HW: Spatial Genomics

Objectives

1. Practice with linked list
    - Inserting a node
    - Removing a node
    - Traversing the linked list
2. Practice with classes
    - Constructors
    - Rule of three
    - Encapsulation
        - Public/Private
3. Additional practice with File I/O
4. Work with dynamic memory

Overview

MERFISH is a massively multiplexed single-molecule imaging technology capable of simultaneously measuring the copy number and spatial distribution of hundreds to tens of thousands of RNA species in individual cells. We have some datasets that contain spatial coordinates of individual cells within the mouse brain tissue samples measured by MERFISH. These datasets are stored in .csv files. The following is a description of the datasets (e.g., cell_metadata1.csv):

Each row corresponds to a cell. The first column is the unique cell ID and the remaining columns are:

A. fov – the field of view containing the cell
B. volume – the volume of the cell in um^3
C. center_x – the x coordinate of the center of the cell in global micron coordinates
D. center_y – the y coordinate of the center of the cell in global micron coordinates
E. min_x, max_x – the minimum and maximum of the bounding box containing this cell in x in global micron coordinates
F. min_y, max_y – the minimum and maximum of the bounding box containing this cell in y in global micron coordinates

The following table shows part of the data in the cell spatial metadata file (cell_metadata1.csv): each cell is identified by a unique ID. The field of view (FOV) is the maximum area visible when looking through the microscope eyepiece (eyepiece FOV) or scientific camera (camera FOV), usually quoted as a diameter measurement (see Figure 1). The first image shows a full mouse coronal slice. The second image shows one FOV. In one FOV, a certain number of cells can be observed. Each cell appears in only one cell.

	fov	volume	center_x	center_y	min_x	max_x	min_y	max_y
110883424764611924400221639916314253469	0	432.141	156.563	4271.33	151.53	161.596	4264.62	4278.03
21019503702334676259087972265037170006 4	11	1586.98	366.555	2724.74	358.606	374.504	2716.03	2733.44
135188247894899244046039873973964001182	0	1351.8	156.509	4256.96	148.29	164.728	4247.66	4266.26
164766962839370328502017156371562646881	0	1080.65	159.965	4228.18	152.178	167.752	4220.56	4235.81
165747897693809971960756442245389760838	0	1652	167.579	4323.87	158.226	176.932	4314.19	4333.55
260943245639750847364278545493286724628	0	1343.38	160.559	4308.8	152.394	168.724	4301.23	4316.37
332515239482476129412625869567854153692	0	960.214	160.613	4296.11	154.446	166.78	4289.03	4303.2
348721203909032302324570638884241493 61	0	1101.28	165.689	4347.52	159.09	172.288	4339.68	4355.36
788245672903925837591491742707887851 95	0	445.436	161.909	4319.22	157.254	166.564	4315.38	4323.07
789127781392866020075979335456245871 49	0	409.057	161.693	4251.35	157.362	166.024	4246.15	4256.54
793483000387047959093834102173243228 59	0	1702.84	174.383	4374.47	165.894	182.872	4364.2	4384.74
112315799428971059837534848038041808853	11	1454.6	353.811	2742.56	345.754	361.868	2733.31	2751.8

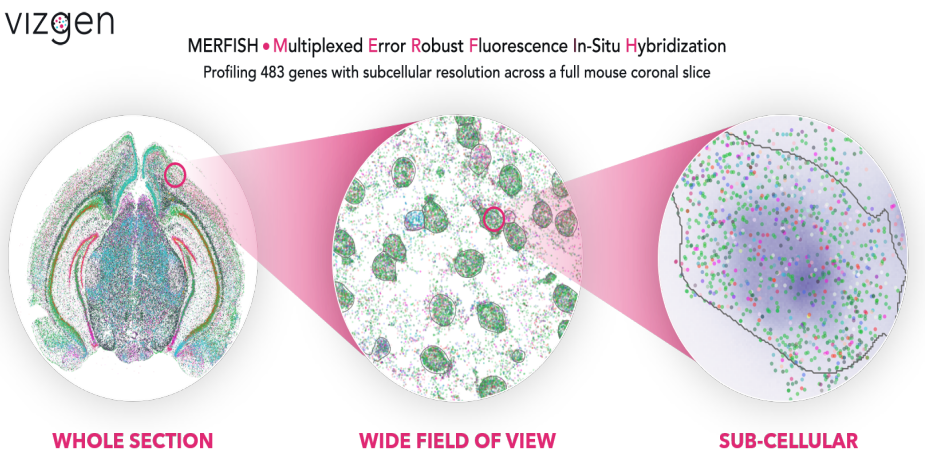

Figure 1

From one FOV, only a very small piece of the tissue can be observed. Therefore, to observe the whole tissue, multiple FOVs are required. To distinguish different FOVs, each FOV is uniquely labeled by one unique integer. Therefore, in the above table, 10 cells are observed in FOV 10 and 2 cells are observed in FOV 11.

Figure 2 shows all the cells of the file cell_metadata1.csv, where each cell is represented by its coordinates (center_x, center_y). The blue, yellow, and red areas are three different FOVs. The

yellow area consists of 193 cells visible in FOV 216, the red area is made up of 230 cells visible in FOV 217, and the blue area contains 215 cells visible in FOV 218.

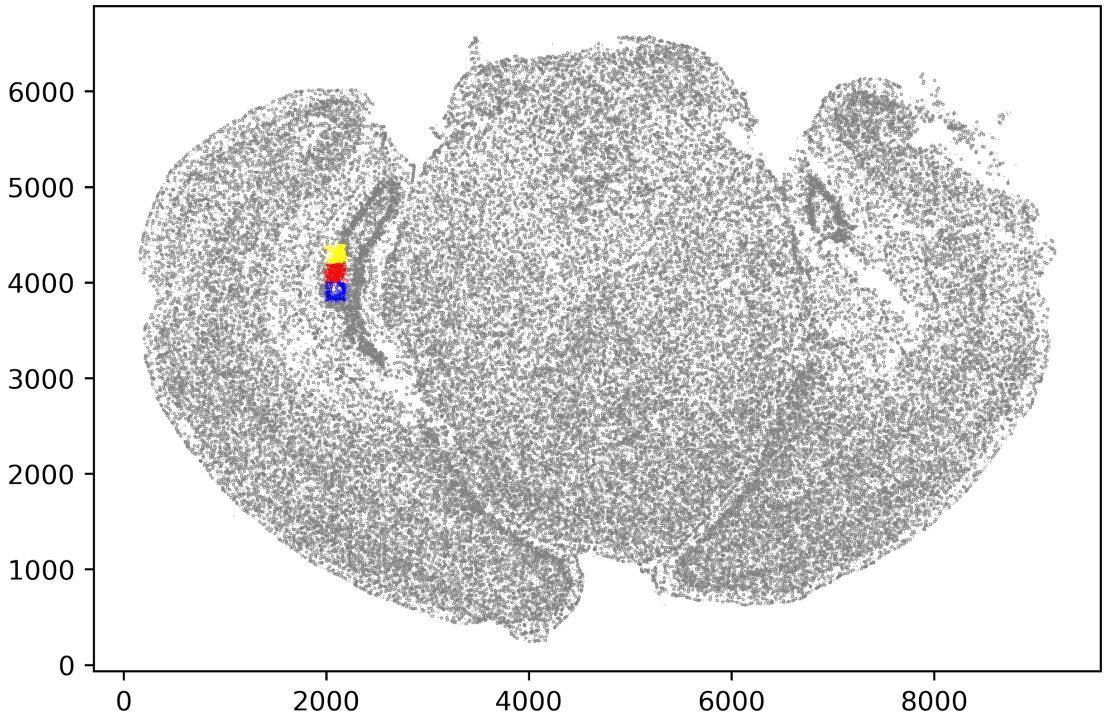

Figure 2

We want to analyze these datasets and compute some statistics of these datasets. In particular, given a FOV k, we want to know how many cells are observed in this FOV, denoting it by $counk(k)$. Suppose there are $m = count(k)$ cells in FOV k, denoted by $cell_1, cell_2, ..., cell_m$. For each cell $cell_i$, let vol_i be its volume. We are interested in computing the average and variance of those cells' volume. Let $avg(k) = (\sum_{i=1}^{m} vol_i)/m$ be the average and $var(k) = (\sum_{i=1}^{m} \left(vol_i - avg(k)\right)^2)/m$ be the variance. We assume that for each cell $cell_i$, vol_i is close to $avg(k)$. If m is large enough and a cell's volume deviates too much from $avg(k)$, we say this cell is an outlier. For example, if a cell's volume is much larger than the average volume of those cells appearing in the same FOV with that cell, then we may guess the cell is not an individual cell but consists of more than one cells.

Roadmap

This is a guide on how to approach this homework. Specific requirements for functions are in the Requirements section.

1. Get the start code and test files from Mimir.

- Code Files
  - `CellData.h`: the class definition for each cell
  - `CellData.cpp`: the definitions for the CellData member functions
  - `Node.h`: the class definition for a node in the linked list
  - `Node.cpp`: the definitions for the Node member functions
  - `LinkedList.h`: the class definition for a linked list, each node contains a cell data
  - `LinkedList.cpp`: the definitions for the LinkedList.h member functions
  - `CellDatabash.h`: the class definition for a database file (.csv file)
  - main.cpp: this file is useful for testing your code locally
- Data Files
  - See section **Overview** and **supporting information** on data file format.
  - Files
    1. cell_metadata1.csv
    2. queries.data
    3. result.data

2. Review the code.
   - Read header and source files
   - Compile and run the initial state of the starter code with the following command line.
     ```
     g++ -std=c++17 -Wall -Wextra -pedantic -g
     -fsanitize=address,undefined CellData.cpp Node.cpp
     LinkedList.cpp CellDatabase.cpp main.cpp
     ```
   - Confirm that the initial state of the starter code executes without errors and with only warnings related to unused variables/parameters and functions not returning values. Go ahead and submit the starter code to Mimir so you can take a look at the test cases.
   - Submit code to Mimir before doing any development.
     - Review the test cases so you know what to expect.
   - Look at the dependencies within each class to determine the best order to implement methods (i.e., functions) of each class. `LinkedList` depends on `Node, Node` depends on `CellData`, and `CellDatabase` depends on `LinkedList. CellData` has no dependencies.
   - Implement the classes.

Hints and Recommendations

- As you develop your code, you only need to implement the methods/functions you need at the moment. You don't have to get everything done at once. It is not a good strategy to do so.
- You can utilize `main.cpp` to test the functions that you have implemented so far. Alternatively, you can create a separate tester file with its own main function.
- Plan and think before you code.
- Pick small problems to solve first, write test cases first, and develop incrementally.
- Use descriptive (long) naming conventions for variables and functions.

- Add comments to the code to describe anything which is not obvious from the code.
- Use whitespace (indentation, newlines) to visually organize code.
- Use and add functions to reduce code duplication and increase abstraction.
- It is the end of the semester. You are busy. Start and finish early.
- Recompile and return (run == test).
    - Check for errors.
    - If no errors, move on
    - Else, start debugging
    - When changing your code to fix errors, before executing or uploading to Mimir, stop and explain to yourself why your bug fix will work.
    - When you get stuck, ask for help on Campuswire and attend office hours. The TAs and the instructors will help you.
        - Read the prior posts before posting a new question (which may be a duplicate).

Requirements

You will implement a program that receives input from two files: a cell spatial metadata file (e.g., cell_metadata1.csv) and a query file (e.g., queries.data). Your implementation has to use a linked list to store the data you read from the spatial metadata file.

- You are required to implement classes `CellData, Node, LinkedList`, and `CellDatabase`. You are required to implement any of the class member functions not already provided for you in the starter code.
    - When implementing constructors, if necessary, assign default values to data members that do not have a corresponding constructor parameter.
- Use of an unapproved header file will result in a score of 0.
- The program must compile without warnings or errors.
    - `g++ -std=c++17 -Wall -Wextra -pedantic -g -fsanitize=address,undefined CellData.cpp Node.cpp LinkedList.cpp main.cpp`
- The program must run without errors or unhandled exceptions.

Allowed Includes

- `<iostream>`
- `<fstream>`
- `<sstream>`
- `<string>`
- `<cmath>`
- `“CellData.h”`
- `“Node.h”`
- `“LinkedList.h”`
- `“CellDatabase.h”`

Class CellData

We provide `CellData` class definition (`CellData.h`)

You implement in `CellData.cpp`:

- Destructor is given. Do not modify it.
- Constructor
  - Default constructor: set id to "-1", and all other variables -1.
  - Parameterized Constructor: `CellData(std::string id, int fov, double volume, double center_x, double center_y, double min_x, double max_x, double min_y, double max_y);`
  - An overloaded operator< for the class: `bool operator<(const CellData& b);`
    - First ordered by FOV number
    - Then by id
  - An overloaded operator== for the class: `bool operator==(const CellData& b);`
    - returns true if and only if two objects are memberwise identical

Class Node

We provide class definition (`Node.h`)

You implement:

- Constructor
  - Default constructor: `Node()`
  - Parameterized constructor: `Node(std::string id, int fov, double volume, double center_x, double center_y, double min_x, double max_x, double min_y, double max_y);`
- Destructor is given. Do not modify it.

Class LinkedList

We provide class definition in `LinkedList.h`.

You implement in `LinkedList.cpp`:

- Default Constructor: `LinkedList()`;
- Destructor: `~LinkedList();`
- Copy constructor: `LinkedList(const LinkedList& other)`;
- Copy assignment operator: `LinkedList& operator=(const LinkedList& other)`;
- `void insert(string id, int fov, double volume, double center_x, double center_y, double min_x, double max_x, double min_y, double max_y)`: insert a record to the linked list.
  - While inserting into the LinkedList, the nodes in the linked list are
    - First ordered by FOV number
    - Then by id

  For example, after reading 3 lines of the file cell_metadata1.csv, your linked list looks like:

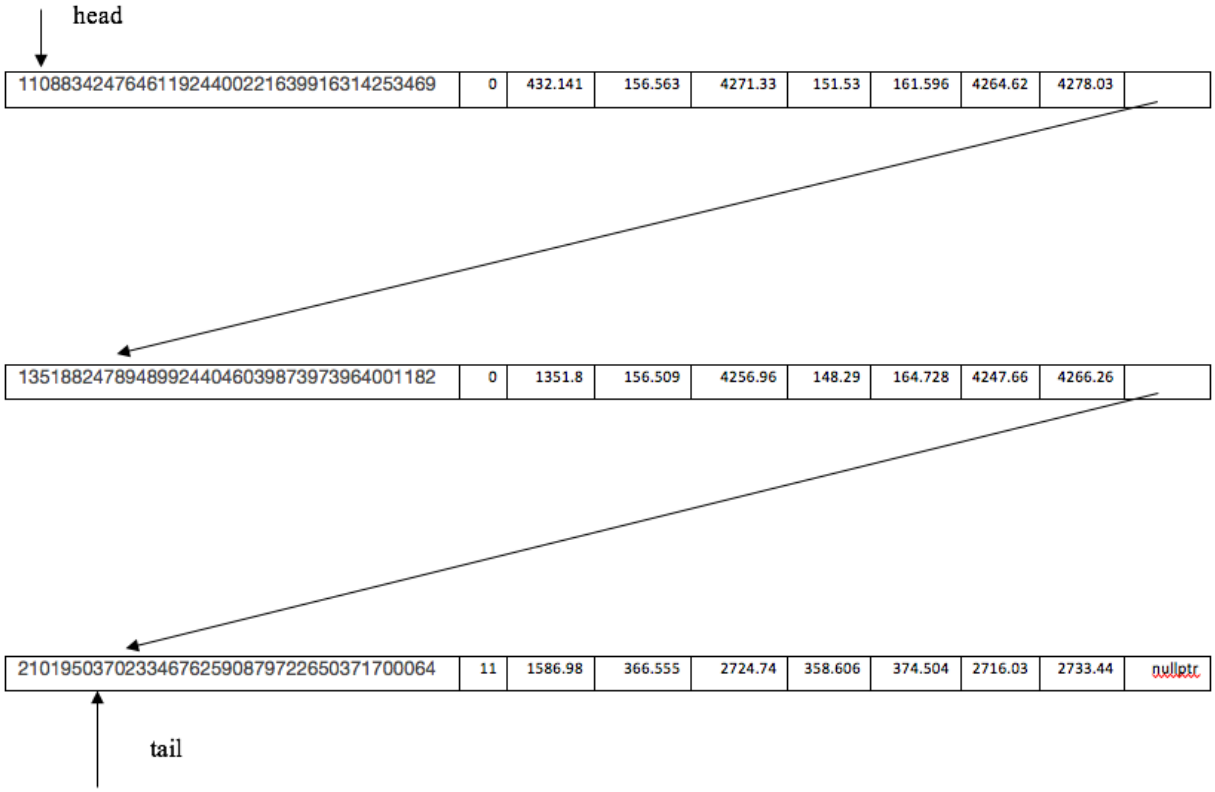

Figure 3

- `void remove(string id, int fov, double volume, double center_x, double center_y, double min_x, double max_x, double min_y, double max_y)`: delete a record from the linked list.
- `getHead()`: this function is supposed to return the pointer to the start Node of the list.
    - Note, a normal linked list would **NOT** have a getHead function since it breaks encapsulation. This is only used for auto-grading!
- `clear()`: clear the content of the linked list and release memory.
- `int countN(int k)`: compute $count(k)$ and return it (see definition in section **Overview**). Implementation is given. Do not modify it.
- `double average(int k)`: compute $avg(k)$ and return it (see definition in section **Overview**).
- `double variance(int k)`: compute and return the variance $var(k)$ (see definition in section **Overview**). Implementation is given. Do not modify it.
- `string outliers(int k, int j, int N)` (see definitions in section **Overview**).
    - if $count(k)$ < N, return a message "Less than N cells in fov k".
    - otherwise, let $delta(k)$ be the square root of $var(k)$. Remove those cells $cell_i$ such that $vol(i)$ is not in the range $[avg(k) - j * delta(k),\ avg(k) + j * delta(k)]$ from the linked list.

Return a message "# cells are removed.", where # is the number of cells removed.

- `void print()`: use this function to print out the content in the linked list.
    - The first line should be "id, fov, volume,center_x,center_y,min_x,max_x,min_y,max_y" **without any whitespaces.**
    - Output format is the same as the format in the input file. id, fov, volume,center_x,center_y,min_x,max_x,min_y,max_y. Items are separated with a comma **without any whitespaces.**
    - When you output, it will look like the input file, with each cell on one line. However, the order should be sorted according to the requirements above.

Class CellDatabase

We provide class definition in `CellDatabase.h`.

You implement in `CellDatabash.cpp`:

- `loadData(const string& filename):` read the cell records from the data file and populate the linked list.
    - If the file could not be opened, output the message "Error: Unable to open" in the console. **Do not** throw an exception.
    - If the input files contain a line with an invalid format, you should output an error message "Error, Invalid input: "followed by that invalid line on the console, and continue executing the program. **Do not** throw an exception. A line is valid if all of the following are true:
        - It consists of **only** id, fov, volume,center_x,center_y,min_x,max_x,min_y,max_y, where each item is separated by a comma **without any whitespaces**.
        - id consists of only number digits.
        - fov is an integer.
        - volume,center_x, center_y,min_x,max_x,min_y,max_y are double and volume is positive.
- `outputData(const string& filename)` is given. Do not modify it.
- `performQuery(string query_filename)`: read the query from the data file and perform the queries. Each line is a query.
    - If the input files contain a line with an invalid format, you should output an error message "Error, Invalid input: " followed by that invalid line on the console, and continue executing the program. **Do not** throw an exception. A line is valid if it is one of the following:
        - AVG k, where k is a positive integer
        - VAR k, where k is a positive integer
        - COUNT k, where k is a positive integer
        - OUTLIER k, j, N, where k and N are positive integers
    - For each valid query, output the query, followed by **a colon, a whitespace** and then the result of that query. The result of the query is defined as follows:

- AVG k, is the value $avg(k)$
- VAR k, is the value $var(k)$
- COUNT k, is the value $count(k)$
- OUTLIER k, j, N, is the value returned by outliers(k, j, N)

Submission

The source files to submit are named:

- `CellData.h` and `CellData.cpp`
- `Node.h` and `Node.cpp`
- `LinkedList.h` and `Linkedlist.cpp`
- `CellDatabase.h` and `CellDatabase.cpp`
- `main.cpp`

Supporting Information

Data File Format

A cell spatial metadata file (e.g., cell_metadata1.csv): the file contains the spatial metadata for each of the detected cells. This data has been obtained from VIZGEN. Each row corresponds to a cell. The first column is the unique cell ID and the remaining columns are:

- fov – the field of view containing the cell
- volume – the volume of the cell in um^3
- center_x – the x coordinate of the center of the cell in global micron coordinates
- center_y – the y coordinate of the center of the cell in global micron coordinates
- min_x, max_x – the minimum and maximum of the bounding box containing this cell in x in global micron coordinates
- min_y, max_y – the minimum and maximum of the bounding box containing this cell in y in global micron coordinates

The following shows part of the data in the cell spatial metadata file:

```
,fov,volume,center_x,center_y,min_x,max_x,min_y,max_y
110883424764611924400221639916314253469,0,432.141,156.563,4271.33,151.53,161.596,4264.62,4278.03
135188247894899244046039873973964001182,0,1351.8,156.509,4256.96,148.29,164.728,4247.66,4266.26
164766962839370328502017156371562646881,0,1080.65,159.965,4228.18,152.178,167.752,4220.56,4235.81
165747897693809971960756442245389760838,0,1652,167.579,4323.87,158.226,176.932,4314.19,4333.55
260943245639750847364278545493286724628,0,1343.38,160.559,4308.8,152.394,168.724,4301.23,4316.37
332515239482476129412625869567854153692,0,960.214,160.613,4296.11,154.446,166.78,4289.03,4303.2
348721203909032302324570638884241493 61,0,1101.28,165.689,4347.52,159.09,172.288,4339.68,4355.36
788245672903925837591491742707887851 95,0,445.436,161.909,4319.22,157.254,166.564,4315.38,4323.07
789127781392866020075979335456245871 49,0,409.057,161.693,4251.35,157.362,166.024,4246.15,4256.54
793483000387047959093834102173243228 59,0,1702.84,174.383,4374.47,165.894,182.872,4364.2,4384.74
112315799428971059837534848038041808853,11,1454.6,353.811,2742.56,345.754,361.868,2733.31,2751.8
210195037023346762590879722650371700064,11,1586.98,366.555,2724.74,358.606,374.504,2716.03,2733.44
225678205628489765024995115598818613245,11,1567.74,349.653,2762.59,340.354,358.952,2754.91,2770.27
226970627149036985512064404522770557414,11,1475.14,341.229,2780.95,332.362,350.096,2772.41,2789.49
337569741715169530762455174860904566910,11,1291.63,330.051,2794.88,322.102,338,2786.66,2803.1
101749104962864163101921170104921674883,12,1121.36,347.949,2814.53,340.756,355.142,2807.77,2821.29
113425335266016764770886165837953002620,12,1038.5,334.503,2814.2,327.904,341.102,2804.53,2823.88
124728226247165472610070765672758045 48,12,691.103,355.185,2885.54,348.1,362.27,2879.26,2891.81
126576512189555517054861881579697262527,12,626.296,286.983,2985.27,280.708,293.258,2980.78,2989.77
```

A query file (e.g., queries.dat): the file contains a list of queries (i.e., requests for information) that your program should calculate based on the data in the spatial metadata file. Each line in the file contains one query. There are four types of queries:

- AVG k, where k is a positive integer
- VAR k, where k is a positive integer
- COUNT k, where k is a positive integer
- OUTLIER k, j, N, where k and N are positive integers

Your program should read these two files and generate a file **result.data** with queries and their results. Regardless, **queries.data** might look like this:

AVG 217
VAR 217
COUNT 217
OUTLIER 217 2 200
AVG 11

Then your program is expected to produce a file named **result.data** with the following content:

AVG 217: 403.424
VAR 217: 66562.9
COUNT 217: 230
OUTLIER 217 2 200: 12 cells are removed.
AVG 11: 1475.22

To illustrate the result, we draw the histogram for the FOV 217 in cell_metadata1.csv, where X-axis represents the volume while Y-axis gives the information about frequency.

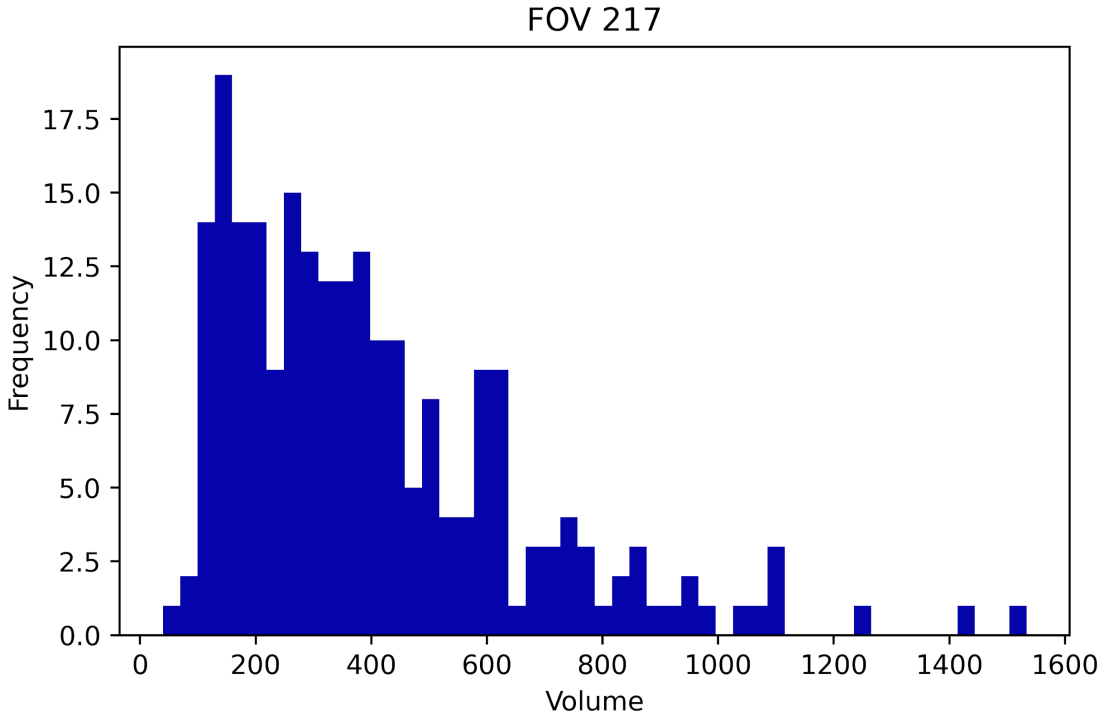

Figure 4