Particle Tracing (2D, Steady Vector Field) “RK-4” Implementation

(A slice of hurricane dataset U V variables)

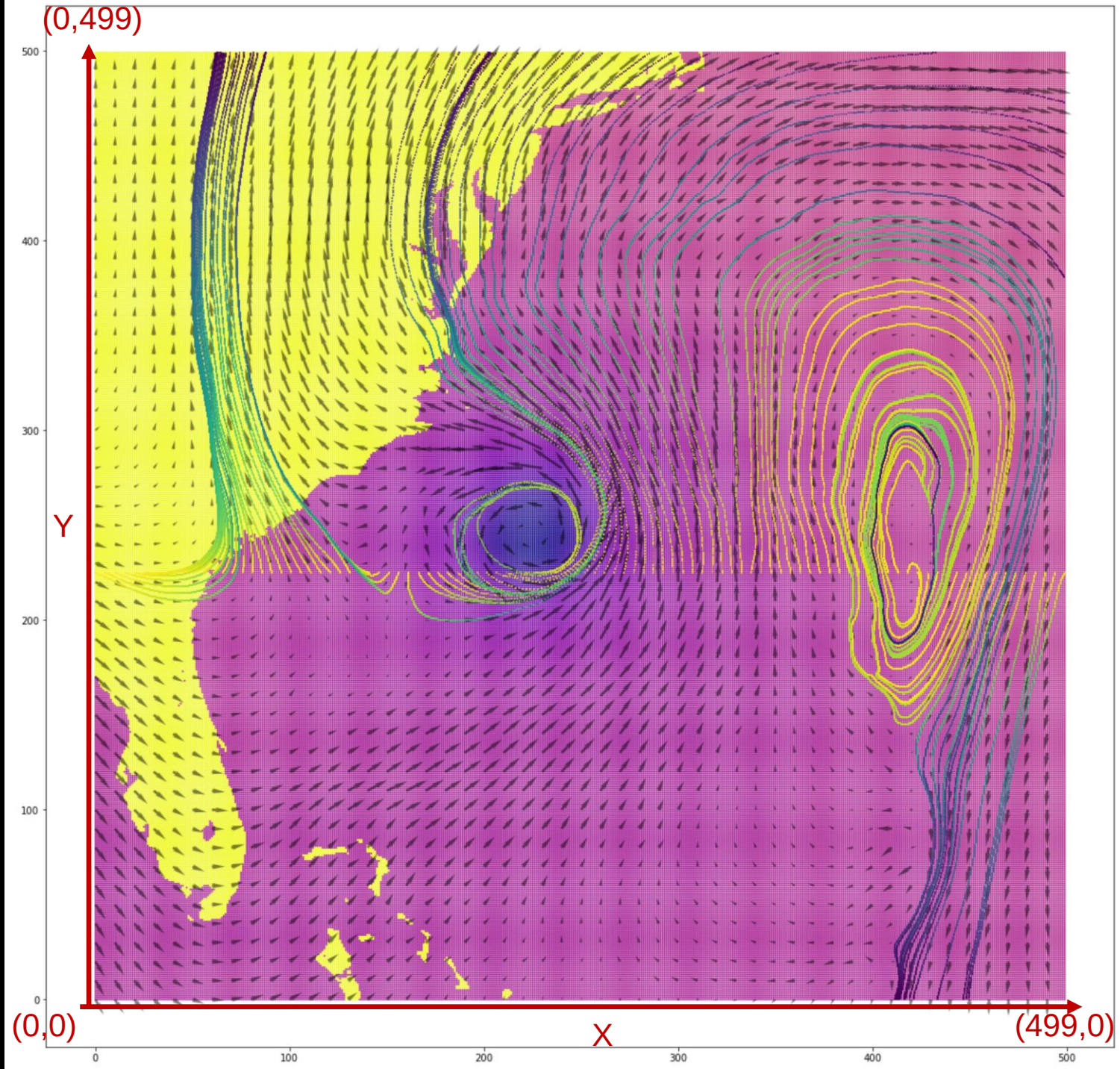
This is a 500X500 2D dataset

Data value at a grid point is a [U, V] vector.

This image is arrows (hedgehogs) vector field visualization (at every 10 grid points)

The region of this dataset is close to to Florida. The background color is just the map (with pressure) for you to know the geographical/hurricane information. You do not need the map/pressure information to compute the particle path.

Each curve: a particle path
Color on a curve: #steps of the particle trance
In this example, I set 100 seeds between [0, 255] -> [499,255], set delta t to 0.025, and set steps to 10000

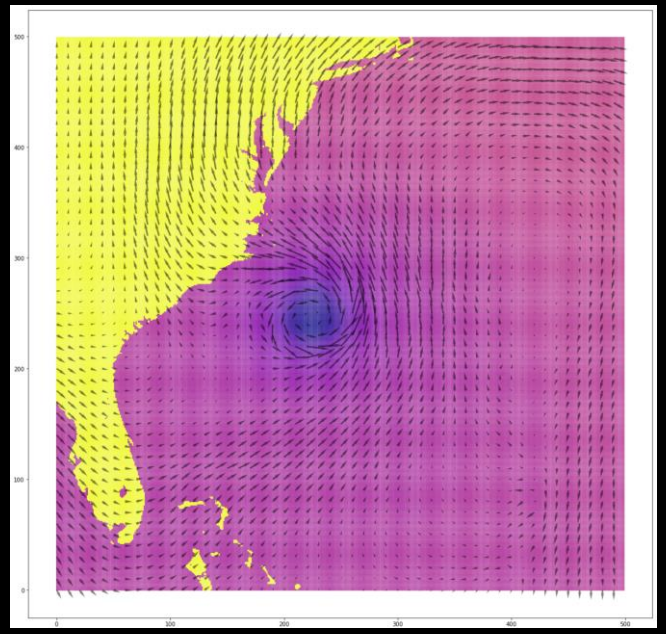


Files

- pathTracing.ipynb: code template and you should complete this homework in this file and submit this file
- flowData.npy: data set (you need them in the working folder)

Template

You can directly run the template. But it only show the arrow visualization and map/pressure background.



```python
import numpy as np
import matplotlib.pyplot as plt
import math

uData = []
vData = []

##### data loading and setup/plot image
##### argument "showBgMap": show or not to show the background map
##### DO NOT modify this function
def Initialize( showBgMap = False ):
    global uData
    global vData

    loadFlowData = np.load("flowData.npy")

    ### flow data
    uData = loadFlowData.item().get('uData')
    vData = loadFlowData.item().get('vData')

    plt.rcParams['figure.figsize'] = [20, 20]

    ### plot backgroup images
    presMapX = loadFlowData.item().get('presMapX')
    presMapY = loadFlowData.item().get('presMapY')
    presMapV = loadFlowData.item().get('presMapV')
    if showBgMap == True:
        plt.scatter(presMapX, presMapY, s=1, c=presMapV, cmap='plasma')

    ### plot arrows
    x_pos = loadFlowData.item().get('x_pos')
    y_pos = loadFlowData.item().get('y_pos')
    x_direct = loadFlowData.item().get('x_direct')
    y_direct = loadFlowData.item().get('y_direct')
    plt.quiver(x_pos, y_pos, x_direct, y_direct, scale = 50,  headwidth = 2, headlength = 5, alpha = 0.5)


##### x, y: location. Return: 2D vector at [x,y] (x and y are integer only)
##### return: a 2D vector [u, v], u is horizontal direction (right is postivie), v is the vertical direction (up is pos
##### DO NOT modify this function
def getDataVector(x, y):
    return uData[(499-y)+500*x], vData[(499-y)+500*x]


##### this function generates "numSeeds" points(seeds) from [startX, startY] to [endX, endY]
##### Return: seeds -> a list of [x, y]
def generateSeeds(startX, startY, endX, endY, numSeeds):
    seeds = []
    x = np.linspace(startX, endX, numSeeds )
    y = np.linspace(startY, endY, numSeeds )
    seeds = np.zeros([numSeeds,2])
    seeds[:, 0] = x
    seeds[:, 1] = y
    return seeds

##### this function plot a particle tracing result on the screen
##### argument "path": particle tracing result. It is represetned by a list of [x, y]. (x, y) can be floating point
##### You may not want to modify this function
def drawOneParticleTracingResult( path ):
    ps = np.array(path)
    c = np.linspace(1, 0, num=ps.shape[0])
    plt.scatter(ps[:,0], ps[:,1], c=c, s=3, marker='_')

##### (TODO) WORK on this function
##### compute ONE particle tracing result
##### you should use "getDataVector())" to get the vector you want on the grid point
##### if you need the vector between grid points, you have to implment the vector interpolation by your self
##### I do not mind the computation is efficnet or not
##### "seed": one seed (x,y)
##### "t": delta t (i suggest 0.025)
##### "steps": how many stesp for this particle tracing process
def particleTrace(seed, t, steps):
    print("implment your path tracing algorithm here!!!")

##### main (if you want, you can change the arguments in the generateSeeds() and the last argument in particleTrace() )
Initialize(True) ## set False to disable backgroup image display (faster a little bit)

seeds = generateSeeds(0, 225, 499, 225, 100) ##this line generates seeds (seeds to generate the graph on the homework c

# for seed in seeds: ###iterate through all seeds
#     path = particleTrace(seed, 0.025, 10000) # compute one particle tracing of the "seed", result is a list of [x,y ]
#     drawOneParticleTracingResult(path) # draw one particle path

plt.show()
```

Template

Initialization(): load dataset, and draw background map and arrows for you

Do not modify this function

You can set the input argument to False to disable the background map display

```
##### data loading and setup/plot image
##### argument "showBgMap": show or not to show the background map
##### DO NOT modify this function
def Initialize( showBgMap = False ):
    global uData
    global vData

    loadFlowData = np.load("flowData.npy")

    ### flow data
    uData = loadFlowData.item().get('uData')
    vData = loadFlowData.item().get('vData')

    plt.rcParams['figure.figsize'] = [20, 20]

    ### plot backgroup images
    presMapX = loadFlowData.item().get('presMapX')
    presMapY = loadFlowData.item().get('presMapY')
    presMapV = loadFlowData.item().get('presMapV')
    if showBgMap == True:
        plt.scatter(presMapX, presMapY, s=1, c=presMapV, cmap='plasma')

    ### plot arrows
    x_pos = loadFlowData.item().get('x_pos')
    y_pos = loadFlowData.item().get('y_pos')
    x_direct = loadFlowData.item().get('x_direct')
    y_direct = loadFlowData.item().get('y_direct')
    plt.quiver(x_pos, y_pos, x_direct, y_direct, scale = 50,  headwidth = 2, headlength = 5, alpha = 0.5)
```

Template

getDataVector(x,y): return the vector[u, v] at [x,y] location to you. (x and y must be integer and 0<=x, y<=499)

u: horizontal direction (same as x). Positive means left to right

v: vertical direction (same as y). Positive means bottom to top

```
##### x, y: location. Return: 2D vector at [x,y] (x and y are integer only)
##### return: a 2D vector [u, v], u is horizontal direction (right is postivie),
##### DO NOT modify this function
def getDataVector(x, y):
    return uData[(499-y)+500*x], vData[(499-y)+500*x]
```

Template

generateSeeds(): generate seeds between two points you give

[startX, startY]: the first point

[endX, endY]: the second point

numSeeds: generate how many seeds between above two locations

```
##### this function generates "numSeeds" points(seeds) from [startX, startY] to [endX, endY]
##### Return: seeds -> a list of [x, y]
def generateSeeds(startX, startY, endX, endY, numSeeds):
    seeds = []
    x = np.linspace(startX, endX, numSeeds )
    y = np.linspace(startY, endY, numSeeds )
    seeds = np.zeros([numSeeds,2])
    seeds[:, 0] = x
    seeds[:, 1] = y
    return seeds
```

Return: a list of x y locations

[[x1, y1],
[x2, y2],
[x3, y3],
...
...]

Template

drawOneParticleTracingResult():
draw one path for you

Input “path”: a list of x y positions

[[x1, y1],
[x2, y2],
[x3, y3],
…
…]

```
##### this function plot a particle tracing result on the screen
##### argument "path": particle tracing result. It is represetned
##### You may not want to modify this function
def drawOneParticleTracingResult( path ):
    ps = np.array(path)
    c = np.linspace(1, 0, num=ps.shape[0])
    plt.scatter(ps[:,0], ps[:,1], c=c, s=3, marker='_')
```

Template

particleTrace(seed, t, step): the function you should implement to generate one particle path of one seed

Seed: one seed (a x, y position)

t: delta t

steps: number of steps

```
##### (TODO) WORK on this function
##### compute ONE particle tracing result
##### you should use "getDataVector())" to get the vector you want on the
##### if you need the vector between grid points, you have to implment th
##### I do not mind the computation is efficnet or not
##### "seed": one seed (x,y)
##### "t": delta t (i suggest 0.025)
##### "steps": how many stesp for this particle tracing process
def particleTrace(seed, t, steps):
    print("implment your path tracing algorithm here!!!")
```

You may want to return a path that consists of x, y positions to represent the particle path you compute

Template

Main function

You can uncomment the for loop after you implement particleTrace()

```
##### main (if you want, you can change the arguments in the generateSeeds() and the la
Initialize(True) ## set False to disable backgroup image display (faster a little bit)

seeds = generateSeeds(0, 225, 499, 225, 100) ##this line generates seeds (seeds to gene

# for seed in seeds: ###iterate through all seeds
#     path = particleTrace(seed, 0.025, 10000) # compute one particle tracing of the "s
#     drawOneParticleTracingResult(path) # draw one particle path

plt.show()
```

Requirements

- Implement RK-4 algorithm

- When the RK4 algorithm runs, you may have to get a vector that is not right on a grid point (ex: get a vector at position [100.2, 50.7]. In this case, you should access vector at [100, 50], [101, 50], [100, 51], [101, 51] to interpolate the vector at [100.2, 50.7]
  - You can simply interpolate u and v individually by bi-linear interpolation algorithm