

UNIVERSITY OF GENOVA

POLYTECHNIC SCHOOL

DIME

**Department of Mechanical, Energy, Management
and Transportation Engineering**



**BACHELOR THESIS
IN
MECHANICAL ENGINEERING**

**Interactive visualization of Big Data
and Real-time data**

Supervisor:

Chiar.^{mo} Prof. Ing. Alessandro Bottaro

Co Supervisor:

Dott. Ing. Joel Guerrero

Candidate:

Raffaello Daniele

July

Interactive visualization of Big Data and Real-time data

Abstract

This aim of this thesis is to explore the implementation of interactive data visualization for engineering applications. Improving efficiency in engineering systems led to a raise in the complexity of resolution methods. As a result, in the recent years there has been a rapid growth of Big Data methodologies throughout the scientific research. Not only datasets are growing in size, but they are also becoming more and more heterogeneous. Therefore, to design effective tools for navigation and analysis has become quite challenging.

The scope of this dissertation is to determine whether the JavaScript open source libraries D3, Dc and crossfilter are meeting the requirements for data analytics and visual display used in everyday life. Therefore, a thorough analysis of the above mentioned libraries and their abilities of handling substantial amount of data while remaining highly responsive to data filtering and exploration has been conducted. After their eligibility for working with big data files has been confirmed, a feasibility study on the libraries's integration to real-time data analysis has been carried out through the implementation of websocket servers with the objective of determining whether data visualization could be paired with computer simulations for design optimization.

Acknowledgments

Firstly, I would like to thank Professor Alessandro Bottaro for offering me the opportunity to work on this project and for the immense independence, he granted me.

Furthermore, I would like to express my gratitude to Joel Guerrero for his availability to help me throughout the entire thesis.

I would like to thank my family, my friends and my girlfriend for their constant support.

Contents

Abstract	I
Acknowledgments	II
1 - Introduction	1
2 –Data Processing Tools	4
2.1 – Programming Languages.....	4
2.1.1 – HyperText Mark-up Language (HTML).....	11
2.1.2 – Cascading Style Sheets (CSS)	12
2.1.3 – JavaScript (JS)	13
2.2 – JavaScript Libraries	17
2.2.1 – Data-Driven Documents (D3.js)	17
2.2.2 – Crossfilter Library (crossfilter.js)	18
2.2.3 – Dimensional Charting Libray (Dc.js)	21
3 – Data Exploration	22
3.1 – Big Data analysis through data visualization	22
3.2 - Real-time data visualization	27
3.2.1 – Design Optimization	27
3.2.2 – Real-time data acquisition through a websocket server.....	30
4 - Conclusions	32
Appendix	33
References	38
Nomenclature	39

1. Introduction

No longer than a decade ago the word “Big data” was introduced in our lexicon to refer to the ever growing data analysis trend that is quickly conquering areas that most of the time break far away from the scientific domain. Particularly, giant tech companies such as Google, Amazon, Facebook and others are the primary users and developers of data analysis, by collecting click-stream data and communications. This allows these companies to develop new advertising and retail strategies.

As Philip Decamp cited - “Nearly every person with a computer or phone is both a frequent contributor and a consumer of information services that fall under the umbrella of Big Data”. [1]

To refer this concept back to the engineering environment, however, the impact of Big Data has been just as effective. For example, in energy systems or in the design optimization. Additionally, the constant strive for improving efficiency led engineers to design ever-complex iterative models that would converge to optimal solutions. However, these developments require a substantial number of simulations, hence a high computing power that only computers can provide. The gathered data is often displayed in plain text or in the form of tables, which are never the best solutions for data reading or analysis. Alternatively, the most efficient way to summarize what extremely large amounts of data are, is to refer to their statistical properties such as the mean, the median, the variance etc. However, by doing so there is a chance of losing valuable information concerning the data set. English statistician Francis Ascombe, in an attempt to counter the general conception among statisticians that “numerical calculations are exact, but graphs are rough”, provided one example demonstrating the above mentioned theory. Ascombe provided his results in an article called Ascombe’s Quartet shown in Fig. 1.1.

I		II		III		IV	
X	Y	X	Y	X	Y	X	Y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

Fig. 1.1 – Datasets from Ascombe’s Quartet

In the Ascombe's quartet, the four datasets appear to have nearly identical descriptive statistics as shown in Fig. 1.2 below.

Statistical property	Value
Sample size	11
Mean (x)	9
Variance (x)	11
Mean (y)	7.50
Variance (y)	4.122
Correlation	0.816
Linear regression	$Y = 3.00 + 0.5000X$

Fig. 1.2 – Descriptive Statistics of Ascombe's Quartet

Yet, when graphed, these four datasets tell a completely different story, appearing in different forms on scatter plot charts as shown in Fig. 1.3.

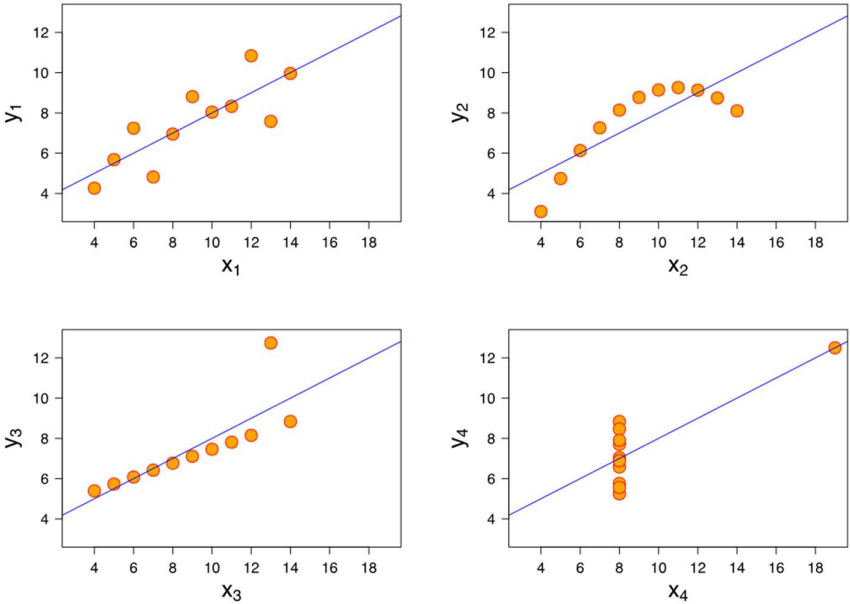


Fig. 1.3 – Ascombe's Quartet graphed through scatter plot charts

- Dataset I consists of a set of points that appear to follow a rough linear relationship with little variance
- Dataset II fits a neat curve but does not follow a linear relationship
- Dataset III looks like a tight linear relationship between x and y, except for one outlier
- Dataset IV appears to be x constant except for one outlier

Hence, data visualization can be considered just as important as statistical data analysis. By placing data in a visual context, people are able to visualize patterns, trends that otherwise would go undetected in a text based, plain data or statistical summary.

Although data visualization allows exploring huge amount of data in a confined space, the constant growth of datasets that are gathered and analysed every day is starting to challenge even the most advanced software programs specifically built for data analytics. Therefore, there is a constant challenge to find the most recently updated tool kit for analysing data. These programs can also be cost effective. Another issue that engineers and developers are facing is represented by the presence of “dirty data” in datasets. This data represents casual points that do not influence a potential pattern. Therefore, it becomes challenging, when confronted to large data files, to retrieve meaningful and valuable information. For this reasons the latest programs / analytics approaches allow for interactive data visualization, hence accelerating the process of data filtering and identification of “dirty data” that needs to be deleted as it only burdens the workload the computer has to provide. Among the multiple software/program choices available for data manipulation and data visualization, a decision was made to implement the JavaScript open source libraries:

- Data Driven Documents (D3.js)
- Crossfiler.js
- Dimensional Charting (Dc.js)

Finally, throughout the course of this thesis, analysis will be carried out to determine whether these libraries are suitable for interactive Big Data analysis and visualization in the context of engineering applications.

2. Data Processing Tools

As mentioned in the previous chapter, the aim of this project is to show how data visualization can be generated through the implementation of the JavaScript library D3.js. It will be used both a JavaScript and HTML approach. In order to manipulate the data and create the dashboards, we will need help from other JavaScript libraries such as Dc.js and Crossfilter.js. Hence, it has been decided to dedicate some time to explain what these languages and libraries are and how they work.

2.1 Programming languages

2.1.1 HyperText Mark-up Language (HTML)

To convey information about a document's structure or presentation, mark-up information is added to the document. Mark-up languages are widely used in everyday computing. For instance, word processors use codes that indicate the structure and presentation of a document. However, while the word processors write the necessary mark-up to produce a document behind the scenes, HyperText Mark-up Language (HTML) is not a behind the scenes mark-up language.

It is a computer language that allows website creation. The functionalities of HTML are:

- HyperText is the method by which it is possible for the user to move around on the web, by clicking on special texts called hyperlinks that transport the user to the next page. The fact that it is hyper just means it is not linear. For instance, the user can go to any place on the internet any time he needs by clicking on links; there is no standard order to accomplish the actions.
- Mark-up is what HTML tags, commonly known as elements, do to the text found in between them. Tags mark the text as a certain type of text such as italicised.
- HTML is a language, as it has code-words and syntax like any other language.

The HTML code is based on tags, which provide all the instructions for formatting the text between each other. Tags will start with an angle bracket (less than sign) '<' and will end with another angle bracket (this time the greater than sign) '>'. Also, tags can be used to inform the processing program, in this case the web browser, how to operate the text. Because of their various uses, the html tags can be divided in categories, depending on the role they play in the manipulating the web page. It is possible to identify different categories such as:

- The basic html tags that are used to create and structure our web page. These particular tags enable the user to create titles, headings, paragraphs and leave comments on the source code as a guidance for the programmers who wish to examine the file.
- Formatting tags allow the user to implement, only to a certain extent, the document's styling. For more complex styles, the Cascading Style Sheet (CSS), a programming language that allows manipulating furthermore more the text, page layout and colours will be implemented.

- Forms and input tags allow the user to insert additional features to our document such as buttons, legends and drop down lists.
- The links tags let the user create links for navigating to different pages from the initial web page.
- The programming tags allow the web developers to implement different programming languages inside an html document.

For better understanding the topic, the different HTML tags used in the course of the thesis are presented below in Tab. 2.1.

Tab. 2.1 – Different HTML elements used in the course of the thesis

HTML Element	Description
Basic Tags	
<!DOCTYPE>	Allows the processing program to know which version of html is in use
<html>	Allows the processing program to know that what follows is html language
<head>	Defines information about the document
<title>	Defines a title for our document which will be displayed in the web tab
<body>	Defines the document body, where we insert the text that will appear on our web page
<h1> ... <h6>	Defines the html heading which are not to be confused with the document's title
<p>	Allow the programmer to add a paragraph element
 	Inserts a single line break
<meta>	Defines metadata about the html document
<!-- ... -->	Allows the programmer to add comments in the source code

Formatting Tags	
	Defines bold text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<center>	Allows to centre the text
	Defines an unordered list
	Defines an ordered list
<table>	Defines a table in our html document
<th>	Defines a table's header
<tr>	Defines a table's row
<td>	Defines a cell in our table
Input tags	
<button>	Defines a clickable button
<fieldset>	Groups related elements in a form
<legend>	Defines a caption for a <fieldset> element
<select>	Defines a drop down list

Links	
<code><a></code>	Defines a hyperlink which allows the user to navigate to other web pages
<code><link></code>	Defines a relationship between a document and an external resource (most used to link to style sheets)
Semantics	
<code><style></code>	Defines a style information for an html document
<code><div></code>	Defines a section in a document
Programming	
<code><script></code>	Defines a client-side script. It allows the user to write a code in another programming language

So far, the html tags have been described in their simplest form. However, it is common practice to add attributes to these tags in order to modify their characteristic. The attributes have to be placed in the opening tag and they should always contain the following two parts:

- The name that represents the property to be set. For instance it is possible to add to a paragraph element `<p>` the name `align` in order to set the alignment of the paragraph in the page.
- The value of the attribute is the value of the property previously set. Using the example above, it is possible to add as a value: “left” to the name `align`. The value has to always be written within quotations.

It is important to note that attributes values are case-insensitive, which means there is no distinction between capital and lowercase letters. However, the World Wide Web Consortium recommends lowercase for HTML v.4.

Among the various attributes that are listed in Tab. 2.2, it is of great importance for the reader’s comprehension to analyse the three core attributes:

- ID attribute
- Class attribute
- Style attribute

As the name indicates, the *id* attribute shown in Fig. 2.1 allows to the user to uniquely identify any element within the html page. This could be useful in several occasions, such as when, the user has different elements with the same name and needs to access one specific element or when he is working in another file source and he needs to link the code to an html element (for example between the JavaScript dc.js chart library and the html element <div>).

```

1 <!DOCTYPE html>
2 <html lang="">
3 <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title></title>
7 </head>
8
9 <body>
10
11 <div id="row-chart">
12   <h4>
13     This is a row Chart
14   </h4>
15 </div>
16
17
18 <div id="bar-chart">
19   <h4>
20     This is a bar chart
21   </h4>
22 </div>
23
24 </body>
25 <script>
26
27   var rowChart1 = dc.rowChart("#row-chart")
28   var barChart1 = dc.barChart("#bar-chart");
29
30   //Rest of the code...
31
32 </script>
33
34 </body>
35 </html>

```

Fig. 2.1 – Code showing the use of the “id” attribute

More general than the id attribute, the *class* attribute shown in Fig. 2.2 allows selecting one or more elements that have the same attribute name. It can be used by CSS or JavaScript to perform certain tasks for elements with the specified class name.

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Title</title>
7
8 <style>
9   .test {
10     background-color: green;
11     color: white;
12     padding: 10px;
13   }
14 </style>
15
16 </head>
17
18 <body>
19
20   <h2 class="test">Experiment 1</h2>
21
22   <h2 class="test">Experiment 2</h2>
23
24   <h2 class="test">Experiment 3</h2>
25
26 </body>
27 </html>
28

```

Fig. 2.2 – Code showing the use of the “class” attribute

The style attribute shown in Fig. 2.3 allows to the user to implement the CSS styling directly inside the designated element. As it is specific to the element, it will override any style set globally such as styles specified in the <style> tag or in an external style sheet.

```

1 <!DOCTYPE html>
2 <html lang="">
3 <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Title</title>
7   <style>
8     .test {
9       background-color: green;
10      color: white;
11      padding: 8px;
12    }
13  </style>
14 </head>
15 </head>
16 </head>
17 <body>
18   <h2 class="test" style="background-color:blue;text-align:center">
19     This is the Experiment 1</h2>
20
21   <h2 class="test">This is the Experiment 2</h2>
22 </body>
23 </html>
24 </html>
25

```




Fig. 2.3 – Code showing the use of the “style” attribute

Tab. 2.2 – Different HTML attributes used in the course of the thesis

HTML Attribute	Belongs to	Description
Data	<object>	Specifies the URL of the resource to be used by the object
Download	<a>, <area>	Specifies that the target will be downloaded when a user clicks on the hyperlink
Hidden	Global attributes	Specifies that an element is not yet, or is no longer relevant
Href	<a>, <area>, <base>, <link>	Specifies the URL of the page the link goes to
Lang	Global attributes	Specifies the language of the element’s content
OnClick	All visible elements	Script to be run when the element is being clicked
Onmouseover	All visible elements	Script to be run when a mouse pointer moves over an element
Rows	<textarea>	Specifies the visible number of lines in a text area
Src	<script>, <source>	Specifies the URL of the media file

For the reader’s sake, further information concerning HTML elements and attributes can be found on the World Wide Web Consortium official website [2].

The beginning of an HTML document will always begin with a `<!DOCTYPE>` declaration. This is not an HTML tag but an instruction which tells the browser which version of HTML the document will be written in. Once it is clear to the web browser which version is being used, the opening HTML tag, `<html>` needs to start the document so that the browser understands the following code is actually HTML code. The user can then proceed to build the `<head>` and the `<body>` containers that are the two main blocks of the document shown in Fig. 2.5.

The `<head>` element shown in Fig 2.4 is the part of the document that will not be displayed in the browser once the page is loaded. It contains information such as the page `<title>`, the relative links to the styling through CSS and other metadata which is essentially data about data.

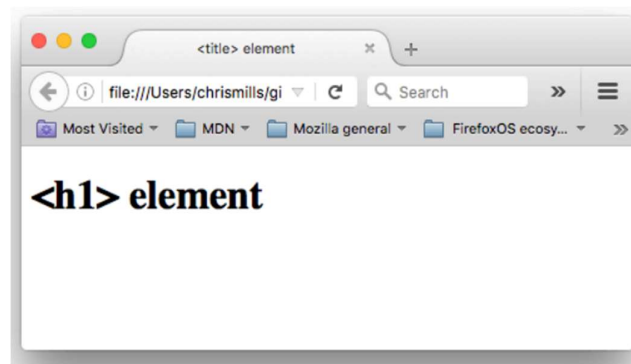


Fig. 2.4 – In this figure it is possible to see the difference between the header and the title element

(The `<title>` element is metadata that describes the overall HTML document, unlike the `<h1>` tag, which belongs to the `<body>` element and is considered as the page's title.)

Adding metadata is crucial as it helps the search engines to keep track and index the web page. For example, the use of the `'lang'` attribute to the opening `html` tag will allow the HTML document to be indexed more effectively by the search engine if the language is set.

During the years, adding metadata in the `<head>` element became important to the point that the World Wide Web Consortium (W3C) developers decided to add a `<meta>` tag. This is used for specifying the document's character encoding as shown in Fig. 2.6 and Fig. 2.7. For instance, UTF-8, a universal character set that includes every character from every human language. This means that the web page will be able to display any language requested. On the other hand, if the user implements the characters encoding under ISO-8859-1 the page rendering could be different from what expected, with the web not recognizing certain characters.

Moreover, to improve the interaction between the web page and the user, the latter should set the viewport. This represents the user's visible area of a web page, which can vary according to the devices in use. The viewport is added by including the `<meta>` viewport element, which gives the browser instructions on how to control the page's dimensions and scaling. The `width=device-width` sets the width of the page to follow the screen-width of the device in use.

The <body> element, also known as the document body element, is perhaps the most important part of the html document. It contains the web page's structure with the CSS styling and the JavaScript used to modify the rendering.

The following element can be combined between each other and are part of the main features of the documents body:

- DIV: it is used for hierarchical containers and static banners
- Block elements formed by paragraphs, lists, forms, tables, figures ect...
- Horizontal rules, and the Address element
- Text and character level mark-up including emphasis, images, math, hypertext links and miscellaneous elements.

2.1.2 Cascading Style Sheets (CSS)

Cascading Style Sheets is a style language that is designed to modify the presentation of a document by changing colors, fonts and layout. In an HTML document, the CSS styling is usually specified in the <head> element. However, any time the user needs to apply a certain style to different documents, a .css file containing the necessary style can be created and then called in the different HTML codes through the <link> element.

A css file contains the style rules for every element to which it is desired to modify the presentation. The anatomy of a CSS rule is shown in Fig. 2.8. It is possible to differentiate:

- The selector, which communicates to the browser which HTML element we wish to style
- The property, which identifies the property to apply to the element (such as the background-color, the text color and the margins.)
- The value of the property that in the example of the text color, refers to which color to use
- The property and the value pair up to for a declaration.

Generally, complex styles are created by combining together multiple declarations (each separated by a semicolons) within one selector. Also, the rule needs to be enclosed in between curly brackets.



Fig. 2.8 – CSS rule's anatomy

2.1.3 JavaScript (JS)

JavaScript (JS) programming language can be categorized as:

- High Level Language (HLL) that has a higher abstraction from the computer, as it is generally independent from a computer's hardware architecture. Thus, high level programming languages are more user friendly than low level languages.
- Interpreted Programming Language as the code has to be parsed, interpreted, and executed each time it is run.
- Dynamic, in computer science, these languages are able to execute many actions at runtime. Those may include extension of the program or adding new code by extending objects.
- Weakly typed: contrary to strongly type languages that check the type of a variable before performing an operation on it, weakly typed languages do not check the type of variables. Additionally, weakly typed languages perform implicit casts.

Regardless of the programming language that is being used, a program will always start with some declarations, followed by functions inside which there will be implemented instructions for arithmetical, logic, relational operations.

Declarations are used to provide the data type of each data contained in the program. Data inside JavaScript can be of different types as shown in Fig. 2.9:

- Boolean: they represent the 'true' & 'false' values. They are often used for conditional statements.
- Null: it is a variable defined to have a null value.
- Undefined: a variable that has not been defined yet. A variable that is not defined, will produce the 'not defined' error in return.
- Number: the number data type can handle integers and floats. The 'number' type can handle positive, negative numbers and decimal places.
- String: In computer science, text is treated as strings (which are grouping of characters). In order to define a string text has to be put in between simple or double quotes, for example: "Hello World" or 'Hello World').
- Object: an object is a collection of properties. A property is an association between a name (key) and a value.

```
var boolean    = true;
var property   = null;
var number     = 4;
var string     = "Hello World";
var object     = { type: car,
                  Manufacturer: Ferrari,
                  Model: "430",
                  BHP: 500
                }
```

Fig. 2.9 – JavaScript code displaying the different types of data

In the Tab.2.3 below, different types of operators that JavaScript provides in order to manipulate data are presented.

Tab. 2.3 – JavaScript operators for creating expressions

Arithmetic Operators	
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (Division remainder)
++	Increment
--	Decrement
Comparison Operators	
==	Equal to
===	Equal value and equal type
!=	Not equal
!==	Not equal value or not equal type
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
?	Ternary operator
Logical Operator	
&&	Logical And
	Logical Or
!	Logical Note

When writing a code, the user usually wants to perform a certain action only on the occurrence of a specific condition. For this purpose, the conditional statements shown in Fig. 2.10 can be implemented. In JavaScript the following statements can be used:

- The *if* statement which will execute a certain action only if a specified condition is true
- The *else* statement, if the precedent condition turns out to be false will execute another operation
- The *else if* statement to specify a new condition to be tested if the first condition is false
- The *switch* statement that specifies many alternative blocks of code to be executed. It is a faster alternative to using the If statement followed by many Else if statements.

```

1 // If statement + Else if statement + Else statement // Switch statement
2
3 var time = 15;
4 var greeting;
5
6 if (time < 10) {
7     greeting = "Good morning";
8 }
9 else if (time < 20) {
10    greeting = "Good day";
11 }
12 else {
13    greeting = "Good evening";
14 }
15
16
17
18
19
20
21
22
23
24

```

```

// Switch statement
switch (new Date().getDay()) {
case 0:
    day = "Sunday";
    break;
case 1:
    day = "Monday";
    break;
case 2:
    day = "Tuesday";
    break;
case 3:
    day = "Wednesday";
    break;
case 4:
    day = "Thursday";
    break;
case 5:
    day = "Friday";
    break;
case 6:
    day = "Saturday";
}

```

Fig. 2.10 – JavaScript code for the conditional statements

Furthermore, programs are generally presented as blocks of functions, designed to execute a particular task such as a calculation. One of the main reasons to apply a function could be to write reusable code that can produce different results each time the code is run.

A JavaScript function is defined with the ‘*function*’ keyword followed by its ‘name’ and the parentheses ‘()’. The function’s code has to always be placed in between curly brackets ‘{}’ as shown in Fig. 2.11.

When a function is generated, it can be invoked (called) in any part of the code. The invocation can be an event (when a user clicks on a button) or it can be automatic (self-invoked function). Once the function reaches, the ‘return’ element found in between the curly brackets it will stop executing.

```

1 // Funtion that converts fahrenheit degrees in celcius degrees
2
3 function toCelsius(f) {
4
5     return (5/9) * (f-32);
6 }
7
8 //Invocation of the toCelsius function
9 var x = toCelsius(77);

```

Fig. 2.11 – JavaScript code for a function

Together with HTML and CSS, JavaScript is a powerful tool for web creation. While HTML and CSS are mostly implemented for rendering the structure and the style of static web pages, JavaScript enables the user to create dynamically updating content, animating images, control multimedia and interactivity. Another useful feature that comes with JavaScript, is

the functionality built on top of the core JavaScript language, the so-called Application Programming Interfaces (APIs). They are constructs made available in programming languages to allow developers to create complex functionality more easily. APIs abstract more complex code, providing some easier syntax to use in its place, they use one or more JavaScript objects.

For example, if the user wants to program 3D graphics, it will be easier to do it by implementing an API written in a high-level language such as JavaScript or Python, rather than using some low-level language such as C++ that directly controls the computer GPU).[3]

JavaScript APIs fall into two categories:

1-Browser APIs:

Built into the browser, they are able to expose data from the computer environment and perform complex activities. Some examples of browser APIs are:

- Document Object Model (DOM) API that allows to manipulate HTML and CSS
- Geolocation API that retrieves geographical information
- Canvas API that allows to create animated 2D and 3D graphics

2-Third Party APIs:

Third party API are not built into the browser, therefore the user needs to grab their source code and information from different the web sources. Examples of third party APIs are:

- The Twitter API
- The Google Maps API

To recap, when a web page is loaded, the JavaScript code will be executed by the JS engine after, after the HTML and CSS have been combined together in order to construct the webpage. By running the JS code last, it can be ensured that the structure and style of the document are set before starting to dynamically modify the content.

2.2 JavaScript Libraries

2.2.1 Data-Driven Documents Library (D3.js)

Data-driven documents (D3.js) is an open-source JavaScript library built to work with documents based on data, usually in .csv, .tsv and JSON format. It produces dynamic and interactive data visualization on web browsers by using HTML, SVG and CSS. [4]

Together with the different properties that D3 provides, some of the most important are:

- DOM Manipulation: modifying the documents with the DOM API can be tiresome, as it would require for a manual iteration and bookkeeping of the temporary state. By implementing D3, it is possible to access an element and modify its rendering in one code line as shown in Fig. 2.12.

```
1 // DOM Manipulation using the DOM API
2
3 var paragraphs = document.getElementsByTagName("p");
4 for (var i = 0; i < paragraphs.length; i++) {
5     var paragraph = paragraphs.item(i);
6     paragraph.style.setProperty("color", "white", null);
7 }
8
9 // DOM Manipulation using D3.js
10
11 d3.selectAll("p").style("color", "white");
```

Fig. 2.12 – DOM Manipulation through JavaScript and D3.js

- Dynamic Properties: styles, attributes, and other properties can be specified as *functions of data* in D3. Which means that data can drive your styles and attributes as shown in the example below in Fig. 2.13.

```
1 // Dynamic Properties
2
3 d3.selectAll("p").style("color", function() {
4     return "hsl(" + Math.random() * 360 + ",100%,50%)";
5 });
```

Fig. 2.13 – Function for randomly color paragraphs

- D3 provides the transition () function. This is a powerful tool because internally, D3 works out the logic to interpolate between the values and find the intermittent states.

2.2.2 Crossfilter Library (crossfilter.js)

Crossfilter is a JavaScript library for exploring large multivariate datasets in the browser. [5]

It is perfectly suited for big data analytics as it can support datasets with millions of records at extreme speeds (<30ms). Records either can come from a data file or even be built directly in the script, in the form of an array of JavaScript objects or primitives.

Ultimately, in order to be more user friendly crossfilter supports APIs. Some of the APIs are show in Tab. 2.4 below.

Tab. 2.4 – Corssfilter’s APIs for manipulating and filtering data

Crossfilter APIs	
crossfilter ([records])	Constructs a new crossfilter. If records is specified, simultaneously adds the specified records.
crossfilter.add (records)	Adds the specified records to this crossfilter
crossfilter.remove (predicate)	Removes all records that match the current filters from this crossfilter.
crossfilter.all ()	Returns all of the raw records in the crossfilter, independent of any filters.
crossfilter.groupAll ()	A convenience function for grouping all records and reducing to a single value.
Dimension	
crossfilter.dimension (value [, is Array])	Construcs a new dimension using the specified value accessor function. The function must return naturally-ordered values.
Dimension.filter (value)	Filters the records such that this dimension’s value matches value, and return this dimension

<code>dimension.filterExact(value)</code>	Filters records such that this dimension's value equals value, and returns this dimension
<code>dimension.filterRange(range)</code>	Filters record such that this dimension's value is greater than or equal to range[0], and less than range[1], returning this dimension
Group	
<code>dimension.group([groupValue])</code>	Constructs a new grouping for the given dimension, according to the specified groupValue function. By default, the group's reduce function will count the number of records per group. In addition the groups will be ordered by count.

2.2.3 Dimensional Charting Library (Dc.js)

Dc.js is a JavaScript chart library with native crossfilter support, allowing highly efficient exploration on large multi-dimensional datasets. It uses D3.js to render charts in a CSS-friendly SVG format. Charts rendered using the dc.js library are data driven and responsive, therefore they provide instant feedback [6]. Once data has been filtered, dc will set a filter on the respective dimension object. Firstly, the chart sends a redraw message to the other charts belonging to the chart group, via the Chart Registry. Secondly, all the chart belonging to the designated group pull new data from their respective crossfilter groups and transit from the old data to the new data, therefore giving the live interaction.

The different dc charts and their respective methods are grouped into classes. Under the *rownChart* class it is possible to find:

- `dc.rowChart`
- `elastic(Boolean)`
- `gap([number])`
- `renderTitleLabel([Boolean])`
- `ect.`

Supported graph types include:

1. Scatter-plot chart, shown in Fig. 2.14, is a two-dimensional chart. It is often called the correlation chart as it allows to see if two variables are correlated. Later versions are supported with a trend line to make the relationship more visible.

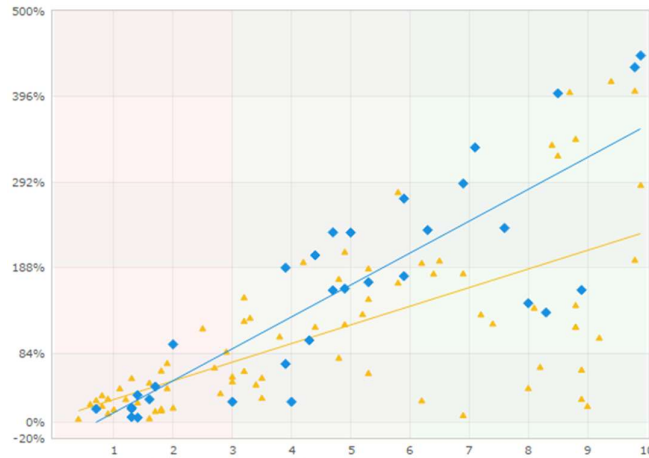


Fig. 2.14 – Scatter-plot chart

2. Bar chart, shown in Fig. 2.15, is used to summarize univariate data sets. Typical bar charts are formed by horizontal axis (the area of interest) and the vertical axis (the response variable). The width of the bars is usually equal and does not have any meaning regarding the underlying data set. Histograms are a type of bar charts, used to depict a certain distribution of data.

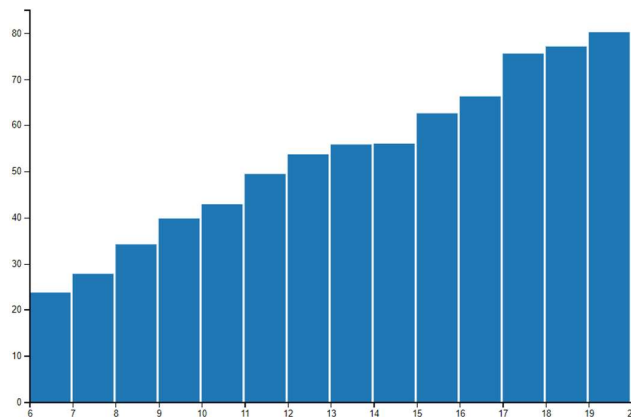


Fig. 2.15 – Bar chart

3. Pie chart, shown in Fig. 2.16, is a circular statistical graphic, divided into slices to illustrate slice proportion. The arc length is proportional to the number of data it represents. Usually they are used to show comparison as the biggest slice can be easily identified.

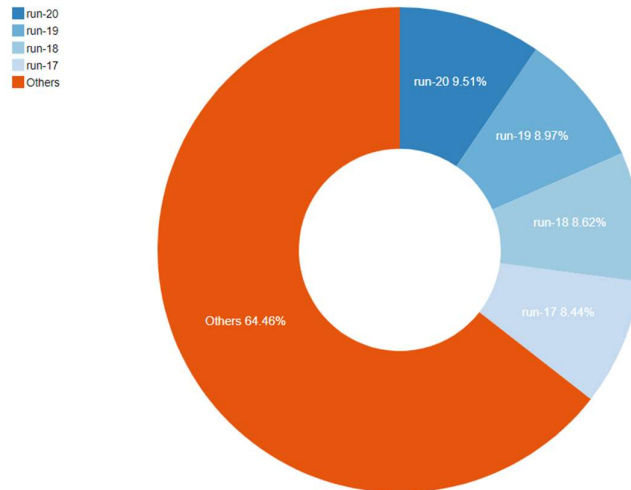


Fig. 2.16 – Pie chart

When creating a dashboard, there is need to upload the above JavaScript libraries in the code, in a specific order. First, D3.js will be loaded as it will allows manipulating all the HTML and CSS elements. Secondly, crossfilter is used to create the necessary dimensions ad groups for filtering the dataset and lastly dc.js which will be uploaded. Dc.js, by leveraging with the two previous libraries, will render our charts.

3. Data Exploration

3.1 Big Data analysis through data visualization

The aim of this project is to display D3, Dc and crossfilter eligibility in processing big data sets. The McKinsey study defines Big Data as “datasets whose size is beyond the ability of typical database software tools to capture, store, manage and analyse.” The data size ranges from a dozen of terabytes (10^{12} bytes) to multiple petabytes (10^{15} bytes). Big Data breaks into four dimensions known as the 4 Vs:

1. Volume: relates to the size of the data records incoming
2. Velocity: relates to how frequently data is generated and processed. This breaks into batch data processing, where data is collected over a period of time to real time data processing where there is a continual input process and output of data.
3. Value: data value can be defined as the valuable information a certain dataset brings. Quite often this information is hidden in the first place and through data analytics all the ‘dirty’ data can be deleted and patterns can be found.
4. Variety: Nowadays data type comes from a great variety of industries such as the financial services, health care, manufacturing, transportation, engineering design. The focus of this project will be on data provided by maritime sensors.

In collaboration with associate professor Giovanni Besio from the civil, chemical and environmental engineering department, a thorough analysis of maritime data was conducted. The aim is to display the different trends of wind and wave properties in the Mediterranean Sea in a period that goes from 1979 to 2017, on an interactive dashboard. The quantities analysed can be seen below:

- Significant Wave Height (H_s): in marine forecast it represents the average of the highest one-third (33%) of waves, measured from the crest that occur in a given period of time. It is measured because larger waves are more significant than smaller waves such due to their impact on coastal erosion and navigation. Statistical distribution of the wave heights is approximated by a Rayleigh distribution as shown in Fig. 3.1. Given the properties of the Rayleigh distribution, it is possible to encounter a wave with double the height of the significant wave height.

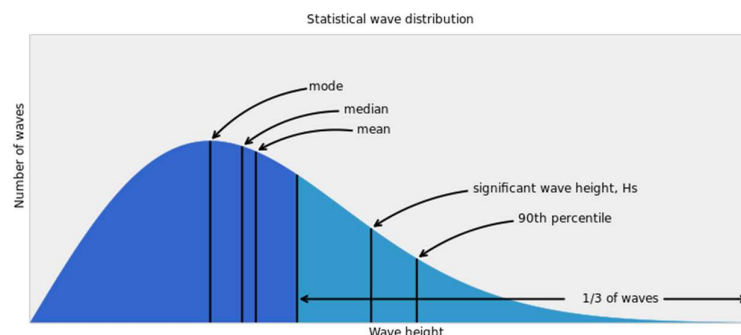


Fig. 3.1– Statistical wave distribution

- Wave mean & peak Period (T_m, T_p): waves with the same height and shape have the same period T (periodic). However in natural sea states this regularity is not present, the motion is irregular and chaotic. Natural states are represented by a superposition of a big number of regular wave trains traveling in different directions with different heights and periods. These trains are called wave components and form the wave spectrum. The wave period represents a key parameter for defining a sea state. Worth noticing, for a given wave height, the larger the period, the more energetic and powerful the swell. Whereas the peak period T_p corresponds with the period of the most energetic wave component, the mean period T_m is calculated as an average from the whole ensemble of wave components. (Only the most energetic component defines the T_p , every single component contributes to the T_m).
- Mean and peak wave direction (Dir_m, Dir_p): wave directions are measured clockwise from North and are evaluated through the mean over the whole 2D spectrum as shown in Fig. 3.2.

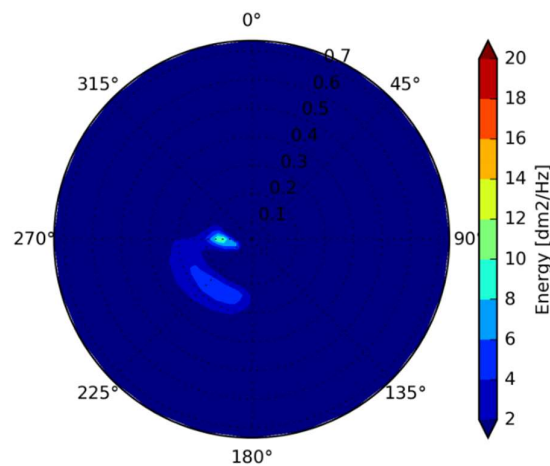


Fig. 3.2– Prediction directional wave spectrum

- Peak wavelength (L_p): the peak wavelength is the length where the spectrum reaches its highest intensity. It corresponds to the wavelength calculated with the peak wave period.
- Mean wavelength (L_m): it corresponds to the wavelength calculated using the mean wave period.
- Directional Spreading (Spread): is the distribution of wave energy with direction. The smaller the directional spread, the larger the amount of wave energy concentrated around the wave direction.
- Wind speed: wind speed is defined as the hourly wind speed average measured at 10m above sea level. Detection of the wind speed is done following the nautical convention (i.e. 'coming from') relative to true north positive clockwise. For example, 0 degrees means from north to south and 90 degrees means from east to west.

The mathematical meaning of these parameters can be found in Appendix A.

The data retrieved from the Mediterranean Sea belongs to a substantial amount of locations, each belonging to a specific cardinal point such as latitude and longitude. A decision was taken to plot all the sites on a map with markers. In order to create the map Leaflet has been used which is a JavaScript library. The creation of the map includes the following steps:

1. Create an index.html file with the basic html structure (see chapter 2, section 2.1)
2. Include the Leaflet CSS in the <head> element via a <link> tag.
3. Include Leaflet.js after Leaflet CSS, either in the <head> or in the <body> element
4. Add a <div> element with its id attribute. The map will show here
5. The markers corresponding to the data retrieved from stations in the Mediterranean Sea are added.

Once the index.html file is run, a map appears on the web page as shown in Fig. 3.3.

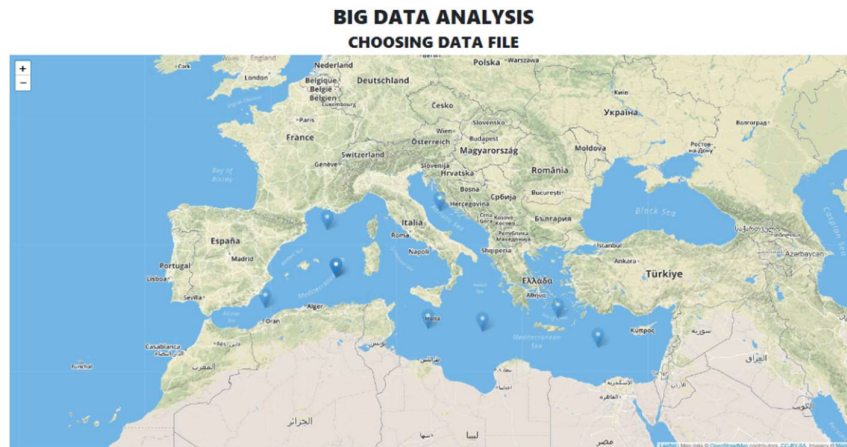


Fig. 3.3–Leaflet map with markers

Moreover, in order to accelerate the access to each data file, code is added to allow the user to get in query string the coordinates of the point clicked in the previous page. The click event will load a file specific for the coordinates. Then the name of the file will be 'lat;lng.csv', where 'lat' and 'lng' are respectively the latitude and the longitude passed via query string as shown in Fig. 3.4.

```
244
245   var urlParams = new URLSearchParams(window.location.search);
246   var coordinates = urlParams.get("coordinates");
247   if (!coordinates || coordinates.indexOf(";") < 0) {
248     alert('Missing or wrongly typed coordinates');
249   }
250   let latLng = coordinates.split(";");
251   $("#coordinates").html("Coordinates for LAT: " + latLng[0] + " and LNG: " + latLng[1]);
252
```

Fig. 3.4– Code retrieving a data file with specific latitude and longitude via query string, after a mouse click event on the map

When confronted to data, the first issue is about understanding its nature in order to find the best way to display it. After thorough research and data analysis, a general agreement was settled in displaying the data through:

- A year bar chart for displaying and filtering by year the data records
- A month row chart for displaying and filtering the data by month
- Two heat maps describing the frequency aggregation of data records with the same [Significant wave height (H_s); Wave peak period (T_p)] and [Significant wave height (H_s); Wave mean direction (Dir_m)]. This allows to identify if there are any specific sea conditions that have a recurrence.
- A wind sunburst chart is used to replace the traditional wind rose. A sunburst chart can be seen as a pie chart with an inner and outer ring. In the inner ring it is possible to find the different cardinal wind directions and on the outer ring, there are the different wind intensity bands corresponding to each cardinal direction.
- Two additional sunburst charts displaying in the inner rings the cardinal wind directions and on the outer rings, the significant wave height for the first chart and the wave peak period for the second.

Before starting the charts rendering, some data manipulation was necessary. In the first place, wind speed was presented in two separate columns.

- West-East wind direction; as a consequence negative wind speeds indicate an East-West wind direction. Similarly, for the South-North wind direction negative values refer to a South-North wind direction.

For instance, constants have been defined in order to represent the cardinal directions, wind speed intervals, wave peak period and significant wave height intervals as shown in Fig. 3.5. Secondly, the `setWindDirection(d)` is called to parse each data records and return the number of the records belonging to each cardinal direction. Additionally, the wind speeds needed to be classified into different wind speed intervals. Hence, the compound direction had to be defined by calculating its module. Furthermore, the cardinal directions, the function parses through the wave peak period and the significant wave height columns return the number of data belonging to the respective intervals. Also, the `getWaveDirection(d)` function is used to parse through the mean wave direction records and return the number of data belonging to each angle interval previously set.

```

3  const interval1 = "0-40";
4  const interval2 = "40-80";
5  const interval3 = "80-120";
6  const interval4 = "120-160";
7  const interval5 = "160-200";
8  const interval6 = "200-240";
9  const interval7 = "240-280";
10 const interval8 = "280-320";
11 const interval9 = "320-360";
12
13
14 const NORTH = "N";
15 const WEST = "W";
16 const EAST = "E";
17 const SOUTH = "S";
18 const NO_WIND = "No wind";
19
20
21 const MONTHS = [
22   "January",
23   "February",
24   "March",
25   "April",
26   "May",
27   "June",
28   "July",
29   "August",
30   "September",
31   "October",
32   "November",
33   "December"
34 ];
35
36 const SPEED1 = "0-3";
37 const SPEED2 = "3-6";
38 const SPEED3 = "6-9";
39 const SPEED4 = "9-12";
40 const SPEED5 = "12-15";
41

```

Fig. 3.5– Constants created for the setWindDirection(d) function

Once the necessary data creation is settled, the data file can be loaded into the browser via the DOM by default, all data will be passed as an array of objects. It is important to notice that the first row of the file does not get printed. This is due to the data object being loaded by D3, which uses these column names as object properties and therefore, they are converted to object keys. Each array element is presented as a string, thus, it needs to be coerced into a number in order to be manipulated.

Now that the file is uploaded in the browser, the data visualization begins by applying the crossfilter library. By using the necessary dimensions, groups and implementing the dc.js classes, the user will be able to generate the charts (See Appendix C).

Lastly, with the use of the resetFilter (filter) function, reset buttons are created for better data exploration. Through the bootstrap library the layout of the webpage is created.

In order to conclude, once the code is uploaded on the webpage an interactive dashboard is generated. If the user tries to filter data on a chart, he will see all charts uploading the new filtered data with a high computational speed. This shows how D3, Dc and crossfilter are useful tools to manipulate big data. Among the traditional uses of these data sets in weather forecasting for marine navigation, useful insights concerning high sea energy locations can be depicted through data visualization techniques. These locations can be transformed into wave energy plants for sustainable energy production.

3.2 Real-time data visualization

Nowadays the majority of data visualization projects consists of static sets of data that are pulled upon request of the user. Therefore, the data sets only get updated when the user manually refresh the data file. However, this model is slowly becoming obsolete in many applications. With the constant raise of produced data, the user would need to spend most of his time updating the data sets. To overcome this issue real-time data visualization techniques can be implemented.

Real-time data visualization is applicable when the user application needs to keep a “pulse” on, and monitor data passively. This means that on the dashboard the charts will update automatically as long as the connection is kept open. This method can be applied to different areas such as:

- Stock/Financial markets
- Industrial manufacturing
- Scientific Laboratories
- GPS monitoring

This project focuses on the possible applications in the engineering industry environment. More specifically, the implementation of real-time data visualization techniques using D3, Dc, and crossfilter, on design optimization projects.

3.2.1 Design Optimization

To provide the reader with a better understanding, design optimization is defined as the creation of an algorithm to explore the design space and find a variety of high performing designs of an object. The problem formulation includes:

- Design variables: they are parameters that are controllable from the designer. Design variables can represent the geometric quantity such as the thickness of a structural object. These variables can be either continuous (represented by real numbers), discrete (the number of bolts in a beam) or Boolean.
- Constraints: constraints functions determine the feasibility of all the solutions found during the process of optimization. They indicate whether a design should be considered as a possible option. Constraints can set the value of the function to be exactly equal, greater or less than a certain value.
- Objectives: these functions describe the goals of the optimization problem. During the problem formulation, it needs to be specified whether the goal is to minimize or maximize the value of each object function. When the problem has one objective function, it will be called a *single-objective optimization* and it usually presents a single solution. Likewise, when the problem has more than one objective function it is called a *multiple-objective optimization* and usually have a range of solutions.

- Methods: once the optimization problem is set, it can be solved using a variety of approaches. When a problem cannot be solved directly, optimization algorithms are implemented. These algorithms can be divided in two different categories: the *deterministic methods*, that apply a series of defined steps in order to obtain a solution and *stochastic methods* which apply a certain level of randomness while searching for a solution.

In this case, the aim is to optimize the shape of a dagger board of a sail boat as shown in Fig. 3.6. The goals are to maximize the vertical force and minimize the drag coefficient which represent the objective functions. This is then a multi-objective optimization (MOO) problem. There are twelve design variables as shown in Tab. 3.1 and one non-linear constraint which corresponds to the lateral force on the dagger board. For this project all the design variables are bounded and for the non-linear constraint an inequality has been used.

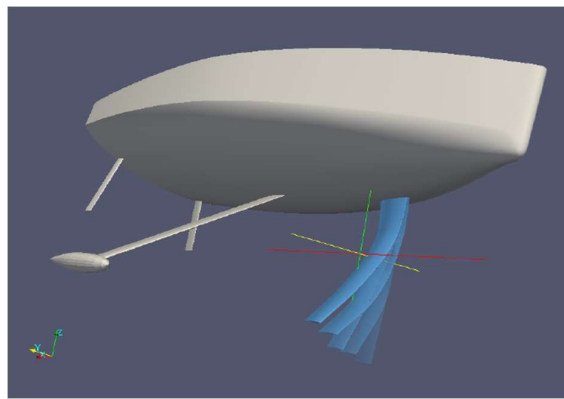


Fig. 3.6– Computer rendering of sail boat to which we wish to apply a design optimization process to the daggerboard

Tab. 3.1 – Quantities of interest in the design optimization process

12 Design variables x	3 airfoil x-c _l	x1, x2, x3
	3 airfoil x-A	x4, x5, x6
	3 wing chord x	x7, x8, x9
	2 wing dihedral x	x10, x11
	1 wing sweep x	x12
2 Objective functions (of)	Drag	obj_fn_1
	Vertical Force	obj_fn_2
1 non-linear constraint	Lateral force	nln_ineq_con_1

The Multi-Objective Genetic Algorithm (MOGA) method was used alongside the surrogate-based optimization (SBO) to perform the MOO. In a SBO, a surrogate model (also known as meta-model) is created to approximate, as show in Fig. 3.7, an original high fidelity model (such as the Computational Fluid Dynamics (CFD) simulations or more expensive experiments). The surrogate acts as a data fit to the observations so that new results can be predicted without recurring to expensive experiments which can either bi-numerical or physical. Once the surrogate is built, the optimization method is applied.

The MOO can be expensive, hence SBO us an economical solution to perform MOO. Moreover, the observations can be used for data mining, data analytics and for initial screening to provide information on the sensitivities of the data.

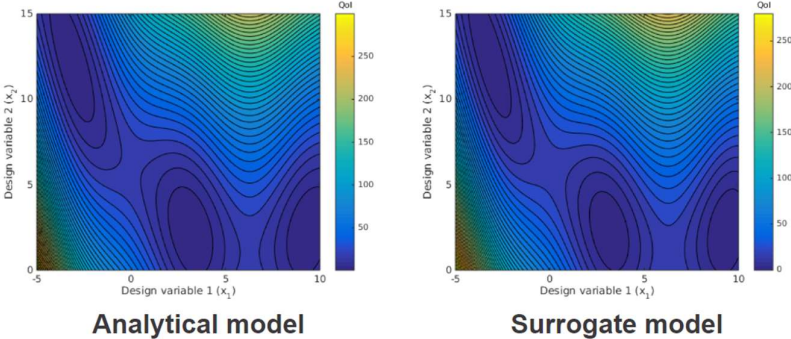


Fig. 3.7– Comparison of a design space obtained by analytical process and a surrogate based process

More information concerning the MOO on the sail boat can be found on Wolf Dynamics website. [8]

A high number of simulations need to be performed by the design optimization methods in order to produce significant results for the user. Therefore, it is good practice to implement data visualization in order to depict data trends in an easier manner. Additionally, the process of data analysis and visualization can be accelerated by implementing a real-time data acquisition.

3.2.2 Real-time data acquisition through a websocket server

The simulations have already been ran hence the data has already been stored. In order to simulate a real-time acquisition a websocket server (a bi-directional client/server connection) returning random data has been generated.

For this project the following instruments in Tab. 3.2 have been implemented.

Tab. 3.3 – Different programming tools implemented for real-time data acquisition

Programming Languages	Python – server side
	JavaScript – client side chart rendering
JavaScript Libraries	D3.js
	Crossfilter.js
	Dc.js
	Python’s tornado library for websockets, IOloop and web

Tornado is a python web framework library [7], mainly used to create servers capable of managing non-blocking input/output (I/O). Usually, when a user requests a web page, a server is called to elaborate and pack the response. The response is then sent via a Transmission Control Protocol (TCP) connection to the client. Once the response is sent to the client, the connection is closed and if the client needs a new request, a connection has to be restored which sometimes might be an issue. With non-blocking servers, such as websockets servers, this is no longer a problem. With a websocket server a permanent bidirectional client/server connection is created and capable of exercising multiple processes while awaiting for new data to be sent. Closing a websocket server connection has to be done manually (for example by closing the web browser).

IOloop is a method that communicates to tornado to listen for websocket’s connections (it tells tornado ‘A connection has been made, keep listening to the server until the connection is manually closed’).

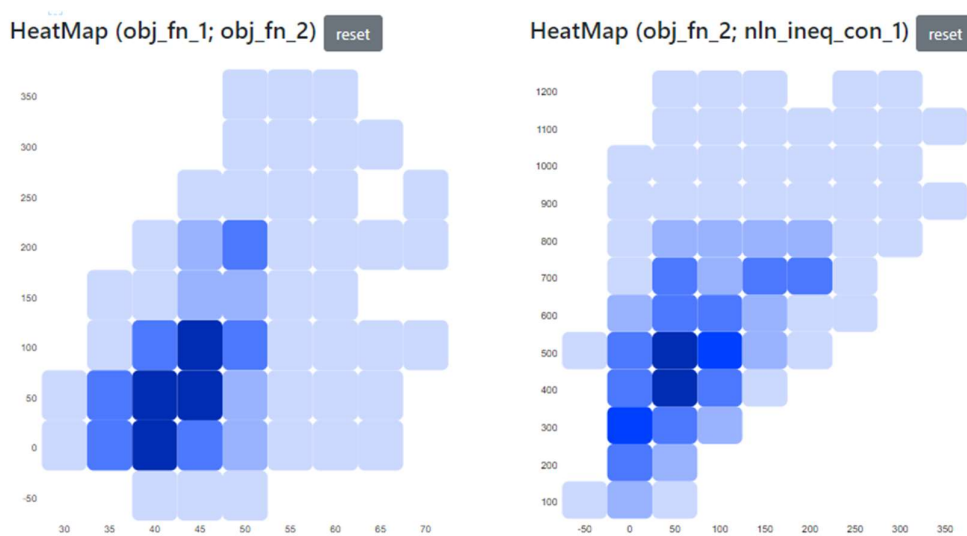
Once all libraries are downloaded, the creation of the websocket server begins. First, all packages are imported. Second, using tornado’s websocket a handler class is created. The handler contains three functions:

1. *open (self)* function, inside which a notice requesting a three seconds delay for sending data once the connection is opened is passed.
2. *on_close(self)* function, returns the message “Connection closed” once the client/server connection is closed

3. `send_data (self)`, it is the most important function where a json object of random data is built and sent via the `self.write_message ()` function.

IOloop is then used to create a timeout that will send data periodically. Furthermore, a websocket web app is built and set to listen on port 8001. In order for the socket to become active, a client-side code needs to open the connection on that port.

Now that the server is ready, it is possible to create the dashboard for data visualization, using D3, Dc and crossfilter libraries. The process is analogous to the one implemented for the first project on maritime data. A decision was made to re-create multiple count heatmaps depicting the frequency distribution for pairs of design variables and object functions. In the original data set, data visualization allows the user to directly locate a data thickening region in the $(obj_fn_1; obj_fn_2)$ and the $(obj_fn_2; nln_ineq_con_1)$ heatmaps as shown in Fig. 3.8. This region enables, on a statistical basis, to locate the optimal regime conditions for the dagger board. In the real time simulation, random data included in a certain range is given by the server, therefore the data thickening locations will not appear.



To recap, this project emphasizes on how D3, Dc, and crossfilter libraries are suitable for real time data analytics (data acquisition and visualization). Making their implementation a potential analytical approach for analysing data in future projects within the DICCA department.

4. Conclusions

In this dissertation has been investigated the use of the JavaScript open-source libraries known as D3.js, Dc.js and Crossfilter.js to perform interactive data visualization. The main objective was to understand whether these libraries could withstand data analysis on large amount of data which is a process known in computer science as Big Data analysis. Google Chrome web browser, although any modern browser would have worked, has been used in order to display the analyses; this required a specific skill set for web creation with the implementation of HTML and CSS programming languages.

To determine the feasibility of this project, datasets from the civil, chemical and environmental engineering department (DICCA) of University of Genoa have been provided. Firstly, maritime records from the Mediterranean Sea over a period of thirty-eight years were analyzed. Although it was possible to recreate some of the graphs used in weather fore-casting engineering (for example the 2D wave spectrum), there were difficulties in generating certain types of charts due to the lack of chart features found in the Dc.js chart library. For instance, it was not possible to recreate the wind and significant wave height rose charts. Instead, it was opted to implement the sunbursts chart, which, were able to display the data just as effectively. The chance of filtering data on the interactive dashboard demonstrated the true power of crossfilter's library. It was possible to easily identify periods of the year where the sea transported more energy through the waves. For this project data concerning only one location was provided. However, if the user had data related to every location, he could be able to identify within the Mediterranean Sea the regions that could theoretically represent the ideal location for a kinetic wave power station. This approach is known as "the feasibility and location study" and is the first step taken by companies producing energy in deciding where to set a new power station.

During the second part of the thesis the capability of these libraries to work with real-time data was tested. Although it was not possible to work with ongoing simulations, it was decided to test the real-time feature by connecting a dashboard (that was generated with data collected from a design optimization project) to a websocket server by using a python library known as Tornado. The server would send random bounded data to the dashboard. The result was successful as all the charts were changing in correspondence to the data that was being sent. This demonstrated the potential use of this data visualization approach for real time computer simulations such as design optimization (DO), computational fluid dynamics (CFD). This would result in faster data gathering and displaying, reducing the costs.

Although the use of these JS libraries may seem restrictive for data visualization and analysis, D3, Dc and Crossfilter represent a valuable alternative to their competitors in displaying and analyzing data as they do not come with any cost and the fact of being open source allows for a faster generation of new features.

Appendix

For the sake of brevity, only the most relevant scripts are reported.

Set wind direction function:

```
79 v function setWindDirection(d) {
80
81     var wd = [];
82
83     var westEast = d.uw;
84     var southNorth = d.vw;
85
86     var absWE = Math.abs(westEast);
87     var absSN = Math.abs(southNorth);
88     var windForce = Math.sqrt(Math.pow(absWE, 2) + Math.pow(absSN, 2));
89
90 v     if (southNorth > 0) {
91         wd.push(SOUTH);
92     }
93 v     else if (southNorth < 0) {
94         wd.push(NORTH);
95     }
96
97 v     if (westEast > 0) {
98         wd.push(WEST);
99     }
100 v    else if (westEast < 0) {
101        wd.push(EAST);
102    }
103
104 v    if (wd.length === 0) {
105        wd.push(NO_WIND);
106    }
107
108    d.windDirection = wd.join("-");
109
110 v    if (windForce <= 3) {
111        d.windSpeed = SPEED1;
112    }
113 v    else if (windForce > 3 && windForce <= 6) {
114        d.windSpeed = SPEED2;
115    }
116 v    else if (windForce > 6 && windForce <= 9) {
117        d.windSpeed = SPEED3;
118    }
119 v    else if (windForce > 9 && windForce <= 12) {
120        d.windSpeed = SPEED4;
121    }
122 v    else if (windForce > 12) {
123        d.windSpeed = SPEED5;
124    }
125
126 v    if (d.Hs < 1) {
127        d.waveHeight = "0-1";
128    }
129 v    else if (1 < d.Hs && d.Hs < 3.5) {
130        d.waveHeight = "1-3.5";
131    }
132    else { d.waveHeight = "High"; }
133
134 v    /* *** */
135
136 v    if (d.Tp < 3) {
137        d.peakPeriod = "0-3";
138    }
139 v    else if (3 < d.Tp && d.Tp < 6) {
140        d.peakPeriod = "3-6";
141    }
142 v    else if (6 < d.Tp && d.Tp < 9) {
143        d.peakPeriod = "6-9";
144    }
145 v    else if (9 < d.Tp && d.Tp < 12) {
146        d.peakPeriod = "9-12";
147    }
148    else { d.peakPeriod = "Long"; }
149
150 }
```

Get wave direction function:

```
169 ▼ function getWaveDirection(d) {
170
171     var result2 = [];
172
173     var wavedirection = d.Dirm;
174
175 ▼   if (wavedirection < 40) {
176       result2.push(interval1);
177   }
178
179 ▼   else if (wavedirection > 40 && wavedirection < 80) {
180       result2.push(interval2);
181   }
182
183 ▼   else if (wavedirection > 80 && wavedirection < 120) {
184       result2.push(interval3);
185   }
186
187 ▼   else if (wavedirection > 120 && wavedirection < 160) {
188       result2.push(interval4);
189   }
190
191 ▼   else if (wavedirection > 160 && wavedirection < 200) {
192       result2.push(interval5);
193   }
194
195 ▼   else if (wavedirection > 200 && wavedirection < 240) {
196       result2.push(interval6);
197   }
198
199 ▼   else if (wavedirection > 240 && wavedirection < 280) {
200       result2.push(interval7);
201   }
202
203 ▼   else if (wavedirection > 280 && wavedirection < 320) {
204       result2.push(interval8);
205   }
206
207   else result2.push(interval9);
208
209
210   return result2;
211
212 }
```

Reset Filter function:

```
742 ▼ function resetFilter(filter) {
743     let chart = null;
744 ▼   switch (filter) {
745       case "wind":
746         chart = windDirChart;
747         break;
748       case "year":
749         chart = yearChart;
750         break;
751       case "month":
752         chart = monthChart;
753         break;
754       case "heatmap1":
755         chart = heatmapOne;
756         break;
757       case "heatmap2":
758         chart = heatmapTwo;
759         break;
760       case "sunburst1":
761         chart = windSunburstChart;
762         break;
763       case "sunburst2":
764         chart = waveheightSunburstChart;
765         break;
766       case "sunburst3":
767         chart = wavepeakperiodSunburstChart;
768         break;
769       case "wave":
770         chart = waveDirChart;
771         break;
772 ▼   case "ALL": {
773     yearChart.filterAll();
774     monthChart.filterAll();
775     heatmapOne.filterAll();
776     heatmapTwo.filterAll();
777     windSunburstChart.filterAll();
778     waveheightSunburstChart.filterAll();
779     wavepeakperiodSunburstChart.filterAll();
780     dc.redrawAll();
781     return;
782   }
783 }
784 ▼ if (chart) {
785   chart.filterAll();
786   dc.redrawAll();
787 }
788 }
```

Websocket Handler:

```
13 ▼ class WebSocketHandler(websocket.WebSocketHandler):
14
15 ▼     def check_origin(self, origin):
16         return True
17
18     # on open of this socket
19 ▼     def open(self):
20         print('Connection established.')
21         # ioloop to wait for 3 seconds before starting to send data
22 ▼         ioloop.IOLoop.instance().add_timeout(
23             datetime.timedelta(seconds=3), self.send_data)
24
25     # close connection
26 ▼     def on_close(self):
27         print('Connection closed.')
28
29     # Our function to send new (random) data for charts
30 ▼     def send_data(self):
31         print("Sending Data")
32         # create a bunch of random data for various dimensions we want
33         x1 = random.uniform(0, 0.8)
34         x2 = random.uniform(0, 0.8)
35         x3 = random.uniform(0, 0.8)
36         x4 = random.uniform(0, 1)
37         x5 = random.uniform(0, 1)
38         x6 = random.uniform(0, 1)
39         x7 = random.uniform(0.2, 0.4)
40         x8 = random.uniform(0.2, 0.4)
41         x9 = random.uniform(0, 0.4)
42         x10 = random.uniform(0, 10)
43         x11 = random.uniform(0, 40)
44         x12 = random.uniform(-16, 6)
45         obj_fn_1 = random.uniform(30, 75)
46         obj_fn_2 = random.uniform(-50, 400)
47         nln_ineq_con_1 = random.uniform(0, 1300)
48
49     # create a new data point
50 ▼     point_data = {
51         'x1': x1,
52         'x2': x2,
53         'x3': x3,
54         'x4': x4,
55         'x5': x5,
56         'x6': x6,
57         'x7': x7,
58         'x8': x8,
59         'x9': x9,
60         'x10': x10,
61         'x11': x11,
62         'x12': x12,
63         'obj_fn_1': obj_fn_1,
64         'obj_fn_2': obj_fn_2,
65         'nln_ineq_con_1': nln_ineq_con_1
66         # 'x': time.time()
67     }
```


Leaflet Map code:

```
52 ▾ (function () {
53
54     var POINT_WITH_DATA = [38.754083, 5.756836];
55
56 ▾     var EXTRA_POINTS = [
57         [41.967659, 4.96582],
58         [34.885931, 18.852539],
59         [43.197167, 15.029297],
60         [36.597889, -0.571289],
61         [35.101934, 13.974609],
62         [35.88905, 25.576172],
63         [33.72434, 29.179688]
64     ];
65
66     var mymap = L.map('mapid').setView(POINT_WITH_DATA, 7);
67
68 ▾     L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access_token=pk.eyJ1IjoibWFwYm94I
69         maxZoom: 18,
70         attribution: 'Map data © <a href="https://www.openstreetmap.org/">OpenStreetMap</a> contri
71             <a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, ' +
72             'Imagery © <a href="https://www.mapbox.com/">Mapbox</a>',
73         id: 'mapbox.streets'
74     }).addTo(mymap);
75
76
77
78     var popup = L.popup();
79 ▾     function onMapClick(e) {
80         popup
81             .setLatLng(e.latlng)
82             .setContent("You clicked the map at " + e.latlng.toString())
83             .openOn(mymap);
84     }
85     mymap.on('click', onMapClick);
86
87
88
89
90     var marker = L.marker(POINT_WITH_DATA)
91         .on('click', () => markerOnClick(POINT_WITH_DATA))
92         .addTo(mymap);
93
94 ▾     for (const point of EXTRA_POINTS) {
95         L.marker(point, { opacity: 0.5 })
96             // to make other points clickable
97             .on('click', () => markerOnClick(point))
98             .addTo(mymap);
99     }
100
101 ▾     function markerOnClick(point) {
102         window.location.href = "data.html?coordinates=" + point[0] + ";" + point[1];
103     }
104
105 })();
```

References

- [1] Decamp, P. (2012). [online] Media.mit.edu. Available at:
https://www.media.mit.edu/cogmac/publications/decamp_phd_thesis.pdf
- [2] W3schools.com. (2018). *HTML Tutorial*. [online] Available at:
<https://www.w3schools.com/html/default.asp>
- [3] MDN Web Docs. (2018). *Introduction to web APIs*. [online] Available at:
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
- [4] Bostock, M. (2018). *D3.js - Data-Driven Documents*. [online] D3js.org. Available at:
<https://d3js.org/>
- [5] Square.github.io. (2018). *Crossfilter*. [online] Available at:
<http://square.github.io/crossfilter/>
- [6] Dc-js.github.io. (2018). *dc.js - Dimensional Charting Javascript Library*. [online]
Available at: <https://dc-js.github.io/dc.js/>
- [7] Tornadoweb.org. (2018). *Tornado Web Server — Tornado 5.1 documentation*. [online]
Available at: <http://www.tornadoweb.org/en/stable/>
- [8] Wolfdynamics.com. (2018). *Wolf Dynamics - Multiphysics simulations, numerical optimization, and data analytics*. [online] Available at:
<http://www.wolfdynamics.com/component/content/article.html?id=82>

Nomenclature

Notations

H_s	Significant wave height[m]
T_m	Mean period [s]
T_p	Peak period [s]
Dir_m	Mean direction [degN]
Dir_p	Peak period [degN]
L_m	Mean wavelength [m]
L_p	Peak wavelength [m]
uw	West-East wind velocity at 10m [m/s]
vw	South-North wind velocity at 10m [m/s]

Definitions and acronyms

D3	Data-Driven Documents
Dc	Dimensional Charting
HTML	HyperText Mark-up Language
CSS	Cascading Style Sheet
W3C	World Wide Web Consortium
JS	JavaScript
HLL	High Level Language
API	Application Programming Interfaces
DOM	Document Object Model
SVG	Scalable Vector Graphics
DO	Design Optimization
MOO	Multi-Objective Optimization
MOGA	Multi-Objective Genetic Algorithm
SBO	Surrgate-based Optimization
CFD	Computational Fluid Dynamics
TCP	Transmission Control Protocol
I/O	Input/Output