Sifting Through the Noise: Finding the Stochastic Gravitational Wave Background from Virgo

Marie Olivia Sykes
Ursinus College

Giancarlo Cella
INFN-Pisa

(Virgo Collaboration)

(Dated: 7 August 2023)

The primordial section of the Stochastic Gravitational Wave Background has the possibility of being detected within the kilohertz detection band when taking into account redshift from Inflationary effects. Even still, sensitivity and noise levels on detectors like the Virgo Interferometer limit the ability to perceive this. Using Kurtosis as a part of Independent Component Analysis when analyzing the detection readings as well as other channels, Gaussian sources of noise can be limited and discarded.

I. INTRODUCTION

Gravitational waves were first predicted by Einstein in 1916 as an implication of General Relativity occurring when two massive objects interact with each other [1]. Major detections typically consist of two objects, such as black hole and/or neutron star interactions, including collisions and mergers. These interactions cause ripples across the fabric of space-time that are large enough to be detected [2].

The first attempt to detect gravitational waves was the Weber Bar, which published results in 1969 though the results were never corroborated [3]. Research first began proposing interferometers with arms in the kilometer-scale in the 1970s and LIGO first opened in 1999 before later joining LIGO-Virgo in 2007 [4]. KAGRA, a Japanese gravitational wave detector collaboration, joined LIGO-Virgo and created LIGO-Virgo-KAGRA (LVK) in October 2019. [5].

In 2015, the first gravitational wave was detected by LIGO-Virgo, proving Einstein's theory nearly one hundred years later [4]. Gravitational waves provide an alternate source of information that studies relying on electromagnetic radiation or subatomic particles cannot provide. They barely interact with matter allowing them to be relatively unaffected by their path across the universe, and eminate from various sources. Some sources, such as a binary black hole merger, would otherwise be undetectable [6]. Electromagnetic counterparts to gravitational wave detection have been sought, but as the detectors cannot determine a small enough area of the sky to search for a possible pairing, nothing has been proven so far.

A. Virgo Interferometer

FIG. 1: An aerial view of the Virgo Interferometer. Credit to Nicola Baldocchi/EGO

The Virgo Interferometer is located at the European Gravitational Observatory in

Cascina, Italy, just outside of Pisa in the middle of farmland.The interferometer first began construction in 1997 and began observations in 2007, forming LIGO-Virgo. The Advanced Virgo upgrade was completed in 2017 and made its first detection in August 2017, two into the second LIGO-Virgo observing run. The interferometer consists of two three kilometer arms pointed north and west. The blue tunnels seen in the figure, painted the colour so they do not disturb the landscape, encases a smaller tunnel and serves as a protective barrier [7]. The detector is sensitive to waves between a few Hertz well into the thousands of kilohertz range (kHz) [8].

After the laser is produced at the square building between the two tunnels, it is split by the beam-splitter, and the light travels down both tunnels until it reaches a mirror held at the end where it returns to the center. The return beams then interact and cause an interference pattern. In theory, if there is no detection, the lasers should take an equal amount of time to return and cancel out. If there is a detection, the space the laser travels is stretched and the return time is altered. Since there is noise that interferes with the laser's pattern, this is not exactly how it plays out [7].

The detector reading composed of the difference in the input and output channels can be described as

$$s_i = h_i + n_i \tag{1}$$

where time is the lower index i, h_i represents the *detection* signal, and n_i represents the *noise* interference.

As the interferometer detections are composed of noise, to tell whether or not a gravitational wave has arrived, the signal must be stronger than the noise or one has to be able to reduce the noise in the system. In order to improve the detections of any type of gravitational wave, reducing the noise is essential.

Over the summer, I became a Virgo collaborator and attended the summer's quarterly meeting. Virgo Week consisted of reports from the various committees and analysis groups, research updates, and communications

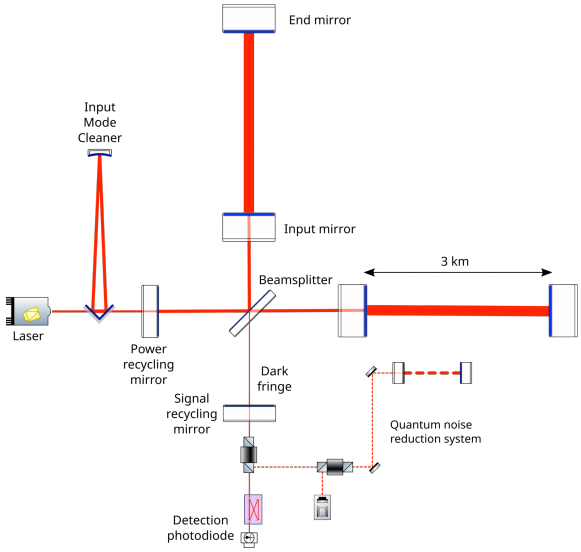

FIG. 2: A diagram of the Virgo Interferometer.

from the Virgo Early Career Scientists (VECS) group. One night VECS organized a social dinner for the attendees. VECS included in their announcements the Women of Virgo Project, an outreach and visibility project interviewing women about their work in science, and I volunteered to be an interviewee and interviewer for their project. These interviews are hosted on the LIGO-Virgo Instagram in the form of reels and posts. This week allowed me to see how collaborations work on such a large scale, pulling various research groups and specialities together to pursue a common goal. One interesting thing a researcher there told me was that there were flowers growing on the field between the arms but they had to cut them down because the wind passing through their leaves caused too much noise, which is also why there aren't any trees in the clearing around Virgo.

The LIGO-Virgo-KAGRA Observing Run 4 began in May 2023. KAGRA observed for the first few weeks of O4, though Virgo has yet to join. Virgo aims to join O4 later this year after initially not joining due to a faulty mirror. The detector is being calibrated with hopes of achieving O3 sensitivity.

B. Stochastic Gravitational Wave Background

The Stochastic Gravitational Wave Background (SGWB) is composed of various sources of gravitational waves across a variety of wavelengths, such as the ongoing ripples produced by the beginning of compact binary interactions as well as primordial gravitational waves, the latter half of which has yet to be seen. If detected, they could provide insight further into the early universe than otherwise seen [2]. Some studies predict the primordial sector to exist somewhere above 10^{-15} though others predict that they may be detectable by LVK when accounting for redshift. Primordial waves could be detectable at wavelengths of up to 100 Hertz [9–11].

In June 2023, the SGWB was shown to have strong evidence for elements existing within the nanoHertz range with origins from supermassive black hole collisions from galaxy mergers after the International Pulsar Timing Array (IPTA) coordinated to arrange a release of their fifteen year datasets. The NANOGrav dataset strongly suggests evidence up to 3σ for the gravitational-wave background. Their report analyzed signals from 67 pulsars using a power-law spectrum and a Bayes factor over 10^{14}. As the power law posterior seen in their Figure 1a does not quite fit to their observations, this could indicate that there might be something else, such as primordial waves, alongside the waves from compact binary systems [12].

Traditionally the stochastic gravitational wave background searched for using a mixture of Bayesian statistics but a filter must be used to separate the astrophysical foreground where compact binaries, such as what the IPTA has detected, obscure the much smaller primordial waves. In a 2020 report, one team developed a method to filter out this larger non-Gaussian foreground while keeping the stochastic background [9].

II. METHODS

To eliminate Gaussian sources of noise, Independent Component Analysis (ICA) was employed using Kurtosis statistical analysis. This methodology can be employed to find linear dependency between different channels of information, both interferometer and other, to eliminate noise and predict glitches. This project primarily focuses on its ability to reduce noise to notice SGWB in the detector channels.

A. Kurtosis and Independent Component Analysis

Independent Component Analysis (ICA) is a type of blind source data analysis which aims to extract the useful underlying data from a wide array of information using various statistical techniques to separate the noise from the source signal, even with little knowledge of the actual source [13].

ICA operates by mixing several channels in order to find a statistically independent signal inside [13]. This could be especially useful to find some type of non-Gaussian disturbance within the Virgo detectors.

In order to optimize Kurtosis for the channels, I found all the possible values of the fourth and second momenta with the goal of minimizing and maximizing the Kurtosis Value.

Kurtosis is a form of higher-order statistics used commonly in ICA. Kurtosis can be used to measure non-Gaussianity using a weight vector multiplied against the applicable channel values. The basis of the analysis consists of

$$K = \frac{S_4}{S_2^2} \tag{2}$$

$$S_4 = w_i w_j w_k w_l \Sigma s_i s_j s_k s_l \tag{3}$$

$$S_2 = w_i w_j \Sigma s_i s_j \tag{4}$$

$$\tag{5}$$

and follows these conditions

$$\vec{W} = X_1 + X_@ + X_3 + ... + X_D \quad (6)$$
$$X_1 = \cos(\Theta_1) \quad (7)$$
$$X_2 = \sin(\Theta_1)\cos(\Theta_2) \quad (8)$$
$$X_3 = \sin(\Theta_1)\sin(\Theta_2)\cos(\Theta_3) \quad (9)$$
$$X_D = \sin(\Theta_1)\sin(\Theta_2)\sin(\Theta_3)...\cos(\Theta_D) \quad (10)$$
$$1 = X_1^2 + X_2^2 + X_3^2 + ... + X_D^2 \quad (11)$$

where Θ_1 through Θ_{D-1} can be any value between 0 and Π and Θ_D ranges between 0 and 2Π [14].

B. The Code

The goal of the code is to interpret multiple arrays each referred to as "channels" with a fixed set of elements or datapoints in them. Each of these channels represent one detection channel or reading from the Virgo Interferometer, such as the laser input, laser output, magnometer, and seisometer.

The code has the ability to read csv files, though it can also generate simulated data directly from the project itself. This can all be seen in the Appendix A.

The code keeps a dedicated logfile documenting the values produced in the dataset.

As seen by equations 6 through 11, each of the components of each weight contains various permutations of theta values. The significant part when calculating the weight value is to ensure that

To calculate the Kurtosis, the code takes the "correlation" matrix created with the stored values of the various combinations of the channel values and compares those values with the various weight values. Though C++ cannot naturally perform recursive measures, something simulating was implemented to achieve this effect.

For example, if there were three possible channels, the correlation matrix for S_2 and S_4 would hold all the values making up each combination of the numbers one through three in the placement of i, j, k, and l.

1. The Simplified Code

In a simplified version of the code, data simulation consisted of using set of weights, a default standard deviation (sigma) value of 1.0, and a random generation key.

Map was used to instead of using a recursive mimic to find all the possible permutations with however many channels to calculate the second and fourth momentas for Kurtosis. Kurtosis was generated using one angle in this version and no recursive effects.

Only two and three channels were used in the simplified version of the code, each with an artificial weight the goal was to recover. In order to settle on the weights, a process called Annealing was employed with increment values of 1000, 2500, and 5000 which determined how many times it could adjust

Two .csv files were created from this process for the annealing process as well as Kurtosis. The Kurtosis file paired theta between 0 and 2Π (incrementing by .1) with the Kurtosis value at that specific theta. The annealing file should have documented the predicted weights, though it seems it assumed there should have been three weights with inputs of two.

Running the code once generated three trials (trials of increments of 1000, 2500, 5000) with ten different sets of weights.

A .txt file documented the weights used, increments, sigmas, accompanied each trial.

C. Channel Data and Simulation

The model for the data we explored is very simple. We suppose there are several time series, which can be written as

$$s_i^A = w^A d_i + n_i^A \quad (12)$$

where n_i^A is Gaussian noise, d_i a non Gaussian disturbance and w^A are *weights* which parameterize the coupling of the disturbance to each channel.

D. C++ and Coding Supplements

C++ was utilized to build the project for its object-oriented focus. The GNU Scientific Library (GSL) was employed for its features regarding vector and matrix analysis,

To learn C++ during this project, I used Chat GPT to teach me how to build sections of the code and to quickly navigate the GSL documentation. It also could help in debugging sections of the code and finding solutions to error messages. The AI was not always able to complete the task at hand and provided faulty suggestions, but it generally taught me enough.

In edition to C++, Mathematica was employed to provide examples and outlines of the future coding project. Working with a symbolic computation language provided a simple approach to experimenting with the procedure before integrating the different parts into one coding projects. Signal and noise separation and probability division techniques were two particular focuses of the Mathematica code. Additionally, Mathematica was used to generate the final graphs and to generate data for the simplified version of the code.

III. RESULTS

The main research conducted this summer was the creation of the toolset and development. Preliminary results from the simplified code are as follows.

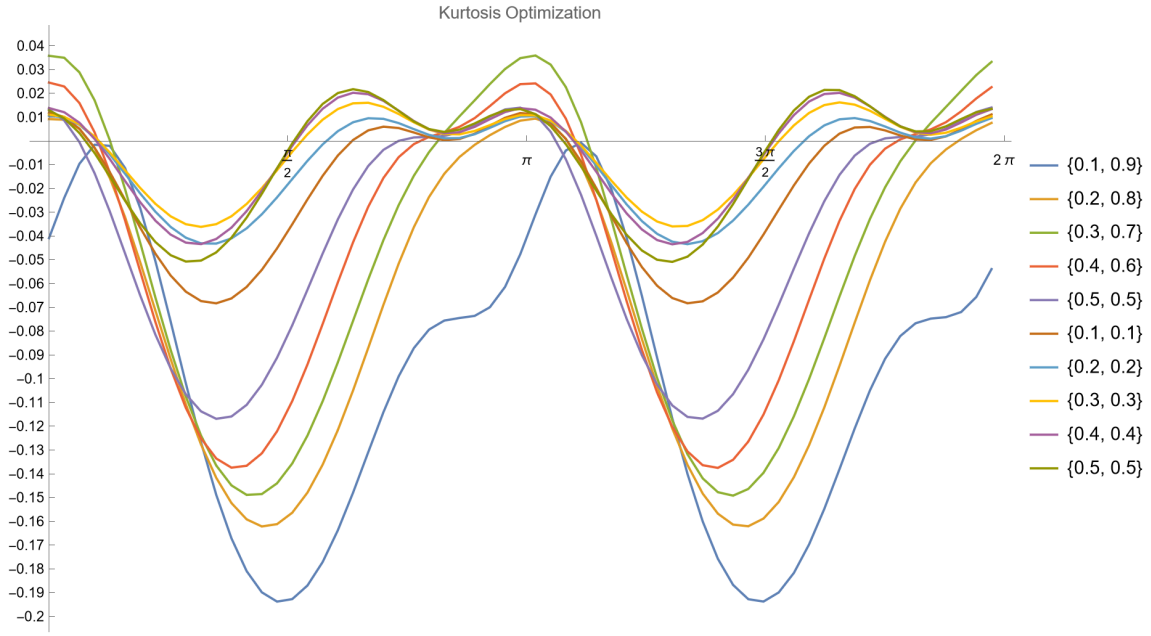

FIG. 3: A collection of the different weight values in response to using increments of 1000

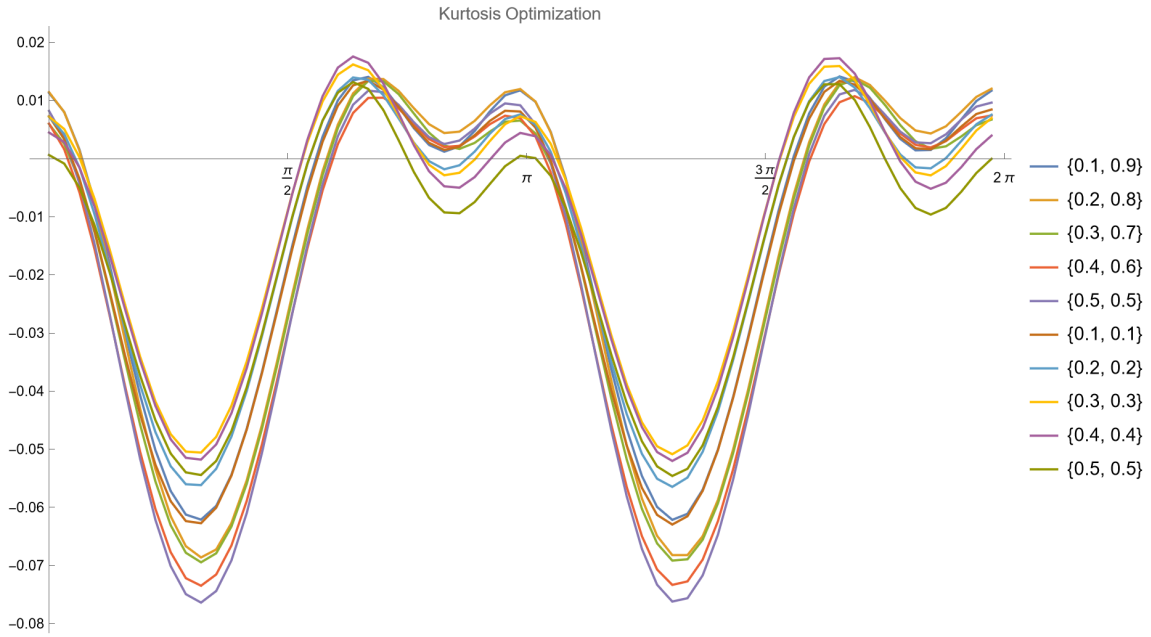

FIG. 4: A collection of the different weight values in response to using increments of 2500

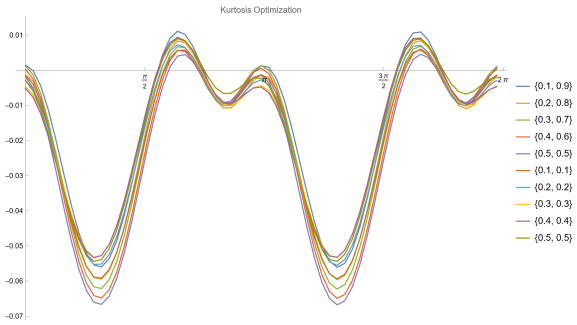

FIG. 5: A collection of the different weight values in response to using increments of 5000

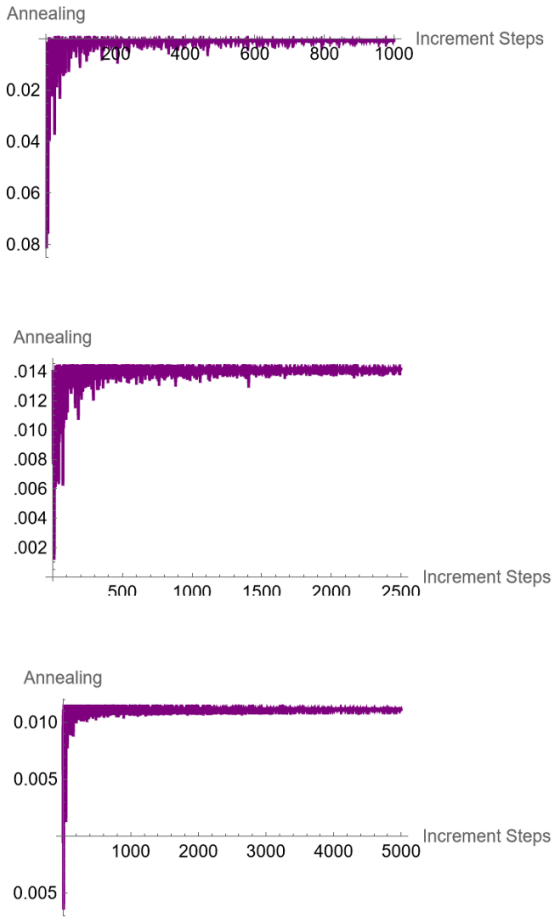

FIG. 6: The difference between the annealing certainty in 1000, 2500, and 5000 increment steps

IV. ANALYSIS

The Kurtosis graphs demonstrated a clear show of Gaussianity and Non-Gaussianity providing the ability to separate a Non-Gaussian signal in further analysis.

The Annealing plots represent the eventual narrowing of the process of determining the weights inputted into the system. When analyzing increments of 1000, the graph is less sure than with increments of 5000. 1000 is significantly less sure than 2500, but the difference between 2500 and 5000 is not extreme, as seen by 6. Optimization is key as to not spend more resources on what will not actually affect the experiment.

The code needs to be further refined to demonstrate the results of predicting the weight values.

V. CONCLUSION

Overall, the data simulation seems to be effective though the code needs further development in order to demonstrate the proof of concept. When optimizing for the annealing process, it seems like 2500 steps is close to the ideal amount of steps. The annealing process seems to be promising in evaluating the weighted values, which will be significant when describing non-simulated situations.

VI. FUTURE WORK

To further improve the code, I would adjust the wave functions to have the expected signal to noise ratio. Currently the stochastic signal is too loud for what is expected. If this technique can show a much smaller signal can still be seen, then it could be easier to translate into real detection analysis.

I would eliminate the option to create multiple runs at once of the data to streamline the process. I may or may not keep data simulation within Mathematica. It was a simple enough process, but if the code can support the generation of data, that would be preferred.

One major addition I would make is rewrite the input channels to read from a .txt file a list of inputs such as whether it is simulating data directly or reading a .csv file. This will simplify the process.

I will also investigate whether a Monte Carlo simulation would be more effective in calculating the weights rather than Annealing.

Additionally, this current model can be applied to find glitches in the detectors.

One bug I found in the simplified code at the end related to the code not being able to properly read the length of an array, as seen by the .txt file for annealing printing over three spaces instead of two, and it would be worthwhile to ensure this does not affect the other sections of the code that depend on evaluating for a length of two or three.

Further testing of this analysis to ensure the results are correct is required as well as including a way to account for non-Gaussian noise.

ACKNOWLEDGMENTS

I would to thank Dr. Paul Fulda, Dr. Peter Wass, Dr. Kathryn McGill, and Dr. Nathaniel Strauss for providing a space to research gravitational waves and travel abroad via the University of Florida's International Research Experience for Undergraduates program as well as the University of Florida Physics Department itself. I would also like to thank my mentor, Dr. Giancarlo Cella, for advising me over the summer and teaching me about noise analysis research and the Virgo Collaboration. His guidance over the summer has taught me so much about the field and I look forward to continuing this work. Thank you to NSF grant PHY-1950830 for funding this research and my summer abroad.

Stefano Rinaldi as well as many other graduate and masters students at the University of Pisa were vital to my success here as a researcher but also to thriving in Pisa as well. They have been great friends and have helped

Phia and I learn all about the city. They made our summer here truly memorable.

I would like to thank the Virgo Collaboration, The University of Pisa, and INFN-Pisa for hosting me over the summer. Thank you to Dr. Jon Ward (UF IREU alum '12) and Dr. Tom Carroll for introducing me to the program, especially to Tom for encouraging me to further my General Relativity knowledge and to pursue it as a research topic. Thanks as well to the rest of the Ursinus College Physics and Astronomy Department for supporting me over the years and forming my research skills. I would also like to thank my parents, brother, and cat, for all their support. To the other IREU students, I have loved getting to hear about everyone's adventures and misadventures and supporting each other along the way.

Finally I would like to thank Phia Morton, the IREU student I spent my summer with. I loved spending my summer with you and traveling across Italy. We certainly had quite the adventure.

Appendix A: Appendix A: Code

This coding project included both header files and source files. For the purposes of condensing the information, only source files are included in the appendix.

1. Main()

```
int main() {
    string logName;

    cout << "Kurtosis Analysis Initialized" << endl;
    cout << endl << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

    cout << "What Would You Like to Analyze?" << endl;
    cout << "   0: Simulated Data" << endl;
    cout << "   1: Real Data" << endl;
    input = getUserInput(valid1);

    int numChannels = 0;
    int numData = 0;

    if (input == 0) {
        cout << "Initializing Data Simulation" << endl << endl;

        logName = beginLog(); //CHECK make sure this works

        vector<gsl_matrix> runCollection;
        runCollection = Data::runSimulation(numChannels, numData);
    }

    else {
        cout << "Initializing Data Analysis" << endl << endl;

        logName = beginLog(); //CHECK make sure this works

        cout << "DATA UPLOAD" << endl << endl;

        // CHECK so this needs to get fixed before I put it through
        // readColumnsFromASCII(filename, startColumn, endColumn);

        // real data needs to go in here

        // make sure this has numChannels and numData edited somewher

    }

    // CHECK so I'm just making a matrix to practice running things through
    // so this needs to be updated so that way things go into a matrix
    gsl_matrix* testMatrix = gsl_matrix_alloc(numChannels, numData);

    // CHECK make sure everything necessary was written to the log

    string anyKey;
    cout << "Select Any Key to Continue to Kurtosis: " << endl;
    cin >> anyKey; // I think this will just look for some input a3nd will accept it
    cout << endl;

// KURTOSIS

    // GRIDS
        cout << "Finding the Probability of a Point in a Location" << endl;
        cout << "Select Grid Type: " << endl;
        cout << "   0: Sphere" << endl;
        cout << "   1: Square" << endl;
        cout << "   2: Weighted Rectangles" << endl;
        input = getUserInput(valid2);

        vector<string> probabilityMenu = {"Sphere", "Square", "Weighted Rectangles"};
        cout << "Sorting the Probability By " << probabilityMenu(input) << endl;


        probability(SSMATRIX CHECK, double x, double y, double STEP, int gridType);

        double STEP;
        cout << "Input Step Size" << endl;
        STEP = getValidDoubleInput();

        sortPointsIntoGrid(&dataView.matrix, STEP);

        // Calculate probability for a specific point
        double x = 2.3; // Change these values as per your requirement
        double y = 3.2;
        double probability = calculateProbability(&dataView.matrix, x, y, STEP, false); // Use regul
        cout << "Probability using regular grid: " << probability << endl;

        probability = calculateProbability(&dataView.matrix, x, y, STEP, true); // Use weighted grid
        cout << "Probability using weighted grid: " << probability << endl;

    // ANALYSIS
        gsl_matrix* weights = gsl_matrix_alloc(numChannels, numData);
        *weights = *DSphere::Weights(); //CHECK maybe this needs to iterate through every run
       // gsl_matrix**** weights4D = gsl_matrix_alloc(numChannels, numData);
       // *weights = *DSphere::Weights(); //CHECK maybe this needs to iterate through every run
        // CHECK do the math for weights
        double kurtosisValue = 0.0;

       // for (int i = 0; i < NUM OF RUNS; i++);

        kurtosisValue = kurtosisS4S2(weights, testMatrix);
        //CHECK idk what this returns at the moment

// GRAPHS

    cout << "Would You Like to Display a Graph?" << endl;
    cout << "   0: Create Graphs" << endl;
    cout << "   1: Quit the Program" << endl;
    input = getUserInput(valid1);

    if (input == 0) {
        // graphs
        cout << endl;
    }

/* BEYOND GRAPHS */

    cout << "Would You Like to Do Something Else?" << endl;
    cout << "   0: Else" << endl;
    cout << "   1: Quit the Program" << endl;
    input = getUserInput(valid1);

    if (input == 0) { //CHECK this was formatting a little funky there might be some semicolon somewh
        //CHECK what else would I want this to do
        cout << endl;
```

```
    }

    cout << "program successfully printed to " << logName << endl;

    closeLog();
    return 0;
}
```

2. evalKurtosis

```
gsl_matrix* corMatrix2D(gsl_matrix* runMatrix) { // make correlation matrix
 // INPUT:   a matrix run of channels, num channels
 // OUTPUT: Correlation Matrix
 // for 2 Dimensions

 numChannels = runMatrix->size2;
 channelLength = runMatrix->size1;
 N = static_cast<size_t>(numChannels);

 correlation2D = gsl_matrix_alloc(N, N);
 // i1 and i2 will always be between [0,numChannels)

 for (size_t i = 0; i < N; i++) { // channel i
 element1 = 0.0;
 element2 = 0.0;
 mult = 0.0;
 value = 0.0;

 for (size_t j = i; j < N; j++) { // channel j
 for (size_t k = 0;  k < channelLength; k++) {// dot each component
 element1 = gsl_matrix_get(runMatrix, i, k);
 element2 = gsl_matrix_get(runMatrix, j, k);
 mult += element1 * element2;
 }

 value = (1.0 / N) * mult;
 gsl_matrix_set(correlation2D, i, j, value);
 gsl_matrix_set(correlation2D, j, i, value);
 }
 }

 return correlation2D;

}

gsl_matrix**** corMatrix4D(gsl_matrix* runMatrix) {
 // INPUT:   a matrix run of channels, num channels
 // OUTPUT: Correlation Matrix
 // for 4 Dimensions

 numChannels = runMatrix->size2;
 channelLength = runMatrix->size1;
 N = static_cast<size_t>(numChannels);

 ****correlation4D = ****create4DMatrix(N, N, N, N);

 for (size_t i = 0; i < N; i++) { // channel i
 element1 = 0.0;
 element2 = 0.0;
 element3 = 0.0;
 element4 = 0.0;
 mult = 0.0;
 value = 0.0;

 for (size_t j = i; j < N; j++) { // channel j
 for (size_t k = j; k < N; k++) { // channel k
 for (size_t l = k; l < N; l++) { // channel l
 for (size_t m = 0; m < channelLength; m++) {// dot each component
 element1 = gsl_matrix_get(runMatrix, i, m);
 element2 = gsl_matrix_get(runMatrix, j, m);
 element3 = gsl_matrix_get(runMatrix, k, m);
 element4 = gsl_matrix_get(runMatrix, l, m);
 mult += element1 * element2 * element3 * element4;

 value = (1.0 / N) * mult;

 set4DMatrixValue(correlation4D, i, j, k, l, value);
 set4DMatrixValue(correlation4D, i, l, k, j, value);
 set4DMatrixValue(correlation4D, j, i, k, l, value);
 set4DMatrixValue(correlation4D, j, l, k, i, value);
 set4DMatrixValue(correlation4D, k, i, j, l, value);
 set4DMatrixValue(correlation4D, k, j, i, l, value);
 set4DMatrixValue(correlation4D, l, i, j, k, value);
 set4DMatrixValue(correlation4D, l, j, i, k, value);
 }
 }
 }
 }

 return correlation4D;
}

double expS2(gsl_matrix* weights, gsl_matrix* correlation) { // Exp[s^2] = WiWjCij
 // INPUT:   weights, correlation matrix
 // OUTPUT: Expectation for S^2
 // CHECK weights needs to be a solid input

 value = 0.0;

 N = weights->size2; // weights and correlation should be the same size and the num of channels

 for (size_t i = 0; i < N; i++) { // column
 for (size_t j = 0; j < N; j++) { // row
 value += gsl_matrix_get(weights, i, j) * gsl_matrix_get(correlation, i, j);
 }
 }
 eS2 = value;

 return eS2;
}

double expS4(gsl_matrix**** weights, gsl_matrix**** correlation) { // Exp[s^2] = WiWjCij
 // INPUT:   weights, correlation matrix
 // OUTPUT: Expectation for S^4

 //N = weights->size2;

 for (size_t i = 0; i < N; i++) { // i1
 for (size_t j = 0; j < N; j++) { // i2
 for (size_t k = 0; k < N; k++) { // i3
 for (size_t l = 0; l < N; l++) { // i4
 value += get4DMatrixElement(weights, i, j, k, l) * get4DMatrixElement(correlation, i, j, k, l);
 }
 }
 }
 }
 eS4 = value;

 return eS4;
}

double kurtosisS4S2(gsl_matrix* weights, gsl_matrix* runMatrix) {

 numChannels = runMatrix->size2;
 channelLength = runMatrix->size1;
 N = static_cast<size_t>(numChannels);

 // CHECK convert weights to 2D and 4D

 correlation2D = gsl_matrix_alloc(N, N); // CHECK should I be defining the same matrix in two differ
 correlation4D = create4DMatrix(N, N, N, N); // CHECK this could be more efficient

 weights2D = gsl_matrix_alloc(N, N);
 weights4D = create4DMatrix(N, N, N, N);

 *correlation2D = *corMatrix2D(runMatrix);
 *correlation4D = *corMatrix4D(runMatrix);

 s2 = expS2(weights, correlation2D);
 s4 = expS4(weights4D, correlation4D);

 kurtosis = s4 / (pow(s2, 2)); // CHECK do we want the -3 or not
 // CHECK either way we need to demonstrate the max and min
 // CHECK so we're evaluating kurtosis for the various linear combinations so we get that angle graph
 // correlation is already calculated once and we're set there
 // weights will vary and then we'll get a function in main that we can graph


 // CHECK print something to display and log
 // CHECK this needs to return something
 return kurtosis;
}
```

3. linearCombination

```
vector<gsl_vector*> matrixToNestedVectors(gsl_matrix* runMatrix) {
 int chanSize = runMatrix->size2;
 int chanLength = runMatrix->size1;
 double valueV;
 vector<gsl_vector*> runNestedVector;
```

```
 for (int i = 0; i < chanSize; i++) {
 gsl_vector* channelVector = gsl_vector_alloc(chanLength); // Allocate a new vector for each channel
 for (int j = 0; j < chanLength; j++) { // Corrected the loop counter from i to j
 valueV = gsl_matrix_get(runMatrix, i, j);
 gsl_vector_set(channelVector, j, valueV); // Set the value directly, don't assign the result
 }
 runNestedVector.push_back(channelVector);
 }

 return runNestedVector;
}

// Recursive function for nested loops with D dimensions
template <size_t D>
void nestedLoop(std::vector<int>& indices, const std::vector<int>& loopLimits) {
 if (D == 0) {
 // Base case: Print the indices when D becomes zero (all nested loops are unrolled)
 for (size_t i = 0; i < indices.size(); ++i) {
 std::cout << indices[i] << " ";
 }
 std::cout << std::endl;
 }
 else {
 // Recursive case: Loop through all indices of the D-th dimension
 for (int i = 0; i < loopLimits[D - 1]; ++i) {
 indices[D - 1] = i;
 nestedLoop<D - 1>(indices, loopLimits); // Recursive call for the next dimension
 }
 }
}
```

4. dataGenV3

```
Data::Data() { /* constructor */
    mean = 0.0;
    stddev = 1.0;
    LIO = true;
    other = false;
    // do I need to have runMatrix constructed over here? built a prototype

    nameOfRun = "basic name";
}

Data::~Data() {  /* deconstructor */
    nameOfRun = " ";
}

void Data::Channel::setName(const string& name)
{
    channelName = name;
}

void Data::Channel::Display() {
    cout << "Channel Name: " << channelName << endl;
    cout << "Data Type: ";
    if (detectionChan /*true*/) {
        cout << "Laser Input/Output" << endl;
    }
    else {
        cout << "Noise" << endl;
    }
    /* cout << "Channel Data: " << channelName << endl; */

    printGSLVector(channelVector);

    cout << endl;

}

void Data::dataGen(Channel& chanName, int n) { /* call twice for channels for in/out the range */
    /* n = size of array */

    chanName.channelVector = gsl_vector_alloc(n);
    chanName.detectionChan = true;
    chanName.dataPoints = n;

    gsl_rng_env_setup();
    gslRng = gsl_rng_alloc(gsl_rng_mt19937);
    gsl_rng_set(gslRng, 5); /* this was gen() */

    for (int i = 0; i < n; ++i) {
        val = gsl_ran_flat(gslRng, -1.0, 1.0);
        sample = sin(val);
        noise = gsl_ran_gaussian(gslRng, stddev) + mean;
        sValue = sample + noise;
        gsl_vector_set(chanName.channelVector, i, sValue);
    }
    chanName.Display();
}

void Data::gaussianChannelGen(Channel& chanName, int n) { /* so this is for the non-arms */
    chanName.channelVector = gsl_vector_alloc(n);
    chanName.detectionChan = false;

    for (int i = 0; i < n; ++i) {
        noise = gsl_ran_gaussian(gslRng, stddev) + mean;
        sValue = noise;
        gsl_vector_set(chanName.channelVector, i, sValue);
    }
}


gsl_matrix* Data::dataCall(int numberChannels, int numberData) {

    /* this is what is getting called by the class instance when you generate a run in main() */

    Channel* c;
    /* generate channel 1 and 2 */
    c = new Channel;
    c->setName("L1");
    dataGen(*c, numberData);
    channels.push_back(*c);

    c = new Channel;
    c->setName("L2");
    dataGen(*c, numberData);
    channels.push_back(*c);

    channelAdditional = numberChannels - 2;
    channelNum = 0;

    for (int i = 0; i < channelAdditional; i++) {
        channelNum = i + 3;
        nameTheChannel = "Channel" + channelNum; /* why do I have too many characters in a const */
        c = new Channel;
        c->setName(nameTheChannel);
        gaussianChannelGen(*c, numberData);
    }

    runMatrix = gsl_matrix_alloc(numberChannels, numberData);
    // #C = columns; #D = rows;

    for (int i = 0; i < numberChannels; i++) { // columns
        for (int j = 0; j < numberData; i++) { //rows
            gsl_matrix_set(runMatrix, i, j, gsl_vector_get(channels[i].channelVector, j));
        }
    }

    return runMatrix;
}

vector<gsl_matrix*> Data::runSimulation(int& numChannels, int& numData) {

    cout << "SIMULATED DATA RUN1" << endl << endl;

    Data run1;
    run1.runIteration = "Run1";

    cout << "RUN 1" << endl;

    numData = 20; // default
    cout << "Input Number of Data Points: " << endl;
    numData = getPositiveIntegerInput();
    cout << "Number of Data Points: " << numData << endl;

    numChannels = 5; // default
    cout << "Input Number of Channels (including Laser Input (L1) and Laser Output (L2): " << endl;
    numChannels = getPositiveIntegerInput();
    cout << "Number of Channels: " << numChannels << endl;

    runCollection.push_back(run1.dataCall(numChannels, numData));
    //CHECK idk if this works

    runNumber = 1;

    do {
        cout << "Would You Like to Add Another Data Run?" << endl;
        cout << "   0: yes" << endl; /* false */
        cout << "   1: no" << endl; /* true */
        input = getUserInput(valid1);

// CHECK all of this needs to be fixed with the redefinitions and etc
        if (!answer) {
            runNumber += 1;
            nameOfRun = "Run" + std::to_string(runNumber);
```

```
            Data runName;
            runName.runIteration = nameOfRun;
            runCollection.push_back(runName.dataCall(numChannels, numData)); /* idk if this is a pointer right */
        }
    } while (!answer);

    return runCollection;

    summarizeSimulation(runNumber, numChannels, numData);

}



void Data::dataLog(ofstream& logFile, Data& dataInstance) {
    nameOfRun = toString(dataInstance);

    logFile << nameOfRun << ": " << endl;
    logGSLMatrix(dataInstance.runMatrix);
    logFile << endl;

    logFile << "Kurtosis Values: " << endl;
    logFile << "   L1: " << "print value" << endl;
    logFile << "   L2: " << "print value" << endl;
    for (int i = 3; i == 5; i++) { //CHECK update 5 to be numberChannels from data object
        logFile << "   C" << i << " print value" << endl;
    }
    logFile << endl;

    //as long as I use matrix get this should work

    // CHECK hopefully this works

}
```

5. dataImport

```
DataImport::DataImport() {
    cout << endl;
}

DataImport::~DataImport() {
    cout << endl;
}

vector<vector<double>> DataImport::readColumnsFromASCII(const string& filename, size_t startColumn, size_t endColumn) {

    // Open the ASCII file
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Error opening the file: " << filename << endl;
    }

    string line;
    while (getline(file, line)) {
        vector<double> rowData;
        stringstream ss(line);
        string cell;

        size_t column = 0;
        while (getline(ss, cell, ',')) {
            if (column >= startColumn && column <= endColumn) {
                double value;
                try {
                    value = stod(cell);
                }
                catch (const invalid_argument& e) {
                    cerr << "Error converting data to double: " << cell << endl;
                    value = 0.0; // Set to default value in case of conversion error
                }
                rowData.push_back(value);
            }
            ++column;
        }

        if (!rowData.empty()) {
            columns.push_back(rowData);
        }
    }

    file.close();
    return columns;
}
```

6. 4DMatrix

```
gsl_matrix**** create4DMatrix(int I1, int I2, int I3, int I4)  {
    N1 = I1;
    N2 = I2;
    N3 = I3;
    N4 = I4;

    gsl_matrix**** matrix4D = new gsl_matrix***[N1];

    for (int i = 0; i < N1; ++i) {
        matrix4D[i] = new gsl_matrix**[N2];

        for (int j = 0; j < N2; ++j) {
            matrix4D[i][j] = new gsl_matrix*[N3];

            for (int k = 0; k < N3; ++k) {
                matrix4D[i][j][k] = gsl_matrix_calloc(N4, N4);
                // Initialize the individual 2D matrices here if needed
            }
        }
    }
    return matrix4D;
}

void set4DMatrixValue(gsl_matrix**** matrix4D, int i, int j, int k, int l, double value) {
    gsl_matrix_set(matrix4D[i][j][k], l, l, value);
}

double get4DMatrixElement(gsl_matrix**** matrix4D, int i, int j, int k, int l) {
    return gsl_matrix_get(matrix4D[i][j][k], l, l);
}

void free4DMatrix(gsl_matrix**** matrix4D) {
    for (int i = 0; i < N1; ++i) {
        for (int j = 0; j < N2; ++j) {
            for (int k = 0; k < N3; ++k) {
                gsl_matrix_free(matrix4D[i][j][k]);
            }
            delete[] matrix4D[i][j];
        }
        delete[] matrix4D[i];
    }
    delete[] matrix4D;
}
```

```
void DataImport::Properties() {
    cout << "Input File Name: " << endl;
    cin >> fileName;
    cout << "Start Column: " << endl;
    cin >> startColumn;
    cout << "End Column: " << endl;
    cin >> endColumn;

    // possibly set for rows. this code automatically reads the entire column
    // maybe make the header for the column the name of the channel. name that somewhere

    cout << endl;

    vector<vector<double>> columns = readColumnsFromASCII(fileName, startColumn, endColumn);

    // Print the extracted data
    for (size_t i = 0; i < columns.size(); ++i) {
        for (size_t j = 0; j < columns[i].size(); ++j) {
            cout << columns[i][j] << "\t";
        }
        cout << endl;
    }

}
void DataImport::uploadData() {
    Properties();

    // so rn it's a double vector and like this isn't sustainable but like it'll be what we deal wit
}
```

7. formatting

```
void printGSLVector(const gsl_vector* vec) {
    cout << "[";
    for (size_t i = 0; i < vec->size; ++i) {
        cout << gsl_vector_get(vec, i);
        if (i != vec->size - 1) {
            cout << ", ";
        }
    }
    cout << "]" << endl;

}

void printGSLMatrix(const gsl_matrix* matrix) {
    if (matrix == nullptr) {
        cout << "Matrix is nullptr." << endl;
        return;
    }

    rows = matrix->size1;
    cols = matrix->size2;

    for (size_t i = 0; i < rows; ++i) {
        for (size_t j = 0; j < cols; ++j) {
            value = gsl_matrix_get(matrix, i, j);
            cout << value << " ";
        }
        cout << endl;
    }
}

int getUserInput(const vector<int>& validOptions) {
    while (true) {
        cin >> input;

        if (std::cin.fail() || std::find(validOptions.begin(), validOptions.end(), input) == validOptions.end()) {
            cout << "Input a Valid Response: \n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // Ignore any extra input
            break;
        }
    }

    return input;
}

double getValidDoubleInput() {
    while (true) {
        cin >> value;

        if (std::cin.fail()) {
            cout << "Invalid input. Please enter a valid double.\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // Ignore any extra input
            break;
        }
    }

    return value;
}

bool isPositiveInteger(int num) {
    return num > 0;
}

int getPositiveIntegerInput() {
    int num;
    cin >> num;
    while (true) {
        if (cin >> num && isPositiveInteger(num)) {
            // Input is valid, break out of the loop
            break;
        }
        // Input is invalid, clear the error flag and ignore the rest of the input
        cout << "Invalid input. Please enter a positive integer." << std::endl;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
    return num;
}

template<typename T>
string toString(const T& x)
{
    std::ostringstream ss;
    ss << x;
    return ss.str();
}
```

8. grid

```
double domainMin;
double domainMax;
double rangeMin;
double rangeMax;

double domainDiff;
double rangeDiff;
double domainStep;
double rangeStep;

gsl_vector* domainBox;
gsl_vector* rangeBox;

double x;
double y;

int column;
int row;

int numColumns;
int numRows;

vector<vector<double>> divisions;

gsl_matrix* createDataMatrix(gsl_matrix* runMatrix) {
    size_t numRows = runMatrix->size1;
    gsl_matrix* dataMatrix = gsl_matrix_alloc(numRows, 2);
    double value;
    for (int i = 0; i < numRows; i++) { // rows
        for (int j = 0; j < 2; j++) { // columns
            value = gsl_matrix_get(runMatrix, i, j);
            gsl_matrix_set(dataMatrix, i, j, value);
        }
    }
    return dataMatrix;
} // this returns a matrix with just the first two columns which should be the channels


void analyzeDomainAndRange(const gsl_matrix* dataMatrix, double& domainMin, double& domainMax, double
    // Initialize domain and range with the first data point
    domainMin = gsl_matrix_get(dataMatrix, 0, 0);
    domainMax = gsl_matrix_get(dataMatrix, 0, 0);
    rangeMin = gsl_matrix_get(dataMatrix, 0, 1);
    rangeMax = gsl_matrix_get(dataMatrix, 0, 1);

    // Loop through the data matrix to find min and max
    for (size_t i = 0; i < dataMatrix->size1; ++i) {
        double x = gsl_matrix_get(dataMatrix, i, 0);
        double y = gsl_matrix_get(dataMatrix, i, 1);

        if (x < domainMin) domainMin = x;
        if (x > domainMax) domainMax = x;
        if (y < rangeMin) rangeMin = y;
        if (y > rangeMax) rangeMax = y;
    }
}

void squareDimensions(const gsl_matrix* dataMatrix, double STEP) {
    analyzeDomainAndRange(dataMatrix, domainMin, domainMax, rangeMin, rangeMax);
    logFile << "Grid Dimensions" << endl;
    logFile << "Domain Min: " << domainMin << endl;
    logFile << "Domain Max: " << domainMax << endl;
    logFile << "Range Min: " << rangeMin << endl;
    logFile << "Range Max: " << rangeMax << endl << endl;

    // Calculate the grid dimensions
    numColumns = static_cast<int>((domainMax - domainMin) / STEP) + 1;
    numRows = static_cast<int>((rangeMax - rangeMin) / STEP) + 1;

    // Initialize the grid with zero counts
    gsl_matrix* grid = gsl_matrix_calloc(numRows, numColumns);

    // Loop through the data and populate the grid
    // size1 is the rows
    for (size_t i = 0; i < dataMatrix->size1; ++i) {
```

```
        x = gsl_matrix_get(dataMatrix, i, 0);
        y = gsl_matrix_get(dataMatrix, i, 1);

        column = static_cast<int>((x - domainMin) / STEP);
        row = static_cast<int>((y - rangeMin) / STEP);

        gsl_matrix_set(grid, row, column, gsl_matrix_get(grid, row, column) + 1);
    }

    // Output the grid
    cout << "Grid with step size " << STEP << ":\n";
    for (int row = 0; row < numRows; ++row) {
        for (int col = 0; col < numColumns; ++col) {
            cout << gsl_matrix_get(grid, row, col) << " ";
        }
        cout << "\n";
    }

    gsl_matrix_free(grid);
}

vector<vector<double>> weightedBoxDimensions(const gsl_matrix* dataMatrix, int N) {
    analyzeDomainAndRange(dataMatrix, domainMin, domainMax, rangeMin, rangeMax);

    vector<double> xPoints;
    vector<double> yPoints;

    // Extract the points' x and y coordinates from the data matrix
    for (size_t i = 0; i < dataMatrix->size1; ++i) {
        double x = gsl_matrix_get(dataMatrix, i, 0);
        double y = gsl_matrix_get(dataMatrix, i, 1);
        xPoints.push_back(x);
        yPoints.push_back(y);
    }

    // Sort the x and y coordinates
    std::sort(xPoints.begin(), xPoints.end());
    std::sort(yPoints.begin(), yPoints.end());

    // Calculate the boundaries for each division
    vector<vector<double>> divisions(N + 1, vector<double>(2, 0.0));

    for (int i = 0; i <= N; ++i) {
        int index = static_cast<int>((i * dataMatrix->size1) / N);
        if (index >= dataMatrix->size1) {
            index = dataMatrix->size1 - 1;
        }

        divisions[i][0] = xPoints[index];
        divisions[i][1] = yPoints[index];
    }

    return divisions;
}

double probability(const gsl_matrix* dataMatrix, double x, double y, double STEP, int gridType) {
    analyzeDomainAndRange(dataMatrix, domainMin, domainMax, rangeMin, rangeMax);

    if (gridType == 0) {
        cout << "sphere" << endl;
    }
    else if (gridType == 1) {
        double N = (domainMax - domainMin) / STEP;
        divisions = weightedBoxDimensions(dataMatrix, N);
        numColumns = divisions.size() - 1;
        numRows = divisions[0].size() - 1;
        // ... Additional logic to set columns and rows based on divisions ...
        // CHECK

    } else {
        numColumns = static_cast<int>((domainMax - domainMin) / STEP) + 1;
        numRows = static_cast<int>((rangeMax - rangeMin) / STEP) + 1;
    }

    column = static_cast<int>((x - domainMin) / STEP);
    row = static_cast<int>((y - rangeMin) / STEP);

    int totalCount = dataMatrix->size1;
    int boxCount = gsl_matrix_get(dataMatrix, row, column);

    return static_cast<double>(boxCount) / static_cast<double>(totalCount);
}

void printProbabilityMatrix(const gsl_matrix* dataMatrix, double STEP, int gridType) {
    analyzeDomainAndRange(dataMatrix, domainMin, domainMax, rangeMin, rangeMax);

    if (gridType == 0) {
        cout << "sphere" << endl;
    }
    else if(gridType == 1) {
        divisions = weightedBoxDimensions(dataMatrix, 10);
        numColumns = divisions.size() - 1;
        numRows = divisions.size() - 1;
        // ... Additional logic to set columns and rows based on divisions ...'
        // CHECK fix the 10
    }
    else {
        numColumns = static_cast<int>((domainMax - domainMin) / STEP) + 1;
        numRows = static_cast<int>((rangeMax - rangeMin) / STEP) + 1;
    }

    gsl_matrix* grid = gsl_matrix_calloc(numRows, numColumns);

    int totalCount = dataMatrix->size1;

    for (size_t i = 0; i < dataMatrix->size1; ++i) {
        x = gsl_matrix_get(dataMatrix, i, 0);
        y = gsl_matrix_get(dataMatrix, i, 1);

        column = static_cast<int>((x - domainMin) / STEP);
        row = static_cast<int>((y - rangeMin) / STEP);

        gsl_matrix_set(grid, row, column, gsl_matrix_get(grid, row, column) + 1);
    }

    // Normalize the grid by dividing the counts by the total count
    gsl_matrix_scale(grid, 1.0 / totalCount);

    // Print the probability matrix
    cout << "Probability Matrix with step size " << STEP << ":\n";
    for (int row = 0; row < numRows; ++row) {
        for (int col = 0; col < numColumns; ++col) {
            cout << gsl_matrix_get(grid, row, col) << " ";
        }
        cout << "\n";
    }

    gsl_matrix_free(grid);
}
```

9. graph

```
void S1vS2(const gsl_vector* s1, const gsl_vector* s2) {
    if (s1->size != s2->size) {
        cerr << "Error: Vectors should have the same size for scatter plotting." << endl;
        return;
    }

    Gnuplot gp;
    gp << "plot '-' with points title 'Scatter Plot'" << std::endl;
    for (int i = 0; i < s1->size; ++i) {
        gp << gsl_vector_get(s1, i) << " " << gsl_vector_get(s2, i) << endl;
    }
    gp << "e" << endl;
}

// CHECK Linear Combination???

void anglevKurtosis() {
 // all the different angles and show Kurtosis Value

 cout << endl;
}

void plotChannels(Gnuplot& gp, const gsl_vector** vectors, int n, double xmin, double xmax, bool aut
    double ymin;
    double ymax;
    double value;

    if (autoRange) {
        for (int i = 0; i < n; ++i) {
            ymin = gsl_vector_min(vectors[i]);
            ymax = gsl_vector_max(vectors[i]);
            if (i == 0) {
                ymin = ymax = gsl_vector_get(vectors[i], 0);
            }

            for (int j = 0; j < vectors[i]->size; ++j) {
                value = gsl_vector_get(vectors[i], j);
                if (value < ymin) ymin = value;
                if (value > ymax) ymax = value;
            }

            if (i == 0) {
                gp << "set yrange [" << ymin << ":" << ymax << "]\n";
            }
            else {
```

```
            gp << "set yrange [" << (ymin < gp.get_yrange()[0] ? ymin : gp.get_yrange()[0]) CHECK
                << ":" << (ymax > gp.get_yrange()[1] ? ymax : gp.get_yrange()[1]) << "]\n";
        }
    }
    else {
        gp << "set xrange [" << xmin << ":" << xmax << "]\n";
    }

    gp << "plot ";
    for (int i = 0; i < n; ++i) {
        gp << "'-' with lines title 'Vector " << i + 1 << "'";
        if (i < n - 1) {
            gp << ", ";
        }
    }
    gp << endl;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < vectors[i]->size; ++j) {
            gp << j << " " << gsl_vector_get(vectors[i], j) << endl;
        }
        gp << "e" << endl;
    }
}
```

10. log

```
string beginLog() {
    cout << "Input Logfile Name: " << endl;
    cin >> logName;
    logName = logName + ".txt";
    cout << "Logfile Name: " << logName << endl;
    cout << endl;

    logFile.open(logName);
    // include 'app' to append to the file (but I want a new one each time)
    /* ofstream and ios are std */
    /* ios means input/output stream */

    if (!logFile.is_open()) {
        cerr << "Error opening log file." << endl; // also cerr
    }
    return logName;
}

void channelLog() {

    logFile << "Channel Summary" << endl;

    // print out the numbers generated for the log

}

void runLog() {


    logFile << "Kurtosis Analysis: " << endl;
    logFile << "   CHECK print matrix: " << endl;
    logFile << endl;

}

void simulationLog(int numRuns) {
    if (logFile.is_open()) {
        logFile << "Simulation Summary" << endl;

        // replace int numRuns with the data structure I would pass in
        // add a way to count the runs and then the columns/rows in each matrix
        // get rid of void
        for (int i = 0; i < numRuns; i++) {
            logFile << "Print for Each Run like Run: num" << endl;
            // call runLog and channelLog
        }
    }
}

void graphLog() {
    if (logFile.is_open()) {
        logFile << "print graphs or maybe just the type of graphs generated " << endl;
    }
}


void summarizeSimulation(int runNumber, int numChannels, int numData) {
    if (logFile.is_open()) {
        cout << "Data Summary: " << endl;
        cout << "   Number of Runs: " << runNumber << endl;
        cout << "   Number of Channels: " << numChannels << endl;
        cout << "   Number of Data Points: " << numData << endl;
        cout << endl;

        logFile << logName << endl;
        logFile << endl << "------------------------------" << endl;
        // this prints to the log

        logFile << "Data Summary: " << endl;
        logFile << "   Number of Runs: " << runNumber << endl;
        logFile << "   Number of Channels Per Run: " << numChannels << endl;
        logFile << "   Number of Data Points Per Channel: " << numData << endl;
        logFile << endl;

        // call from formatting the part where I print the log summaries in simulationLog()

    }
}

void logGSLVector(const gsl_vector* vec) {
    logFile << "[";
    for (size_t i = 0; i < vec->size; ++i) {
        logFile << gsl_vector_get(vec, i);
        if (i != vec->size - 1) {
            logFile << ", ";
        }
    }
    logFile << "]" << endl;

}

void logGSLMatrix(const gsl_matrix* matrix) {
    if (matrix == nullptr) {
        logFile << "Matrix is nullptr." << endl;
        return;
    }

    size_t rows = matrix->size1;
    size_t cols = matrix->size2;

    for (size_t i = 0; i < rows; ++i) {
        for (size_t j = 0; j < cols; ++j) {
            double value = gsl_matrix_get(matrix, i, j);
            logFile << value << " ";
        }
        logFile << endl;
    }
}

void closeLog() {
    if(logFile.is_open()) {
        logFile.close();
    }
}
```


[1] A. Einstein, Approximative Integration of the Field Equations of Gravitation (1916), volume 6: The Berlin Years: Writings, 1914-1917 (English translation supplement) Page 201 (213 of 462).

[2] LIGO Lab CalTech, Sources and types of gravitational waves ().

[3] D. Lindley, A Fleeting Detection of Gravita-

tional Waves, Physics **16**, 19 (2005), publisher: American Physical Society.
[4] LIGO Lab CalTech, Timeline ().
[5] LIGO Lab CalTech, The science of LSC research ().
[6] LIGO Lab CalTech, Why detect them? ().
[7] Virgo Interferometer, Detector ().
[8] Virgo Interferometer, Sensitivity ().
[9] S. Biscoveanu, C. Talbot, E. Thrane, and R. Smith, Measuring the primordial gravitational-wave background in the presence of astrophysical foregrounds, Physical Review Letters **125**, 241101 (2020), arXiv:2009.04418 [astro-ph, physics:gr-qc].
[10] V. Mandic and A. Buonanno, Accessibility of the pre-big-bang models to LIGO, Physical Review D **73**, 063008 (2006), publisher: American Physical Society.
[11] W. Giarè and A. Melchiorri, Probing the inflationary background of gravitational waves from large to small scales, Physics Letters B **815**, 136137 (2021).
[12] G. Agazie, A. Anumarlapudi, A. M. Archibald, Z. Arzoumanian, P. T. Baker, B. Bécsy, L. Blecha, A. Brazier, P. R. Brook, S. Burke-Spolaor, R. Burnette, R. Case, M. Charisi, S. Chatterjee, K. Chatziioannou, B. D. Cheeseboro, S. Chen, T. Cohen, J. M. Cordes, N. J. Cornish, F. Crawford, H. T. Cromartie, K. Crowter, C. J. Cutler, M. E. DeCesar, D. DeGan, P. B. Demorest, H. Deng, T. Dolch, B. Drachler, J. A. Ellis, E. C. Ferrara, W. Fiore, E. Fonseca, G. E. Freedman, N. Garver-Daniels, P. A. Gentile, K. A. Gersbach, J. Glaser, D. C. Good, K. Gültekin, J. S. Hazboun, S. Hourihane, K. Islo, R. J. Jennings, A. D. Johnson, M. L. Jones, A. R. Kaiser, D. L. Kaplan, L. Z. Kelley, M. Kerr, J. S. Key, T. C. Klein, N. Laal, M. T. Lam, W. G. Lamb, T. J. W. Lazio, N. Lewandowska, T. B. Littenberg, T. Liu, A. Lommen, D. R. Lorimer, J. Luo, R. S. Lynch, C.-P. Ma, D. R. Madison, M. A. Mattson, A. McEwen, J. W. McKee, M. A. McLaughlin, N. McMann, B. W. Meyers, P. M. Meyers, C. M. F. Mingarelli, A. Mitridate, P. Natarajan, C. Ng, D. J. Nice, S. K. Ocker, K. D. Olum, T. T. Pennucci, B. B. P. Perera, P. Petrov, N. S. Pol, H. A. Radovan, S. M. Ransom, P. S. Ray, J. D. Romano, S. C. Sardesai, A. Schmiedekamp, C. Schmiedekamp, K. Schmitz, L. Schult, B. J. Shapiro-Albert, X. Siemens, J. Simon, M. S. Siwek, I. H. Stairs, D. R. Stinebring, K. Stovall, J. P. Sun, A. Susobhanan, J. K. Swiggum, J. Taylor, S. R. Taylor, J. E. Turner, C. Unal, M. Vallisneri, R. v. Haasteren, S. J. Vigeland, H. M. Wahl, Q. Wang, C. A. Witt, O. Young, and T. N. Collaboration, The NANOGrav 15 yr Data Set: Evidence for a Gravitational-wave Background, The Astrophysical Journal Letters **951**, L8 (2023), publisher: The American Astronomical Society.
[13] J. V. Stone, *Independent Component Analysis: A Tutorial Introduction* (MIT Press, Cambridge, Mass, 2004).
[14] A. Tharwat, Independent component analysis: An introduction, Applied Computing and Informatics **17**, 222 (2020), publisher: Emerald Publishing Limited.