c: Find Average Velocity Vector

After solving the question, u represents velocity in y, v represent velocity in x

```
% [rows,columns] = deal(504,555);
% [avg_x_y] = average_velocity
```

Written Report

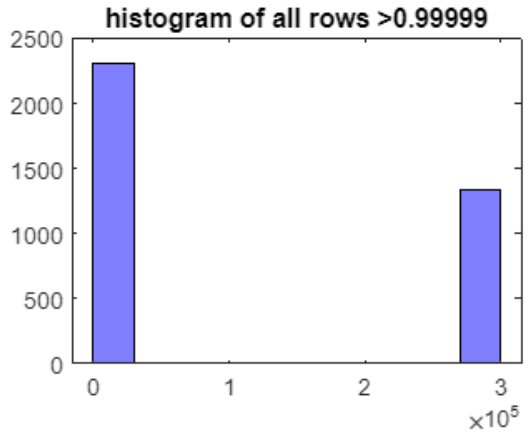
Q2: Identify long-range correlations

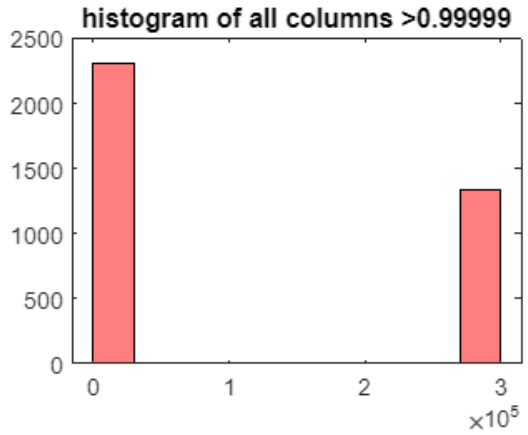
1. Calculating entire data set is expensive. Don't try that.
2. Set threshold of correlation high so only a few correlation would be found.
3. a cluster of points might be correlated with another cluster.
4. It is suggested to find a correlation for each direction separately.

As achieving element-wise accuracy is expensive and potentially not desired, since geographically clustered points would likely correlated with ot her geographically clustered data, I devised the following candidate search scheme.

1. Using a series of transformation, including permutation, reshape, and filter, constructed a 2_D Table whose columns represents a linear index of rows and columns of original data(a scalar that can track rows columns of original data). i.e. matrix of the dimension of 100 by 279720.
2. Find correlationship matrix of local neighbors.If there exists a correlation greater than 0.99999, indexes are added to lists.
3. Neighbors are constructed by the following. After filtering zeros, leaving 233904 data points, I constructed a search space in size of 15000. Therefore, making severak 15000 by 150000 correlation matrices.
4. Converted diagonal entries to zero and made the matrix into a triangular matrix by converting lower parts to 0's.
5. Found all pair-wise correlations with greater than 0.99999 within neighbors.

```
[rows,columns] = deal(504,555);
[A, B] = readData(rows, columns);
[T, a, b] = CalculateAndPlotHistogram(A, rows, columns);
```



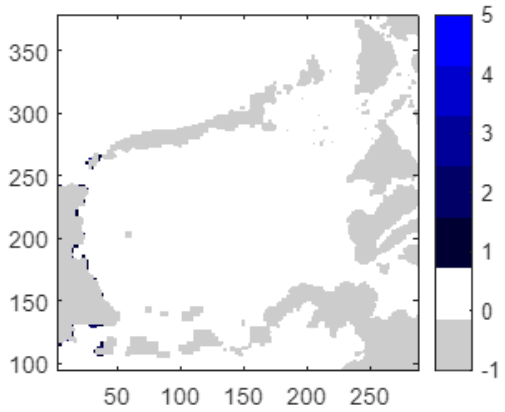



```
land_table = T(:,~any(table2array(T)));
land = [cellfun(@str2num, land_table.Properties.VariableNames)];
[positive,negative] = CalculatePosandNeg(a, b, T);
```

Since paris of coordinates are found, from upper triangular matrix, two sets of histogram were made. The first being the first coordinates and second being seconds coordinates. Even though they look identical, they are not necessarilly the same as I took correlation matrix to be upper triangular, which means non-symmetric matrix.

This histogram demonstrate there are two clear clusters. Now, it became possible to reduce down the search space by only considering points appeared in the histogram. Adding two histogram made about 7300 points and there are unique elements of 1702 points.

```
MapImagesc(positive,negative, rows, columns, land);
set(gca,'YDir','normal')
map = [0.8 0.8 0.8; 1 1 1; 0 0 0.2;0 0 0.4 ; 0 0 0.6; 0 0 0.8; 0 0 1];
colormap(map)
colorbar

xlim([3 289])
ylim([95 379])
```




```

```

-1 value indicates the land, 0 indicates no correlation found in the sample, and higher integers indicate how many correlations there were. The maximum correlation found were 5.

It can be seen that the almost absolute correlation(correlation>0.99999) came from the simple fact that the land only permites current to move in one direction. The magnitudes are quite similiar too since waves are shielded by lands so that external currents would not influence the currents of the shores.

There doesn't exist negatively correlated places whose relatedness is less than -0.99999. When threshold are lowered however to -0.93, a few points can be found.

```
fprintf('There are total of %d numbers of uniquely negatively correlated points',
size(unique(negative),2))
```

```
There are total of 12 numbers of uniquely negatively correlated points
```

```
fprintf('All of them are also parts of positively correlated parts, however')
```

```
All of them are also parts of positively correlated parts, however
```

```
disp('')
disp('')
fprintf('ismember(unique(negative),positive)')
```

```
ismember(unique(negative),positive)
```

```
disp(ismember(unique(negative),positive))
```

```
   1   1   1   1   1   1   1   1   1   1   1   1
```

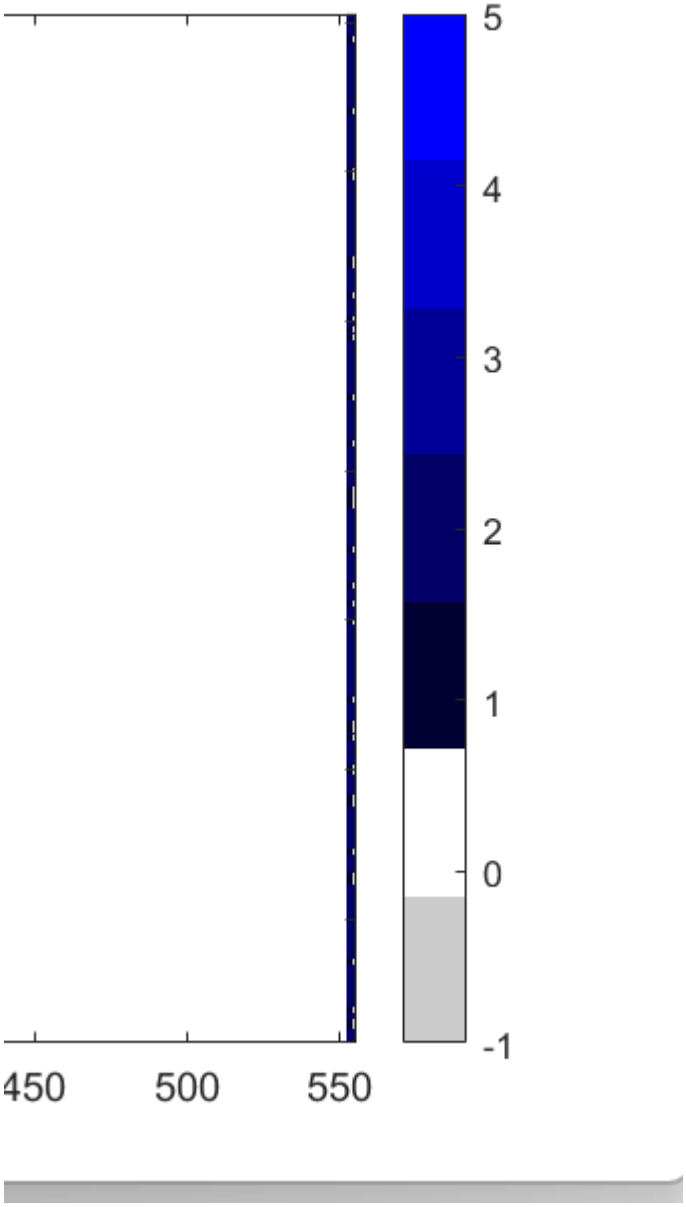
```
fprintf('They are parts of positively correlated pairs')
```

```
They are parts of positively correlated pairs
```

```
temp = zeros(rows,columns);
temp(land) = -1;
map_phillipe = temp;
save variables map_phillipe
```

Lastly,a few limitations are explained.

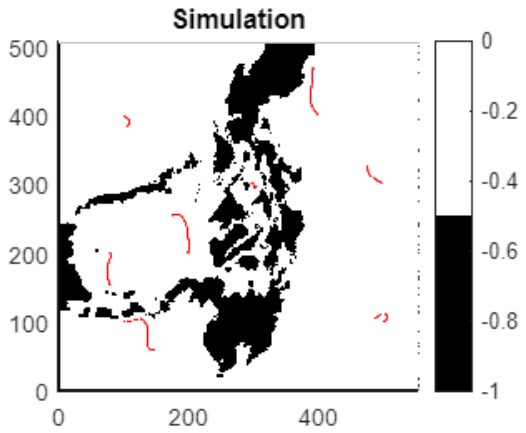
1. A band of positive correlations appeared on the right edge of the graph. I disregarded these as data set error as I can see a few land masses which doesn't appeared in the map.
2. Candidates vs rest correlations were not performed since it already yielded satisfactory results in the search space of 15000 matrix distance.



Q3: Simulating Particle Movement in flows

```
% clc
% clear
f = clf("reset");
load variables.mat
[rows,columns] = deal(504,555);
[u,v] = readData(rows,columns);
xp_final = cell(1,5);
xp_final{1} = simulate_path(100,100, u, v);
xp_final{2} = simulate_path(200,200,u,v);
```

```
xp_final{3} = simulate_path(300,300,u,v);
xp_final{4} = simulate_path(400,400,u,v);
xp_final{5} = simulate_path(100,500,u,v);
xp_final{6} = simulate_path(100,300,u,v);
xp_final{7} = simulate_path(400,100,u,v);
xp_final{8} = simulate_path(300,500,u,v);
xp_final{9} = simulate_path(200,80,u,v);


hold on
imagesc(map_phillipe)
map = [0 0 0; 1 1 1];
set(gca,'YDir','normal')
xlim([0 555])
ylim([0,510])
title('Simulation')
colormap(map)
colorbar
for i  = 1:9
    temp = xp_final{i};
    plot(temp(2,:),temp(1,:),'r');
end
hold off
```



Simulation Algorithm

Following is the explanation of the simulation algorithms..

1. Epsilon, ϵ, the step enumeration is 10^6.
2. Time interval, T_{interval}, is therefore, $\frac{300h}{10^6}$.
3. Did not utilized break command. as the needed time was sufficiently small to consider. Further, not utilizing break command would further demonstrate the correctedness of the simulation.

Following is the specific methodology this report utilized.

1. Speed Datas are read by matrices with dimensions of 504X555X100, which corresponds to rows, columns, and time vector.
2. For a given initial location, following recursion was used; $X(t+1) = X(t) + T_{\text{interval}} * \text{speed}(\text{at } X(t))$.
3. Notice that location is determined by dropping off decimals. $i.e. 3.9 \rightarrow 3$.
4. After enumerating $\epsilon = 10^6$ steps, all coordinates were stored and plotted.

I utilized this methods since animation is supported in MATLAB interface but not in PDF.

Sample Initial Coordinates

Following is the simulation of 9 points. The way to read coordinate is (y,x,u,v).

```
xp{1} = simulate_path(100,100, u, v);
xp{2} = simulate_path(200,200,u,v);
xp{3} = simulate_path(300,300,u,v);
xp{4} = simulate_path(400,400,u,v);
xp{5} = simulate_path(100,500,u,v);
xp{6} = simulate_path(100,300,u,v);
xp{7} = simulate_path(400,100,u,v);
xp{8} = simulate_path(300,500,u,v);
xp{9} = simulate_path(200,80,u,v);
```

Notice that the first input is y coordinate and second is x coordinate. These specific points were chosen to demonstrate a path complete simulation.

Intermediate State

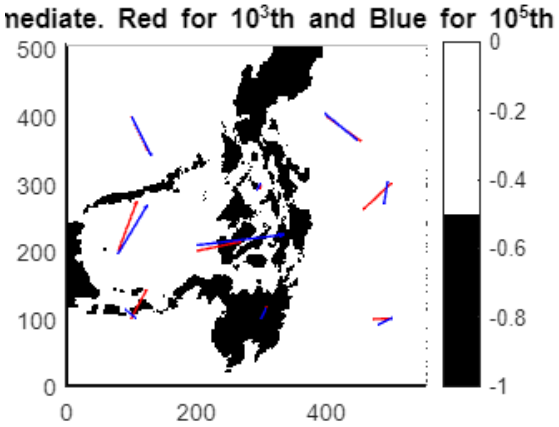
From the land data, the map was plotted. Then, the flow simulation is plotted on top of the map.

9 points are chosen to demonstrate path completeness of the simulation.

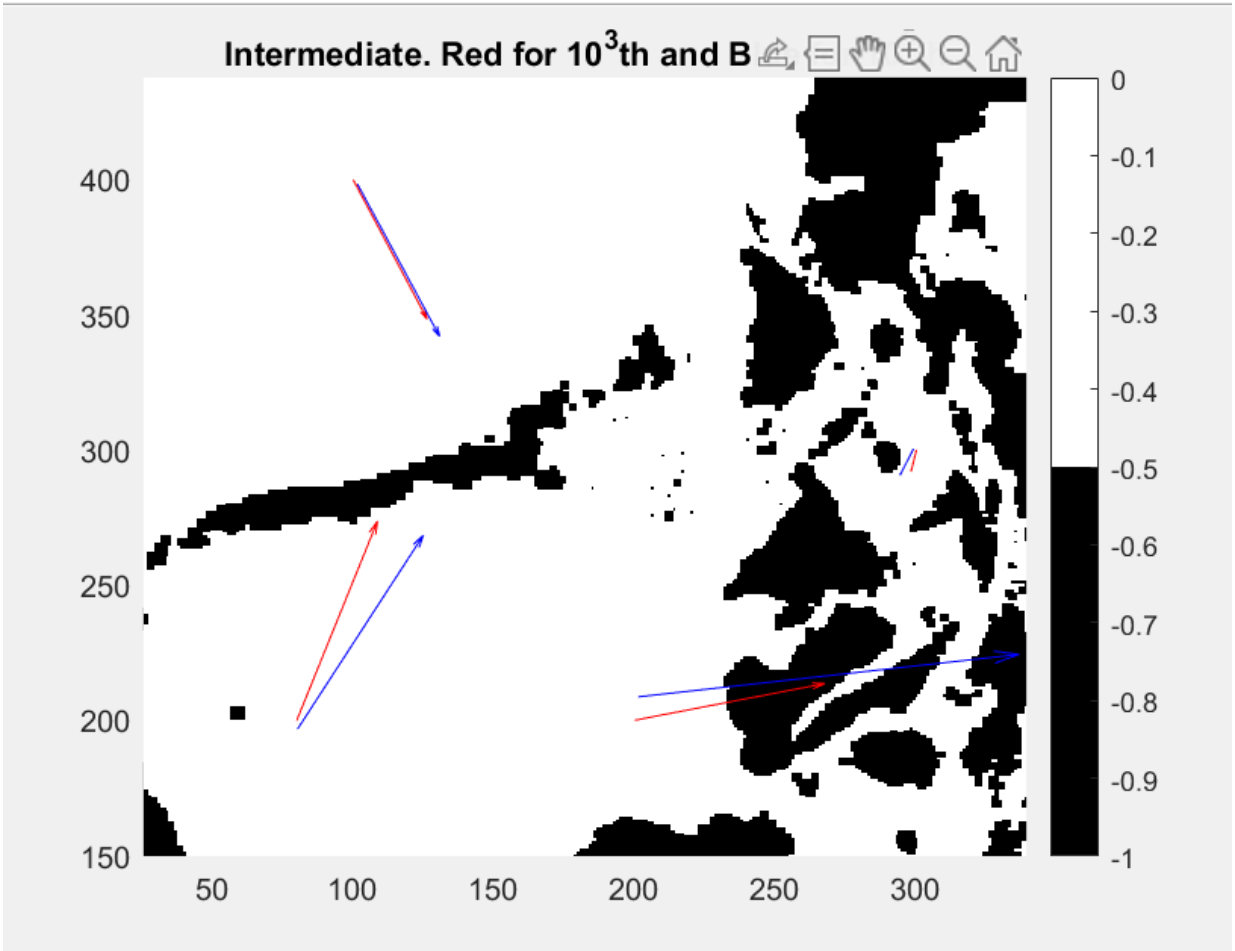
There were 10^6 numbers of enumerations. 10^3th and 10^5th enumerations were shown.

```
f = clf("reset");

epsilon = 10^6;
t_interval = 100/epsilon;
t = fix(1:t_interval:100);
t(1,size(t,2):epsilon)=1;

hold on
imagesc(map_phillipe)
map = [0 0 0; 1 1 1];
set(gca,'YDir','normal')
```

```
xlim([0 555])
ylim([0,510])
title('Simulation')
colormap(map)
colorbar
for i  = 1:9
    temp = xp_final{i};
    quiver(temp(2,10^3),temp(1,10^3),
100*v(fix(temp(2,10^3)),fix(temp(1,10^3)),t(10^3)),...
    100*u(fix(temp(2,10^3)),fix(temp(1,10^3)),t(10^3)),'r');

    quiver(temp(2,10^5),temp(1,10^5),
100*v(fix(temp(2,10^5)),fix(temp(1,10^5)),t(10^5)),...
    100*u(fix(temp(2,10^5)),fix(temp(1,10^5)),t(10^5)),'b');

end
title('Intermediate. Red for 10^3th and Blue for 10^5th')
hold off
```



Where red indicate 1000th enumeration and blue represent 10000th enumeration. The magnitudes of each arrows are scaled by 100 times since they are invisible when zoomed out.

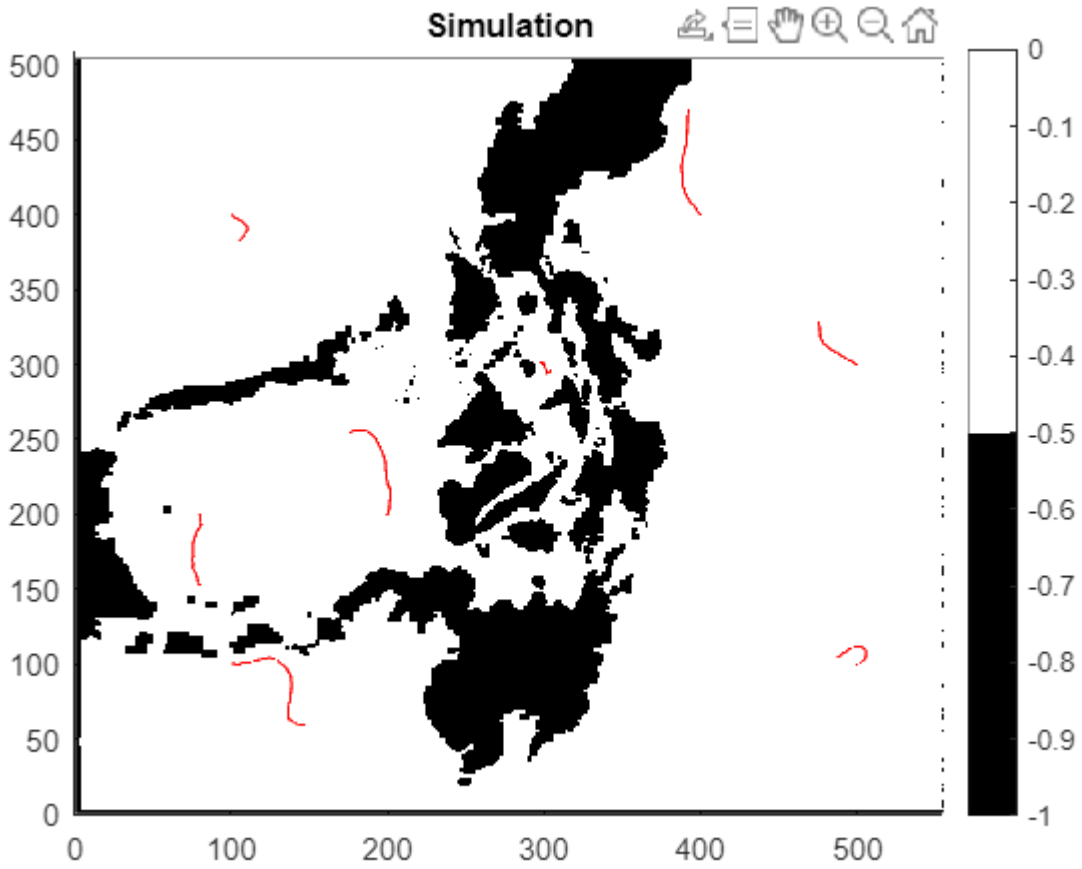


A specific region is magnified to give an insight.

Provides a plot of the final state of the simulation

Noticeably, it can be seen that in the middle of the island, the point with coordinate of (300,300), it travels very slowly, which concurs with out common sense. And the point with coordinate of (100,300) is not effectively shown, since it begins on land and stays on one point.

Therefore, this paper asserts that the flow simulation yields satisfactory results.



Animation is supported in MATLAB, but cannot be exported as PDF. Therefore, following the methodology introduced by the recitation, trajectories are stored and plotted.

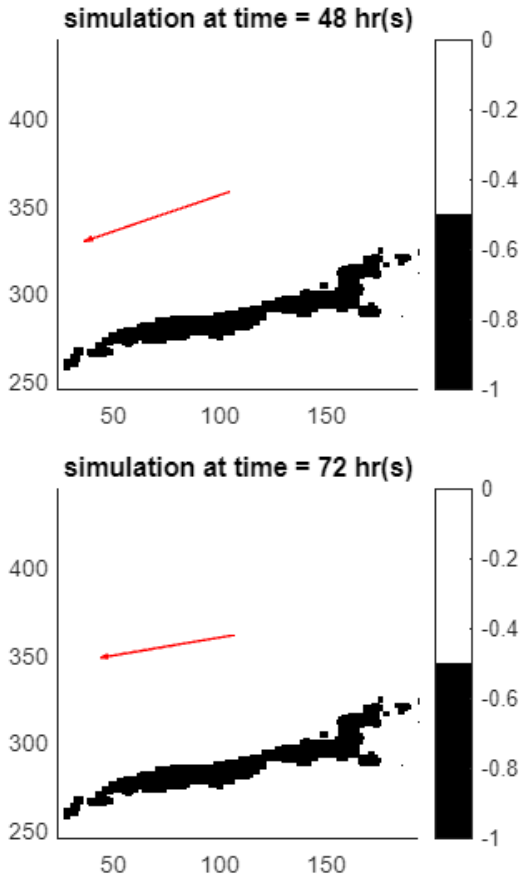
Q3-b: Plane Crash

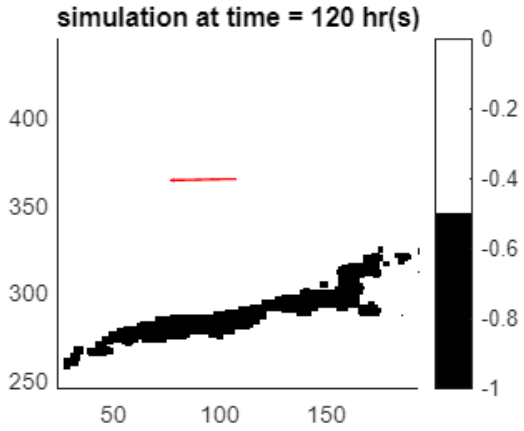
Plotting Simulations

1. Initial positions are (100, 350)
2. Time at (48,72,120)hrs translates to [16 24 40]*10^4th enumerations.
3. Used the same simulation process introduced in Q3-a.

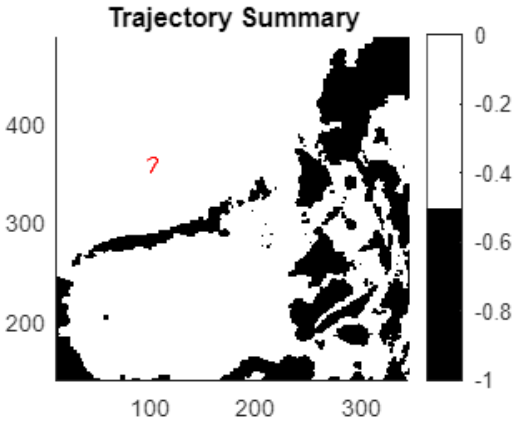
```
f = clf("reset");

initial_x = [101, 351];
time = [48 72 120];
time_of_interest = time./300*epsilon;

xp2 = simulate_path(initial_x(2),initial_x(1),u,v);
for i = 1:3
    figure()
    hold on
    imagesc(map_phillipe)
    map = [0 0 0; 1 1 1];
```

```
    set(gca,'YDir','normal')
    xlim([0 555])
    ylim([0,510])
    colormap(map)
    colorbar
    temp = xp2;
    temp2 =  time_of_interest(i);

quiver(temp(2,temp2),temp(1,temp2),50*v(fix(temp(2,temp2)),fix(temp(1,temp2)),t(temp
2)),...
        50*u(fix(temp(2,temp2)),fix(temp(1,temp2)),t(temp2)),'r')
    xlim([24 193])
    ylim([245 445])
    hold off
    title(sprintf('simulation at time = %.f hr(s)',time(i)))
end
```






```
figure()
hold on
imagesc(map_phillipe)
map = [0 0 0; 1 1 1];
set(gca,'YDir','normal')
xlim([0 555])
ylim([0,510])
colormap(map)
colorbar
plot(xp2(2,:),xp2(1,:),'r')
hold off
title('Trajectory Summary')
xlim([12 345])
ylim([140 488])
```



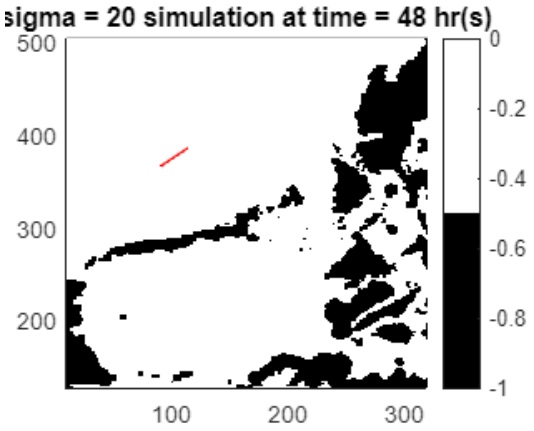
Two or more additional choices with variance

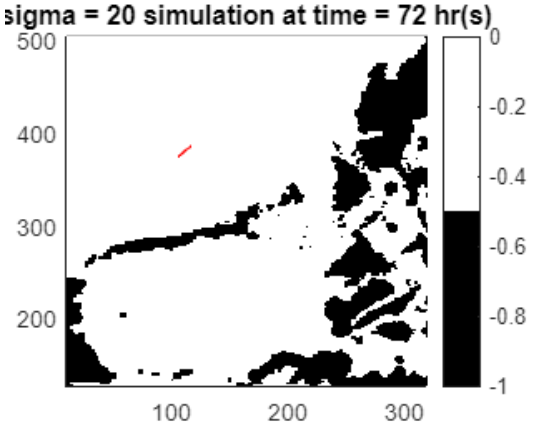
1. Initial position = $[100, 350] + \text{randn}(1, 2) * \sigma$ where randn is normally distributed random numbers and σ is a standard deviation. It was assumed that variance is not a function of orientation.
2. The rest of the procedures are the same as before.
3. random seed is set for the convenience of writing a report.

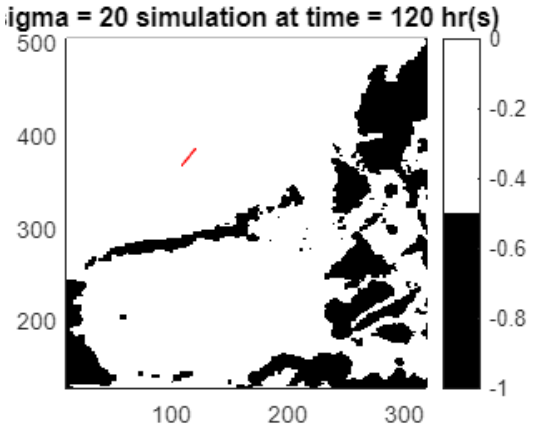
```
rng('default')
f = clf("reset");
```

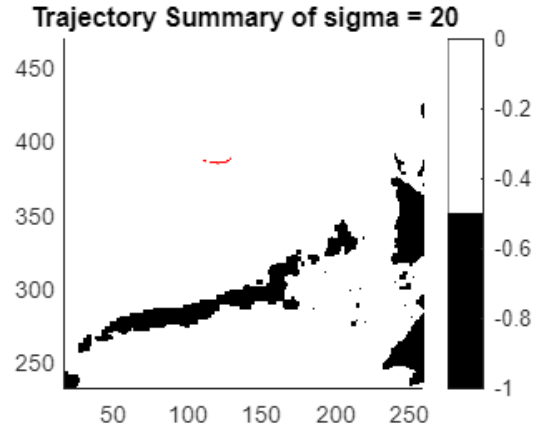
```
sigma_chosen = [20,30];


for j = 1:2
    initial_x = [100 350] + randn(1,2)*sigma_chosen(j);
    fprintf('the initial position is given as following')
    disp(initial_x)
    time = [48 72 120];
    time_of_interest = time./300*epsilon;

    xp2 = simulate_path(initial_x(2),initial_x(1),u,v);
    for i = 1:3
        figure()
        hold on
        imagesc(map_phillipe)
        map = [0 0 0; 1 1 1];
        set(gca,'YDir','normal')
        colormap(map)
        colorbar
        temp = xp2;
        temp2 =  time_of_interest(i);

quiver(temp(2,temp2),temp(1,temp2),50*v(fix(temp(2,temp2)),fix(temp(1,temp2)),t(temp
2)),...
            50*u(fix(temp(2,temp2)),fix(temp(1,temp2)),t(temp2)),'r')
        xlim([10 320])
        ylim([127 505])
        hold off
        title(sprintf('sigma = %.f simulation at time = %.f
hr(s)',sigma_chosen(j) ,time(i)))
    end
    figure()
    hold on
    imagesc(map_phillipe)
    map = [0 0 0; 1 1 1];
    set(gca,'YDir','normal')
    xlim([0 555])
    ylim([0,510])
    colormap(map)
    colorbar
    plot(xp2(2,:),xp2(1,:),'r')
    hold off
    title(sprintf('Trajectory Summary of sigma = %.f', sigma_chosen(j)))
    xlim([17 260])
    ylim([232 470])
end
```

```
the initial position is given as following
  110.7533  386.6777
```

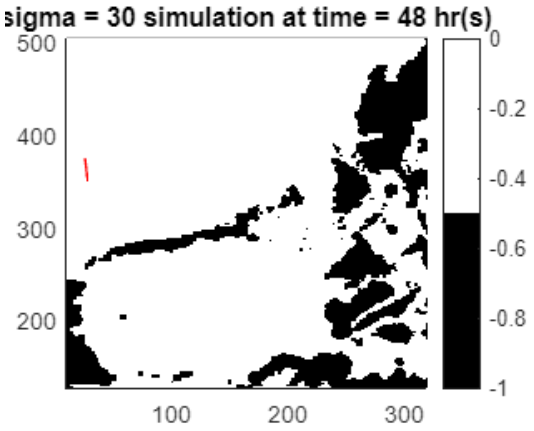


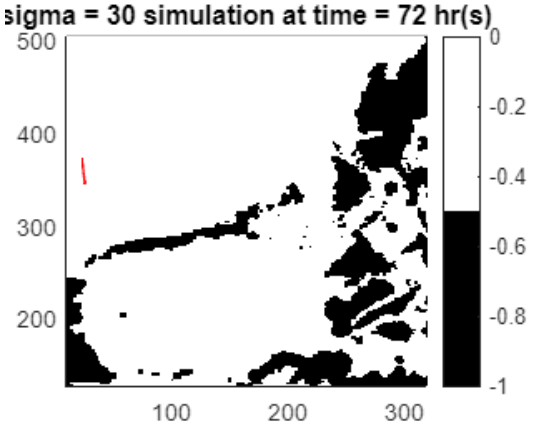


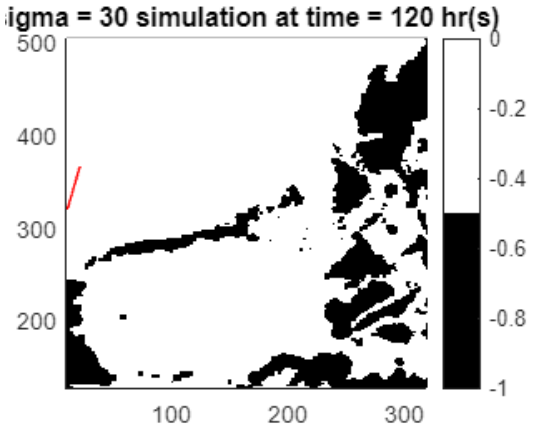


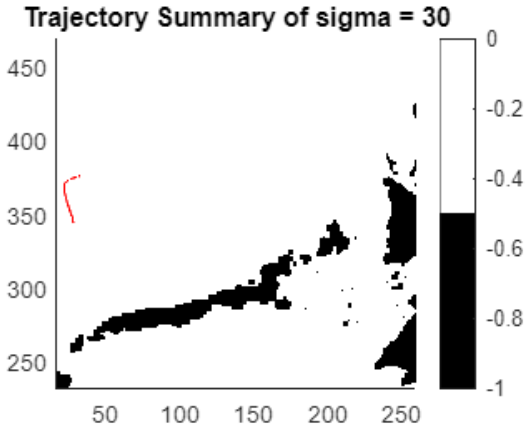



```
the initial position is given as following
   32.2346  375.8652
```


sigma = 30 simulation at time = 48 hr(s)
500
400
300
200
100
200
300
0
-0.2
-0.4
-0.6
-0.8
-1

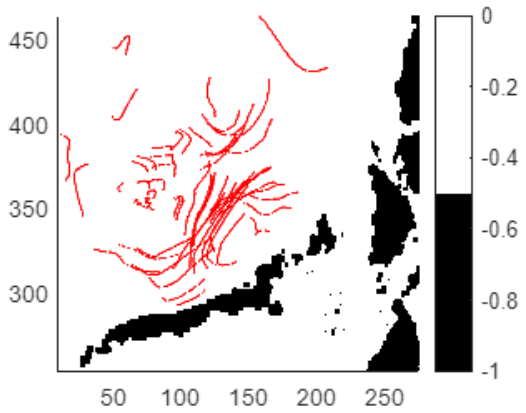

sigma = 30 simulation at time = 72 hr(s)
500
400
300
200
100
200
300
0
-0.2
-0.4
-0.6
-0.8
-1


igma = 30 simulation at time = 120 hr(s)
500
400
300
200
100
200
300
0
-0.2
-0.4
-0.6
-0.8
-1


Trajectory Summary of sigma = 30
450
400
350
300
250
50
100
150
200
250
0
-0.2
-0.4
-0.6
-0.8
-1

Search Space

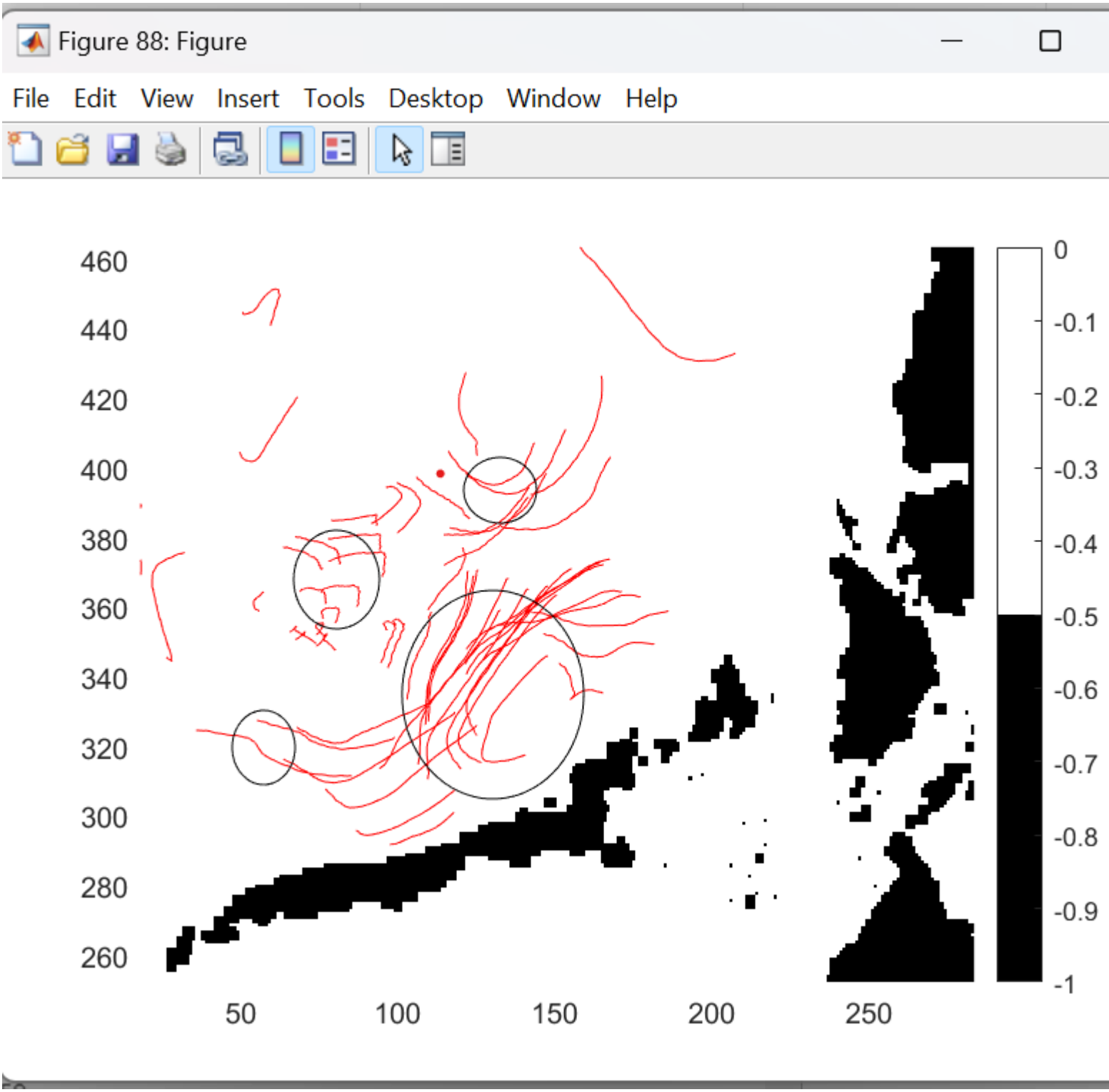
```
rng('default')
sigma_chosen = 30;

f = clf("reset");
figure(5)
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
xlim([10 275])
ylim([253 464])
colormap(map)
colorbar

for i = 1:60
    initial_x = [100 350] + randn(1,2)*sigma_chosen;
    xp2 = simulate_path(initial_x(2),initial_x(1),u,v);
    plot(ax,xp2(2,:),xp2(1,:),'r')

end
```



from simulation of 60 initial points with the standard deviation of 30, most of flows head to either (60,370) region or (120, 320).

In conclusion, for the upper bound, the search space would be x from 0km to 600km and y from 900km to 1380km.

To narrow down promising candidates, this paper propose the following locations of interest.



Written Portion#2

Estimating Flows with Gaussian Processes

Notice: The enumeration size was already $\epsilon = 10^6$. No further adjustment mentioned in the course is needed.

4-a

1. State Kernel and reason
2. Identify parameters
3. State Search Space(domain)
4. state k in k-fold partition
5. state parameters
6. Provide metrics of cost/performance over the search space

As suggested by the course, the kernel function, $K(z_i, z_j) = e(\sigma)^2 * e^{\left(-\frac{(z_i - z_j)^2}{e(l)^2}\right)}$, was used. Notice that squared exponential kernal is the most commonly used covariance function and is often default function that programs utilizes. Having degree of freedoms with σ, signal standard deviation, and l, characteristic length scale, this kernel function captures mosts of the gaussian processes. Trivially, z represents time data from 1 through100 and e represents the natural exponent..

The point (x,y) = (66,166) was chosen. Using Gaussian fitting function with initial point(1.5, 0,2), fitting parameters were found. The search space was not specified(entire space). Notice that local optimized variable near (1.5, 0.2) will be returned. And 10-fold partition was used for the cross-validation

The results of 10-fold partitioned training is the following, where the first row and second row represent l and σ, and the third row represents RMSE error of training. There are two sets of parameters as there are u and v.

```
region_x = 66;
region_y = 166;
modelfun = @(X1,X2,theta) theta(2)^2*exp(-(pdist2(X1,X2).^2)/(2*theta(1)^2));
time=1:100;
time = time';

point1_x = reshape(u(166,66,:), 100,1);
point1_y = reshape(v(166,66,:), 100,1);

theta0 = [1.5, 0.2];
gpr =
fitrgp(time,point1_x,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
gpr2 =
fitrgp(time,point1_y,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
% ypred = resubPredict(gprMdl);
% plot(time,point1_x)
% hold on
% plot(time,ypred,'r')
parameters = zeros(3,10);
dataset = cell(1,10);
parameters2 = zeros(3,10);
dataset2 = cell(1,10);
```

```
for i = 1:10
    parameters(:,i) = [gpr.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset{i} = gpr.Trained{i}.ActiveSetVectors;
    parameters2(:,i) = [gpr2.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset2{i} = gpr2.Trained{i}.ActiveSetVectors;
end
parameters
```

```
parameters = 3×10
    1.8659    1.8482    1.8852    1.8329    1.8702    1.8741    1.8636    1.8602 ···
    0.1176   -0.1174   -0.1197   -0.1175   -0.1192   -0.1189    0.1204   -0.1159
    0.0055    0.0082    0.0060    0.0054    0.0050    0.0062    0.0068    0.0060
```

```
parameters2
```

```
parameters2 = 3×10
    1.4449    1.4179    1.4372    1.4158    1.4372    1.4475    1.4753    1.4556 ···
    0.2203   -0.2115    0.2134   -0.2123    0.2130   -0.2172   -0.2199    0.2161
    0.0055    0.0082    0.0060    0.0054    0.0050    0.0062    0.0068    0.0060
```

$$\mu(X_{\text{test}}|X_{\text{training}}) = \mu(\text{training}) + \Sigma_{12}(\Sigma_{22} + 0.001 * I)^{-1}(X_2 - \mu_2)$$

$$\Sigma(X_{\text{test}}|X_{\text{training}}) = \Sigma_{11} - \Sigma_{12}(\Sigma_{22} + 0.001 * I)^{-1}\Sigma_{21}$$

This paper interpreted the equations as following. There were 10 partitions(observations) and performed cross-validation.

```
data_partition = cell(1,10);
manual_mu = zeros(1,10);
manual_sigma = zeros(1,10);

data_partition2 = cell(1,10);
manual_mu2 = zeros(1,10);
manual_sigma2 = zeros(1,10);
for i = 1:10
    data_train = point1_x(gpr.Trained{i}.ActiveSetVectors);
    data_partition{i} = setdiff(point1_x,data_train);
    data_train2 = point1_y(gpr2.Trained{i}.ActiveSetVectors);
    data_partition2{i} = setdiff(point1_y,data_train);
end

for i =1:10
    rest = setdiff(1:10,i);
    covmat = cov(horzcat(data_partition{:}));
    x1 = point1_x(setdiff(1:100,gpr.Trained{i}.ActiveSetVectors));
    x2 = reshape(point1_x(gpr.Trained{i}.ActiveSetVectors),10,9);
    manual_mu(i) = mean(x1) + covmat(i,rest)*inv(covmat(rest,rest) +
0.001*eye(9))*(mean(x2)-mean(x2,'all'))';
    manual_sigma(i) = covmat(i,i) - covmat(i,rest)*inv(covmat(rest,rest) +
0.001*eye(9))*covmat(rest,i);
```

```
    covmat2 = cov(horzcat(data_partition2{:}));
    y1 = point1_x(setdiff(1:100,gpr2.Trained{i}.ActiveSetVectors));
    y2 = reshape(point1_y(gpr2.Trained{i}.ActiveSetVectors),10,9);
    manual_mu2(i) = mean(y1) + covmat2(i,rest)*inv(covmat2(rest,rest) +
0.001*eye(9))*(mean(y2)-mean(y2,'all'))';
    manual_sigma2(i) = covmat2(i,i) - covmat2(i,rest)*inv(covmat2(rest,rest) +
0.001*eye(9))*covmat2(rest,i);
end
fprintf('The manually computed mu_x for 10-fold is')
```

```
The manually computed mu_x for 10-fold is
```

```
disp(manual_mu')
```

```
    0.0422
   -0.0381
   -0.0087
    0.0726
    0.0229
    0.0180
   -0.0163
   -0.0140
    0.1040
   -0.0959
```

```
fprintf('The manually computed sigma_x for 10-fold is')
```

```
The manually computed sigma_x for 10-fold is
```

```
disp(manual_sigma')
```

```
    0.0004
    0.0011
    0.0005
    0.0009
    0.0005
    0.0003
    0.0002
    0.0012
    0.0008
    0.0015
```

```
fprintf('The manually computed mu_y for 10-fold is')
```

```
The manually computed mu_y for 10-fold is
```

```
disp(manual_mu2')
```

```
    0.0731
   -0.0274
    0.0084
    0.1400
    0.0269
    0.0101
   -0.0728
   -0.0263
```

```
    0.0226
   -0.0091
```

```
fprintf('The manually computed sigma_y for 10-fold is')
```

```
The manually computed sigma_y for 10-fold is
```

```
disp(manual_sigma2')
```

```
   1.0e-03 *

    0.1108
    0.1108
    0.1108
    0.1108
    0.1108
    0.1108
    0.1108
    0.1108
    0.1108
    0.1108
```

Notice that those are means and standard deviation observed from squared exponential function, not necesaarily the parameters of fitting function.

$$y = f(\sigma, l) \text{ and } \mu = \text{mean}(y), \text{and } \sigma = \text{sd}(y)$$

Returned to optimizing and calculating squared exponential function, on the entire data set to simplify into a single set of parameters,

```
test = fitrgp(time,point1_x);
fprintf('Reported Loglikelihood \n using the entire data points: %d ',
test.LogLikelihood)
```

```
Reported Loglikelihood
 using the entire data points: 1.911955e+02
```

```
fprintf(['Found Parameters \n' ...
    ' (l,sigma) = %d %d'],test.KernelInformation.KernelParameters)
```

```
Found Parameters
 (l,sigma) = 1.868220e+00 1.186886e-01
```

```
param1 = test.KernelInformation.KernelParameters;
test2 = fitrgp(time,point1_y);
fprintf('Reported Loglikelihood \n using the entire data points: %d ',
test2.LogLikelihood)
```

```
Reported Loglikelihood
 using the entire data points: 5.282511e+01
```

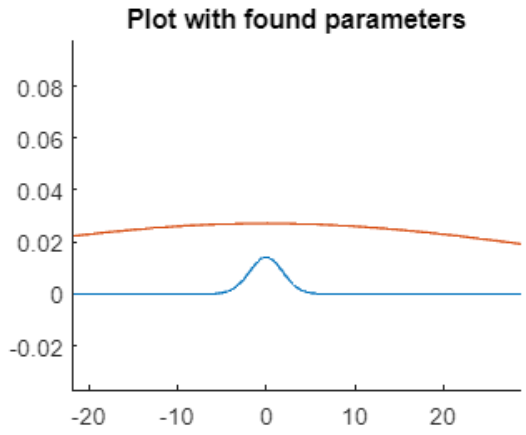
```
fprintf(['Found Parameters \n' ...
    ' (l,sigma) = %d %d'],test2.KernelInformation.KernelParameters)
```

```
Found Parameters
 (l,sigma) = 3.456556e+01 1.646961e-01
```

```
param2 = test2.KernelInformation.KernelParameters
```

```
param2 = 2×1
   34.5656
    0.1647
```

```
hold on
fplot(@(x) param1(2)^2*exp(-(x.^2)/(2*param1(1)^2)))
title('Plot with found parameters')
fplot(@(x) param2(2)^2*exp(-(x.^2)/(2*param2(1)^2)))
hold off
legend('off')
xlim([-21.8 28.8])
ylim([-0.037 0.098])
```




```
param_summary{:,1} = [parameters ; parameters2];
```

4-b

3 or more locations;

point = (100,400)

```
region_x = 100;
region_y = 400;
modelfun = @(X1,X2,theta) theta(2)^2*exp(-(pdist2(X1,X2).^2)/(2*theta(1)^2));
time=1:100;
time = time';
point1_x = reshape(u(region_y,region_x,:), 100,1);
point1_y = reshape(v(region_y,region_x,:), 100,1);

theta0 = [1.5, 0.2];
gpr =
fitrgp(time,point1_x,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
```

```
gpr2 =
fitrgp(time,point1_y,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
parameters = zeros(3,10);
dataset = cell(1,10);
parameters2 = zeros(3,10);
dataset2 = cell(1,10);
for i = 1:10
    parameters(:,i) = [gpr.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset{i} = gpr.Trained{i}.ActiveSetVectors;
    parameters2(:,i) = [gpr2.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset2{i} = gpr2.Trained{i}.ActiveSetVectors;
end

test = fitrgp(time,point1_x);
fprintf('Reported Loglikelihood \n using the entire data points: %d ',
test.LogLikelihood)
```

```
Reported Loglikelihood
 using the entire data points: 1.381469e+02
```

```
fprintf(['Found Parameters \n' ...
    ' (l,sigma) = %d %d'],test.KernelInformation.KernelParameters)
```

```
Found Parameters
 (l,sigma) = 1.242384e+01 9.381508e-02
```

```
param1 = test.KernelInformation.KernelParameters;
test2 = fitrgp(time,point1_y);
fprintf('Reported Loglikelihood \n using the entire data points: %d ',
test2.LogLikelihood)
```

```
Reported Loglikelihood
 using the entire data points: 1.416709e+02
```

```
fprintf(['Found Parameters \n' ...
    ' (l,sigma) = %d %d'],test2.KernelInformation.KernelParameters)
```

```
Found Parameters
 (l,sigma) = 1.896641e+01 1.065520e-01
```

```
param2 = test2.KernelInformation.KernelParameters
```

```
param2 = 2×1
   18.9664
    0.1066
```

```
param_summary{:,2} = [parameters ; parameters2];
```

point = (200,200)

```
region_x = 200;
```

```
region_y = 200;
modelfun = @(X1,X2,theta) theta(2)^2*exp(-(pdist2(X1,X2).^2)/(2*theta(1)^2));
time=1:100;
time = time';
point1_x = reshape(u(region_y,region_x,:), 100,1);
point1_y = reshape(v(region_y,region_x,:), 100,1);

theta0 = [1.5, 0.2];
gpr =
fitrgp(time,point1_x,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
gpr2 =
fitrgp(time,point1_y,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
parameters = zeros(3,10);
dataset = cell(1,10);
parameters2 = zeros(3,10);
dataset2 = cell(1,10);
for i = 1:10
    parameters(:,i) = [gpr.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset{i} = gpr.Trained{i}.ActiveSetVectors;
    parameters2(:,i) = [gpr2.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset2{i} = gpr2.Trained{i}.ActiveSetVectors;
end

test = fitrgp(time,point1_x);
fprintf('Reported Loglikelihood \n using the entire data points: %d ',
test.LogLikelihood)
```

```
Reported Loglikelihood
 using the entire data points: 1.288950e+02
```

```
fprintf(['Found Parameters \n' ...
    ' (l,sigma) = %d %d'],test.KernelInformation.KernelParameters)
```

```
Found Parameters
 (l,sigma) = 2.419464e+00 1.723263e-01
```

```
param1 = test.KernelInformation.KernelParameters;
test2 = fitrgp(time,point1_y);
fprintf('Reported Loglikelihood \n using the entire data points: %d ',
test2.LogLikelihood)
```

```
Reported Loglikelihood
 using the entire data points: 5.818354e+01
```

```
fprintf(['Found Parameters \n' ...
    ' (l,sigma) = %d %d'],test2.KernelInformation.KernelParameters)
```

```
Found Parameters
 (l,sigma) = 1.629184e+03 8.207116e-05
```

```
param2 = test2.KernelInformation.KernelParameters
```

```
param2 = 2×1
10^3 ×
    1.6292
    0.0000
```

```
param_summary{:,3} = [parameters ; parameters2];
```

point = (500,250)

```
region_x = 500;
region_y = 250;
modelfun = @(X1,X2,theta) theta(2)^2*exp(-(pdist2(X1,X2).^2)/(2*theta(1)^2));
time=1:100;
time = time';
point1_x = reshape(u(region_y,region_x,:), 100,1);
point1_y = reshape(v(region_y,region_x,:), 100,1);

theta0 = [1.5, 0.2];
gpr =
fitrgp(time,point1_x,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
gpr2 =
fitrgp(time,point1_y,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
parameters = zeros(3,10);
dataset = cell(1,10);
parameters2 = zeros(3,10);
dataset2 = cell(1,10);
for i = 1:10
    parameters(:,i) = [gpr.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset{i} = gpr.Trained{i}.ActiveSetVectors;
    parameters2(:,i) = [gpr2.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset2{i} = gpr2.Trained{i}.ActiveSetVectors;
end

test = fitrgp(time,point1_x);
fprintf('Reported Loglikelihood \n using the entire data points: %d ',
test.LogLikelihood)
```

```
Reported Loglikelihood
 using the entire data points: 1.711238e+02
```

```
fprintf(['Found Parameters \n' ...
    ' (l,sigma) = %d %d'],test.KernelInformation.KernelParameters)
```

```
Found Parameters
 (l,sigma) = 8.501412e+01 4.624519e-05
```

```
param1 = test.KernelInformation.KernelParameters;
test2 = fitrgp(time,point1_y);
fprintf('Reported Loglikelihood \n using the entire data points: %d ',
test2.LogLikelihood)
```

```
Reported Loglikelihood
 using the entire data points: 1.223793e+02
```

```
fprintf(['Found Parameters \n' ...
    ' (l,sigma) = %d %d'],test2.KernelInformation.KernelParameters)
```

```
Found Parameters
 (l,sigma) = 2.490179e+01 1.969363e-01
```

```
param2 = test2.KernelInformation.KernelParameters
```

```
param2 = 2×1
   24.9018
    0.1969
```

```
param_summary{:,4} = [parameters ; parameters2];
```

In order to observe a pattern, parameters from 10 partitioned regression were stored.

```
test = abs(horzcat(param_summary{:}))
```

```
test = 6×40
    1.8659    1.8482    1.8852    1.8329    1.8702    1.8741    1.8636    1.8602 ···
    0.1176    0.1174    0.1197    0.1175    0.1192    0.1189    0.1204    0.1159
    0.0055    0.0082    0.0060    0.0054    0.0050    0.0062    0.0068    0.0060
    1.4449    1.4179    1.4372    1.4158    1.4372    1.4475    1.4753    1.4556
    0.2203    0.2115    0.2134    0.2123    0.2130    0.2172    0.2199    0.2161
    0.0055    0.0082    0.0060    0.0054    0.0050    0.0062    0.0068    0.0060
```

```
var(test,0,2)
```

```
ans = 6×1
    0.1471
    0.0022
    0.0001
    0.1494
    0.0019
    0.0001
```

I was expecting that all waves would exhibit periodic pattern with similiar time interval. Meaning, the parameter l would be similiar regardless of the location.

However, it turned out that the 2nd and 5th parameters have low variances. Meaning, some sorts of congruency. were found on $\sigma's$ regardless of the direction of the flow(1~3 means x direction whereas 4~6 means y direction). Surprisingly, the priod of oscillations are significantly different for points I selected but the scale of oscillation is rather similiar.

Notice that 3rd and 6th are the variance of RMSE error.

4-c

2 other choices of τ

Going Back to the point = (66,166)

There were 10-fold regressions and each of them estimates signal noises based on input data.

```
region_x = 66;
region_y = 166;
modelfun = @(X1,X2,theta) theta(2)^2*exp(-(pdist2(X1,X2).^2)/(2*theta(1)^2));
time=1:100;
time = time';

point1_x = reshape(u(166,66,:), 100,1);
point1_y = reshape(v(166,66,:), 100,1);

theta0 = [1.5, 0.2];
gpr =
fitrgp(time,point1_x,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
gpr2 =
fitrgp(time,point1_y,'KernelFunction',modelfun,'KernelParameters',theta0,'KFold',10)
;
parameters = zeros(3,10);
dataset = cell(1,10);
parameters2 = zeros(3,10);
dataset2 = cell(1,10);
for i = 1:10
    parameters(:,i) = [gpr.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset{i} = gpr.Trained{i}.ActiveSetVectors;
    parameters2(:,i) = [gpr2.Trained{i}.KernelInformation.KernelParameters;
rmse(predict(gpr.Trained{i},time),point1_x) ];
    dataset2{i} = gpr2.Trained{i}.ActiveSetVectors;
end
```

Among them, 1st and 10th partitioned set's sigma was printed.

```
Tau = gpr.Trained{1}.Sigma
```

```
Tau = 0.0076
```

```
Tau = gpr.Trained{10}.Sigma
```

```
Tau = 0.0078
```

those training sets estimate parameters as follows

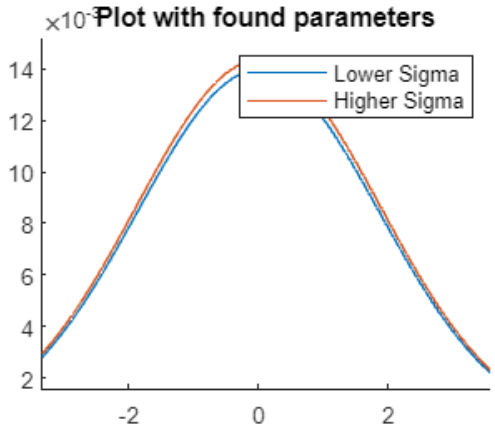
```
param1 = gpr.Trained{1}.KernelInformation.KernelParameters
```

```
param1 = 2×1
    1.8618
    0.1180
```

```
param2 = gpr.Trained{10}.KernelInformation.KernelParameters
```

```
param2 = 2×1
    1.8738
   -0.1196
```

```
figure()
hold on
fplot(@(x) param1(2)^2*exp(-(x.^2)/(2*param1(1)^2)))
title('Plot with found parameters')
fplot(@(x) param2(2)^2*exp(-(x.^2)/(2*param2(1)^2)))
hold off
legend('Lower Sigma','Higher Sigma')

xlim([-3.33 3.57])
ylim([0.0016 0.0152])
```



Just like the case of the t-distribution, higher sigma resulted in bigger tale and lower certainty around mean. Having different tau results in different bayesian inference updates.Tau, noise level, will influence how much we can trust our data. With higher uncertainty due to tau and sigma will put weight on realization of data(testing) compared to prior knowledge(training data).

In conclusion, compared to Problem 4.a, these results are different.

4-d

I used function called fitrgp(fitting Gaussian process regression model) with default option(squared exponential). Used default fitting function, Squared exponential function, was used.

In order to compare the performance, one needs to derive parameters from mu and sigma.

$$f = (e^{\sigma})^2 * e^{\frac{-(x^2)}{2(e^t)^2}}$$

When $\mu = \text{mean}(f)$ and $\sigma = \text{sd}(f)$ one needs to find the following parameters.

```
region_x = 66;
region_y = 166;
modelfun = @(X1,X2,theta) exp(theta(2))^2*exp(-(pdist2(X1,X2).^2)/
(2*exp(theta(1))^2));
time=1:100;
time = time';

point1_x = reshape(u(166,66,:), 100,1);
point1_y = reshape(v(166,66,:), 100,1);

theta0 = [1.5, 0.2];
test = fitrgp(time,point1_x,'KernelFunction',modelfun,'KernelParameters',theta0);
fprintf('(sigma,l) = (%.5f,%.5f)',test.KernelInformation.KernelParameters)
```

```
(sigma,l) = (1.07151,-2.06740)
```

Just comparing values of mu and sigma, there is obvious difference between what I manually calculated and the values the library returned. One main difference I can notice is that the computed noise standard deviation is significantly different.

$\tau = 0.001$

But the library returned,

```
tau_returned_by_library = test.Sigma
```

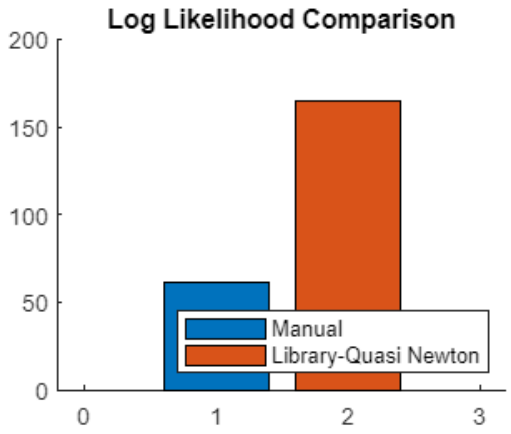
```
tau_returned_by_library = 0.0236
```

To provide additional details on the library and provide explanations why they are different from 4.a, a few key insights from algorithm would be mentioned. For manual calculation, we had linear projection matrix to find posterior statistics. However, the optimized algorithm utilizes a series of methods called the quasi-Newton optimizer with symmetric rank-1based(SR1) with iterative process with a stepsize.

Despite not understanding the details of algorithm, it seems clear that it is advance methods compared to matrix calculation we manually performed.

```
modelfun = @(X1,X2,theta) theta*manual_sigma(5)^2*exp(-(pdist2(X1,X2).^2)/
(2*manual_mu(1)^2));
theta0 = [1];
test2 = fitrgp(time,point1_x,'KernelFunction',modelfun,'KernelParameters',theta0);
figure()
hold on
bar(1,test2.LogLikelihood)
bar(2,test.LogLikelihood)
```

```
hold off
title('Log Likelihood Comparison')
legend('Manual','Library-Quasi Newton',Location='southeast')
```



p5 Estimating Unobserved Flow Data(time wise)

1. point = (66, 166)
2. Will predict flow for each day. Time intervals will be divided into 3.
3. The predicted means
4. The predicted standard deviation as a 3σ band.
5. The observed data ppoints

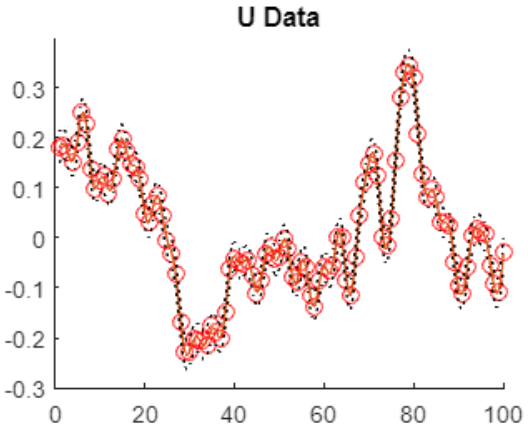
The time-stamps with intervals of one day is chosen. Before increasing predictions with an infinitesimal level, it seems to natural to check the logic and soundness through observing integer points Since the velocities of currents are evolving over time continuously, it seems natural to choose prior means as previous observed time plus epsilon.

$$f(t+\delta t) = f(t) + \delta t * f'(t) + \Theta(2)$$

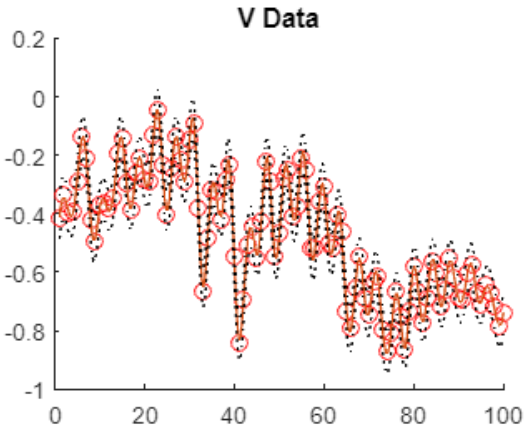
Where theta represents taylor's expansion with terms with degree higher than 2 and ignored for the sake of simplicity.

```
% region_x; region_y; u ; v;
time = linspace(1,100,300)';
modelfun = @(x1,x2,theta) exp(theta(2))^2*exp(-(pdist2(x1,x2).^2)/
(2*exp(theta(1))^2));
point1_x = reshape(u(region_y, region_x, :),1, 100)';
point1_y = reshape(v(region_y, region_x, :),1, 100)';
temp = reshape(1:100,100,1);
gpr = fitrgp(temp, point1_x, KernelFunction=modelfun,KernelParameters=[2 2]);
[ypred, ysd, yint] = predict(gpr,time);
figure()
scatter(temp,point1_x,'r')
hold on
plot(time,ypred)
plot(time,ypred+3*ysd, 'k:')
```

```
plot(time,ypred-3*ysd,'k:')
hold off
title('U Data')
```

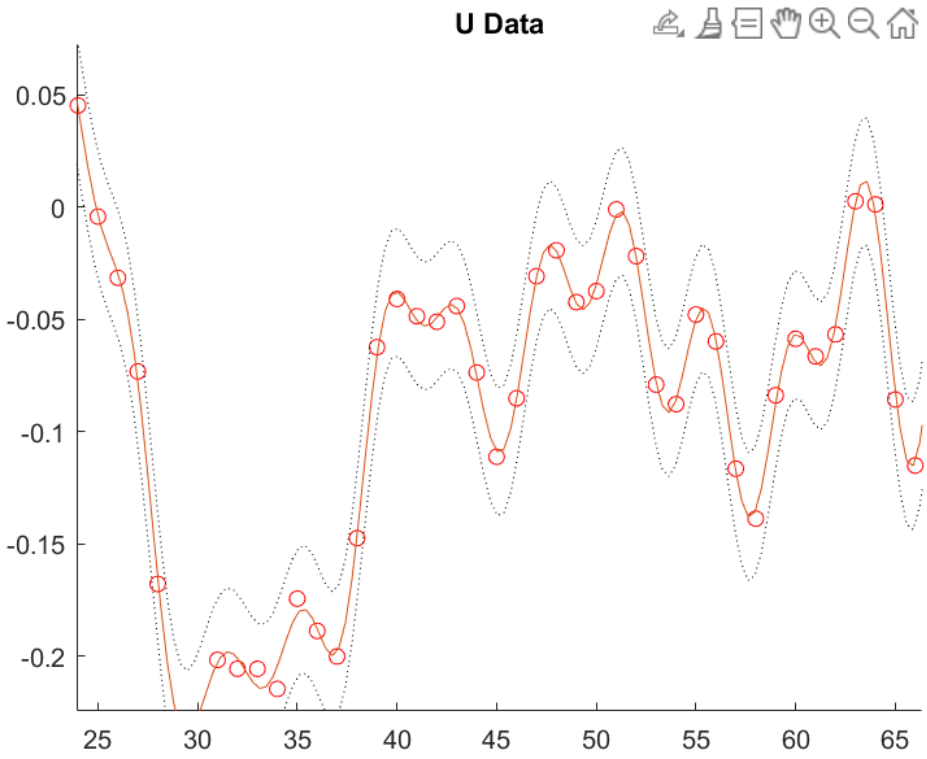


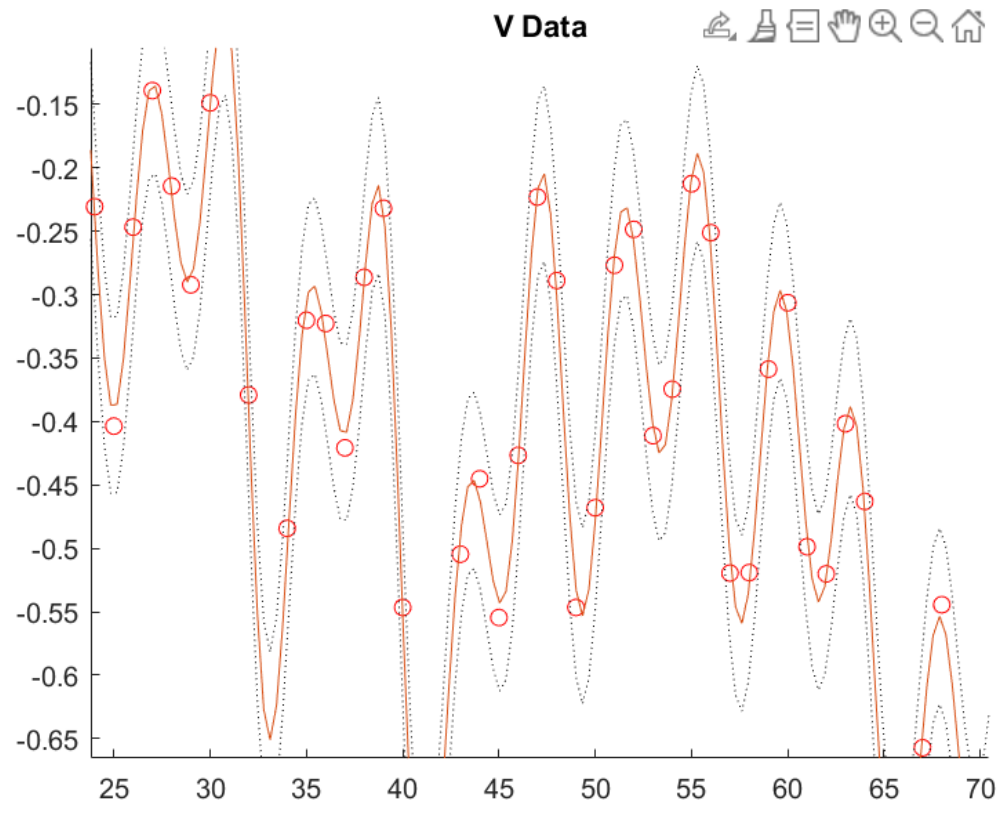

```
% legend('True Response','yprediction','yprediction+3\sigma','yprediction-3\sigma')

gpr2 = fitrgp(temp, point1_y, KernelFunction=modelfun,KernelParameters=[0.0001 -1]);
[ypred2, ysd2, yint2] = predict(gpr2,time);
figure()
scatter(temp,point1_y,'r')
hold on
plot(time,ypred2)
plot(time,ypred2+3*ysd2, 'k:')
plot(time,ypred2-3*ysd2,'k:')
hold off
title('V Data')
```




```
% legend('True Response','yprediction','yprediction+3\sigma','yprediction-3\sigma')
```

red dots are actual data, red lines are predictions, and greys are 3σ confidence intervals. A portion of the graph is magnified as it is not interactable in PDF format.


U Data
0.05
0
-0.05
-0.1
-0.15
-0.2
25
30
35
40
45
50
55
60
65


V Data
-0.15
-0.2
-0.25
-0.3
-0.35
-0.4
-0.45
-0.5
-0.55
-0.6
-0.65
25
30
35
40
45
50
55
60
65
70

P6: Longer time-scale simulation

As stated from the problem, this paper utilizes the fitting results performed at location (66, 166).

For those who are not used to anonymous function, the model function is rementioned in a standard mathematical formula.

$$\text{modelfun} = (e^{\sigma})^2 * e^{\frac{-(x^2)}{2(e^t)^2}}$$

```
disp('The model of function is ')
```

```
The model of function is
```

```
disp(gpr.KernelFunction)
```

```
    @(x1,x2,theta)exp(theta(2))^2*exp(-(pdist2(x1,x2).^2)/(2*exp(theta(1))^2))
```

```
disp('Parameters found:')
```

```
Parameters found:
```

```
disp('For U')
```

```
For U
```

```
disp(gpr.KernelInformation.KernelParameters)
```

```
    0.6250
   -2.1313
```

```
disp('For V')
```

```
For V
```

```
disp(gpr2.KernelInformation.KernelParameters)
```

```
    0.3861
   -1.5118
```

Due to multiple simulation, epsilon was reduced to 10^3. The search space is 40 by 40 grid.To make it coarse, every third coordinate was plotted.

The first search is at 200,200.

Initial State

```
region_x =200
```

```
region_x = 200
```

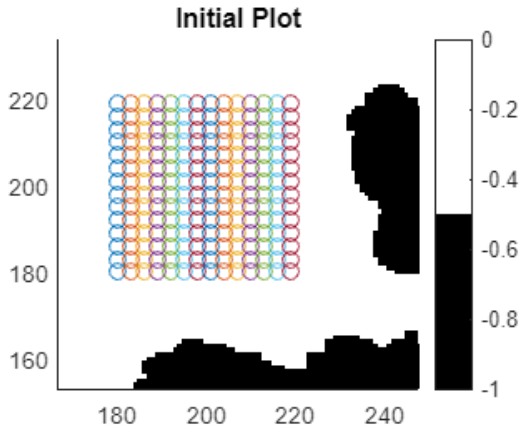
```
region_y = 200
```

```
region_y = 200
```

```
epsilon = 10^3;
time = linspace(1,100, epsilon);
time_interval = 100/epsilon;
unit_conversion = 24;

x = region_x-20:3:region_x+20;
y = region_y-20:3:region_y+20;

xp_final = -1*ones(2,size(time,2),size(x,2),size(y,2));
for i = 1:size(x,2)
    for j = 1:size(y,2)
        xp_final(:,1,i,j) = [x(i) ;y(j)];
    end
end

for i = 1:size(x,2)
    for j = 1:size(y,2)
        for k = 2:epsilon
            temp = fix(xp_final(:,k-1,i,j));
            if  any(temp ==0) || temp(1) >  size(u,2) || temp(2) > size(u,1) ||
(u(temp(2),temp(1),fix(time(k))) && v(temp(1),temp(2),fix(time(k))) == 0)
                break
            else
                temp2 = [predict(gpr,time(k)); predict(gpr2,time(k))]
+ [u(temp(2),temp(1),fix(time(k))) - u(region_y,region_x,fix(time(k)));
v(temp(2),temp(1),fix(time(k))) - v(region_y,region_x,fix(time(k)))];
                xp_final(:,k,i,j) = xp_final(:,k-1,i,j) +
temp2*time_interval*unit_conversion;
            end
        end
    end
end
```

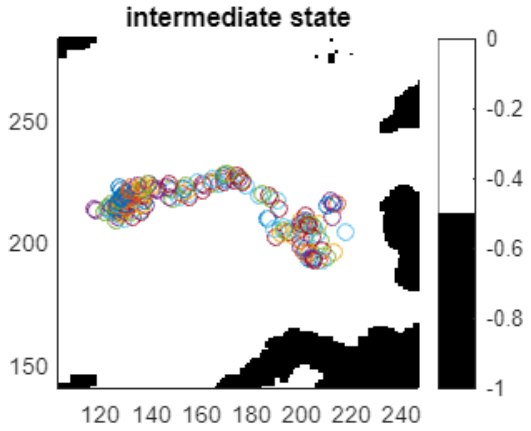
```
rng('default')
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        % ind = find(xp_final(1,:,i,j) ==-1,1);
        scatter(ax,xp_final(2,1,i,j),xp_final(1,1,i,j));
    end
```

```
end
xlim([166.7 247.8])
ylim([153 234])
title('Initial Plot')
```



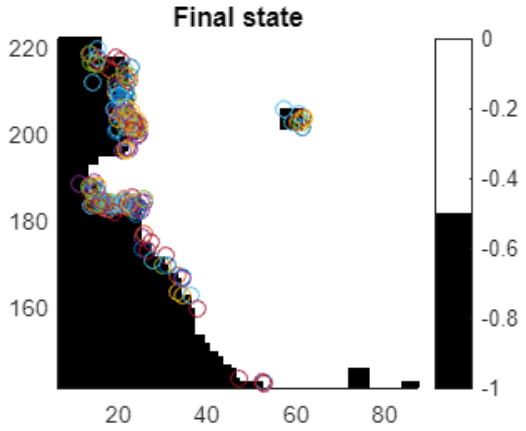
Intermediate State(a half point)

```
rng('default')
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        ind = find(xp_final(1,:,i,j) ==-1,1);
        scatter(ax,xp_final(2,floor(ind/2),i,j),xp_final(1,floor(ind/2),i,j))


    end
end
xlim([103 247])
ylim([140 284])
title('intermediate state')
```



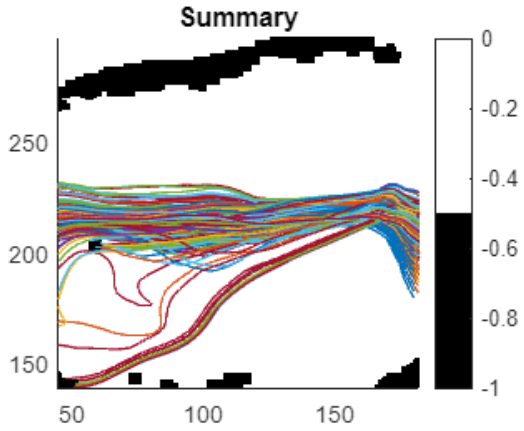
Final State(a half point)

```
rng('default')
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        ind = find(xp_final(1,:,i,j) ==-1,1);
        if ind
            scatter(ax,xp_final(2,ind-1,i,j),xp_final(1,ind-1,i,j))
        else
            scatter(ax,xp_final(2,end,i,j),xp_final(1,end,i,j))
        end
    end
end
title('Final state')
xlim([6.6 87.6])
ylim([141 222])
```



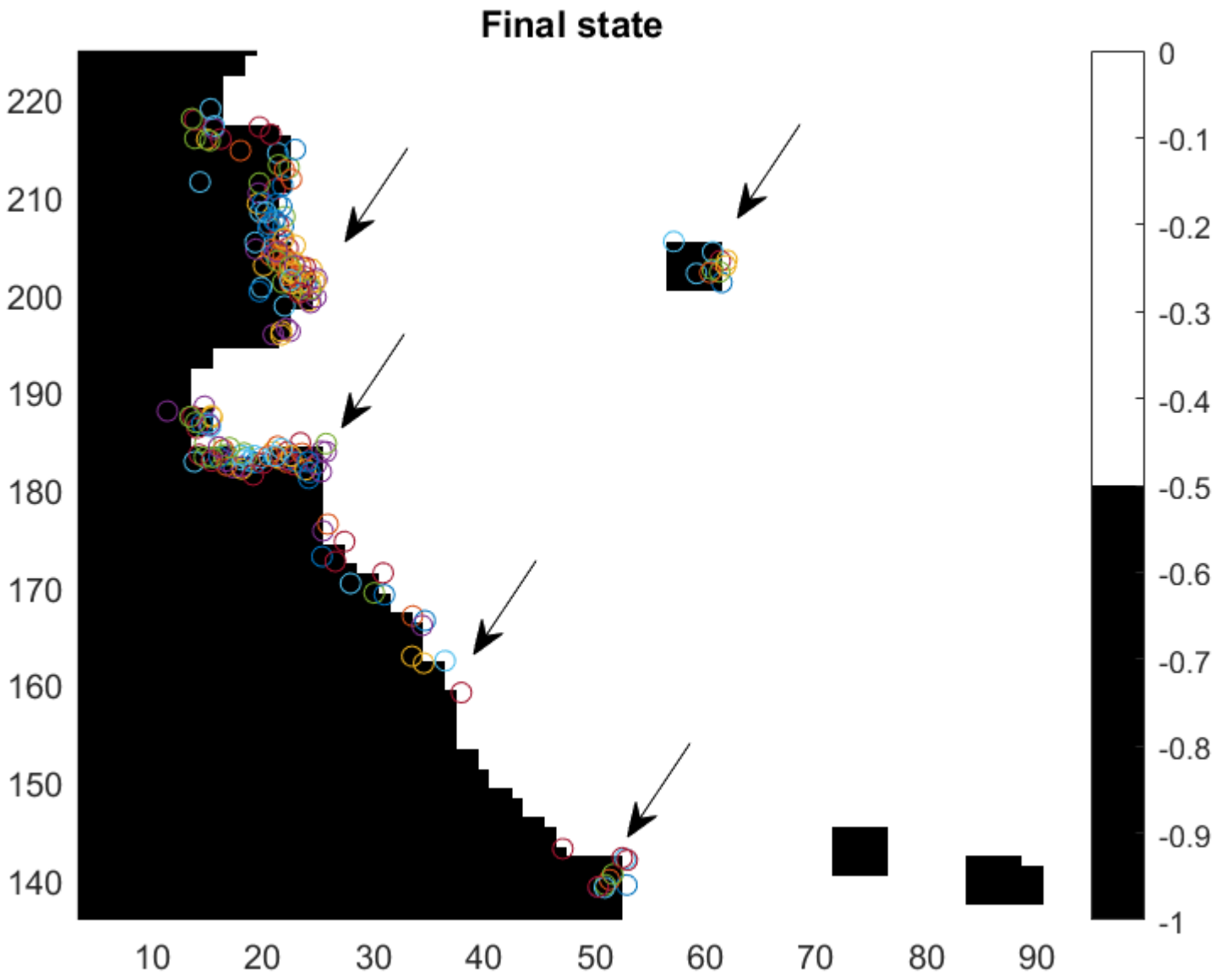
Summary

```
rng('default')
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        ind = find(xp_final(1,:,i,j) ==-1,1);
        if ind
            plot(ax,xp_final(2,1:ind-1,i,j),xp_final(1,1:ind-1,i,j))
        else
            plot(ax,xp_final(2,1:end,i,j),xp_final(1,1:end,i,j))
        end
    end
end
title('Summary')

xlim([45 182])
ylim([139 297])
```



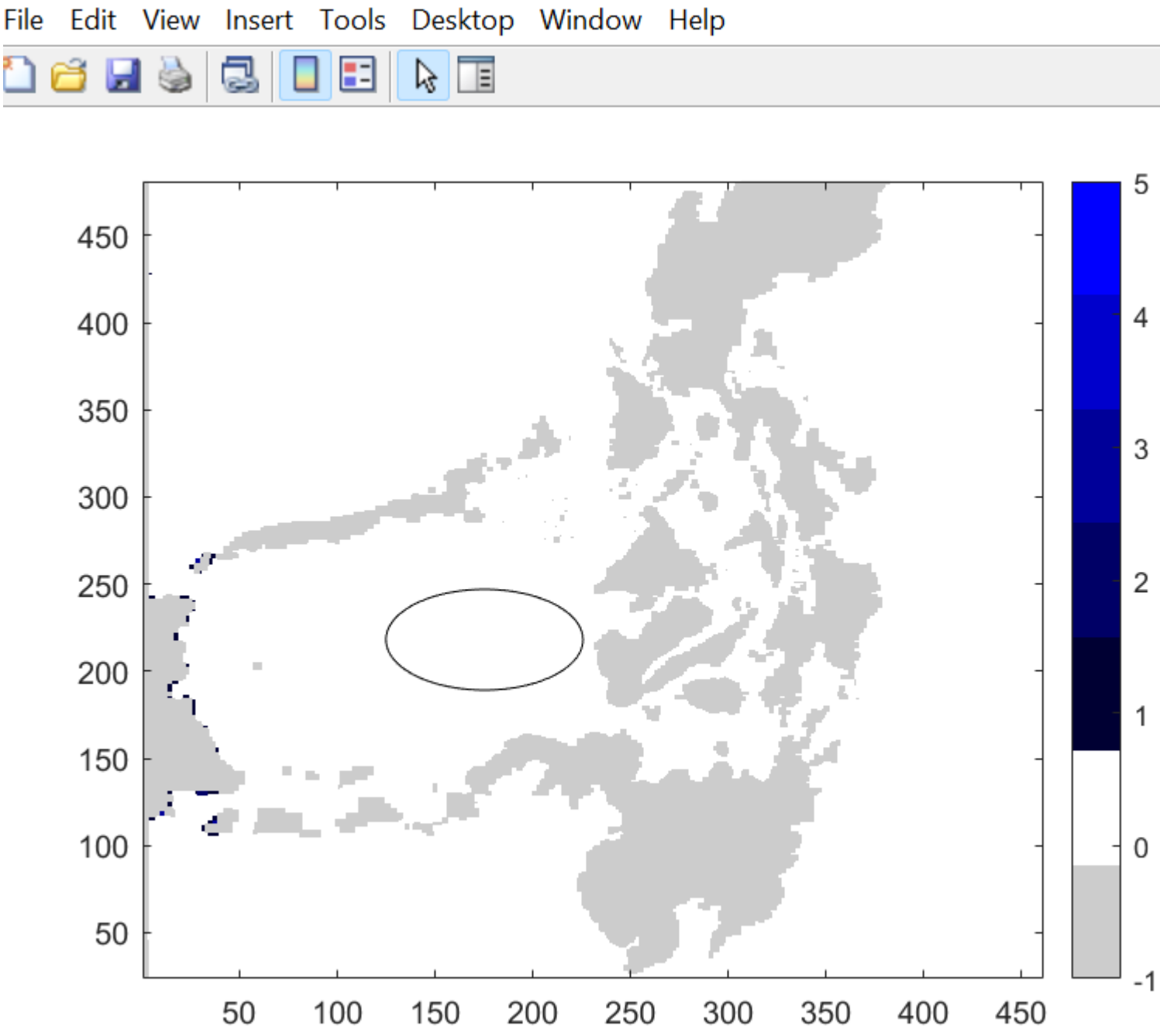
Location on the coast:

Since Color scheme is a little messy, the locations of interests are pointed with arrows.



Location on the ocean

In the vincinity of (200,200), the simulation ended up on coasts.Simple reason is that the location 200,200 is surrounded by land mass by every sides.



Provides plots for other choice of sigma

Already different potential points near the point of interests were plotted.Therefore, choosing another point using randnormal distribution would yield no different result.

Since no points ended up in the ocean, another point that is highly likely end up in the ocean is simulated.

```
region_x = 450
```

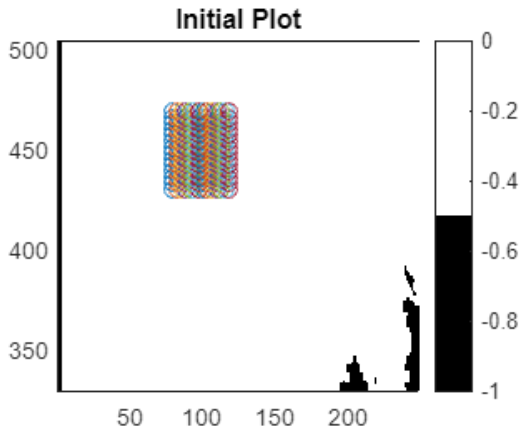
```
region_x = 450
```

```
region_y = 100
```

```
region_y = 100
```

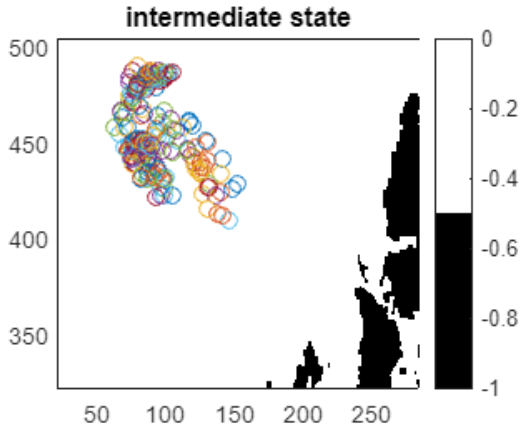
```
epsilon = 10^3;
time = linspace(1,100, epsilon);
time_interval = 100/epsilon;
unit_conversion = 24;

x = region_x-20:3:region_x+20;
y = region_y-20:3:region_y+20;

xp_final = -1*ones(2,size(time,2),size(x,2),size(y,2));
for i = 1:size(x,2)
    for j = 1:size(y,2)
        xp_final(:,1,i,j) = [x(i) ;y(j)];
    end
end

for i = 1:size(x,2)
    for j = 1:size(y,2)
        for k = 2:epsilon
            temp = fix(xp_final(:,k-1,i,j));
            if  any(temp ==0) || temp(2) >  size(u,2) || temp(1) > size(u,1) ||
(u(temp(2),temp(1),fix(time(k))) && v(temp(1),temp(2),fix(time(k))) == 0)
                break
            else
                temp2 = [predict(gpr,time(k)); predict(gpr2,time(k))]
+ [u(temp(2),temp(1),fix(time(k))) - u(region_y,region_x,fix(time(k)));
v(temp(2),temp(1),fix(time(k))) - v(region_y,region_x,fix(time(k)))];
                xp_final(:,k,i,j) = xp_final(:,k-1,i,j) +
temp2*time_interval*unit_conversion;
            end
        end
    end
end
rng('default')
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar
```

```
for i = 1:size(x,2)
    for j = 1:size(y,2)
        % ind = find(xp_final(1,:,i,j) ==-1,1);
        scatter(ax,xp_final(2,1,i,j),xp_final(1,1,i,j));
    end
end

title('Initial Plot')

xlim([1 249])
ylim([329 505])
```



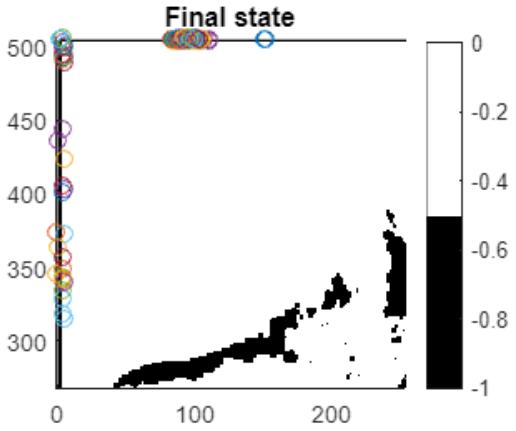
Intermediate State(a half point)

```
rng('default')
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        ind = find(xp_final(1,:,i,j) ==-1,1);
        scatter(ax,xp_final(2,floor(ind/2),i,j),xp_final(1,floor(ind/2),i,j))


    end
end

title('intermediate state')

xlim([22 284])
ylim([322 505])
```

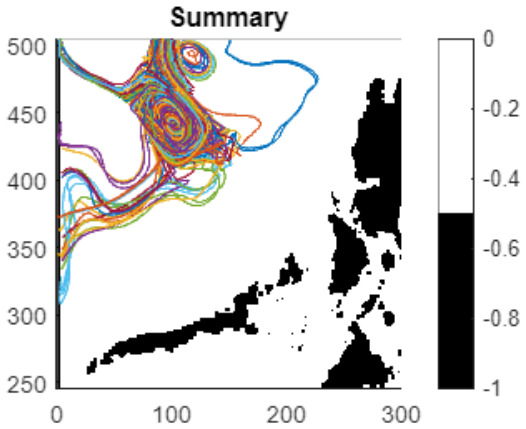


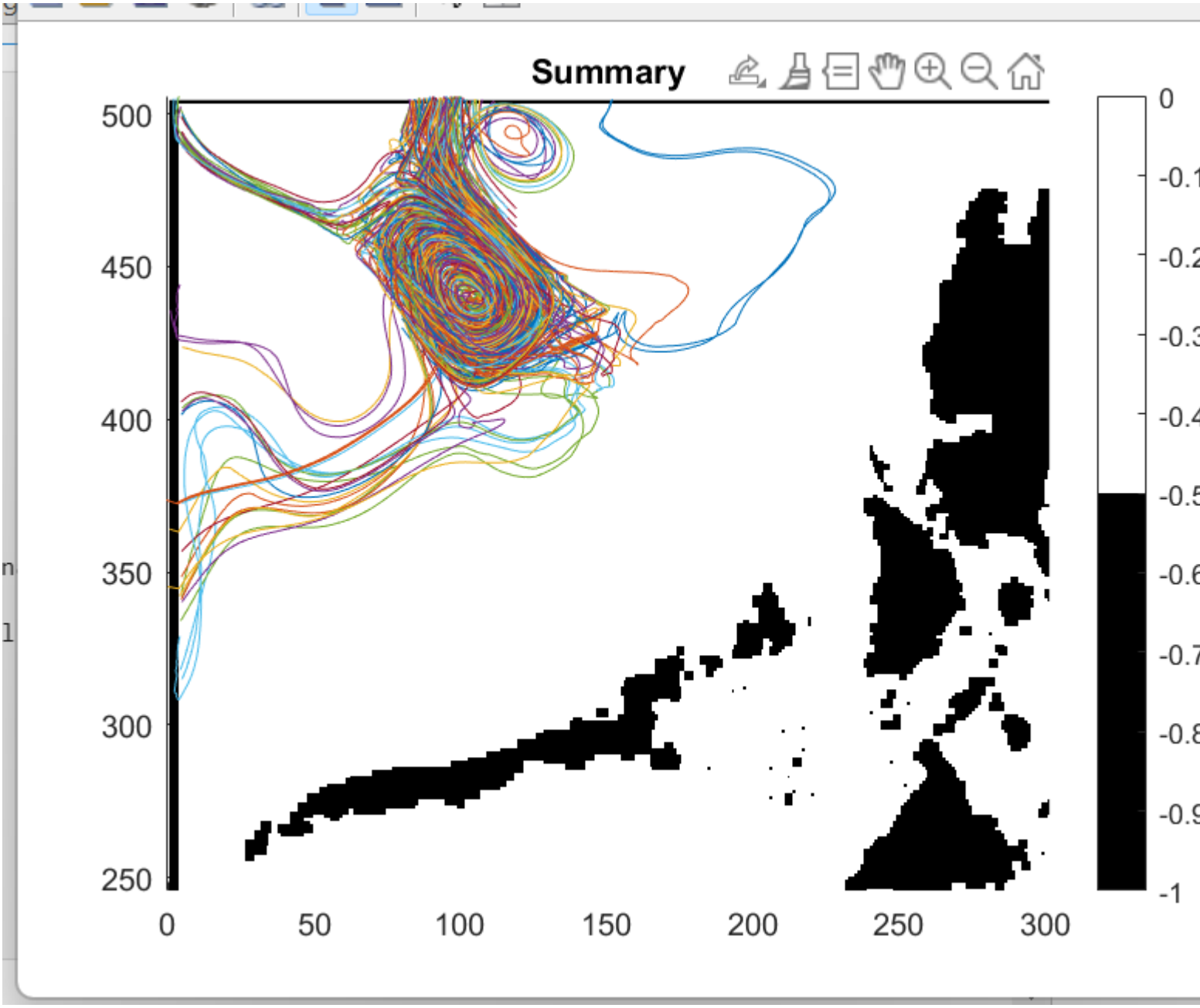
Final State

```
rng('default')
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        ind = find(xp_final(1,:,i,j) ==-1,1);
        if ind
            scatter(ax,xp_final(2,ind-1,i,j),xp_final(1,ind-1,i,j))
        else
            scatter(ax,xp_final(2,end,i,j),xp_final(1,end,i,j))
        end
    end
end
title('Final state')

xlim([-1 254])
ylim([268 506])
```




```
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        ind = find(xp_final(1,:,i,j) ==-1,1);
        if ind
            plot(ax,xp_final(2,1:ind-1,i,j),xp_final(1,1:ind-1,i,j))
        else
            plot(ax,xp_final(2,1:end,i,j),xp_final(1,1:end,i,j))
        end
    end
end
title('Summary')

xlim([-1 301])
ylim([246 506])
```


Summary
500
450
400
350
300
250
0
100
200
300
0
-0.2
-0.4
-0.6
-0.8
-1



It looks like boosting $3\text{hrs} \rightarrow 3\text{days}$ resulted in boosting speed vector $\frac{\text{km}}{h}$ by 24 times. Therefore, most of simulation seems to be reaching the edges.

Additionally, I boosted the speed vector by the squared exponential waveform at location (66,166). It means waves are not likely to be trapped as external waveform estimation forces them to get out of the curl, which can be inferred from the shape of the graph above.

6-b randomly chosen locations

Despite the prompt suggested using iterative process with generating random numbers, it has the following problems;

1. I originally wrote codes for grid search, with matrix multiplication. Using for loop with a few dozen points will take a significant time.
2. When there is a grid search, why would one use random probability that depends on random seed.
3. It is also hard to know how many points are good enough to stop, further complicating trade off between complexity and computation.

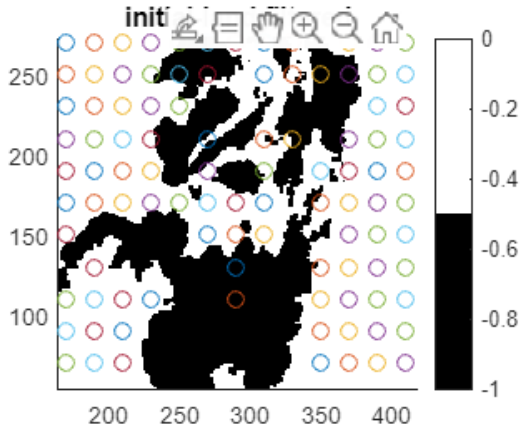
If one is adhering to the principle of randomness, one can easily choose a few points out of the grid using random choice, which is a trivial thing to do.

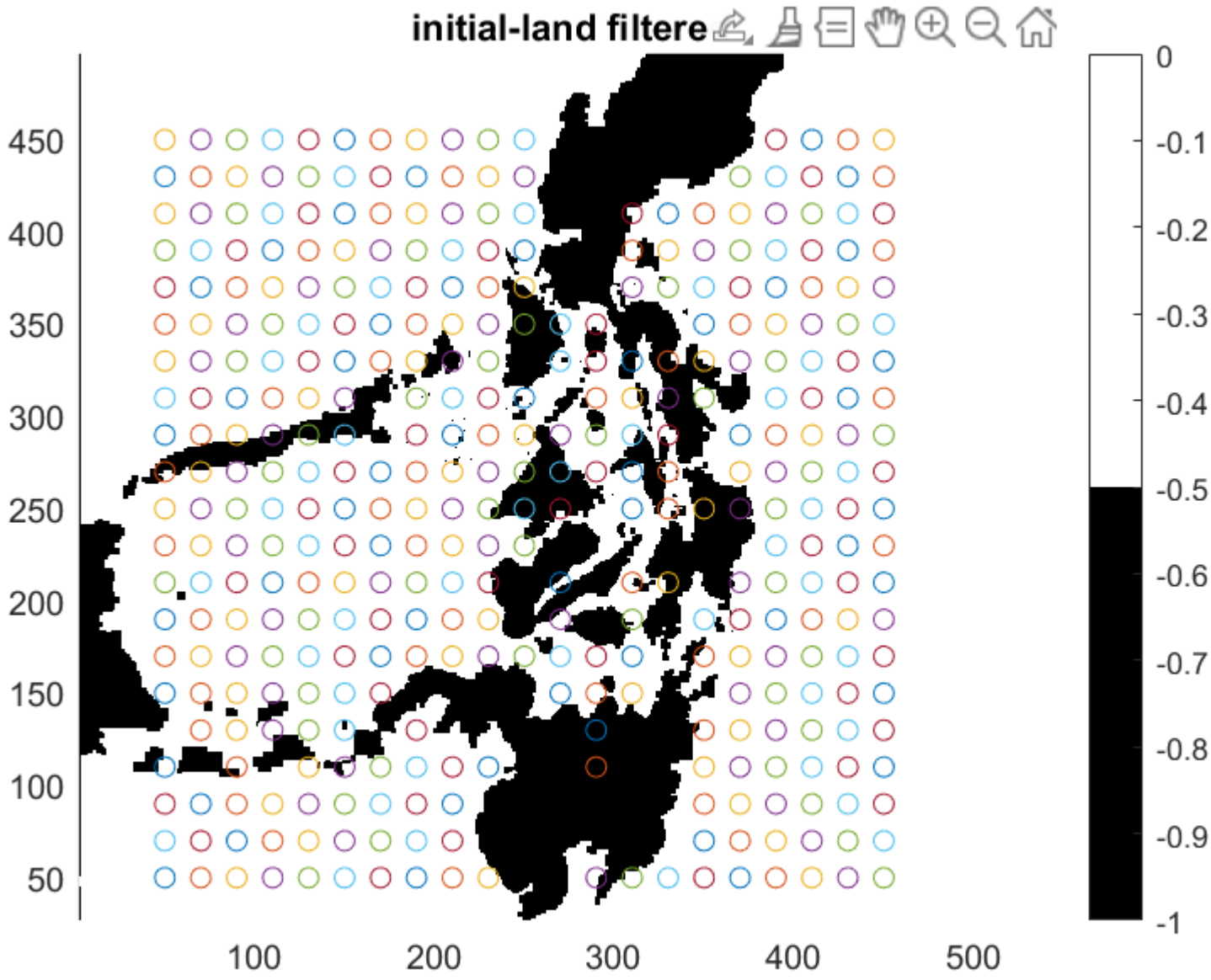
```
region_x = 250;
region_y = 250;
epsilon = 10^3;
time = linspace(1,100, epsilon);
time_interval = 100/epsilon;
unit_conversion = 24;

x = region_x-200:20:region_x+200;
y = region_y-200:20:region_y+200;

xp_final = -1*ones(2,size(time,2),size(x,2),size(y,2));
for i = 1:size(x,2)
    for j = 1:size(y,2)
        xp_final(:,1,i,j) = [x(i) ;y(j)];
    end
end

for i = 1:size(x,2)
    for j = 1:size(y,2)
        for k = 2:epsilon
            temp = fix(xp_final(:,k-1,i,j));
            if  any(temp ==0) || temp(2) >  size(u,2) || temp(1) > size(u,1) ||
(u(temp(2),temp(1),fix(time(k))) && v(temp(1),temp(2),fix(time(k))) == 0)
                break
            else
                temp2 = [predict(gpr,time(k)); predict(gpr2,time(k))]
+ [u(temp(2),temp(1),fix(time(k))) - u(region_y,region_x,fix(time(k)));
v(temp(2),temp(1),fix(time(k))) - v(region_y,region_x,fix(time(k)))];
                xp_final(:,k,i,j) = xp_final(:,k-1,i,j) +
temp2*time_interval*unit_conversion;
            end
        end
    end
end
```

Initial

```
rng('default')
f = clf("reset");
```

```
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        ind = min(find(xp_final(1,:,i,j) ==-1,1), find(xp_final(2,:,i,j) ==-1,1));
        if ind<=3
            continue
        else
            scatter(ax,xp_final(2,1,i,j),xp_final(1,1,i,j))
        end
    end
end
title('initial-land filtered')

xlim([164 419])
ylim([54 273])
```

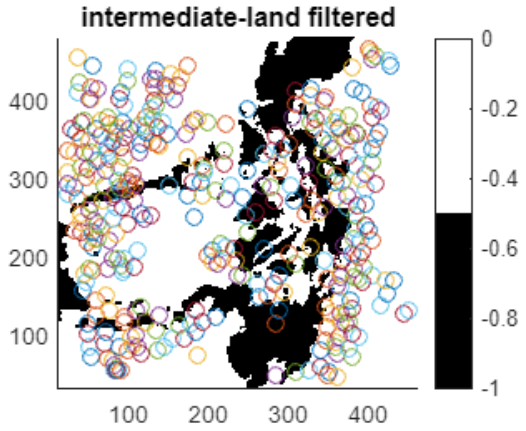




Intermediate

```
rng('default')
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
```

```
        ind = find(xp_final(1,:,i,j) ==-1,1);
        if ind<=3
            continue
        else
            scatter(ax,xp_final(2,floor(ind/2),i,j),xp_final(1,floor(ind/2),i,j))
        end
    end
end
title('intermediate-land filtered')

xlim([12 463])
ylim([31 481])
```

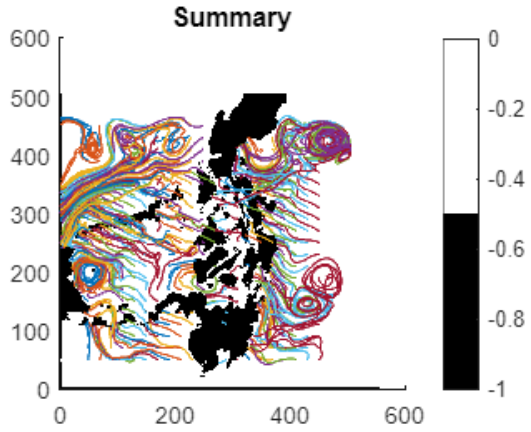



```

```

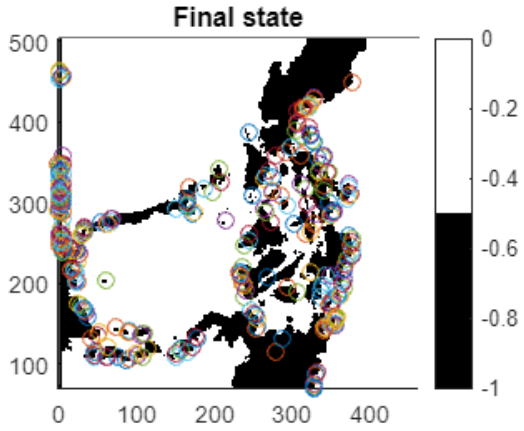
Final and summary

```
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        ind = find(xp_final(1,:,i,j) ==-1,1);
        if ind
            plot(ax,xp_final(2,1:ind-1,i,j),xp_final(1,1:ind-1,i,j))
        else
```

```
            plot(ax,xp_final(2,1:end,i,j),xp_final(1,1:end,i,j))
        end
    end
end
title('Summary')
```

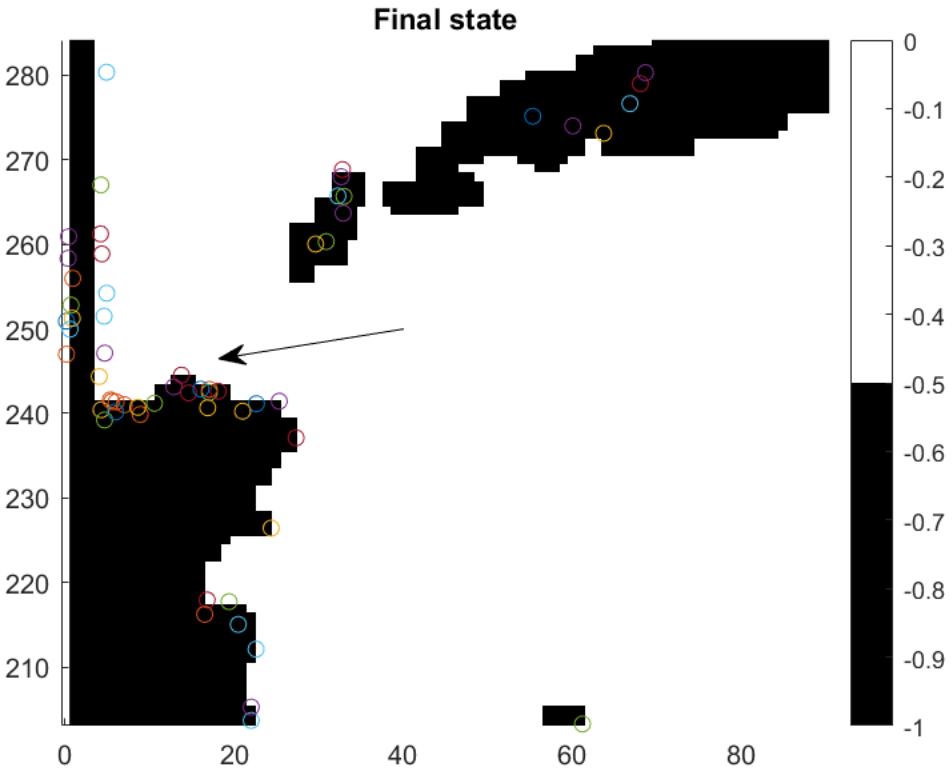



```
rng('default')
f = clf("reset");
figure()
hold on
h = imagesc(map_phillipe);
ax = gca;
set(gca,'YDir','normal')
map = [0 0 0; 1 1 1];
colormap(map)
colorbar

for i = 1:size(x,2)
    for j = 1:size(y,2)
        ind = find(xp_final(1,:,i,j) ==-1,1);
        if ind<=3
            continue
        else
            if ind
                scatter(ax,xp_final(2,ind-1,i,j),xp_final(1,ind-1,i,j))
            else
                scatter(ax,xp_final(2,end,i,j),xp_final(1,end,i,j))
            end
        end
    end
end
title('Final state')

xlim([-0 410])
ylim([78 487])
```

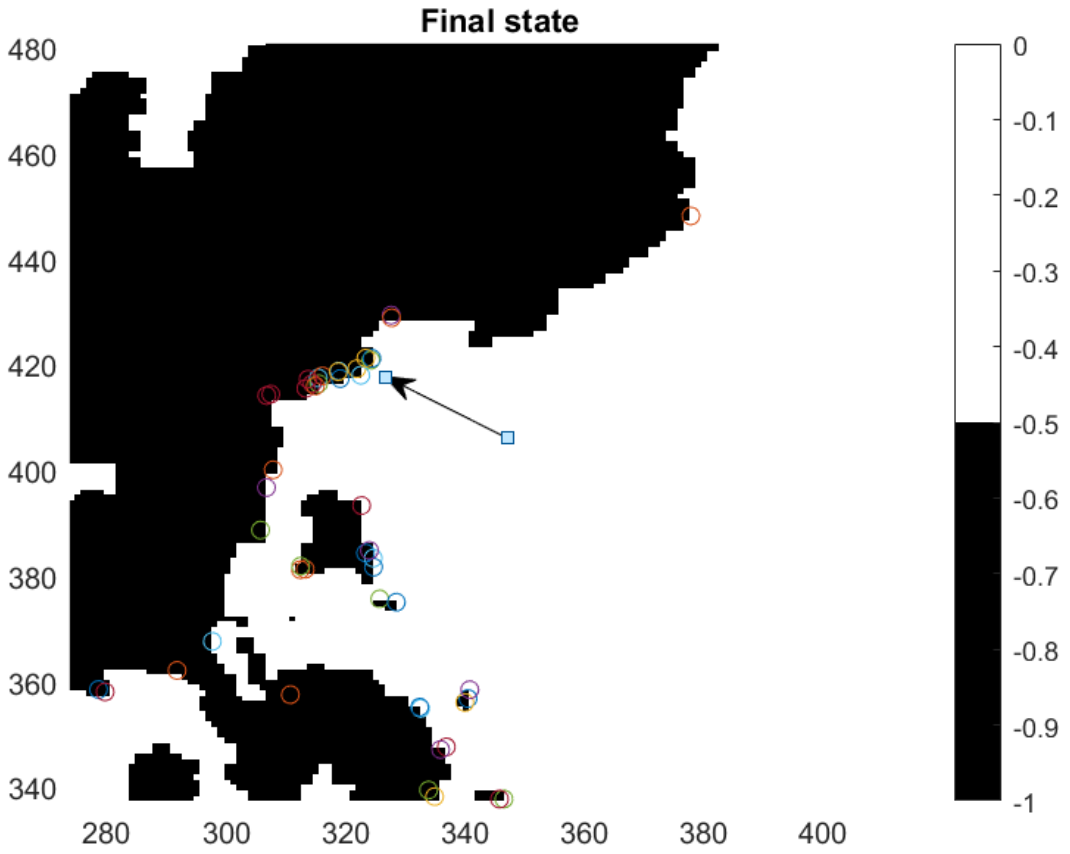


3 Candidates

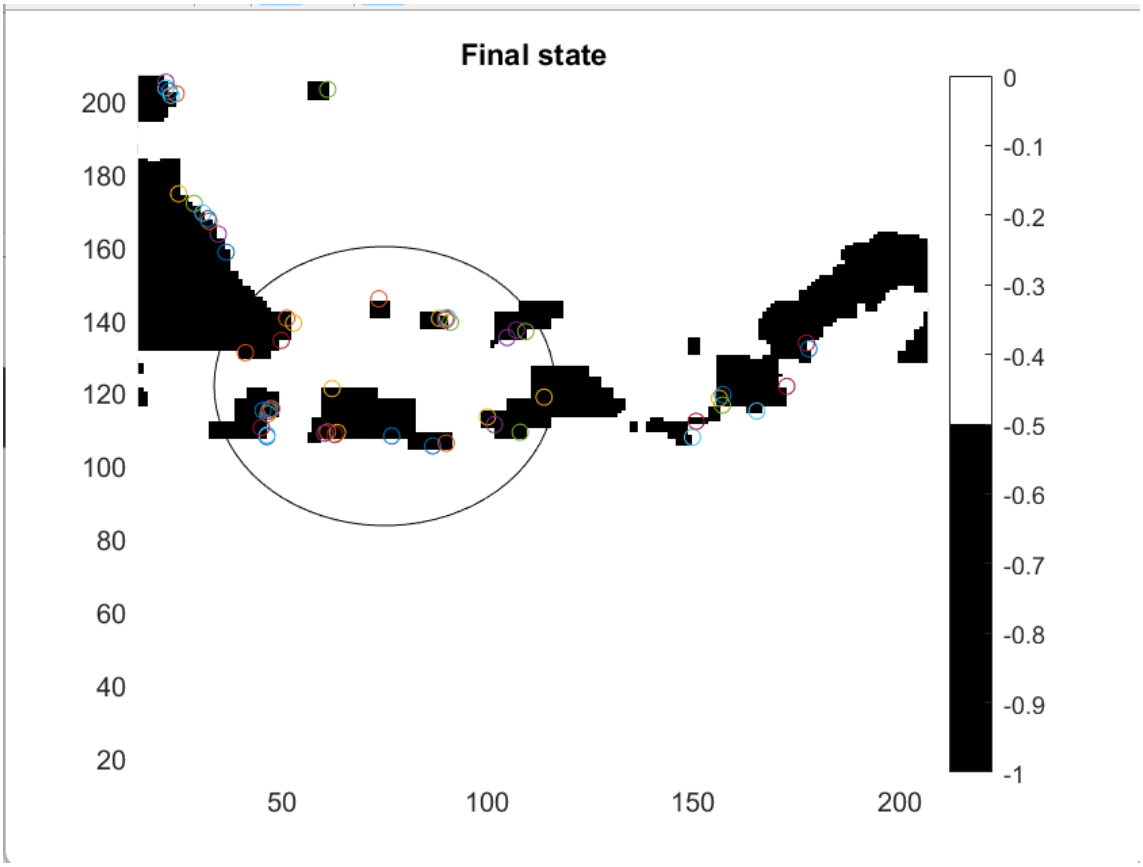
My first candidate:



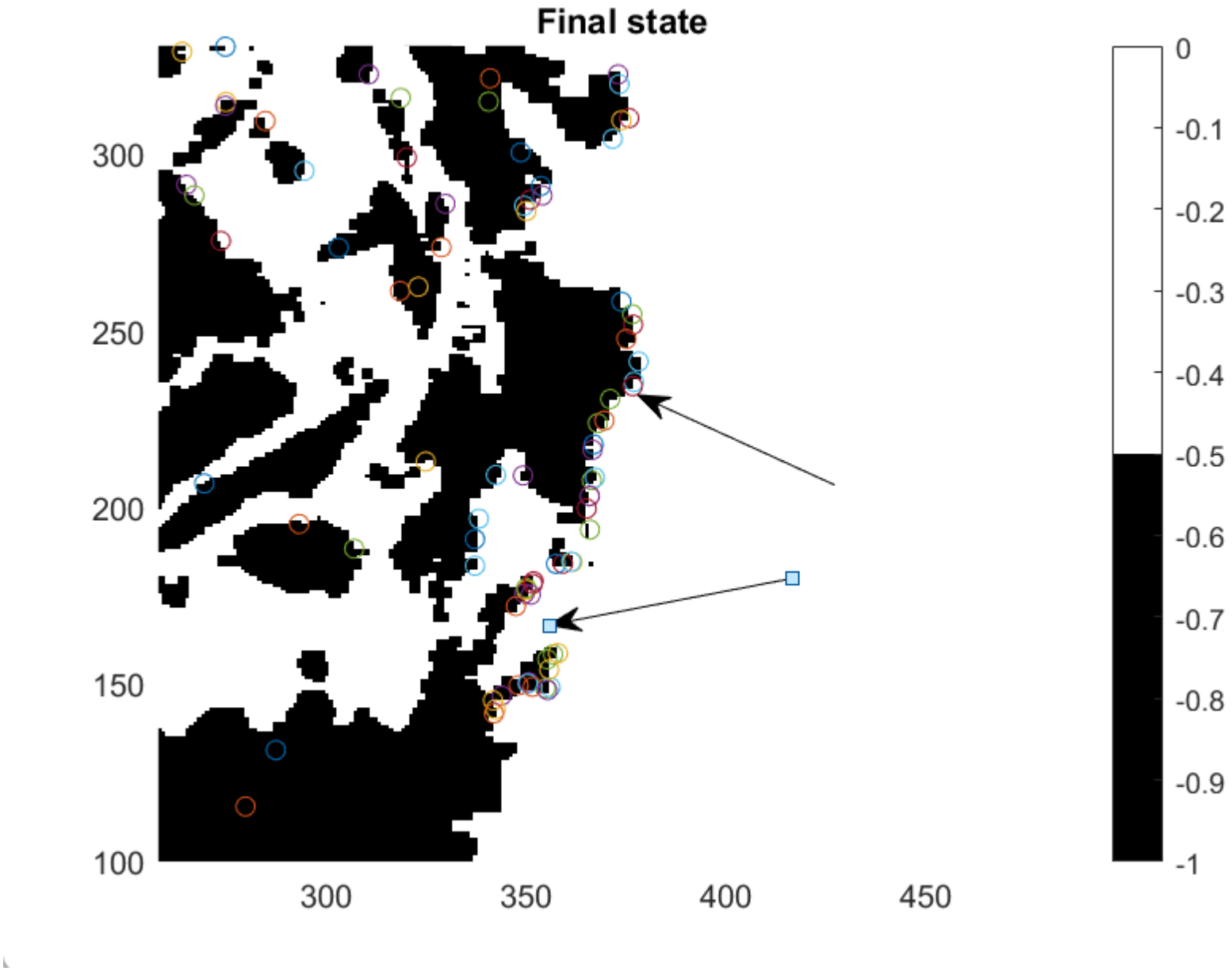
My second candidate:



My last candidate:



I found those candidates by observing points density per circular area. For example, there were several occasions where points are condensed along the coastline.



However, building an monitary station that can observe 100 km t50km long is absurd. Therefore, only considering density of points within a circular area, the candidates were proposed.

```
function variance = variance_of_data()
    [rows,columns] = deal(504,555);
    A = zeros(rows,columns,100);
    for i =1:100
        u = readmatrix(sprintf('%du.csv',i));
        v = readmatrix(sprintf('%dv.csv',i));
        A(:,:,i) = hypot(u,v);
    end
    variance = var(A,0,[3]);
```

```
end

function [x,I] = max_x()
[rows,columns] = deal(504,555);
A = zeros(rows,columns,100);
for i =1:100
    u = readmatrix(sprintf('%du.csv',i));
    A(:,:,i) = u;
end
[x,I] = max(A,[],'all');
end


function [avg_x_y] = average_velocity()
    [rows,columns] = deal(504,555);
    A = zeros(rows,columns,100,2);
    for i =1:100
        u = readmatrix(sprintf('%du.csv',i));
        v = readmatrix(sprintf('%dv.csv',i));
        A(:,:,i,1) = u;
        A(:,:,i,2) = v;
    end
    avg_x_y = mean(A, [1 2 3]);
end

function [A, B] = readData(rows, columns)
A = zeros(rows,columns,100,2);
for i =1:100
    u = readmatrix(sprintf('%du.csv',i));
    v = readmatrix(sprintf('%dv.csv',i));
    A(:,:,i,1) = u;
    A(:,:,i,2) = v;
end
B = A(:,:,:,2);
A = A(:,:,:,1);
end

function [T, a, b] = CalculateAndPlotHistogram(A, rows, columns)
mat_correlation = reshape(permute(A,[3 1 2]), 100, rows*columns);
T = array2table(mat_correlation,variablename = string(1:279720));
filtered_T = T(:,any(table2array(T)));

for i =1:floor(233904/15000)
    sample = filtered_T(:,i:i+14999);
    cor_mat = corrcoef(table2array(sample));
    cor_mat(boolean(eye(15000))) = 0;
    [r,c] = find(abs(triu(cor_mat))>0.99999);
    var_name = sample.Properties.VariableNames;
    store_index_A{i} = var_name(r);
    store_index_B{i} = var_name(c);
```

```
end

sample = filtered_T(:,15000*15+1:end);
cor_mat = corrcoef(table2array(sample));
cor_mat(boolean(eye(size(cor_mat,1)))) = 0;
[r,c] = find(abs(triu(cor_mat))>0.99999);
var_name = sample.Properties.VariableNames;
store_index_A{16} = var_name(r);
store_index_B{16} = var_name(c);

index_A = cat(2,store_index_A{:});
index_B = cat(2,store_index_B{:});
index_A_converted = [cellfun(@str2num, index_A)];
index_B_converted = [cellfun(@str2num, index_B)];
figure()
a = histogram(index_A_converted,"FaceColor",'b','FaceAlpha',0.5);
title('histogram of all rows >0.99999')
figure()
b = histogram(index_B_converted,'FaceColor','r','FaceAlpha',0.5);
title('histogram of all columns >0.99999')
end

function [positive,negative] = CalculatePosandNeg(a, b,T)
search_space = unique([a.Data b.Data]);
size(search_space);
cor_mat = corrcoef(table2array(T(:,search_space)));
cor_mat(boolean(eye(size(cor_mat,1)))) = 0;
[r,c] = find(triu(cor_mat)>0.99999);
[negative_r,negative_c] = find(triu(cor_mat)<-0.93);
positive = [search_space(r) search_space(c)];
negative = [search_space(negative_r) search_space(negative_c)];
end

function MapImagesc(positive,~, rows, columns, land)
binc = min(positive):max(positive);
counts = hist(positive,binc);
data = zeros(rows,columns);
data(binc) = counts;
data(land) = -1;
imagesc(data);
end

function xp = simulate_path(initialx, initialy,u,v)
epsilon = 10^6;
t_interval = 100/epsilon;
t = fix(1:t_interval:100);
t(1,size(t,2):epsilon)=1;

% [x,y] = meshgrid(1:columns,1:rows);
figure
```

```
xp = zeros(2,epsilon+1);
xp(:,1) = [initialx,initialy];
for i = 1:epsilon
    temp = fix(xp(:,i));
    speed = [v(temp(1),temp(2),t(i));u(temp(1),temp(2),t(i))];
    xp(:,i+1) = xp(:,i) + t_interval*speed;
end
end



function xp = particles_simulate_path(initialx, initialy,u,v,gpr1,gpr2)
epsilon = 10^6;
t_interval = 100/epsilon;
t = fix(1:t_interval:100);
t(1,size(t,2):epsilon)=1;

[x,y] = meshgrid(initialx-10:initialx+10,initialx-10:initialx+10);


figure
xp = zeros(2,epsilon+1);
xp(:,1) = [initialx,initialy];
for i = 1:epsilon
    temp = fix(xp(:,i));
    speed = [v(temp(1),temp(2),t(i));u(temp(1),temp(2),t(i))];
    xp(:,i+1) = xp(:,i) + t_interval*speed;
end
end
```