Problem Chosen	2024 MCM/ICM Summary Sheet	Team Control Number
ABCDEF		1111111

2024 Mathematical Contest in Modeling (MCM) Summary Sheet

Summary

This study investigates the data of the geomagnetic monitoring sensors through **matrix** approach, **KD Tree(KDT)** model and **Heuristic Filtering Based on Nearest Neighbor Algorithm(HF-BNNA)**. By analyzing data from all sensors in specific range of time, we address three key questions:

Most Representative Data In 4 hours(Problem 1):
We introduce a matrix approach to accurately calculate the difference between the measured data and the initial data of each sensor. We further utilize the calculations in the difference matrix to find the most typical value for each time point in the given 4 hours.

Most Representative Data In 24 hours(Problem 2):
We use the identical approach concerning matrix in problem 1. However, the high requirement of the computer's VRAM caused by the huge amount of data prevent us using the method continuously. As some experiments are taken into practice, KDT are eventually chosen to seek out the consequences of the typical values in 24 hours' time point.

Signal Variation From Specific Object. (Problem 3):

Our findings highlight that:
We can solve some problems base on established mathematics model. For example. Filter out the effect of noise on the magnetic field. Measure magnetic field of certain object.Analyze factors causing geomagnetic field change.Predict the occurrence of natural disasters and take appropriate measures. Learn about the movement of the liquid core, and between the outer core and the mantle interaction.

Keywords: Earth's Magnetic Field; Sensor; Time Point; Matrix; KDT; HFBNNA.

Contents

1 Introduction

1.1 Problem background

The Earth's magnetic field is not static; it is constantly changing due to various natural and external influences. One of the major factors that affect the magnetic field is solar activity, such as solar flares and coronal mass ejections, which interact with the Earth's magnetic field through the solar wind. These interactions can result in geomagnetic storms, which can disrupt power grids, communication systems, and navigation technologies, with potentially serious consequences for modern society.

The Earth's magnetic field is also influenced by processes occurring within the Earth itself. The movement of the liquid iron in the outer core, as well as the complex interactions between the outer core and the mantle, play a significant role in shaping the geomagnetic field. Over time, these internal dynamics lead to changes in the strength, direction, and structure of the magnetic field, which can be observed at the Earth's surface.

To understand and monitor these variations, a network of geomagnetic observatories is required. These observatories would continuously measure changes in the Earth's magnetic field and provide crucial data for scientific research, earthquake prediction, national defense, and various technological applications. Accurate measurements of the magnetic field require sensitive sensors that can detect small variations in magnetic field strength and direction. Some data processing are needed in order to find apparent characteristic and accurate results concerning to the intensity and direction of the Earth's magnetic field.

1.2 Restatement of the Problem

Based on the background information and the constraints outlined in the problem statement, we need to address the following issues:

- How to select the best sensors to represent the magnetic field of a specific time.
- How to perform the selection algorithm efficiently on a large amount of data.
- How to extract the original signal from the data of many sensors and eliminate noises.

1.3 Our work

Our work is illustrated in Figure 1.

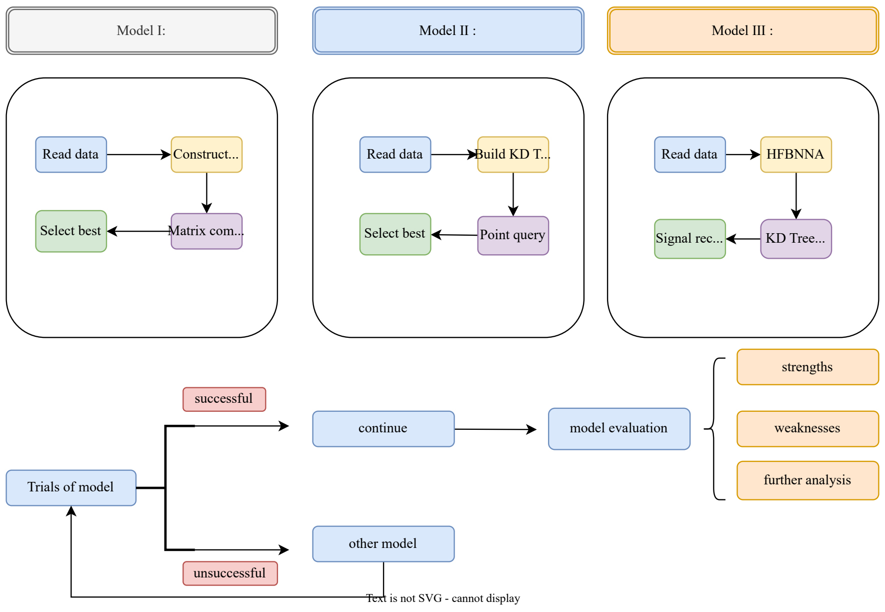

Figure 1: Our work

2 Assumption and Justifications

Given that practical problems often involve numerous complex factors, the first step is to make reasonable assumptions in order to simplify the model. Each assumption is then followed by a corresponding explanation.

- **Assumption**: The geomagnetic field and magnetic field from a specific objects are uniform in space.
- **Justification**: To make the variation of data be consistent with data from sensors from different unknown locations only uniform magnetic field's model can be established with relative accurate data.
- **Assumption**: The data from the sensors is accurate enough to model the magnetic field.
- **Justification**: The amount of the sensors is large and for the geomagnetic data is nearly consistent.

3 Data processing

To speed up data I/O, we first combine all the sensor data into a dictionary format stored in JSON file. Then, the data is rearranged as a matrix, its dimension is the time stamps, the second dimension represents every sensors, the final dimension contains data from all channels.

4 Model for problem 1 & 2

4.1 Problem analysis

Problem 1 and 2 required finding a variation from a sensor such that the difference of this value and data from other sensors is the minimum. To evaluate the difference, we utilized the mean-squared error(MSE) between the selected data and other sensors'value in the same time point.

$$\mathrm{MSE} = \frac{1}{N} \sum_{(x_i, y_i, z_i) \in P_t} \left[(x_i - x_t)^2 + (y_i - y_t)^2 + (z_i - z_t)^2 \right] \tag{1}$$

By minimizing the MSE, we can find the most representative data from all sensors in a specific time according to our second assumption. In other word, the task is for every time stamp, to select a sample for all sensors whose MSE with data of other sensors is the smalseness. Taking no account of that the targeted value must be selected from an existing data and only consider one channel. The target function is

$$S = \sum (x_i - x_t)^2 \tag{2}$$

Deriving the partial differential to x_t

$$P = \frac{\partial S}{\partial x_t} = \sum 2(x_i - x_t) \tag{3}$$

Let $P = 0$, $x_t = \frac{1}{N} \sum x_i$ which is the mean of all data meaning that the ideal value is the mass of center of all data. To obtain the optimized choices, it is required to find the existing point that is closest to the mass of center at every time.

4.2 Differential matrix

4.2.1 Description

It is known that there are 353 geomagnetic monitoring sensors.Each sensor send back a set of data for each moment.A set of data includes the strength of the magnetic field in the three directions of x,y,z.In order to establish a suitable mathematical model to ensure that the variation in the measurement data from all sensors between 24:00 and 4:00 (midnight to 4:00 AM) is consistent based on appendix 1.Each moment,we need select a variation from one sensor, such that the difference between the variation of each sensor is minimized. To establish a mathematics model based on our problem analysis.

First of all,eliminate the effect of different initial values for each sensor caused by different position.

Then, to select the data points that is the closest to the mass of center, we need to calculate the distance between every samples and the mass of center meaning that for a certain time, every differences between values from every sensors should be calculated. For problem 2, there are 353 sensors and 2881 time stamps in 24 hours and x,y,z three directions. The amount of calculation is $2881 * 353 * 353 * 3 = 1076995587$ which is huge for non-parallel algorithm. However, we can remodel this problem to matrix operation.

Take a simple example of deriving every difference between elements of vector $[1, 2, 3]$. The following steps can be taken to get the result.

1. Expand the vector to a matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \tag{4}$$

2. Transpose the matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} \tag{5}$$

3. Do element-element subtract between the original and transposed matrix.

$$M = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \tag{6}$$

For the newly obtained matrix M, its x-th row and y-th column represent the difference between the x-th element and the y-th element of the original vector.

4. Replace every element in M with its square.

$$M_1 = \begin{bmatrix} 0 & 1 & 4 \\ 1 & 0 & 1 \\ 4 & 1 & 0 \end{bmatrix} \tag{7}$$

5. Sum up M_1 along its columns to get a new vector.

$$V_1 = [5, 2, 5] \tag{8}$$

The x-th value of V_1 is the square sum of differences between the x-th value in the original vector and its other elements.

6. Select the minimum value from V_1The minimum value of V_1 is 2 (its second element),thus, the second value of the original vector (2) is the value that is closest to the mass of center of $[1, 2, 3]$

For the data from the sensors, this method can also be appied with the following steps.

1. Rearrange the data

Form the data to a tensor with a shape of:

$$M_0 : [\text{num of time stamps} \times \text{num of sensors} \times 3] \tag{9}$$

2. Expand the matrix along the second dimension

$$M_1 : [\text{num of time stamps} \times \text{num of sensors} \times \text{num of sensors} \times 3] \tag{10}$$

3. Subtract with its transposed as M_2

$$M_2 = M_1 - M_1^T : [\text{num of time stamps} \times \text{num of sensors} \times \text{num of sensors} \times 3] \quad (11)$$

4. Perform element-wise square on M_2

$$M_3 = M_2.^2 : [\text{num of time stamps} \times \text{num of sensors} \times \text{num of sensors} \times 3] \quad (12)$$

5. Sum up M_3 along the third dimension

$$M_4 : [\text{num of time stamps} \times \text{num of sensors}] \quad (13)$$

6. Find the indices of maximum value of every time stamps and map the obtained indices to the values of the original vector.

4.2.2 Implementation and results

For both problem 1 and 2, we used some simple operation of PyTorch framework along with CUDA to accelerate our matrix calculation on GPU, the solution of both problems can be obtained within one second, however, it required large memory for large matrix especially for problem 2. Table 1 shows the optimal sensors of a few time stamps of problem 2.

Table 1: Results illustration

Time stamp	Optimal sensors indexed
19:23:36	41
19:24:06	330
19:24:36	210
19:25:06	120
19:25:36	42
19:26:06	320
19:26:36	254
19:27:06	151
19:27:36	291

4.3 KD Tree

4.3.1 Description

Considering problem 2 that the sensors data from 24 hours is requied to be processed, the size of matrix constructed for differential calculation is extremely large ($4311 \times 2881 \times 2881 \times 4$). It requires approximately 25GB of GPU memory if the data is stored in 64-bit floting point form which exceed the limit of household PC. Although we rent a A800 GPU to obtain the accurate results, this is not pratical in because of limited access to top performance devices.

KD-Tree(KDT) however, is a data structure that can reduce the time complexity of searching algorithm to the nearest neighbor from $O(n)$ to $O(\log(n))$. In this case, we build a KDT for every time stamps and query the nearest neighbors of the center of mass. Take some ponits on a two-dimensions surface as an example, KDT can be built by the following steps:

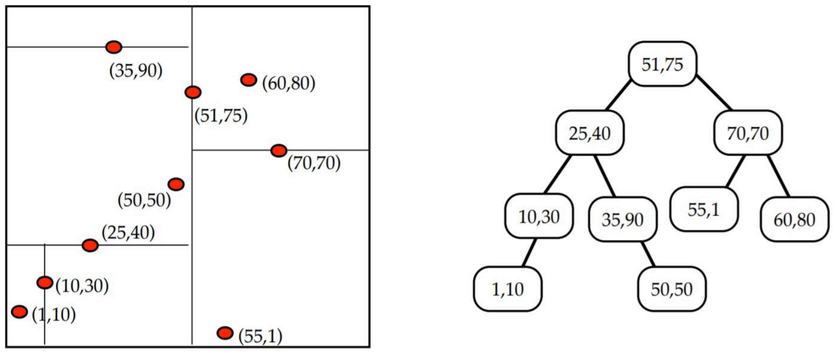

Figure 2: Build up KD-Tree

1. Choose the splitting axis: The building process begins by selecting the axis along which the points will be split. This is done based on the depth of the current node. At depth 0 (root node), the axis is typically the x-axis; at depth 1, the y-axis is chosen; and at depth 2, the z-axis is used. This pattern repeats cyclically (x → y → z → x → …).
2.

5 Model Evaluation

5.1 Strengths

Matrix Method: Our approach concerning difference matrix is quite simple at the level of thought, which is understandable for readers. And its result is the most accurate in the all methods we use.

KDT: The model does not have high requirement for computer's VRAM. And the speed of calculation and comparison is much faster.

HFBNNA: In high-frequency band processing, noise can be a major challenge. Neural networks can learn to filter out unwanted noise and focus on the meaningful signal components, improving the quality of the processed data.

5.2 Weaknesses

Matrix Method: If there is a large amount of data, it's difficult for a normal computer to deal with the model's algorithm in a short period of time. That is the reason why when the range of time is prolonged to 24 hours, other methods are taken into consideration instead of the matrix approach.

5.3 Further Discussion

Accurate measurements of the geomagnetic field are crucial for space weather prediction. Solar activities such as solar wind and geomagnetic storms can influence the Earth's magnetic field, causing phenomena like geomagnetic storms. By monitoring changes in the geomagnetic field, it is possible to predict geomagnetic storms in advance and issue warnings to mitigate potential impacts on satellite communications, power grids, and navigation systems. Space weather prediction helps protect spacecraft, satellites, and ground infrastructure, ensuring their proper functioning during periods of intense solar activity.

References

[1] G. Strang, *Introduction to linear algebra*. SIAM, 2022.

[2] N. A. Gershenfeld, *The nature of mathematical modeling*. Cambridge university press, 1999.

[3] 陈宝林, 最优化理论与算法. 清华大学出版社有限公司, 2005.

[4] M. M. Meerschaert, "数学建模方法与分析." 北京: 机械工业出版社, 2005.

[5] Y. Ju, 线性代数. 清华大学出版社有限公司, 2002.

[6] E. A. Bender, *An introduction to mathematical modeling*. Courier Corporation, 2000.

Appendix A

```
#ifndef KD_TREE_H
#define KD_TREE_H
#include <cmath>
#include <limits>
#include <vector>
#include <algorithm>
using LD=long double;
struct Point {
    LD x{},y{},z{};
    int index_in_sensor{};
};
using T_Point_VEC=std::vector<Point>;
class KdTree {
public:
    struct KDNode {
        Point point;
        KDNode* left;
        KDNode* right;
          explicit KDNode(const Point& point) : point(point), left(nullptr),
right(nullptr) {}
    };

private:
    static LD distance(const Point &p1, const Point &p2) {
        return sqrt(pow((p1.x - p2.x),2) +
              pow(abs(p1.y - p2.y),2)  +
              pow(abs(p1.z - p2.z),2));
    }
    static bool compare(const Point &p1, const Point &p2, int depth) {
        if(depth%3==0) {
            return p1.x<p2.x;
        }
        if(depth%3==1) {
            return p1.y<p2.y;
        }
        return p1.z<p2.z;
    }
    KDNode *root{nullptr};
    static KDNode* r_build_tree(std::vector<Point> &points, int depth=0) {
        if(points.empty()) {
            return nullptr;
        }
        int axis=depth%3;
        std::ranges::sort(points, [depth](const Point &p1, const Point &p2) {
            return compare(p1,p2,depth);
        });
        int median=points.size()/2;
        auto *new_node=new KDNode(points[median]);
        std::vector left(points.begin(), points.begin()+median);
        std::vector right(points.begin()+median+1, points.end());
        new_node->left=r_build_tree(left, depth+1);
        new_node->right=r_build_tree(right, depth+1);
        return new_node;
    }
    static void r_find_nearest(KDNode *root, const Point &query, KDNode*& best,
LD &bestDist, int depth=0) {
        if (root == nullptr) {
            return;
        }

        // Use squared distance for more precision
        LD dist = distance(query, root->point);
        if (dist < bestDist) {
            bestDist = dist;
            best = root;
        }

        int axis = depth % 3;
        KDNode *next_node = nullptr;
        KDNode *other_node = nullptr;

        // Decide which side to search first
        if ((axis == 0 && query.x < root->point.x) || (axis == 1 && query.y <
root->point.y) || (axis == 2 && query.z < root->point.z)) {
            next_node = root->left;
            other_node = root->right;
        } else {
            next_node = root->right;
            other_node = root->left;
        }

        // Explore the next node (recursive search)
        r_find_nearest(next_node, query, best, bestDist, depth + 1);

        // Check if we need to explore the other side
        if ((axis == 0 && abs(query.x - root->point.x) < bestDist) ||
            (axis == 1 && abs(query.y - root->point.y) < bestDist) ||
            (axis == 2 && abs(query.z - root->point.z) < bestDist)) {
            r_find_nearest(other_node, query, best, bestDist, depth + 1);
            }
    }


public:
    void build(std::vector<Point> &points) {
        root = r_build_tree(points);
    }
    struct NearResult {
        Point best_point;
        LD best_dist{};
    };
    [[nodiscard]] NearResult find_nearest(const Point &query) const {
        LD best_dist{std::numeric_limits<LD>::infinity()};
        KDNode *best_node=nullptr;
        r_find_nearest(root, query, best_node, best_dist, 0);
        if(best_node!=nullptr) {
            return {best_node->point, best_dist};
        }
        return {Point(0,0,0),-1};
    }
};
#endif //KD_TREE_H
```