Data Processing Website Design Document

# Overview

The main goal of this side project is to create a website where the user can input data in the form of a lab data table, specify how the data will be processed, and have the data be automatically processed and graphed afterwards. This eliminates the need for the user to perform tedious and repetitive lab calculations, as well as automatically formatting the user’s data tables and graphs for them.

## Terms

### Curve straightening

Curve straightening (also known as linearizing a graph) is a process that involves manipulating raw data and plotting that manipulated data as a straight line (a linear relationship) so that the raw data can be further analyzed. (<https://sites.google.com/site/apphysics1online/appendices/2-data-analysis/graph-linearization>)

### Uncertainty

Uncertainty is a term used to represent the range between the measured value and the actual value of something. For example, the quantity (2.2±0.5) meters has an uncertainty of 0.5 meters. (<https://www2.southeastern.edu/Academics/Faculty/rallain/plab194/error.html>)

# Sequence Diagram



Note: this flowchart may not be 100% accurate, but it is a rough outline of how data could be handled in my project.

# Frameworks/Libraries

## Angular

Angular can be used to organize the look of the website *(Note: as it is the only front-end framework that I have worked with so far)*. Angular users can also make API services in Angular *(Note: but I am not super familiar with them)* to connect the website’s front-end with its backend.

## Handsontable

Handsontable is a library that can be used to display spreadsheets on the website. Spreadsheet data is usually organized through 2D arrays. *(Note: I am not entirely sure if the API should also model the spreadsheet data as a 2D array or through a different model)*.

## Custom API

An API can be used to upload a user’s inputted data to a database. This can be done using a custom API with the ability to create, read, update, and delete a user’s inputted data. The API can be called when the user wants to see a processed data table or when they want to see a graphical plot of their data. The API should be able to interface with DataTable or dataTableSchema objects (objects that represent a user’s inputted data table and curve straightening instructions).

## MongoDB

The data that will be analyzed or displayed through this website will be stored in MongoDB. MongoDB can also be used to process data (altering a copy of the raw data to change it to a desired effect).

## Chart.js

Chart.js is a framework that can be used to plot the data stored in MongoDB. The user can toggle chart options once a chart is displayed to customize the layout of their chart.

## Node.js

Node.js can serve as a backend for a web app, so it can be used for this side project as well. In this app, the backend will be responsible for actually processing the data based on the user’s inputted instructions.

## Math.js

If the user enters a custom method to curve straighten their graph, their string instructions about how to curve straighten the graph will need to be parsed. Math.js has a built-in parser to do just that. *(Note: this framework might not be needed as parsing custom curve straightening methods will require a custom parser to process how the* ***uncertainties*** *corresponding to those data points will be processed, and that might be too complex for now).*

# Overall Design

## How the user can give curve straightening instructions

It is a little difficult to calculate uncertainties based on the instructions that the user gave, as we would need to have an additional parser to determine the “class” of function the user inputted so that we can deal with the uncertainties properly. For example, if the user wanted to plot “y” vs “x^a”, the relative uncertainties for “x” would be multiplied by “a” to produce a new relative uncertainty for the “x^a” term. However, if the user wanted to plot “y” vs “ln(x)”, then the uncertainties for “ln(x)” would be equal to “(1/x)\*(absolute uncertainty for that “x” data point)”.

In short, there is no “one size fits all” method to process uncertainties, so we may need the user to choose from a predetermined list of curve straightening strategies. Instead of having the user dictate a custom curve straightening strategy, we can include common functions in a drop-down menu for the user to use for curve straightening. We can make a custom Instruction class that can keep track of common curve straightening instructions for the user to use.

## The Instruction class

The instruction class represents curve a straightening instruction that can be processed in the back-end.

### Instruction class member variables

functionClass: string : This string will align with various strings that describe common functions used in the sciences. For example, we could make constant string variables such as LOG\_BASE\_E or X\_TO\_CONST\_PWR to describe which function a user wants to apply to the data point.

constantPower: number : If we are doing the common curve straightening operation of raising a data point coordinate to a constant power (X\_TO\_CONST\_PWR), this power variable would be initialized to specify which power that data point coordinate is being raised to.

constantVariableValue: number

constantVariableUncertainty: number

If we are doing the common curve straightening operation of multiplying or dividing a data point coordinate by the value of a constant variable, these 2 member variables will be able to store the value of that constant.

## How data is stored

When following this guide to make skeleton code for my custom API (<https://medium.com/@dinyangetoh/how-to-build-simple-restful-api-with-nodejs-expressjs-and-mongodb-99348012925d>), they modelled contact information through the following model:

1. var contactSchema = mongoose.Schema({
2. name: {
3. type: String,
4. required: true
5. },
6. email: {
7. type: String,
8. required: true
9. },
10. gender: String,
11. phone: String,
12. create\_date: {
13. type: Date,
14. default: Date.now
15. }
16. });

 A similar schema for the graph data would be:

1. var dataTableSchema = mongoose.Schema({
2. xCoords: { // “x” coordinates of the data table
3. type: Array,
4. required: true
5. },
6. yCoords: { // “y” coordinates of the data table
7. type:Array,
8. required: true
9. },
10. xUncertainties: { // uncertainties corresponding to each “x” coordinate
11. type: Array,
12. required: true
13. },
14. yUncertainties: { // uncertainties corresponding to each “y” coordinate
15. type: Array,
16. required: true
17. },
18. xCurveStraighteningInstructions: Instruction,
19. yCurveStraighteningInstructions: Instruction
20. // these are instructions that the user gives when they want to process the raw data table.
21. });

This is a rough outline of one possible schema that could represent one data table. These dataTableSchema objects could then be stored in the MongoDB database. In the frontend, when the user is generating their data table, the form that they are inputting their data (data points and computation instructions) into would also correspond to this dataTableSchema.

## How data is processed

There are pre-existing conventions that are used to propagate uncertainties (<https://sites.fas.harvard.edu/~scphys/nsta/error_propagation.pdf>). If a data table has uncertainties, its data points (with uncertainties) will be processed according to these conventions. The user’s inputted curve straightening instructions will be parsed and the user’s processed data table can then be created. How this will be done will be explained in more detail in the next section (section 4, outlining the DataTable class).

## How data is graphed

Data will be graphed using the chart.js framework. There exists a chart.js plugin to handle error bars. Sample usage of this plugin is shown in this link (<https://github.com/sgratzl/chartjs-chart-error-bars/blob/main/samples/scatter.html>). Data can be entered in the proper format as “x” or “y” (with the appropriate uncertainties) and plotted accordingly.

# The DataTable class

## About the class

The DataTable class will be used to represent the data table that the user inputted.

*(Note: Starting out this side project, I want to start out adding the most basic features to my web app, so I don’t get overwhelmed. For example, coding out extra functionality (such as plotting graphs of “x” vs “y” where there are multiple trials) might get added later when I have a working app that can analyze single-trial graphs of “x” vs “y” variables.)*

## Member variables

xCoords:Array : an array of all the x coordinates of the data table

yCoords:Array : an array of all the y coordinates of the data table

xUncertainties:Array : an array of all the x uncertainties of the data table

yUncertainties:Array : an array of all the y uncertainties of the data table

xCurveStraighteningInstructions:Instruction : an instruction of how the x variable is manipulated

yCurveStraighteningInstructions:Instruction : an instruction of how the y variable is manipulated

# Possible functions

These are certain functions that can be implemented in the code to process data *(Note: here are ones that I thought of so far).*

## ProcessDataPoint ()

ProcessDataPoint() is a function that receives an instruction, a coordinate, and the uncertainty for that coordinate. It can process the Instruction given to it, process the data point accordingly, and return the processed number and uncertainty in an array in the form [processed number, uncertainty for that processed number].

1. ProcessDataPoint(instruction: Instruction, Coord: number, Uncertainty: number) : array {
2. // Switch statement to process the various instructions
3. // processing the instruction given
4. // return an array in the form [processed number, uncertainty for that processed number]
5. }

## ProcessDataTable()

ProcessDataTable() is a function that receives a data table, processes it, and returns a processed DataTable object.

1. ProcessDataPoint(dataTable: DataTable) : DataTable{
2. // make new arrays (xCoords, yCoords, xUncertainties, and yUncertainties) representing the processed data
3. // loop through the xCoords, yCoords, xUncertainties, and yUncertainties arrays of the data table
4. // call ProcessDataPoint( instruction , coordinate, uncertainty) for each data point that we index through this loop
5. // append to the 4 newly created arrays (refers to the arrays in line 2) as needed
6. // make another dataTable object using these 4 new arrays
7. // return another DataTable object with xCurveStraighteningInstructions and yCurveStraighteningInstructions set to undefined.
8. }