main.cpp

The code is a C++ program that performs various tasks related to data analysis. It is an application with a command-line interface for processing data files, generating histograms, and performing comparisons.

Libraries and Namespaces: The program includes necessary libraries, such as "essential.h" and "functions.h".

Interface: The program's main function begins by displaying an interface logo using the interface() function.

Command-Line Option Handling: The program allows the user to choose between different options: "compare", "hadd", or data analysis.

"compare" Option: If the user chooses "compare", the program prompts for additional options ("standard" or "custom") and performs either standard or custom comparison based on user input. The “custom_compare” and “standard_compare” functions are implemented in the included "functions.h" header file.

"hadd" Option: If the user chooses "hadd", the program prompts for an input path to perform the merging of ROOT files using the hadd_creator function.

Data Analysis Option: If the user doesn't choose "compare" or "hadd", the program proceeds to data analysis.

Data Analysis Loop: The program enters a loop where it processes data files for analysis. It prompts the user for the data folder path, data format path, and sampling time. The files with extension .h5 and .txt are found in the given data folder path using the function file_selector. It then performs the following steps for each file:

Displays information about the current file being processed.

Determines the file's extension to decide whether to use analyser_h5 or analyser_matrix function for analysis.

Calls the appropriate analysis function (analyser_h5 for ".h5" files or analyser_matrix for ".txt" files).

Collects the results and errors from the analysis and stores them in respective vectors.

Histogram Result Writing: After analyzing all files, the program calls the histogram_result_writer function to write the histogram results and errors to appropriate output files. The temp_FoundFiles.txt file is removed.

End of Program: The main function returns 0, indicating successful execution.

Note

Please note that the provided code references functions (interface, custom_compare, standard_compare, hadd_creator, reader, analyser_h5, analyser_matrix, histogram_result_writer, file_selector, is_number, line_counter, splitter, etc.) that are assumed to be defined in the "essential.h" and "functions.h" header files. The detailed functionality of these functions would be defined in those header files or in additional source files that are not included here.

essential.h

COLORS: This header defines color codes to be used in terminal output. It uses escape sequences to set text color in terminal output. Each color is defined as a preprocessor macro, allowing you to use these codes to change text colors in your terminal output. For example, RESET resets the text color to default, RED sets the text color to red, BOLDORANGE sets the text color to a bold orange, and so on.

basich: This header includes a variety of standard C++ libraries and system-related headers.

Included: iostream, fstream, sstream, string, vector, unistd.h, sys/types.h, bits/stdc++.h, signal.h, cstdlib, iomanip, algorithm, filesystem.

rooth: This header includes various ROOT-related headers, which are used for data analysis and visualization using the ROOT framework.

Included: TF1.h, TChain.h, TError.h, TDirectory.h, TFile.h, TTree.h, TMath.h, TString.h, TCanvas.h, TGraph.h, TApplication.h, TBrowser.h, TROOT.h, TH1F.h, TFrame.h, TSystem.h, TStyle.h

functions.h

vector<vector<double>> reader(ifstream &thefile):

The reader function is designed to read data from an input file stream (ifstream) and generate a transposed matrix of double values. This function is named "Hazal's Function" and follows a specific process to convert the input data into a transposed matrix of numerical values.

Parameters

thefile (input): An input file stream (ifstream) that refers to the file from which the data will be read.

Return Value

The function returns a transposed matrix represented as a vector of vectors of double values (vector<vector<double>>).

Function Description

Initialization: The function starts by initializing various variables including line for reading each line from the input file, matrix to store the matrix as a vector of vector of strings, empty to store an empty matrix of double values, n_rows to count the number of rows in the matrix, and n_columns to store the number of columns.

Reading Data: The function reads the input file line by line using the getline function. For each line, it processes the line using an istringstream to extract individual elements. The extracted elements are then stored in a vector called row.

Storing Matrix: The row vector is then added to the matrix vector, representing a row of the matrix. The n_rows counter is incremented. If n_columns is still 0, the n_columns value is set to the size of the row vector, thus recording the number of columns in the matrix.

Transposing Matrix: After reading all the data from the input file, the function proceeds to transpose the matrix. It creates a new matrix called transpose with dimensions n_columns by n_rows.

Conversion to Double: The function loops through each element of the matrix and converts the string values to double values using the stof function. The converted values are stored in the corresponding positions in the transpose matrix.

Displaying Information: The function prints the number of rows and columns in the matrix using the cout stream.

Return Value: Finally, the transposed matrix transpose is returned from the function.

vector<string> splitter(string name, string DELIMITER)

The splitter function is designed to split a given string into multiple substrings using a specified delimiter. The substrings are then stored in a vector of strings, which is returned by the function.

Parameters

name (input): The string that needs to be split into substrings.

DELIMITER (input): The delimiter string used to split the input string into substrings.

Return Value

The function returns a vector of strings (vector<string>) containing the substrings extracted from the input string.

Function Description

Initialization: The function starts by initializing a vector of strings called name_split to store the extracted substrings.

Tokenization: The function uses the C standard library's strtok function to tokenize the input string. It takes the name string and the DELIMITER string converted to a C-style string using .c_str().

Tokenization Loop: The function enters a loop where it repeatedly calls strtok with a NULL pointer as the first argument. This is done to continue tokenizing the input string until no more tokens can be found.

Pushing Tokens: For each extracted token (substring), the function appends it to the name_split vector using the push_back method.

Memory Management: After the loop is completed, the function deletes the token pointer.

Return Value: The function returns the name_split vector containing the extracted substrings.

string file_selector(string path):

The file_selector function is designed to search for all text files (with the ".txt" extension) within a specified folder and its subdirectories. It utilizes the system command to execute a shell command that performs the file search. The paths of the found text files are saved in a temporary text file, and the path of this temporary file is returned by the function.

Parameters

path (input): The path of the folder in which the function will search for text files.

Return Value

The function returns a string containing the path to the temporary text file that lists the paths of the found text files.

Function Description

Temporary File: The function starts by defining a string named found_files_path to store the path of the temporary text file that will hold the paths of the found text files.

System Command: The function constructs a shell command to search for text files within the specified path. It utilizes the find command with the -type f -iname *.txt options to search for regular files (not directories) with the ".txt" extension. The search results are appended to the temporary file. The command is constructed using the input path and the found_files_path.

Execution: The constructed command is executed using the system function. The return value of the system is stored in systemErr.

Error Handling: If systemErr equals -1, an error message is printed to indicate that the execution of the command encountered an error.

Return Value: The function returns the found_files_path string, which contains the path to the temporary file that lists the paths of the found text files.

int line_counter(string path):

The line_counter function is designed to count the number of lines in a text file specified by its path. The function reads the file line by line and increments a counter to keep track of the total number of lines in the file.

Parameters

path (input): The path of the text file for which the number of lines will be counted.

Return Value

The function returns an integer representing the total number of lines in the specified text file.

Function Description

Initialization: The function starts by initializing an unsigned integer variable named lines to store the count of lines in the file.

File Opening: The function opens the specified text file using an input file stream (ifstream) named inputfile.

Line Counting Loop: The function enters a loop that reads the file line by line using the std::getline function. For each line read, the lines counter is incremented.

File Closure: After reading all lines from the file, the function automatically closes the input file stream.

Return Value: The function returns the final count of lines stored in the lines variable.

Note

The line_counter function efficiently counts the number of lines in a file. However, it's important to note that this function assumes that the file exists and can be opened. It's a good practice to include proper error handling to deal with cases where the file might not be accessible or doesn't exist.

bool is_number(const std::string &s):

The is_number function is designed to determine whether a given string can be successfully converted to a numerical value, specifically a double. The function attempts to convert the string to a double using the strtod function from the C standard library and then performs checks to verify if the conversion was successful.

Parameters

s (input): A constant reference to the string that needs to be checked for numerical validity.

Return Value

The function returns a boolean value indicating whether the provided string can be successfully converted to a valid double number.

Function Description

Conversion Attempt: The function starts by declaring a pointer to a character named end and a double variable named val. It uses the strtod function to attempt to convert the input string s into a double value. The end pointer will point to the first character that couldn't be converted.

Validation: The function performs the following checks:

end != s.c_str(): This condition checks if any characters were converted, ensuring that the input string was not empty or consisted entirely of non-numeric characters.

*end == '\0': This condition checks if the end pointer points to the null-terminator, indicating that the entire input string was successfully converted without any remaining non-numeric characters.

val != HUGE_VAL: This condition checks if the conversion result is not equal to HUGE_VAL, which indicates an overflow during the conversion process.

Return Value: The function returns true if all the conditions are met, indicating that the input string can be successfully converted to a double value. Otherwise, it returns false.

Note

The is_number function serves as a useful utility for verifying whether a given string can be interpreted as a numerical value. Keep in mind that this function only checks the format of the input string; it doesn't guarantee that the value is within a specific range or precision.

interface():

The interface function is designed to display a stylized title interface on the console. The interface incorporates ASCII art and custom colors to create an eye-catching title display for the "SAT-Force Analysis Interface".

int find_position(vector<string> vec_string, string search_string):

The find_position function is designed to search for the first occurrence of a given search string within a vector of strings. It iterates through the vector and returns the index of the first element that matches the search string.

Parameters

vec_string (input): A vector of strings in which the search will be performed.

search_string (input): The string to search for within the vector.

Return Value

The function returns an integer representing the index of the first occurrence of the search_string within the vec_string. If the search string is not found, the function returns -1.

Function Description

Initialization: The function starts by initializing an integer variable named position with a default value of -1. This will serve as the indicator for whether the search string was found or not.

Looping Through Vector: The function iterates through the elements of the vec_string vector using a for loop. It checks each element to see if it matches the search_string.

String Comparison: For each element of the vector, the function compares it with the search_string using the equality operator (==).

Matching Element: If a match is found, the function immediately returns the current index i as the position of the first occurrence of the search string.

Return on No Match: If the entire loop completes without finding a match, the function returns the default value of -1 to indicate that the search string was not found.

Note

The find_position function provides a simple way to locate the index of the first occurrence of a search string within a vector of strings. Keep in mind that this function performs a linear search, which might not be efficient for very large vectors. If you need to perform multiple searches or have larger datasets, consider using more advanced data structures like hash maps or binary search trees for faster lookup times.

double summation_vec(vector<double> input, int first):

The summation_vec function is designed to calculate the sum of elements in a given vector of double values, starting from a specified index (first) and continuing until the end of the vector.

Parameters

input (input): A vector of double values for which the sum will be calculated.

first (input): The index from which the summation should start.

Return Value

The function returns a double value representing the sum of elements in the vector input starting from the index first.

Function Description

Initialization: The function starts by initializing a double variable named sum to store the cumulative sum of the vector elements.

Looping through Vector: The function iterates through the elements of the input vector using a for loop. The loop starts from the first index and continues until the end of the vector.

Accumulating Sum: For each element of the vector, the function adds its value to the sum variable.

Return Value: After iterating through all relevant elements, the function returns the accumulated sum stored in the sum variable.

Note

The summation_vec function provides a convenient way to calculate the sum of elements in a vector, starting from a specified index. Keep in mind that the function assumes valid input indices, and you should ensure that the provided index is within the valid range of the vector to avoid unexpected behavior.

vector<vector<string>> hist_reader(ifstream &thefile):

The hist_reader function is designed to read histogram data from an input file stream (ifstream) and organize it into a matrix-like structure, represented as a vector of vectors of strings. Each line from the input file is split using a comma delimiter, and the resulting values are stored as rows in the matrix.

Parameters

thefile (input): An input file stream (ifstream) that refers to the file containing the histogram data.

Return Value

The function returns a matrix-like structure represented as a vector of vectors of strings (vector<vector<string>>) where each inner vector corresponds to a row of histogram data.

Function Description

Initialization: The function starts by initializing a vector of vectors of strings named matrix to store the histogram data.

Reading Data: The function reads the input file line by line using the getline function. For each line, it uses the splitter function to split the line into individual string values based on the comma delimiter.

Row Storage: The resulting vector of string values obtained from the splitter function represents a row of histogram data. This row is then added to the matrix vector.

Return Value: After reading all lines from the input file, the function returns the populated matrix containing the histogram data.

Note

The hist_reader function provides a way to read and structure histogram data from a file. Make sure that the input file format matches the expected structure (comma-separated values in rows) for proper functioning of the function. Additionally, consider validating the data and handling any potential errors that might arise during reading or processing.

void compare_hist(vector<vector<double>> data, vector<string> filters, string hist_type, string output_path, string bin_division):

The compare_hist function is designed to generate histograms for certain combinations of data based on provided parameters. The histograms are created using the ROOT framework (ROOT CERN) and are customized based on the given input.

Parameters

data (input): A vector of vectors of double values representing the data to create histograms from.

filters (input): A vector of strings containing filter information.

hist_type (input): A string representing the type of histogram to create.

output_path (input): A string representing the path where the generated histograms will be saved.

bin_division (input): A string representing the division for binning in the histogram.

Function Description

Histogram Naming: The function constructs a name for the histogram based on the hist_type and filters provided. It appends filter names to the histogram name and replaces certain characters for compatibility.

Data Preparation: The function extracts relevant data from the data vector and stores it in temp_data and temp_std vectors.

Calculating Range: The function calculates the minimum and maximum values for the histogram based on the temp_data.

Creating Histogram: The function initializes a TCanvas and creates a histogram (TH1) named histo based on the calculated range. It fills the histogram with data from temp_data.

Standard Deviation: The function calculates the standard deviation of the histogram data and stores it in the std variable.

Rebinned Histogram: The function calculates the number of bins based on bin_division and the size of temp_data. A new histogram (new_histo) is created with a modified range and the specified number of bins. It's filled with data from temp_data.

Customizing Histogram: The function customizes the histogram's X-axis title based on the hist_type.

Drawing Histogram: The function draws the histogram on the canvas.

Saving Histogram: The function constructs the output file name based on the output_path, histogram name, and PDF extension. It saves the histogram canvas as a PDF file.

Memory Management: The dynamically allocated memory for canvases and histograms is properly deleted.

Note

The compare_hist function integrates ROOT framework functionality to generate and customize histograms based on given data and parameters. Make sure to have the ROOT library properly linked and set up for your project. Additionally, this function heavily relies on ROOT-specific classes and methods, so it may not be portable across different environments.

vector<int> filter(vector<vector<string>> data, string filter):

The filter function is designed to filter data based on a given type and value. It searches for the specified type in the header row of the input data matrix and then identifies the positions of rows that match the filter criteria in terms of both the specified type and value.

Parameters

data (input): A vector of vectors of strings containing the data to be filtered.

filter (input): A string containing the filter in the format "type:value". The function searches for rows that match this filter.

Return Value

The function returns a vector of integers (vector<int>) containing the positions of rows that match the filter criteria.

Function Description

Initialization: The function starts by initializing a vector of integers named positions to store the positions of matching rows.

Filter Extraction: The function extracts the filter type and filter value from the input filter string using the splitter function.

Find Filter Type Position: The function iterates through the header row of the data matrix (at index 0) to find the position of the column that matches the filter type.

Position Validation: The function checks if the filter_type_position is within the bounds of the data rows, displaying an error message if it's not.

Filtering Data Rows: The function iterates through the data rows (excluding the header row) and checks if both the filter type and the value match the specified criteria. If both conditions are met, the row's position is added to the positions vector.

Return Value: After processing all data rows, the function returns the positions vector containing the positions of rows that match the filter criteria.

Note

The filter function serves as a tool for selecting and extracting specific rows from a matrix of data based on filter criteria. Be cautious with data type conversions and ensure that the data matrix is correctly formatted according to the filter type and value positions.

string concatenate_vec(string head_string, vector<string> vector_line, string first, string last, string DELIMITER):

The concatenate_vec function is designed to concatenate elements from a vector of strings into a single string, separated by a specified delimiter. The concatenation starts from the first position (inclusive) and goes up to the last position (exclusive) within the vector.

Parameters

head_string (input): A string that will be prepended to the concatenated result.

vector_line (input): A vector of strings from which elements will be concatenated.

first (input): A string representing the first element to include in the concatenation. An empty string indicates starting from the beginning.

last (input): A string representing the last element to include in the concatenation. An empty string indicates ending at the last element.

DELIMITER (input): A string that will be used as a delimiter to separate concatenated elements.

Return Value

The function returns a single string that is the result of concatenating elements from the vector_line based on the specified range and delimiter.

Function Description

Initialization: The function starts by initializing integer variables i_first and i_last to determine the range of elements to concatenate.

Determine First Position: The function checks if the first is empty. If it is, the i_first is set to 0. Otherwise, it uses the find_position function to locate the index of the first element in the vector_line and adds 1 to it to include the element itself.

Determine Last Position: The function checks if last is empty. If it is, i_last is set to the size of the vector_line. Otherwise, it uses the find_position function to locate the index of the last element in the vector_line.

Concatenation Loop: The function initializes a string new_line with the value of head_string. It then iterates through the elements of the vector_line starting from i_first and up to (but not including) i_last. For each element, it appends the element followed by the DELIMITER to new_line.

Return Value: After the loop, the function returns the new_line string, which is the result of concatenating elements with the specified delimiter.

Note

The concatenate_vec function provides a convenient way to concatenate specific elements from a vector of strings using a delimiter. Make sure to manage the range of elements properly, ensuring that the indices provided by the first and last elements are within bounds and correctly represent the desired range.

void hadd_creator(string hadd_path, string input_path):

The hadd_creator function is designed to generate and execute the hadd command, which is used in the ROOT framework to merge multiple ROOT files into a single file. The function also includes logic to handle cases where the output file already exists and provides an option to delete the existing file and continue or abort the merging process.

Parameters

hadd_path (input): A string representing the path and name of the output merged ROOT file.

input_path (input): A string representing the path to the directory containing the input ROOT files to be merged.

Function Description

Generate hadd Command: The function constructs the hadd command string by combining the hadd_path and input_path with the appropriate flags to merge all the ROOT files in the specified directory.

Check for Existing File: The function checks if a file already exists at the hadd_path location. If it does, it prompts the user to decide whether to delete the existing file and continue or to abort the merging process.

Execute Deletion (Optional): If the user chooses to delete the existing file, the function removes the file using the remove function.

Execute hadd Command: The function executes the hadd command using the system function. This command merges all the ROOT files in the input directory into the specified output file.

Error Handling: The function checks for errors after executing the hadd command and displays an error message if there's an issue with merging the ROOT files.

Note

The hadd_creator function is a useful tool for automating the process of merging ROOT files using the hadd command provided by the ROOT framework. It includes error handling and a user interaction step for cases where there's a possibility of overwriting an existing file. Make sure that the necessary ROOT libraries are linked and set up properly for your project to use the hadd command.

vector<int> filter_intersector(vector<int> first, vector<int> second):

The filter_intersector function is designed to find the intersection of two vectors of integers. It identifies the common elements that exist in both input vectors and returns a new vector containing those common elements.

Parameters

first (input): A vector of integers representing the first input vector.

second (input): A vector of integers representing the second input vector.

Return Value

The function returns a vector of integers (vector<int>) containing the elements that are common to both input vectors.

Function Description

Initialization: The function starts by initializing integer variables n1 and n2 to store the sizes of the input vectors first and second, respectively.

Merging Vectors: The function initializes a vector v with a size of n1 + n2 to accommodate the potential intersection size. This vector will hold the result of the intersection.

Sorting Vectors: Both input vectors, first and second, are sorted using the sort function to prepare them for intersection.

Intersection: The set_intersection function is used to perform the intersection of the sorted input vectors. The result is stored in the v vector.

Vector Resizing: The v vector is resized to the actual size of the intersection using the difference between the iterator it and the beginning of the v vector.

Return Value: The function returns the v vector containing the common elements present in both input vectors.

vector<string> compare_available_options(vector<vector<string>> data, int column, vector<int> positions):

The compare_available_options function is designed to extract and return unique values from a specific column of a matrix-like data structure, either from the entire column or from specific rows indicated by positions.

Parameters

data (input): A matrix-like structure represented as a vector of vectors of strings from which values will be extracted.

column (input): An integer representing the index of the column to extract values from.

positions (input): A vector of integers indicating the positions of rows from which values will be extracted. If empty, values will be extracted from the entire column.

Return Value

The function returns a vector of strings (vector<string>) containing the unique values extracted from the specified column or rows.

Function Description

Initialization: The function starts by initializing an empty vector of strings named v to store the extracted values.

Value Extraction: The function checks whether the positions vector is empty. If it is, the function iterates through all rows (except the header row) in the specified column, adding the values to the v vector.

Specific Row Extraction: If the positions vector is not empty, the function iterates through the positions specified in the vector, extracting the values from the specified column and adding them to the v vector.

Sorting and Deduplication: The v vector is sorted using the sort function, and then duplicates are removed using the unique function in combination with the resize function.

Return Value: The function returns the v vector containing the unique values extracted from the specified column or rows.

Note

The compare_available_options function provides a way to extract and deduplicate values from a specific column or rows of a matrix-like data structure. Ensure that the column index and row positions are valid within the given data to avoid out-of-bounds access. The function can be helpful for obtaining unique options available in a particular column for further analysis or processing.

void custom_compare(string hist_path):

The custom_compare function performs a custom comparison and analysis process based on histograms and user-defined filters. It interacts with the user to determine filtering criteria and generates histograms for the specified combination of filters.

Parameters

hist_path (input): A string representing the path to the histogram data file.

Function Description

Verbose Mode: The function sets the global error level to ignore informational messages.

Source Date Extraction: The function extracts the source date from the provided hist_path.

Histogram Data Reading: The function opens and reads the histogram data file using the hist_reader function. The data is stored in the hist_output vector of vectors.

Initialization: The function initializes variables and vectors for storing filter values, user options, and filtered positions.

Filter Value Collection: The function iterates through available options for comparison based on the histogram data. It prompts the user for filter values and dynamically filters the data based on these values.

Filtered Position Handling: The filtered positions are intersected with previously filtered positions to narrow down the comparison range. If no combinations are found, the function outputs a message and ends.

Result Generation: If combinations are found, the function prompts the user to specify a division factor for histogram binning.

Directory and File Initialization: The function generates directory paths and initializes a ROOT output file for saving the comparison histograms.

Histogram Generation: The function iterates through histogram types, extracting values and errors for each filtered position. It then generates and saves comparison histograms using the compare_hist function.

Result Storage and Reporting: The comparison histograms are written to the ROOT output file. The function outputs a message indicating where the results have been saved.

Note

The custom_compare function automates the process of generating custom comparison histograms based on user-specified filters and available options in a given histogram data file. This function relies heavily on ROOT's functionalities and interactive user input. Ensure that the necessary ROOT libraries are linked and set up properly for your project.

void standard_compare(string hist_path):

The standard_compare function performs a standard comparison and analysis process based on histograms, considering various combinations of source, scintillator, and threshold filters. It generates histograms for these specified combinations.

Parameters

hist_path (input): A string representing the path to the histogram data file.

Function Description

Verbose Mode: The function sets the global error level to ignore informational messages.

Source Date Extraction: The function extracts the source date from the provided hist_path.

Histogram Data Reading: The function opens and reads the histogram data file using the hist_reader function. The data is stored in the hist_output vector of vectors.

Available Options: The function extracts available options for sources, scintillators, and thresholds by utilizing the compare_available_options function.

Print Available Options: The function prints available source, scintillator, and threshold options.

User Input: The function prompts the user to specify a division factor for histogram binning.

Combination Iteration: The function iterates through all combinations of sources, scintillators, and thresholds.

Filter Position Calculation: The function calculates filtered positions based on the source, scintillator, and threshold filters.

Result Generation: If valid positions are found for the combination, the function generates and saves comparison histograms using the compare_hist function. It also outputs the directory where the results are saved.

Note

The standard_compare function automates the process of generating standard comparison histograms for various combinations of sources, scintillators, and thresholds. This function relies heavily on ROOT's functionalities and interactive user input. Ensure that the necessary ROOT libraries are linked and set up properly for your project.

void histogram_result_writer(string data_path, string data_format_path, vector<string> results, vector<string> errors):

The histogram_result_writer function writes histogram results and errors to output files based on specified data paths and formats.

Parameters

data_path (input): A string representing the path to the original data file.

data_format_path (input): A string representing the path to the data format file.

results (input): A vector of strings containing the histogram results to be written.

errors (input): A vector of strings containing error messages to be written (if any).

Function Description

Output Directory Preparation: The function constructs the output directory path and file name based on the current working directory, the data file name, and the "outputs/data" subdirectory. It creates the output directory if it doesn't exist.

Output Files Creation: The function creates output files for histogram results and errors based on the constructed paths. It uses ofstream objects to manage file writing.

Data Format Writing: The function reads the data format file and writes its contents to the histogram result output file. It uses a temporary string temp_data_format to store the header format and writes it to the output file.

Data Results Writing: The function iterates through the results vector and writes each result entry to the output file.

Output and Closure: The function prints messages indicating where the histogram results and errors have been saved. It closes the output files.

Error Writing (Optional): If the errors vector contains any entries, the function writes the error messages to a separate output file.

Note

The histogram_result_writer function simplifies the process of saving histogram results and error messages to output files. Ensure that the provided data format file matches the expected format and that the provided paths are valid. The function is designed to improve organization and reporting when dealing with large-scale analysis processes.

vector<string> analyser_matrix(vector<vector<double>> input, string filename, double ns):

The analyser_matrix function performs analysis on a matrix of data, generating histograms, fitting curves, and calculating various parameters. It saves the results and errors in appropriate files and directories.

Parameters

input (input): A 2D vector representing the matrix of data to be analyzed.

filename (input): A string representing the name of the data file.

ns (input): A double representing the time step in nanoseconds.

Function Description

Output Directory Preparation: The function constructs the output directory path and file names based on the current working directory, the input file name, and specific subdirectories. It creates the necessary directories.

Initialization: The function initializes various variables and data structures for storing results and errors.

Data Analysis Loop: The function iterates through each dataset in the input matrix and performs the following steps for each dataset:

Fits a Landau function to the data using TGraph and TF1 classes.

Calculates peak voltage, peak time, rise time, fall time, and integral of the fall for the fitted curve.

Checks for NaN values in the calculated parameters and handles them by plotting and saving the corrupted data.

Histogram Creation: The function creates histograms for fall time, rise time, integral of the fall, peak voltage, and peak time using the calculated parameters.

Results Generation: The function calculates summary statistics for the analyzed parameters and appends the results to a string.

Root File Saving: The function saves histograms and fits in a ROOT file.

Txt File Saving: The function saves histogram results and error plots in a text file.

Memory Deallocation: The function deallocates memory for histograms and closes the ROOT file.

Finalization and Return: The function stores the results and errors in a vector and prints messages indicating where the output has been saved. It returns the vector containing results and errors.

Note

The analyser_matrix function conducts comprehensive analysis on input data matrices, including curve fitting, histogram generation, parameter calculation, and result/error saving. The function is designed to handle data sets and matrices and facilitate a detailed analysis process. Ensure that the input data matrix is properly populated with data points and the time step is provided accurately.

vector<string> analyser_h5(string filename, double ns):

The analyser_h5 function performs analysis on data stored in an H5 file format. It reads data from the H5 file, fits curves to the data, calculates various parameters, generates histograms, and saves the results and errors.

Parameters

filename (input): A string representing the name of the H5 data file.

ns (input): A double representing the time step in nanoseconds.

Function Description

Output Directory Preparation: The function constructs the output directory path and file names based on the current working directory, the input file name, and specific subdirectories. It creates the necessary directories.

H5 File Opening: The function opens the H5 data file for reading.

Number of Datasets: The function determines the number of datasets (segments) stored in the H5 file.

Data Analysis Loop: The function iterates through each dataset in the H5 file and performs the following steps for each dataset:

Reads data from the H5 dataset.

Determines the location of the peak (positive or negative) based on data summation.

Creates x-axis values (time).

Fits a Landau function to the data using TGraph and TF1 classes.

Calculates peak voltage, peak time, rise time, fall time, and integral of the fall for the fitted curve.

Checks for NaN values in the calculated parameters and handles them by plotting and saving the corrupted data.

Histogram Creation: The function creates histograms for fall time, rise time, integral of the fall, peak voltage, and peak time using the calculated parameters.

Results Generation: The function calculates summary statistics for the analyzed parameters and appends the results to a string.

Root File Saving: The function saves histograms and fits in a ROOT file.

Txt File Saving: The function saves histogram results and error plots in a text file.

Memory Deallocation: The function deallocates memory for histograms and closes the ROOT file and H5 file.

Finalization and Return: The function stores the results and errors in a vector, prints messages indicating where the output has been saved, and returns the vector containing results and errors.

Note

The analyser_h5 function is designed to analyze data stored in H5 file format, including data extraction, curve fitting, histogram generation, and result/error saving. The H5 file should contain datasets with a specific naming convention as indicated in the code. Make sure that the H5 file is correctly formatted and contains the relevant data for analysis.