Table of contents

[1. Abbreviations 5](#_Toc48088907)

[2. Introduction 6](#_Toc48088908)

[3. Objectives 7](#_Toc48088909)

[4. Literature review 8](#_Toc48088910)

[5. Background 9](#_Toc48088911)

[5.1. Programming Scheme 9](#_Toc48088912)

[5.2. Hydrology 11](#_Toc48088913)

[5.2.1. Rainfall 11](#_Toc48088914)

[5.2.2. Runoff 12](#_Toc48088915)

[6. Design Methodology 20](#_Toc48088916)

[6.1. Requirements 20](#_Toc48088917)

[6.2. Architecture 20](#_Toc48088918)

[6.3. External libraries 21](#_Toc48088919)

[6.4. Data sources 23](#_Toc48088920)

[6.4.1. AEMET 24](#_Toc48088921)

[6.4.2. EAUK 25](#_Toc48088922)

[6.4.3. FEMA 26](#_Toc48088923)

[6.4.4. METEOIT 26](#_Toc48088924)

[6.4.5. METEOSTAT 27](#_Toc48088925)

[6.4.6. NOAA 28](#_Toc48088926)

[6.4.7. USGS 29](#_Toc48088927)

[6.4.8. World Bank 29](#_Toc48088928)

[6.5. User interface 30](#_Toc48088929)

[7. Implementation 31](#_Toc48088930)

[7.1. Data 31](#_Toc48088931)

[7.1.1. Functions 33](#_Toc48088932)

[7.2. Analyze 34](#_Toc48088933)

[7.2.1. Stats 35](#_Toc48088934)

[7.2.2. Hydro 37](#_Toc48088935)

[7.2.3. NN (Neural Networks) 41](#_Toc48088936)

[7.3. Visualize 41](#_Toc48088937)

[7.3.1. Functions 42](#_Toc48088938)

[7.4. Maps 44](#_Toc48088939)

[8. Case study 1: rainfall analysis 47](#_Toc48088940)

[8.1. Description 47](#_Toc48088941)

[8.2. Library usage 47](#_Toc48088942)

[8.3. Loading a map for visualization 48](#_Toc48088943)

[8.4. Data 48](#_Toc48088944)

[8.4.1. Data analysis 51](#_Toc48088945)

[9. Case study 2: Derivation of a flooding hydrograph 54](#_Toc48088946)

[10. Discussions 55](#_Toc48088947)

[10.1. Challenges and limitations 55](#_Toc48088948)

[11. Conclusions 57](#_Toc48088949)

[11.1. Future work 57](#_Toc48088950)

[12. References 58](#_Toc48088951)

[13. Appendix 62](#_Toc48088952)

[13.1. Case study 1 62](#_Toc48088953)

[13.2. Usages 71](#_Toc48088954)

**List of figures**

[Figure 1. Sequence diagram for API calls. 9](#_Toc48088754)

[Figure 2. REST API design (TechnoMerge, 2020). 10](#_Toc48088755)

[Figure 3. Example of a hyeteograph and mass curve (Chow, et al., 1988). 11](#_Toc48088756)

[Figure 4. Conceptual idea of travel time (USDA, 2015). 14](#_Toc48088757)

[Figure 5. Dimensionless curvilinear unit hydrograph and equaivalent triangular hydrograph. 16](#_Toc48088758)

[Figure 6. Bucket model representation (Santos, et al., 2018). 18](#_Toc48088759)

[Figure 7. Dependencies of the data module. 32](#_Toc48088760)

[Figure 8. Dependency diagram of analyze module. 34](#_Toc48088761)

[Figure 9. Dependencies of the stats class. 36](#_Toc48088762)

[Figure 10. Dependency diagram of hydro component. 38](#_Toc48088763)

[Figure 11. Dependency diagram of map module. 44](#_Toc48088764)

[Figure 12. Station location. 50](#_Toc48088765)

[Figure 13. Response from console after first request. 51](#_Toc48088766)

[Figure 14 Example of statistics of raw data vs cleaned data. 52](#_Toc48088767)

[Figure 15. Identification of time gaps and data gaps. 53](#_Toc48088768)

**List of tables**

[Table 1. Relationships of m and peak rate flow (USDA, 2007). 17](#_Toc48088769)

[Table 2. Functions inside the data module. 33](#_Toc48088770)

[Table 3. Sources inside the data sources component. 33](#_Toc48088771)

[Table 4. Components of the analyze module. 35](#_Toc48088772)

[Table 5. Functions in the stats component. 36](#_Toc48088773)

[Table 6. Functions in the hydro component. 38](#_Toc48088774)

[Table 7. Dependency diagram of visualize module. 43](#_Toc48088775)

[Table 8. Functions within visualize module. 43](#_Toc48088776)

[Table 9. Functions of map module. 45](#_Toc48088777)

**List of equations**

[Equation 1. Average precipitation using arithmetic mean. 12](#_Toc48089007)

[Equation 2. Average precipitation using Thiessen method. 12](#_Toc48089008)

[Equation 3. Water balance equation. 13](#_Toc48089009)

[Equation 4. SCS intial abstraction 14](#_Toc48089010)

[Equation 5. SCS maximum storage retention. 14](#_Toc48089011)

[Equation 6. SCS runoff. 14](#_Toc48089012)

[Equation 7. SCS Time of concentration. 15](#_Toc48089013)

[Equation 8. Kerby-Kirpich method for time of concentration. 16](#_Toc48089014)

[Equation 9. Kerby time of concentration. 16](#_Toc48089015)

[Equation 10. Time lag and time to peak. 16](#_Toc48089016)

[Equation 11. Dimensionless hydrograph equation. 17](#_Toc48089017)

[Equation 12. SCS Peak Discharge based on peak rate factor. 18](#_Toc48089018)

[Equation 13. SCS time to peak and time step duration. 18](#_Toc48089019)

[Equation 14. Total volume of the DRH . 18](#_Toc48089020)

[Equation 15. Total olume of the DRH in terms of units of depth. 18](#_Toc48089021)

[Equation 16. Discrete convolution equation. 19](#_Toc48089022)

[Equation 17. Bucket model initial flow. 19](#_Toc48089023)

[Equation 18. Bucket model infiltration. 20](#_Toc48089024)

[Equation 19. Bucklet model subsequent flows. 20](#_Toc48089025)

[Equation 20. Bucket model final discharge. 20](#_Toc48089026)

**List of code snippets**

[Snippet 1. Example code for source usage. 25](#_Toc48088782)

[Snippet 2. Example of function call per module or component. 32](#_Toc48088783)

[Snippet 3. Access to data module functions. 34](#_Toc48088784)

[Snippet 4. Object example for retrieval function. 34](#_Toc48088785)

[Snippet 5. Example configuration object for transform function. 34](#_Toc48088786)

[Snippet 6. Example usage of analyze functions. 35](#_Toc48088787)

[Snippet 7. Example of arrays for usage on array class. 37](#_Toc48088788)

[Snippet 8. Array example for arithmetic function. 39](#_Toc48088789)

[Snippet 9. Example object for thiessen function. 40](#_Toc48088790)

[Snippet 10. Example object for syntheticalc function. 40](#_Toc48088791)

[Snippet 11. Example object for dimensionless hydrograph. 40](#_Toc48088792)

[Snippet 12. Example object for hyetogen function. 41](#_Toc48088793)

[Snippet 13. Example objects for unithydrocons function. 41](#_Toc48088794)

[Snippet 14. Example usage for visualize function. 43](#_Toc48088795)

[Snippet 15. Example argument for chart function. 44](#_Toc48088796)

[Snippet 16. Object example for table function. 44](#_Toc48088797)

[Snippet 17. Object examples for styles function. 44](#_Toc48088798)

[Snippet 18. Argument for loader function. 46](#_Toc48088799)

[Snippet 19. Configuration object for renderMap function. 46](#_Toc48088800)

[Snippet 20. Example objects for Layers function. 47](#_Toc48088801)

[Snippet 21. Example of HydroLang library on HTML file. 49](#_Toc48088802)

[Snippet 22. Adding a map to screen. 49](#_Toc48088803)

[Snippet 23. Example commands for downloading data. 50](#_Toc48088804)

[Snippet 24. Adding the station into the map. 51](#_Toc48088805)

[Snippet 25. Using the transformation function. 52](#_Toc48088806)

[Snippet 26. Code for calling statistics for the data. 53](#_Toc48088807)

[Snippet 27. Function calls for time and data gaps identifiers. 53](#_Toc48088808)

[Snippet 28. Downloading the data. 54](#_Toc48088809)

# Abbreviations

* Standardize the use of client, user, developer.
* API: application program interface.
* External library (JavaScript): a library of functions that have been created by third parties which aim to provide with standardized and optimized functionalities for specific problems.
* Lazy-loading: asynchronous loading of interfaces such as images, websites, external libraries, etc.
* EcmaScript: commonly used international agreements for client-side code development of the World Wide Web.
* CDN: content distribution network, used for calling external libraries dynamically.
* REST: acronym for Representational State Transfer.
* AEMET: Spanish Meteorological Agency (Agencia Estatal de Meteorologia).
* CORS: acronym for cross-origin requests in browser technology.
* EAUK: Environmental Agency of the United Kingdom.
* FEMA: United States Federal Emergency Management Agency.
* METEOIT: Italian meteorological network.
* Object (JavaScript Environment): available variables, functions, definitions on the language. It is the main idea for the language’s development (object-based paradigm).
* NOAA: National Oceanic and Atmospheric Administration of the United States.
* USGS: United States Geological Service.
* DWD: Deutscher Wetterdiesnt (German Weather).
* Syntactic sugar: ways of coding that become easier to write.
* Blob: Binary Large Object; a chunk of data.
* PRF: Peak rate flow.
* cfs: Cubic feet per second.
* cumecs: Cubic meters per second.
* UH: Unit hydrograph.
* DUH: Dimensionless unit hydrograph.
* SCS: United States Soil Conservation Service.
* ERH: Effective rainfall hydrograph.

# Introduction

An environmental study requires data. The data depends solely on the typoe of study that is been carried out and the resolution of the data neededfor the research. Many times, a sresearcher has problems just in acquiring data; the amount of sources that can be found online can become overwhelming. Furthermore, once the data has been obtained, it requires preparation that has takes time and efforts from the use rpersepctive. Hydrologic libraries have been used in the scientific communities to simplify the workload that a researcher most due to obtain basic and reliable results. A complete hydrological analysis is done considering

# Objectives

HydroLang uses client-side technology to perform different routines which aim towards the retrieval of data and perform basic analyses from a data and environmental perspective. This has been done by creating a series of components within the framework working as a library in which the user selects a particular function for data manipulation. The main structure of the HydroLang is described in more details on the methodology section of this thesis.

The components defined and created within the library follow the next structure:

* Data
* Analyze
* Maps
* Visualization

The data module has been written so that it can retrieve environmental data from open sources such as governmental agencies, free API’s, etc. which solely depend on the availability of information and the final aim of the user. The data is obtained, manipulated and output through functionalities within the data module.

The analyze module contains three different components: stats, hydro and NN. Stats component contains different tools that process data and performs statistical analyses which provide a cleanup version of the data as required from the user. The hydro component contains subroutines that build up together a complete hydrological analysis. This includes features that work within the framework established by renown authors and community accepted practices.

The maps module consist of mapping tools that allow the user to include layers of data obtained from previous modules or owned by the user (provided they are on the same format) so that it can be included within a map. The layers can be flooding extents, markers or layer of analyzed data. Finally, the visualization module provides the user with charts, reports and other visual tools that aid the user to better analyze the obtained data from previous modules or previously owned.

The modules can be accessed using the chaining properties of high-level modules in JavaScript, and after the functions have been used, they can be saved on array-like objects.

# Literature review

The research question that was used to find similar approaches to what HydroLang was to become was the following:

*“Which applications are available on the literature and on the web with respect to open source, web-based environmental, hydrologic, and hydraulic analyses?”*

To answer the latter previous to the development of the project, the following resources were used:

* GitHub
* Science Direct
* Google Scholar
* Research Gate
* Google

From the results, 5 were papers and projects that used JavaScript; 3 used Java, C, C++; 12 used Python, and finally 6 used PHP or any other language. Overall, the papers can be categorized on the type of results that are obtained based mainly on environmental data, data retrieval from environmental agencies, visualizations applications that used map engines or other web services for deployment, and statistical sorting and manipulation of data.

Table 1. Literature review of web applications.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Study* | *Citing Articles* | *Answer to RQ* | *Main conclusion* | *Search Engine* | *Application type* |
| (Swain, et al., May 2015) | 43 | Review of software falling in 2 main scope definitions. | Review of web GIS software, web development software. The selected applications were evaluated depending on criteria featuring the end means (spatial functions, web configuration, implementation language, etc). | Google/Science Direct/SpringerLink | GIS applications, hydraulic-hydrology |
| (Ames, et al., November 2012) | 112 | Open software used to download open source data for hydrologic and hydraulic analysis | A collaborative project using CUAHSI platform. Able acess to data from internet-based platforms with flexibility on the software to enable new features by introducing plugins. The software has evolutioned since its deployment in 2012. | Science Direct | GIS applications, hydraulic-hydrology, data manipulation. |
| (Raseman, et al., June 2019) | 5 | Library for multi-objective optimization problems in environmental decision making | Interactive visualization library created using HTML, Javascript and CSS deployed using Github repository. The main idea is to use k-mean approach for multi-objective optimization. 5 different libraries created that aim to integrate different purposes. | Science Direct | Environmental decision making, k-means clustering |
| (Heistermann, et al., February 2013) | 86 | Library for geoprocessing of radar images for hydrological purposes. | Open source ibrary of geoprocessing tools which aims to do the preprcoessing of radar imagery data for further hydrological analyses. Written in python and having a community based approach. | Research Gate | GIS applications, radar imagery, hydrology |
| (Ames, 2018) |  | Open source infrastructure for hydrological data | Open source repository with different applications fo hydrological analysis. It allows the user to upload the data and manipulate it for visualization, analysis and work depending on the type fo data. Is part of the CUAHSI framework. Nany papers derived from the platform. | Google | GIS applications, Hydrology, Data management |
| (Foglia, et al., April 2015) | 37 | Open source software | A Horizon 2020 project financed by EU comission consisting in a platform with free open software encompassing several types of approaches for hydrological analysis. | Research Gate | GIS applications, hydraulic-hydrology, data manipulation. |
| (Alcantara Souffront, et al., October 2019) |  | Open repository | Discussion about the hydoinformatic challenges that developing countries have. Specifically introducing a respository in which many tools can be obained written in a variety of languages and addressing different approaches. | Frontiers In | GIS, hydrologic validation, Statistics |
| (Dawson, et al., July 2007) | 243 | Web application | Website hosting a toolbox for hydrological analysis, specifically focusing on evaluation metrics for performance testing. Somewhat outdated considering the scope of the review, but important takeouts can be obtained from the development of the app. | Elsevier | Hydrologic model performance test assessment for forecasting |
| (Delipetrev, et al., 2014) | 32 | Web application | Part of the CUAHSI framework, the web app aims to maange data, support water resources modeling and optimization of resources. Written in PHP, Ajax, Javascript and Java. | Science Direct | GIS, hydrologic validation, Statistics |
| (Roberge, et al., 2017) |  | Web application | Web application for browsing within a webpage regarding data from USGS and german WSV of about 10,000 stream gages worldwide. It uses Javascript, html and CSS. Specifically uses knockout.js and hosted in github | Open research software | Hydrology, GIS visualization |
| (Brendel, et al., July 2019) | 1 | Web application | Web application that aims to probide data analysis features on precipitation. Specifically, hydrological preanalyses for projects. It was written in R with the Shiny web app framework. The app obtais data from different governmental services. | Science Direct | Hydrology, statistical analysis |
| (Swain, et al., November 2016) | 29 | Web application | Web development app focusing on creating a friendly environment for developing applications. The platform serves as a host and provides the developer with tools to generate code easier. Written in Python, HTML. | Science Direct | Hydrological repository |
| (Conner, et al., November 2013) | 18 | Web application | Part of the CUAHSI framework, the application uses relational databses to store data through the framework. Serves as a point for data retrieval. | Science Direct | Hydrological database system |
| (Tyralia & Schumann, 2013) | 4 | Open repository | An interactive framework for developing hydrological models. Everything written in Python. It is built with a core module that serves for ussage between the different applications within the library. | Research Gate | Hydrological model library |
| (Kraft, et al., June 2011) | 43 | Open repository | CMF is a library of hydrological processes for the creation of models, modular and connectible to other models using multiple hypothesis background and based on FVM. Written in C++ but compiled to be using Python. It encompasses many libraries with a core module as a wrapper. | Science Direct | Hydrological library for model creation |
| (Schellekens, et al., 2019) | 180 | Open repository | Part of the open stream project of Deltares, wflow is a set of python programs that can be run using the command line to perform hydrological analysis. Consists of a setup api that wraps over an extensive library. | Github | Hydrological analyses |
| (Ulmo, 2020) |  | Application | Python library for retrieval of weather data from governmental and open source sources. It serves a a basis as a start up for hydrological analyses. It has different libraries build up depending on the institution that it is needed to obtain the info from. | Github | Meteorological data retrieval |
| (MetPy Developers, 2019) |  | Open library application | MetPy is a collection of Python written tools that enable users to read, visualize and perform calculations on weather data. The repository contains data structures with several options aiding on different libraries in Python. No clear wrapper found in the docs. | Github | Data analysis |
| (Georgy, 2016) | 89 | Open library | LHMP is a lumped conceptualized modelling playground designed so that stakeholders and general public are able to visualize hydrological processes. Written in Python, with several LHMP models wrapped up in a core application. | Github | Lumped models |
| (Roberge, 2016) |  | Open library | Hydrofunctions is a library of commandas that is able to retrieve data from USGS NWIS service and extract data by the use of Python modules in order to plot the information and serve it on interactive maps. Written mainly on Python using Jupyter notebooks. | Github | Hydrological analyses |
| (PyMT Devs, 2016) |  | Open library application | PYMT is a python package that aims to provide with the necessary tools for couple modelling working with Basic Model Interface (BMI). It has grid mappers, time steppers, exchange data capabilities. Written in python. | Github | Modelling |

# Background

## Programming Scheme

The library was developed fully using JavaScript. Since its development in the ‘90s, the language has been used mainly on world wide web technologies and has been established as a standard scheme for web browsing. It is lightweight and compilated simultaneously on a browser, making it fully compatible with integrations of rendering markup languages like CSS and HTML. It is based on a prototypical document object model in which everything is considered objects: classes, functions, variables, etc. (Punchatz, 2017). Moreover and because of the support that commercial browsers have, the language has been used for the development of Application Programming Interfaces (APIs), which are sets of code that are ready to allow for the development of programs that would be very difficult or even impossible to do (Mozilla, 2020).

The scripting language has been used for a variety of approaches, including the development of web API’s, which are classified into four main types:

* Open APIs: available for public usage without any restriction. They are subcategorized on the type of service that they serve
* Partner APIs: an amount of higher degree of security is added; perhaps the need for a right or license to access data. This is usually created for particular data.
* Internal APIs: a company-based API that interconnects processes through it.
* Composite APIs: Combination of the features of the latter.

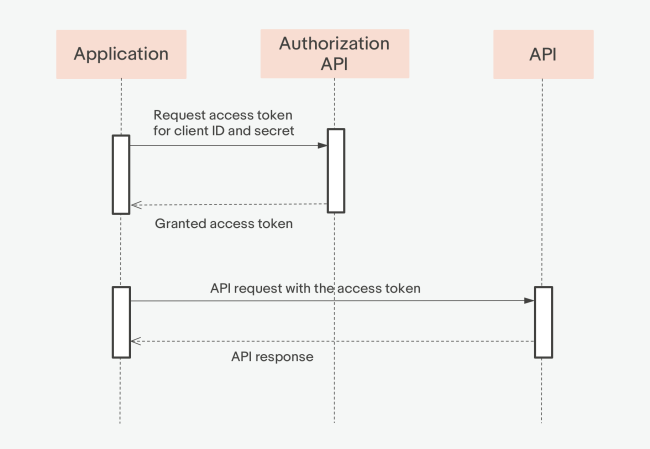


Figure 1. Sequence diagram for API calls.

They are also classified depending on communication level high, low, and web services APIs. The main focus of this research has been on web services, for which there is are particular communication and behavioral approaches (TechnoMerge, 2020):

* SOAP: Simple object access protocol. It has its own communication protocol and complex infrastructure in terms of the level of security and how messages are sent. It is a standardized protocol that only has XML format as proprietary for information diffusion but because of the significant level of security, the calls cannot be preserved.
* XML-RPC: a protocol using only specific XML format to transfer data; it uses a minimum bandwidth and is the antecessor of SOAP.
* JSON-RPC: uses JSON format to transfer data; RPC is one of the methods in which the microservice architecture can be communicated to.
* REST: Representational state transfer is an architectural style for the distribution of hypermedia systems introduced by (Fielding, University of California,). It is information-driven to build APIs. It depends on URI’s (Uniform Resource Identifier) HTTP convention, and the usage of JSON, CSV, and other formats for data retrieval.

REST APIs are the most popular nowadays because of its architecture and ease of implementation. Moreover, environmental agencies and public institutions the architecture more and more for providing data (Rouse, 2020).

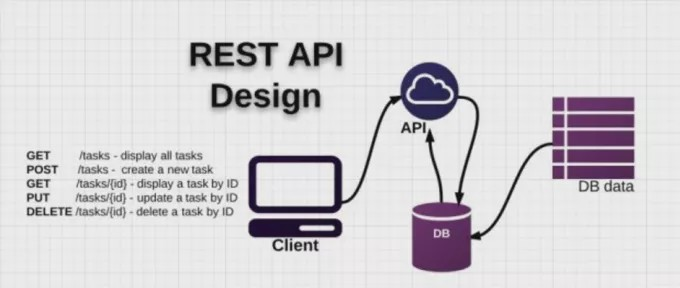


Figure 2. REST API design (TechnoMerge, 2020).

Data types

Objects

Arrays

Downloading using API interface requests.

Statistics and cleaning

## Hydrology

### Rainfall

Rainfall varies in space and time according to patterns that are subjected to global and local factors. This makes of rainfall an unpredictable phenomenon that is studied in terms of space and time. An example of the spatial distribution of rainfall is isohyetal maps, which show the rainfall recorded at a gaged point within an area (Chow, et al., 1988). To determine areal averages of rainfall, the arithmetic means method is used if the gages are uniformly distributed in an area.

Equation 1. Average precipitation using arithmetic mean.

If the stations are located in an area for which they can be more representative, then weights in regards to area percentage area added to calculate the average precipitation. This is called the Thiessen method, which assumes that the points inside a watershed are the same as the nearest gage.

Equation 2. Average precipitation using Thiessen method.

To sum up a rainfall event, it can be either summed in terms of total depth or intensity; from here the rainfall hyetograph and mass curve can be derived. It is usually represented graphically and used for calculation of losses, runoff, etc.

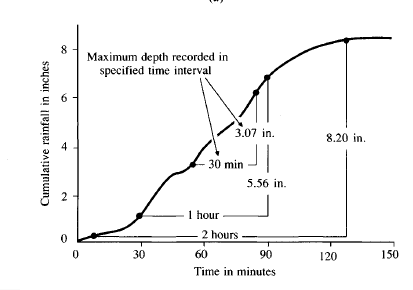
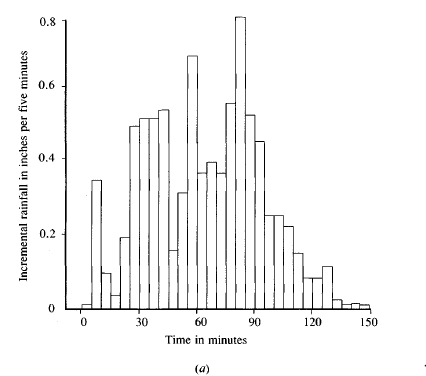


Figure 3. Example of a hyeteograph and mass curve (Chow, et al., 1988).

Rainfall data is usually widely available on higher temporal resolutions. This is linked by the number of stations that are recording the data and its quality. (Koutsoyiannis, 2003) highlights how many countries can only provide hourly or sub-hourly data in terms of daily gauges. Rainfall aggregation means summing the readings of an event at a temporal scale (5 min to hourly data) Disaggregation, on the other end, from high-resolution data obtain a coarser scale. Different methods have been applied on the literature such as empirical methods (Knoesen & Smithers, 2009), cascade models (Muller & Haberlandt, 2018), and, more recently, neural networks (Muller & Haberlandt, 2018). Evaluation metrics have shown the advantages and disadvantages of the approaches but this lies outside the scope of the research.

### Runoff

As described by (Sitterson, et al., 2017) the water balance equation is described as the difference between precipitation, evapotranspiration, storage, and groundwater. The result is the surface runoff that can be observed on land.

Equation 3. Water balance equation.

The relationship between rainfall and runoff has been studied for more than 100 years, with the first publication making a reference to a method being published by Thomas Mulvaney in 1851: the rational method. Being quite simple, it uses rainfall intensity, the drainage area, and a runoff coefficient for determining peak discharges in a basin (Beven, 2012). This runoff coefficient has been widely studied to determine new applicable scenarios in which it changes. This included graphical techniques (Beven, 2012), and more recently the usage of the unit hydrograph to account for the responses of a basin on a rainfall event (Xu, 2002).

The analysis of runoff has become its own focal study by means of models. A runoff model helps understand the hydrological phenomena in the complex system described before. They can be categorized depending on the approach that they take. Empirical structure studies the non-linearity of the inputs and outputs (Klemes, 1982). Conceptual structure takes simplified equations that represent water storage in a catchment (Vaze, et al., 2011); while physical structure applies physical laws and equations based on the real hydrologic responses expected from an area (Sitterson, et al., 2017).

Moreover, the models can also be classified based on the spatial processes which are involved in the catchment. They are lumped, semi-distributed, and distributed (Devi, et al., 2015). Lumped models do not consider spatial variability, and thus the entire catchment is modeled as one single unit. The inputs are averaged throughout the study area and are fast for computation but make many assumptions. Distributed models account for spatial variability, dividing the whole involved area into grids, and calculating all physical calculations per cell. Because of their approach, distributed models are data-intense and require longer computation times. Finally, semi-distributed models take ideas from both lumped and distributed models to make calculations.

#### Curve (SCS) method

An example of a commonly used empirical lumped model is the one developed by the SCS called the CN method. It considers the total drainage area of a watershed or subbasin for a rainfall event, but as a difference between the rational method, it also uses infiltration rates, losses and interceptions, and finally, the temporal distribution of the rainfall.

Rainfall is considered to be uniformly distributed upon a watershed. Initial abstraction is defined as the losses that a watershed has before runoff can begin. Losses are considered surface depressions, evapotranspiration, and infiltration. This is done using the empirical equation:

Equation 4. SCS intial abstraction

S is the maximum retention after runoff begins, calculated from:

Equation 5. SCS maximum storage retention.

The value of CN, or curve number, is calculated based on the soil’s cover type, hydrologic soil group, and antecedent moisture condition. The documentation of the method has been widely extended to include different types of each of the three components in which CN is constrained. The values for CN typically range between 30 and 100 (non-inclusive). Finally, runoff is calculated as the difference between the precipitation and initial abstraction, if applicable, per unit of time.

Equation 6. SCS runoff.

#### Time of concentration

The time that it takes for runoff to form and travel hydraulically through the most distant point of a catchment is defined as time of concentration. It is obtained by the summation of all traveling times in consecutive components of a catchment drainage and it is in direct relation with the shape and peak of a runoff hydrograph (USDA, 2015).

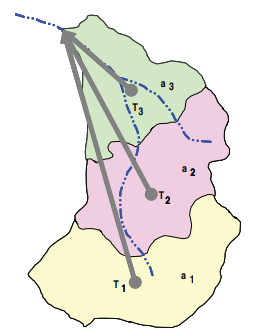


Figure 4. Conceptual idea of travel time (USDA, 2015).

Different approaches have been taken for the calculation of the time of concentration. The SCS watershed lag method spans through a large set of conditions by using the CN described previously. It uses the flow length, average catchment slope, and maximum potential retention.

Equation 7. SCS Time of concentration.

The Kerby-Kirpich equation considers that the total time of concentration is the sum of the overland time and the channel time (Sharify & Hosseini, 2011). For small watersheds where flow is important to consider for the travel time, the overland time is calculated using the overland flow length L; a conversion coefficient K which is 0.828 for metric system and 1.44 for imperial; and a dimensionless retardance coefficient N that depends on the terrain, ranging from 0.02 to 0.80. The channel travel time is calculated using another coefficient K that is 0.0078 for metric system and 0.0195 for imperial; S being the main channel slope, L the channel flow length.

Equation 8. Kerby-Kirpich method for time of concentration.

Finally, an approach considering Manning’s roughness coefficient was introduced by Kerby which uses the coefficient, the overland slope, and the longest path. The formulas change depending on the units which are used.

Equation 9. Kerby time of concentration.

It has been well established the relationships between time of concentration, lag time and time to peak. Lag time is the interval that lies between the center of mass of rainfall and the peak runoff (USDA, 2015); established as 60% of the time of concentration. Time to peak is the time required for the mass of rainfall to reach its highest peak, considered to be 70% of the time of concentration.

Equation 10. Time lag and time to peak.

#### Unit hydrograph

One of the most important ways to make a simple yet powerful hydrological analysis is by using the concept of the unit hydrograph. It is a direct runoff hydrograph that results in a total volume of one unit of rainfall that is uniformly distributed over a basin during a specified unit of time. It requires certain assumptions (Shaw, 1998):

* Effective rainfall should be distributed over a basin.
* Rainfall duration should be uniform during the unit duration; time is invariant.
* There is linearity, superimposition, and proportionality between one hydrograph and another.

A unit hydrograph can be derived from an observed hydrograph resulting from a storm that fulfills the conditions of rainfall uniformly distributed and more or less uniform intensity; analysis of multi-peaked flood hydrographs or from synthetic calculations based on the basin characteristics. Many methods have been developed for synthetic hydrographs. The NRCS developed a dimensionless unit hydrograph from the observation of natural unit hydrographs from catchments varying in size and location that is used as a base start when information is unavailable. It is derived from the gamma distribution in the forms of ratio of discharge Q/Q\_p, the Euler constant e, the gamma equation shape factor m, and the ratio of time t/t\_p (USDA, 2007).

Equation 11. Dimensionless hydrograph equation.

The only parameter that changes from the equation is the shape factor, which is linked to the peak rate factor that changes depending on the terrain characteristics varying from 101 to 566; the values on the lowest end represent less flat areas while the highest values are for very steep terrain (Chow, et al., 1988).

Table 2. Relationships of m and peak rate flow (USDA, 2007).

|  |  |
| --- | --- |
| m | PRF |
| 0.26 | 101 |
| 1 | 238 |
| 2 | 349 |
| 3 | 433 |
| 3.7 | 484 |
| 4 | 504 |
| 5 | 566 |

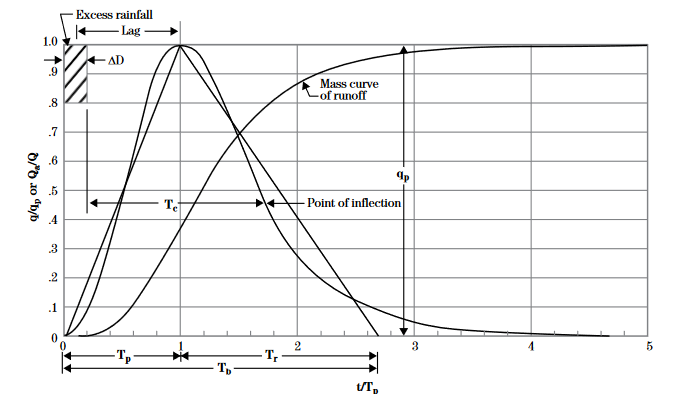


Figure 5. Dimensionless curvilinear unit hydrograph and equaivalent triangular hydrograph.

Equation 12. SCS Peak Discharge based on peak rate factor.

Finally, using the physical characteristics of the basin, the unit hydrograph based upon the idea of a triangular hydrograph. The time to peak is calculated as Tp, considering the total duration of excess unit rainfall D, calculated at the same time using the relation of 0.4\*lag time.

Equation 13. SCS time to peak and time step duration.

By multiplying the entries of the dimensionless unit hydrograph by the required rainfall duration time and the peak discharge, a synthetic unit hydrograph is generated. After subtracting all losses from the rainfall event, the cascade multiplication of the rainfall event entries and the synthetic unit hydrograph results in a flood hydrograph.

If the readings of both discharges of streamflow and a rainfall event are available, then a unit hydrograph can be derived empirically. This by using the concepts previously mentioned of proportionality and uniformity (Raghunath, 2006). Given an event, the total volume from a direct runoff hydrograph (DRH) is obtained as the total summation of the volume times the timestep of the readings of the effective rainfall hyetograph (ERH), once the baseflow has been subtracted.

Equation 14. Total volume of the DRH .

The total volume equivalent in units of depth will be the same as the division of the total volume DRH divided by the total area of the catchment. The unit hydrograph is then calculated as a time series in which the value is in a units of discharge by unit of volume, i.e. m^3/s/cm.

Equation 15. Total olume of the DRH in terms of units of depth.

Once derived, a flooding hydrograph for a rainfall event can be found by doing a discrete convolution of the UH. The rainfall hyetograph must be separated in terms of pulses in which each pulse represents units of equivalent depth The discharge ordinate of the UH is calculated as a convolution spanning through time depending on the number of pulses required.

Equation 16. Discrete convolution equation.

#### Bucket Model

A bucket model replaces the unit hydrograph by representing streamflow as a cascade of state variables that are calculated at each time step (Santos, et al., 2018). The variables included within are vary depending on the type of approach for the model that is being taken, but most well-established models consider evaporation, rainfall, surface flow, infiltration, and baseflow.

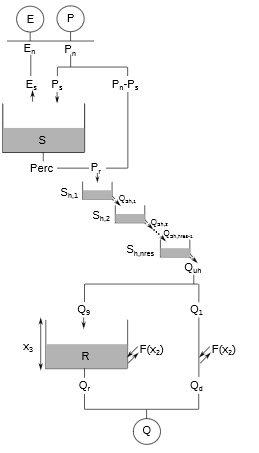


Figure 6. Bucket model representation (Santos, et al., 2018).

A simple bucket flow model calculates initial flow (using the following equation, where FC stands for field capacity and LU is the land use percentage:

Equation 17. Bucket model initial flow.

The soil moisture content is related to the different land uses scenarios. If the result of is bigger than the FC, then there is an overflow of the system. The overflow, then, is calculated as the difference between the and FC. If this is not true, then the overflow is 0. If the flow is bigger than 0, this means that there is interflow (), which is calculated as the multiplication of times the infiltration capacity (:

Equation 18. Bucket model infiltration.

The subsequent iterations consider the latter as a function for overland flow and interflow.

Equation 19. Bucklet model subsequent flows.

Total flow () then is calculated as the sum all the fluxes as:

Equation 20. Bucket model final discharge.

Neural Networks

The main constraint that the implementation of neural network has is the amount of data that is required for training a model. This issue had been a setback in the early years of artificial intelligence. The reliability of the data that is to be used for both feeder and result has a direct effect over the trained model (Vemuri, 1993). The number of training sets for both input and output of models will showcase the percentage of performance that the trained model will have over new datasets. This comes with a toll, of course on the performance of the environment in which the model has been developed. Nowadays, with the advancements in web technologies, browser applications that are being actively maintained, the usage of a native environment is becoming less necessary. The discussion done by (Ma, et al., 2019) showcases the capabilities that running deep learning on the browser

Neural networks have been used in hydrology for solving nonlinear problems such as rainfall runoff, groundwater applications and rainfall distributions. Regarding the latter, numerous papers have shown the advantages that using NN models have on prediction of patters and distribution. In comparison to the stochastic distributions.

# Design Methodology

## Requirements

A major motivation in the conception of HydroLang was to perform hydrological software tasks using only client-side. The main advantages of running the client-side executions are:

* It allows for immediate interaction between the user and the framework’s functionalities.
* A faster execution time; there are no delays from contacting external servers.
* The current usability of web browsers improves using client-side approaches and up to date with the common programming standards.
* The code runs using the technology that the user has within their environment, mainly using the CPU. If required, a dedicated unit like a GPU can also be used.

Code efficiency depends solely on the programming approach that has been taken for a given task. The better the algorithm is the less dependencies between each component required and better performance. A higher module cohesion is preferred, whilst modular dependency creates a code that is not easy to read, understand, or upgrade.

The modules in HydroLang are designed to:

* Being able to retrieve, transform, and upload data in several formats from external sources or local storage to be used on the framework.
* Perform data comprehension using relevant statistical metrics to assess the reliability of the data and conclude if any further actions are required.
* Provide a robust scheme for the study of precipitation events, with the final task of delivering measurable metrics that can be used for other purposes.
* Being able to create a simple neural network model and develop, train, and use models on the browser.
* Visualize and map data so that a user can have a clearer image of what has been achieved through the framework or from external files and what else might be needed for further improvement.

## Architecture

Using a library-oriented microservice architecture (Nadareishvili, et al., 2016), HydroLang has been developed to explore the functionalities that the dynamic coding style of JavaScript has, considering the principles and methodologies of reusable software libraries. By establishing the latter, functions are accessed using instantiation from classes and export functions as established on ECMAScript 6, in which the syntactic sugar has been reduced to chainage, achieved through the usage of basic dot notation (Engelschall, 2017). A decision was made to work within the library creating code wrapping of objects to access the functionalities on lower levels.

A modular approach was then adopted that allows for the protection of the execution context of the given modules while enabling usage of lower-level functions. In general, the following is the modular tree which the library follows:

* Instantiation from classes, instead of inheriting class properties.
* The arguments of all the functionalities in each module are objects or arrays. The objects change depending on the type of functionality.

The functions defined inside a module are exported through an import/export file. This is called the core of the library, which just extends the usage of each module. Finally, within the HydroLang class, the core is imported and used within the class constructor to create HydroLang as an object with the modules being its keys.

The latter approach has made of HydroLang an extendable, maintainable, and modular software in with low tolerances and high performance and becomes scalable and usable for community-based usage. The framework was developed using pure JavaScript with the help of external libraries. Although frameworks for front-end development could have been used, the idea that the software requires an up-to-date browser, with no need for further installations.

## External libraries

To achieve a cleaner, concise, and version compliant code, different libraries have been used within the framework to create new modules or components. The libraries that have been used are (but not limited to):

* JQuery.js
* Google Charts API
* Google Maps API
* Leaflet.js
* D3.js
* Tensorflow.js
* Documentation.js

JQuery is a JavaScript external library that creates a more compact version for requests and object manipulation in either synchronous or asynchronous environments for HTML and JavaScript (JQuery, 2020). It is mostly used for creating HTML documents and webpages enabling traverse manipulation, event handling, and implementation of features for requests using Ajax. The library is very handy regarding data retrieval because of its in-built functionalities and ease of use of calls and requests. It can handle any type of available requests for different types of API’s and databases and allows for cross-referencing manipulation.

Google Charts is a powerful tool developed by Google that enables software developers to use different charting modalities on webpages. The library contains a variety of options that are customizable as required from the developer. The API can be used free of charge and enables the user for a dynamic view of the data due to the protocols and tools implemented within (Google, 2020).

D3 is a library that is mainly focused on the manipulation of documents that are driven by data. The library allows the user for data binding using a Document Object Module and later apply transformation to the document. The library solves data manipulation by applying the most relevant and applicable algorithms that are dependent on the type of data that is to be handled. Just as with all the other libraries, it can be accessed using an API that is called whenever the functions are required (Bostock, 2020).

Google Maps is a service by Google which provides information about geographical regions and sites around the world. Its API has become the most popular library for map generation. It has always been an open-source library but has changed since 2014, afterwards the application changed to a billing scheme depending on the number of calls it on a monthly basis. Nevertheless, because of its ease of implementation, use, and flexibility for developers, it is widely used for web page creators, content creators, etc. The API has been written completely in JavaScript and it can be paired with any type of database to load data (Google Maps Platform, 2020).

Leaflet is an open-source interactive JavaScript library for map generation and rendering. It has been developed by Vladimir Agafonking and out to the public since 2010 (Agafonking, 2020). The main functionality that Leaflet has is its flexibility. Maps can be generated using different types of tile layers, markers, vector layers, image overlays, geoJSON files, interaction features, map control, etc. It can run on most web browsers for either desktop or mobile applications. It has also become one of the most popular libraries for generating maps mainly due to its community-based approach. Plugins have been developed that can add new features to maps in different ways. It is lightweight and can be used on its CDN version with no need for external dependencies. Overall, the API is very well documented and can be coupled with other libraries for creating beautified rendering styles.

Tensorflow is a platform dedicated to machine learning applications used for creating neural network models for different programming languages and different approaches. The platform was created by Google and it has a simple and flexible architecture that allows to create, train, and compile neural network models easily through its high-level APIs. Different options have been implemented by the platform to include different options for models such as recursive algorithms, feed-forward networks, convolutional cores, etc. It can also run on both the browser and if needed/required, a GPU (Tensorflow, 2020).

To generate the documentation from code, all of the features in the framework have used the script tags comments compliant with documentation.js. It is developed based upon Doc.js which reads through the code for comments that have /\*\*\*/ format and generates documentation from that is later published for usage. It is an open-source library that has been used since 2015 and is continuously upgraded to adapt to new programming styles (Documentation.js, 2020)

## Data sources

The data that is being retrieved using HydroLang comes from APIs developed by governmental or open-source institutions. Most of the data has been filtered and cleaned up before it is hosted within the databases, but this limitation is strongly linked to the type of data and geographic locations.

The data sources that have been included within the library have been selected because of the varied information that can be retrieved from them, the locations for which they can be obtained, and the cross-referencing (validation) of the data. The framework currently holds 8 different data resources, namely:

* AEMET
* EAUK
* FEMA
* METEOIT
* METEOSTAT
* NOAA
* USGS
* World Bank

All the data sources provide endpoints that can be tailored based on the information required by the user. These implementations serve as guides on how to integrate data based on external sources.

Data can be extensive and overwhelming, which becomes unhelpful if not filtered properly. Considering that HydroLang is for environmental data, specifically water-related, the final endpoints selected for usage can retrieve precipitation, evaporation, stations, and locations. The endpoints within the library represent an example usage for hydrological sciences and this idea can be extended to other scientific fields. Each endpoint described below has the following format:

1. export default {
2. stations: {
3. endpoint:
4. "https://some.url.com/endpointname",
5. params: {
6. name1: param1,
7. name2: param2
8. },
9. requirements: {
10. needProxy: true,
11. requireskey: true,
12. keyname: "keyname",
13. method: "GET ,POST",
14. },
15. }

Snippet 1. Example code for source usage.

The requirements on the sources explain whether the endpoint requires a proxy server, requires a key and/or the method for retrieval. Some queries need to have a specific format, while others can be flexible. It will all depend on the end data needs and the user should search and understand what each source can provide and what it requires. Links for appropriate documentation and guidance have been included in each source.

### AEMET

Aemet OpenData is an open REST API that allows for the distribution and utilization of meteorological and climatologic information from Spain and surrounding stations. The API provides data about observations, forecasts, climatologic values, satellite information, maps and graphs, maritime predictions and observations, ultraviolet readings, radars, and others (AEMET, 2020). The API requires the generation of a key to be passed into the header of the request, along with a “GET” method. The API also requires a proxy server to avoid CORS violations. The endpoints selected to be used within the library are the following:

* Station data (“stations”): conventional data gathering.
* Daily station data (“daily-stations”): retrieves precipitation data for a given station by giving as arguments initial and final observation dates.

### EAUK

The EAUK created a hydrological REST API which gives access to historical water information from the UK and surrounding areas. The API is on its alpha state and as of August 2020, it provides information regarding flood monitoring, hydrological data, environmental metrics, etc (Environmental Agency UK, 2020). The API is available freely for the public without the need for a key or any other parameter. The measurements that are provided by the source depend solely on the number of available measurement stations that have are being considered. Thus, for some locations, the data is unavailable until a certain date and further manipulation of the data might be considered for gap filling or outlier removal. The retrieved data can be in JSON, CSV, and XML format and uses a “GET” method for request and it does not require a proxy server. The library also provides geoJSON files.

For each request, added metadata is appended to the first lines of the retrieved object. Due to the available data within the application and the ease of use of the source, the following endpoints were considered for the library:

* Flood warnings (“flood-warnings”): considers instantaneous and historical flood warnings within a certain county or coordinates within the UK. It retrieves the data depending on the type of severity of the warning.
* Flooding prone areas (“flood-areas”): instantaneous and historical areas prone to flooding depending on the coordinates and search strings.
* Flooding prone stations (“flood-stations”): retrieves stations that have been historically cataloged for flooding areas. The endpoint has as a parameter the town location, catchment name, etc.
* List stations (“list-stations”): retrieves the stations that are available within a certain location by searching for towns, counties, or coordinates.
* Data stations (“data-stations”): finds historic rainfall information for a given rainfall station depending on the location, start and end date, station reference.

### FEMA

In accordance to open data initiatives, FEMA developed a read-only RESTful API that uses query string parameters to manage query selection. It includes datasets for major disaster declarations across the USA territories sorted depending on city, county, state, or nationwide (FEMA, 2020). The API has different operations that can be built depending on what the request for the end-user is in the forms of logical operators (i.e. equal, not equal, less than, logical negation, etc.). The formats that the API provides are JSON or geoJSON and allows for retrieving data through callback functions without the need for a key. The API accepts the “GET” method for retrieval and CORS has been implemented through the API, meaning that it does not require a proxy server.

The requests are appended with the information that has been requested by the user along with metadata. For the library, the following endpoints have been considered:

* Disaster declarations (“disaster-declarations”): it accepts as data fields the number of disaster, the county or state; the type of incident, and declaration title among others.

### METEOIT

METEOIT API contains meteorological data by giving access through different endpoints from the Italian territories. The service requires the creation of an account and the generation of a unique token that must be included within the header of the request. Because of the lack of implementation of CORS on the API, it requires a proxy server to be present with the request, accepting only “POST” methods for retrieval.

Since its launch, the API has deployed 3 different versions. Because of this, a new key is needed for accessing the latest version, while the other 2 versions have been deprecated but can still be accessed (MeteoNetwork API, 2020).

For the usage of the library, the following endpoints have been included:

* Daily station data (“station-daily”): retrieves meteorological information for a specific station during the last day previous to the request. It accepts as a parameter the station code, the observation data, and data quality. This last one is a parameter that can be added to every endpoint to enable the API to fill data in case there is missing.
* Last dates data in a station (“stations-lastdays”): it retrieves data for specific data for a single station. This endpoint is on the 2nd version of the API. It requires the station code, the data type for the request, and the validity of the station.
* Data per single station per date (“station-singledate”): it retrieves data for multiple stations on a single date. It requires as parameters date, and station validity.
* Data per multiple stations per single date (“stations-singledate”): it retrieves the data from multiple stations for a single date on nearby countries. It requires as parameters the date, country code, and station validity.
* Nearby stations (“nearstations”): it retrieves data from nearby stations to a query from longitude and latitude. It requires as parameters the coordinates, range of search, and station validity.

### METEOSTAT

METEOSTAT is a historical weather API done by private developers that use its own climate model to create projections and statistics of single weather stations on geographical points that are used as searching queries. This model allows the API to obtain data from several places throughout the world and guarantees is always up-to-date (METEOSTAT, 2020). The databases from the API has weather stations that regularly report observations and statistics; it also retrieves historical data from stations that are being provided by governmental organizations like NOAA, Deutscher Wetterdienst and Environment Canada. The information that is downloaded through the API contains all the variables that a certain weather station measure.

A relevant feature of the source is that it is completely free and allows for the connection to historical data. The developers have also included a bulk data endpoint in case large chunks of information are required by the user. The API requires a proxy server and a key to obtain the data and uses the “GET” method for retrieving information.

As part of the framework, the following endpoints from METEOSTAT were included:

* Find stations (“find-stations”): through query and limit as parameters, it obtains the stations that are within a country or basin.
* Nearby station(“nearby-stations”): it retrieves the information about stations that are within certain coordinates. The parameters for the endpoint at latitude, longitude, limit and radius.
* Hourly data per station (“hourlydata-stations”): obtains the hourly data from a specific station. It requires as parameter the station ID, the start and end for the query and the model for retrieval.
* Daily data per station (“dailydata-station”): it retrieves daily data from a single weather station given as parameters the station ID, and the date range.
* Hourly data point (“hourlydata-point”): an experimental version inside the API, it allows the user to retrieve data from any given station in hourly intervals. It requires as parameters the coordinates of a certain location, the start and end dates. It allows for 10 days per request.
* Daily data point (“dailydata-point”): another experimental endpoint, it allows h user to obtain daily information about a specific point in the world by passing by as parameters the coordinates and start and end dates. It allows only 370 days per request.
* Bulk data (“bulkdata”): it allows the user to retrieve information as bulk in gz format. The parameters vary depending on what is required by the end user.

### NOAA

NOAA’s API provides through its Climate Data Online (CDO) server access to current data for users. The REST API provides different types of datasets, data categories, types, locations, stations and data depending on what is required by the end user (NOAA, 2020). The coverage of data will depend on the minimum date available of the data as well as the overall technology used for data retrieval (for instance, radar data).

The API does not require a proxy server for access, but it does require an access key that must be included in the header of the request; it uses the “GET” method for retrieval. For the library, the following endpoints were considered:

* Datasets (“datasets”): retrieves the datasets that are available on the NOAA database system. It does not have any parameters.
* Available stations (“availablestations”): retrieves the available stations within a certain location, extent or coordinate within the US territory.
* Precipitation every 15 min (“prec-15min”): retrieves precipitation of a certain location every 15 min for a given period of time. The parameters it accepts are location id, station id, start and end date, units, limit, include metadata.
* Hourly precipitation (“prec-hourly”): retrieves the information from a certain location or station. It requires as parameters location id, station id, start and end date, units, limits, offset.

### USGS

USGS REST API services provides several web services that are separated depending on the type of information that is required. These services are separated from instantaneous values, site services, daily values, water quality services, groundwater levels and statistics. Each of the services is access through its own endpoint which contains as parameters data types accepted. All of the sites allow of water data retrieval from thousands of sites that are managed and/or monitored by the USGS throughout the US.

The API requires a proxy server to be accessed but it does not require a key and it uses the “GET” method for retrieving the data.

For the framework, the following endpoints were included:

* Instant values (“instant-values”): retrieves instantaneous information from a specific location. The parameters accepted are format, site, stat ID, county ID, start and end date.
* Daily value (“daily-values”): retrieves daily values of a certain location. The parameters accepted are format, site, stat ID, start and end date.

### World Bank

The world bank has a REST API interface available for retrieving historical and forecast data from every country in the world, with the option of going deeper into basin and sub basin area. The API is based on the derivation and extrapolation of 15 global circulation models, which are used by the Intergovernmental Panel on Climate Change reports. The data within the API are modeled estimates of both temperature and precipitation along with back casting of the data. It is important to notice that the information that the API delivers are based on the more on the models and not as much as in instrumental observed data, this because the difficulty of obtaining data specially during the early 1900s.

The data options within the API are averages for months, years and monthly anomalies and can deliver both precipitation and temperature. The available dates have been separated as past and future, from 1920-1999 and 2020-2099 respectively. The requests also can be sent by identifying basin id’s, which is a number varying from 1 to 468 representing regional river basins throughout the world. The responses come in forms of xml, csv or json and are limited by the anomaly data types, the scenarios for future periods and the type of ensemble within the request. The API requires a proxy server to be accessed but it does not require a key and it uses the “GET” method for retrieval.

For the framework, the following endpoints were included:

* Monthly averages per country (“monavgs-country”): retrieves the monthly averages of precipitation per country.
* Annual averages per country (“annualavgs-country”): retrieves annual averages of precipitation per country.
* Monthly averages per basin (“monavgs-basin”): retrieves monthly averages per basin.
* Annual averages per basin (“annualavgs-basin”): retrieves annual averages per basin.
* Daily precipitation per country (“dailyprec-country”): retrieves daily precipitation per country.
* Daily precipitation per basin (“dailyprec-basin”): retrieves daily precipitation within a basin.

## User interface

HydroLang must be onloaded into an HTML file and opened in a browser. From here, the user can obtain all the functionalities of the framework on the browser’s console. Graphs, diagrams and maps are rendered to screen, while results from analytical features are prompted to console.

Despite not having a specific user interface, the HTML file containing the framework can be styled so that it shows the results of all the calculations or graphs in a more “beautified” way by the usage of different available HTML and CSS styles.. The use cases that are included on the repository showcase the latter.

# Implementation

The framework was developed based on the interoperability of the modules, enabling the user to access all the functionalities from the modules independently, removing potential dependency and inheritance issues. For instance, the user can use data retrieved with the framework or import external files compliant with the supported formats to perform statistical, hydrological analyses, or to generate new map layers or charts/tables rendered on screen without the need to create a new HydroLang instance.

Overall, the library has a low cohesion between one module or another. Largest dependency issues lie on the usage of external libraries. This problem has been addressed by writing asynchronous code waiting for all dependencies to be loaded before any usage.

The functions inside each component have been categorized as either main or helper functions. The purpose of helper functions is to use them on mainly on a module instead of on a HydroLang instance, while the main functions are exported for usage on the latter. Access of each function will depend on the level of chainage in which it is located. Functions located on module with components are on a four-level chaining. Functions located on exports modules are located on third-level chaining.

1. hydro[n].module.component.function(arg)
2. hydro[n].module.function(arg)

Snippet 2. Example of function call per module or component.

The arguments and results of a function will vary upon the purpose of each module. To see the argument requirements, argument examples, function calls, and function results, please refer to the usages annex or the framework’s documentation online. The modules explained below highlight the functions that have a *complex behavior* regarding the usage of external libraries or connections with other functions. The flow chart for each function can be found on the repository of the framework.

## Data

Data is subdivided into the data module and the data sources component. The data module handles the queries coming as inputs from the user while the data sources component is used to export the source parameters from the data components and the data sources providers which contains JSON object examples that contain the necessary information for the creation of a valid query.

Table 3. Functions inside the data module.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Component | Function | Overview | Type |
| Data | *data* | retrieve | Main function for retrieving data. Handles queries for the data sources on the library. | Main function |
|  |  | transform | Transform data in JSON format (or Array of Objects) into an array, CSV, JSON or XML file. | Main function |
|  |  | upload | Uploads data for manipulation. The file can be in either JSON or CSV format. | Main function |
|  |  | download | Downloads data in CSV, JSON or XML formats. It calls the transform function for manipulation of raw data or cleaned data. | Main function |

Table 4. Sources inside the data sources component.

|  |  |  |  |
| --- | --- | --- | --- |
| Module | Component | Sources | Overview |
|  | *datasources* | USGS | Retrieve data from USGS for the US territories. |
|  |  | FEMA | Retrieves data from FEMA for the US. |
|  |  | AEMET | Retrieves data from AEMET for the Spanish territory. |
|  |  | EAUK | Retrieves data from EAUK for UK territories. |
|  |  | METEOIT | Retrieves data from METEOIT for Italian territory. |
|  |  | NOAA | Retrieves data from NOAA to US and the world. |
|  |  | METEOSTAT | Retrieves data from METEOSTAT for the world. |
|  |  | WORLDBANK | Retrieves data for Worldbank for the world |

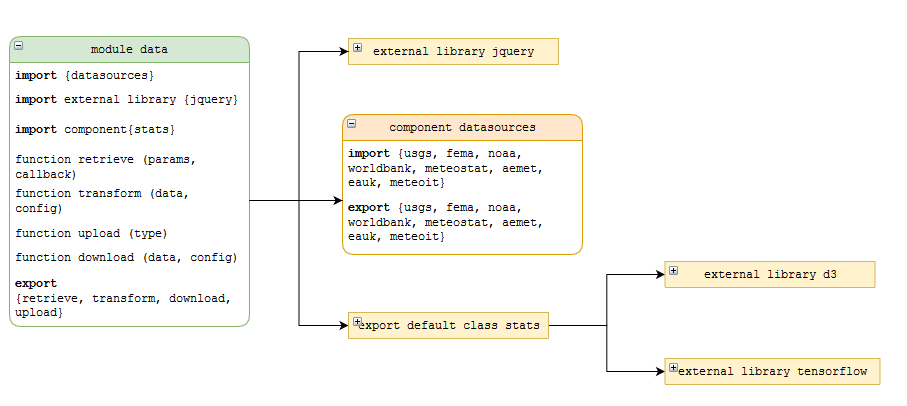


Figure 7. Dependencies of the data module.

The data module has 4 different functions. All functions can be used independently or sequentially, but there is no dependency between one another.

1. hydro[n].data.function(arg)

Snippet 3. Access to data module functions.

### Functions

The *retrieve* function uses as parameters a callback function and a configuration object which contains the source of the data, the final data type required and any other arguments that the source needs. After the function has been initialized, it checks if the object contains a source within the data sources component. If the source does not exist, returns: “No data has been found for the given specifications.” If the source exists, the function checks if the data requires a proxy server or not. This is read from within the data sources specifications. If the source requires a proxy, then the user must specify it within the configuration object; if that is not the case the function returns: “info: please verify if the resource needs a proxy server.” If the source requires some sort of key or token, like before, the user must specify it on the configuration object. Once all the filters have been fulfilled, the request is sent to the API URL and waits for a response (whether positive or negative) to later be prompted to screen. The results of the object can be saved as temporal variables and used in other functions.

1. var example = {
2. source: "name of source",
3. dataType: "type of data",
4. type: "format of data",
5. arguments: {
6. arg1: "name of arg1",
7. arg2: "name of arg2",
8. },
9. nameoftoken: "token",
10. proxyurl: "proxy server as url"
11. };

Snippet 4. Object example for retrieval function.

The *transform* function can use as arguments objects saved from the retrieve function or objects passed by the user from the upload function. If the data needs to be cleaned, then the user can pass a configuration object that states what is required to be saved from the raw object. If the data is ok as it is, then the configuration object must only include to what format should the data be transformed. The data can be transformed from JSON or arrays into CSV, XML, JSON, or arrays. The object is saved on a declared variable. Unless the file is to be transformed into an array or JSON, the saved variable is not very useful.

1. var config = {"type": "CSV, XML or JSON", "keep": ["field1", "field2"]};

Snippet 5. Example configuration object for transform function.

 The *download* function uses the transformation function to convert retrieved data into other formats. It creates a temporal blob of transformed data appended to the HTML file which runs the framework. To initialize the function, a configuration object must be created with similar specifications to the transformation function: the data that is to be saved and the format of the data. After running the function, a new download automatically begins with the data within the specified format.

Finally, the *upload* function can be used for uploading CSV and JSON files. The file is saved into a new variable and from here, it can be manipulated as required.

## Analyze

The module was developed with three components: hydro, stats, and NN. The components are fed into the module and called default classes. The functions of each module are located on the fourth level chainage as static methods and called on usage just as with other functions.

1. hydro[n].analyze.component.function(arg)

Snippet 6. Example usage of analyze functions.

Table 5. Components of the analyze module.

|  |  |  |  |
| --- | --- | --- | --- |
| Module | Component | #functions | Overview |
| Analyze | *stats* | 29 | Tools for performing statistical analyses on data of different formats. The tools differ on the types of arguments that they can accept. |
|  | *hydro* | 10 | Tools for calculation of synthetic rainfall analyses. Contains areal precipitation along with flooding conversions using physical characteristics of a basin. |
|  | *NN* | 4 | Creates neural networks models using a recursive approach. |

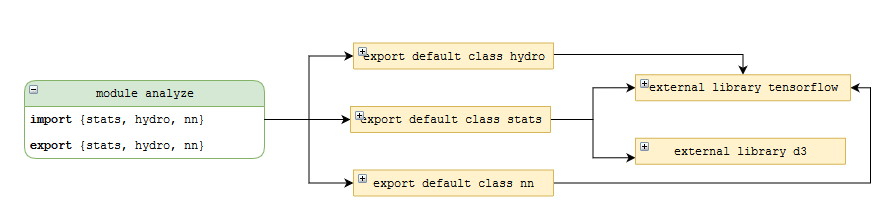


Figure 8. Dependency diagram of analyze module.

### Stats

Stats component has been created to generate basic statistical analysis on data that has been either retrieved or uploaded. All the functions on the class can be accessed by the user; the helper functions are used in the class or other instances too.

The component is used merely for data classification and manipulation. The arguments of functions are 1D and 2D arrays. More specialized functions have been implemented so that they can deal with time-series data, which at the same time can be in either string or number format.

Both D3 and tensorflow.js libraries are being used in the class for different functions. Sums, mean, median, variance, max, and min make use of the already built functions inside of D3, while *fastfourier* uses the fast Fourier analysis tool in tensorflow.js to perform the latter on a given array.

To create ease of usage, a basic statistics function has been created that calls the functions mentioned previously along with count and *stddev* to return a n \* 2 array that specifies basic metrics for a dataset. From here, the user can render a new table screen using the tools on the visualize module.

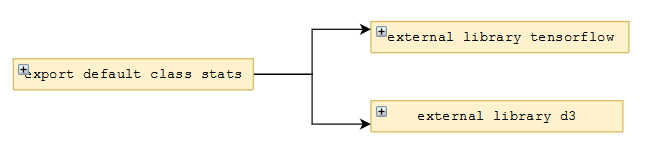


Figure 9. Dependencies of the stats class.

Table 6. Functions in the stats component.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Component | Function | Overview | Type |
| Analyze | *stats* | copydata | Makes a deep copy of a dataset. | Main function |
|  |  | onearray | Returns the data of a time series 2D array. | Main function |
|  |  | datagaps | Identifies gaps in data. Can be fed with time series 2D array or just the data of a time series. | Main function |
|  |  | gapremoval | Removes gaps from data. | Main function |
|  |  | sum | Sums all data in an array. | Main function |
|  |  | mean | Calculates mean of an array. | Main function |
|  |  | stddev | Calculates standard deviation of an array. | Main function |
|  |  | sumsqrd | Calculates sum of squares of an array. | Main function |
|  |  | min | Calculates minimum value of an array. | Main function |
|  |  | max | Calculates maximum value of a dataset. | Main function |
|  |  | unique | Determine unique values within a dataset. | Main function |
|  |  | standardize | Standardize dataset given mean and standard deviation. | Main function |
|  |  | quantile | Calculates a given quantile of a dataset. | Main function |
|  |  | cleaner | Filters out items in an array that are undefined, NaN, null. | Main function |
|  |  | outremove | Removes outliers on a dataset. Requires the identification type (either quantile or normalized). If p1 or p2 are not given, then the default values for either method will be selected. | Main function |
|  |  | itemfilter | Filters out items in an array based on another array. | Main function |
|  |  | correlation | Calculates the Pearson coefficient of bivariate data. | Main function |
|  |  | variance | Calculates the variance of a dataset. | Main function |
|  |  | frequency | Determines the frequency of a dataset. | Main function |
|  |  | interoutliers | Calculates the outliers of a dataset using interquartile method. By default, 25 and 75 quantiles are used. | Main function |
|  |  | normoutliers | Calculates the outliers of a dataset using the normalized data method. By default, -0.5 and 0.5 are used. | Main function |
|  |  | fastfourier | Calculates fast Fourier analysis on a dataset to study seasonality effects. Returns an array with the same dimensions as the input. | Main function |
|  |  | basicstats | Generates an object with basic statistics for a dataset to be plotted using google charts. The values considered for the table are: number of values, minimum value, maximum value, sum, mean, median, standard deviation and variance. | Main function |
|  |  | joinarray | Joins arrays of data. | Helper function |
|  |  | numerise | Transforms arrays of data from strings to number. | Helper function |
|  |  | flatenise | Flatens a dataset | Helper function |
|  |  | dateparser | Parse data to seconds since 1970 for data manipulation. | Helper function |
|  |  | arrchange | Interchanges a m x n matrix to n x m. | Helper function |
|  |  | push | Pushes one array into another array in the end. | Helper function |

A special interest was given on developing functions that can filter outliers or gaps in a dataset. These are *gapremoval, outremove, itemfilter, interoutliers, normoutliers*. These functions take as arguments either 1d arrays with data in number format or 2d arrays representing a time series.

1. var arrex1 = [number1, number2, …]
2. var arrex2 = [[date1, date2, date3, …][number1, number2, number3, …]]
3. var arrex3 = [[number1, number2, number3, …][number1, number2, number3, …]]

Snippet 7. Example of arrays for usage on array class.

Outlier identification and removal functions consider 2 approaches for identifying outliers in data: normalizing the data and establishing limits taking 0 as the center and interquartile identification. The gap identifiers go through all the data and see if the data is not a number and drops it—in the case is a time-series, the correspondent value for date is also removed.

### Hydro

The hydro component was developed focused mainly on rainfall and runoff analyses based upon the theories of conceptual lumped models. It contains basic functions for the study of precipitation data regarding spatial and temporal coverage as well as the derivation of runoff based on physical characteristics using synthetic calculations.

Most of the functions require objects as parameters. The functions can handle metric systems according to the user’s requirements but are constrained by the method derived for calculations. They can be used independently or connected with the results of another. The degree of cohesion between each function is low because results from one function can be used on another but are not constrained to this. For instance, the results of rainfall distribution derived from Thiessen polygons can be used for intensity hyetographs and later for the bucket model, but not constrained to this.

Table 7. Functions in the hydro component.

| Module | Component | Function | Overview | Type |
| --- | --- | --- | --- | --- |
| Analyze | *hydro* | Totalprec | Calculates summation of total precipitation of a given event. Rainfall must be evenly distributed and fed as a 1D array. | Helper function |
|  |  | arithmetic | Calculates arithmetic average for different rain gauges for the same event. Rainfall must be evenly distributed and fed as multiple 1D arrays inside an object with same rainfall period. | Main function |
|  |  | thiessen | Calculates pondered average for different rain gauges for a same event given subbasin areas. Rainfall must be evenly distributed through the area. The passed object should contain 1D array with areas and an array of arrays with precipitation with same rainfall period. | Main function |
|  |  | syntheticalc | Calulates time parameters regarding a rainfall event using different approaches. | Main function |
|  |  | bucketmodel | Calculates rainfall-runoff analyses of an event given parameters as evaporation, baseflow, land uses. The rainfall event should be a time series in which the time variable should be either a valid time string type or number. | Main function |
|  |  | floodhydro | Calculates the flooding hydrograph given a U.H. and the physical characteristics of a basin. Object passed to the function should include rainfall event (time in either string or number formats), unit hydrograph (time in either string or unit format), CN number, storm duration and required timestep. | Main function |
|  |  | dimunithydro | Generates a dimensionless unit hydrograph from the physical characteristics of a basin. Object should include timestep, number of hours of event and peak rate flow. | Main function |
|  |  | ground1d | Calculates 1D groundwater flow/head steady transport using gaussian elimination. | Main function |
|  |  | move | Moves part of an array into another. Arguments should include the array and locations. | Helper function |
|  |  | matrix | Creates an m x n matrix filled with 0s or any other number. | Helper function |
|  |  | equationsystemsolver | Solves linear system in the form Ax = b. Requires a matrix, left vector and right vectors as arguments. | Helper function |

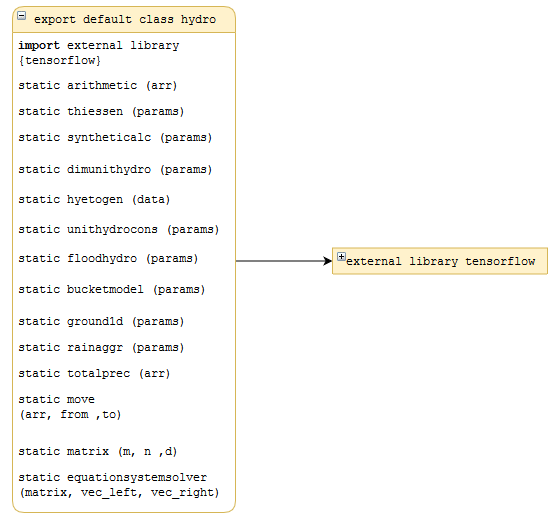


Figure 10. Dependency diagram of hydro component.

*arithmetic* function calculates the mean distribution of a rainfall event caught by stations in the same basin. The arguments for the function are passed as an array of arrays, in which each subarray contains data for the event for each station to be considered. The rainfall event must be of equal length for all the stations and only the data of the event should be fed to the argument.

1. var exarr = [[data1,data2,data3,data4…],[data1,data2,data3,data4….]]

Snippet 8. Array example for arithmetic function.

*thiessen* function calculates the average precipitation for a given rainfall event in a basin considering there is one station per subbasin. It takes as arguments an object that describes the readings of a time series and the areas per each basin, both as arrays. If a sub-basin contains more than one station, then the arithmetic function should be first used to calculate the average of that sub-basin.

1. var rainf = [[data1,data2,data3,data4…],[data1,data2,data3,data4….]]
2. var areas = [area1, area2, …]
3. var example = {thiessenprec: rainf, thiessenareas: areas}

Snippet 9. Example object for thiessen function.

*syntheticalc* function is used for deriving duration parameters for the creation of a unit hydrograph based on approaches from SCS, Kerby-Kirpich, and Kerby. The argument of the function is an object describing the method required, the units of the calculation (SI or metric), and additional parameters that depend on the method.

* SCS method: the value for CN, longest path (in feet or meters) and average basin slope.
* Kerby-Kirpich: the longest path (in feet or meters), and average slope for both the main channel and the basin.
* Kerby: the manning coefficient, the longest path (in feet or meters), and slope.

The results from the function will return an object with keys of time of concentration and lag time in hours.

1. var example = {type: "type", units: "SI or metric", args: {L: "length in app units", slope: "in percentage"}}

Snippet 10. Example object for syntheticalc function.

*dimunithydro* function creates a dimensionless hydrograph based on the Gamma distribution for calculating peak rate flows (PRF). It accepts as argument an object describing the distribution type—for now, only Gamma has been used but others will be added in the future—the peak rate flow in cfs, the required time step, and the number of hours for an event. The peak rate flow varies from 101 to 566. This yields the “m” parameter in the Gamma distribution which is later used to derive the hydrograph. If the PRF is on the lower end, then the basin flooding area should have a flat slope and the response time of the basin would be longer. If on the contrary, it is on the highest end, then the basin area should be a very steep basin. This should be considered by the user previous to usage.

1. var example = {distribution: {type: "gamma", PRF: "101-566"}, timestep: number, numhours: number}

Snippet 11. Example object for dimensionless hydrograph.

*hyetogen* function creates an intensity hydrograph in pulses for a time-series event from a 2D array. It accepts either a time as string or number. It accepts as arguments the rainfall event and the aggregation required. For instance, from a 15 minute-data for 2 hours, create a 1-hour intensity hyetograph.

1. var example = {event: data (2d array) timereq = number}

Snippet 12. Example object for hyetogen function.

*unithydrocons* creates a unit hydrograph from a time-series of either a dimensionless unit hydrograph or an observed discharge hydrograph. The argument will depend on which of the latter is selected. For the dimensionless case, it accepts as arguments the drainage area in square miles or square kilometers, the time of concentration in hours, and the dimensionless unit hydrograph created from the *dimunithydro* function. The unit hydrograph is constructed by multiplying the entries of the dim hydro by the drainage area and divided by the time to peak.

For the observed hydrograph uses an empirical approach for deriving the latter. The function accepts as arguments the drainage area in the same units as the observed hydrograph (SI: cfs—square feet, metric: cumecs—square meters), precipitation intensity in cm/hr or in/hr, and baseflow in cumecs or cfs. From the observed hydrograph, a direct runoff hydrograph (DRH) is calculated using the area of the basin, removing the observed baseflow; the final hydrograph will have units of cumecs/cm or cfs/in. The unit hydrograph is then calculated by dividing the DRH by its total volume in depth (inches or cms). Finally, the function returns an object with the UH and the total volume calculated for the rainfall event in cms or in (this is used later on the floodhydro function).

1. var ex1 = {drainagearea: number, hydro: 2darray(dimhydro), type: "dim", config: {tconcentration: number(hours)}}
2. var ex2 = {drainagearea: number, hydro: 2darray(obshydro), type: "obs", config: {baseflow: number}}

Snippet 13. Example objects for unithydrocons function.

*floodhydro* function generates a flooding hydrograph based on the physical characteristics of a basin. It has two different methods for this: SCS and empirical method. The SCS calculates abstractions based on the CN number (for more information, please refer to the background chapter). This number is changed later based on the type of unit that the user specifies; using a unit hydrograph it then calculates the composite hydrograph which creates a hydrograph generated from the rainfall event. The method has been optimized for usage with a UH constructed using a DUH.

The empirical method calculates runoff considering the total volume of a unit hydrograph in terms of depth from the unithydrocons function. Using the principle of superposition and proportionality, the flooding hydrograph is calculated based on the convolution of the unit hydrograph, using rainfall intensity in terms of pulses. This calls the hyteogen function which generates hourly intensities. The pulses are found by subtracting the

NEED TO FINISH

### NN (Neural Networks)

The NN component using the open-source, client-side service of Tensorflow.js, used for artificial intelligence. The module included within HydroLang creates a sequential algorithm model using the recursive method. The inputs are fed directly to the neurons, with preestablished activation functions. The number of inputs, neurons, and outputs are fed to the model by the user depending on the final outcome needed. It creates a 1 layer of neurons, with the number of neurons to be chosen by the user as well as the number of outputs.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Overview | Type |  |
| Analyze | *NN* | createModel | Creates a new sequential neural network model. | Main function |  |
|  |  | convertToTensor | Converts JavaScript arrays into tensorflow tensors. | Main function |  |
|  |  | trainModel | Trains an exisiting model with already created tensors. | Main function |  |
|  |  | prediction | Based on a trained model, predict data. | Main function |  |

## Visualize

The visualization module (named visualize) contains 3 different main functions, with one of them (i.e. styles) depending on the two others. They contain basic styling features accordingly to guidelines from Google Charts API documentation. The helper function defined within the module calls in the google charts API from the library components. It was decided to include the caller within the framework to avoid calling it from external servers.

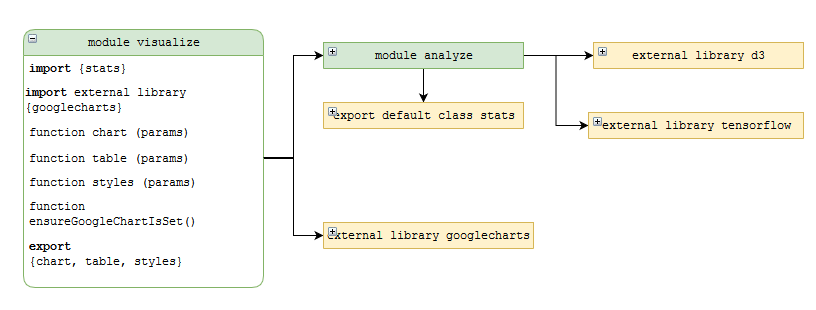


Table 8. Dependency diagram of visualize module.

1. hydro[n].visualize.function(arg)

Snippet 14. Example usage for visualize function.

The functions are located on the third level chainage accessed through the HydroLang instance as with the other modules. The arguments for the main functions are objects that have tailored with different arguments, depending solely on the final requirements from the user end and the API’s usage. Examples of function calling and arguments types are found in the usages appendix.

Table 9. Functions within visualize module.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Component | Function | Overview | Type |
| Visualize | *visualize* | chart | Draws data as a chart depending on the options passed by the user. The data must be in array format. Options available are scatter, column, line, timeline and histograms. | Main function |
|  |  | table | Draws a table from data provided by the user. The data must include a header and passed as an array. | Main function |
|  |  | styles | Contains different styles available to use by feeding fewer arguments than by directly using the chart and table functions. The drawing styles are meant just as options in case the user does not want to create a style object. | Main function |
|  |  | ensureGoogleChartIsSet | Promise for calling the google charts component. Calls in all the charts and tables that are rendered in the library. | Helper function |

### Functions

The *chart* function builds up an n-dimensional chart. It accepts as arguments an object that specifies the data that is to be drawn, the chart type, the name of a new page division, and the options which specify criteria dependent on the chart type. All the specifications of the charts can be found on the Google Charts API page. It creates a new division with the name of the graph title, the name of the page division with a fixed 1000px, and 500px of width and height.

1. var example = {
2. chartType: "scatter, column, histogram, line or timeline",
3. data: (ndim data as array),
4. divID: "name of page division",
5. //the different options can be found on google charts doc.
6. options: {
7. option1: option1,
8. option2: option2
9. },
10. }

Snippet 15. Example argument for chart function.

The *table* function creates a n\*dimensional table with headers. Accepts an object with the data, division name, and an array specifying the data types of each column. It creates a new divisor where the table is appended to. The colors of the table are fixed to blue on the header and white on the content. The data should be an n-dimensional array with the data.

1. var example = {
2. data: (ndim data as array),
3. divID: "name of page division",
4. datatype: ["datatype1", "datatype2",...]
5. }

Snippet 16. Object example for table function.

*styles* function was created to create both graphs and tables that are considering certain options based on the documentation of the API. The user can access the same types of charts and tables available on the module by passing an object with fewer inputs. Afterward, the selected chart will be drawn using the latter functions.

1. var example1 = {
2. data: (ndim data as array),
3. draw:: "chart",
4. config: {chart: "chartype", div: "divname", title: "titlename"}
5. }
7. var example2 = {
8. data: (2d array),
9. draw:: "table",
10. config: {div: "divname"}
11. }

Snippet 17. Object examples for styles function.

## Maps

The map module enables the user to visualize a map for the selection of stations, render geodata files, and generate queries from referenced points. It uses two different map engines: leaflet and google maps. The module provides a generalized single interface for all the map functionalities implemented, allowing the user for choosing the underlying map engine. As a demonstration of these features, HydroLang supports the previous engines (only leaflet fully implemented). This idea can be extendable to other engines in the future.

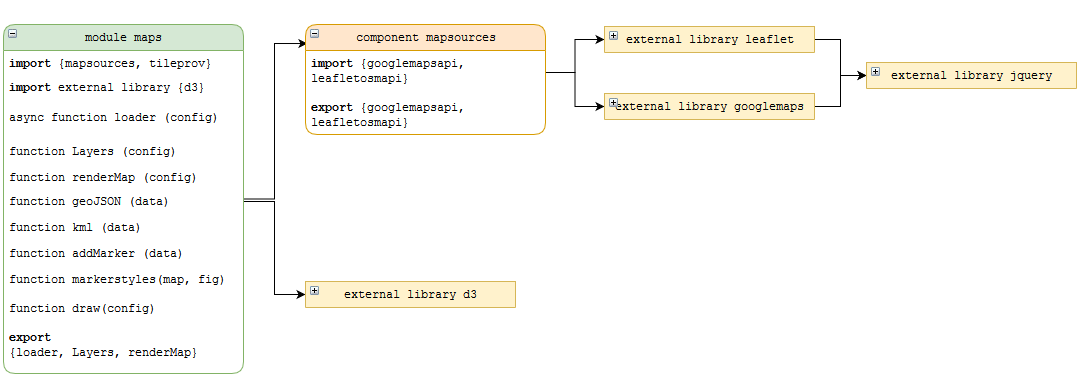


Figure 11. Dependency diagram of map module.

The map module has 7 functions, in which 3 can be used from the map instance. The arguments on each function is a configuration object that changes depending on the function.

Table 10. Functions of map module.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Component | Function | Overview | Type |
| Map | maps | loader | Loads maps from leaflet or google. | Main function |
|  |  | Layers | Add layers of data for both google or leaflet or both geodata, kml, drawing tools, markers or remove layers. Main function used to draw new layers into an existing map. | Main function |
|  |  | renderMap | Appends a map loaded into a div located on the header of an html page. The loader function must be defined first before to load the necessary libraries. | Main function |
|  |  | geoJSON | geoJSON function for rendering georeferenced layers into screen. Used through Layers function. | Helper function |
|  |  | kml | Creates a layer of kml data passed through an object to an exisiting map. Used through Layers function. | Helper function |
|  |  | Marker | Creates a new maker into an existing map given the coordinates. Used through Layers function. | Helper function |
|  |  | markerstyles | Changes the style of a marker depending on the marker shape itself. Used through Marker function. | Helper function |
|  |  | draw | Appends a drawing tool on a map. Can create similar shapes as in markers and layers can be saved. Used through Layers function. | Helper function |

The *loader* function is in charge of calling the necessary libraries for either provider and append them to the HTML file in use. It has a very simple call but has to be used previously to other functions in the module. It accepts an object as an argument which specifies the map type to be used: google or osm (reference for leaflet).

1. var map = {maptype: config.mapType};

Snippet 18. Argument for loader function.

*renderMap* is the function that renders the map to screen by appending a new division into the HTML file. The loader function should have been already called; requires as arguments an object that specifies the map type, the coordinates, and zoom of the map, and another object that specifies the type of tile to be used. The tiles can be found on the tileprov library; the user is able to use any of the tile types specified on the latter by passing the name of the provider to the configuration object. The function then calls the Layers function to render a tile from the provided sources.

1. var mapconfig = {
2. maptype: "osm",
3. lat: "value",
4. lon: "value",
5. zoom: "value",
6. layertype: {
7. type: "tile",
8. name: "OpenStreetMap",
9. },
10. };

Snippet 19. Configuration object for renderMap function.

*Layers* is the main function of the module. It has been tailored to deal with all sorts of layer manipulation within an existing map. It requires a configuration object that changes depending on the type of action the function should do. The data should be on the specified format for each call and can be from either external sources—like the sources included on the framework—or data from the user’s local storage.

Like the map, the layers as saved on the module. All the helper functions on the module are called by the Layers function to add new layers including geoJSON data, markers, KML data, drawing tools, or new tiles to an existing map. If the user wants to remove a certain layer, that is also done through the function by specifying the layer name on the configuration object. To see the different types of layers that can be added to the map, please see the usage annex.

1. var marker = {
2. maptype: "osm",
3. layertype: {
4. type: "marker",
5. markertype: "rectangle, circle, circlemarker, polyline, etc…",
6. name: "1st marker"
7. }
8. };
10. var remove = {
11. maptype: "osm",
12. layertype: {
13. type: "removelayers",
14. name: "layername"
15. },
16. };

Snippet 20. Example objects for Layers function.

# Case study 1: rainfall analysis

## Description

As part of a feature within the library, a neural network model has been developed for analyzing rainfall disaggregation patterns from 1 hour to 15 min data. The data requirements are both 15 min and 1-hour interval data with identified 1 hour, 2 hours, and 4 hours storm events. Case study 1 is the development of the latter by using the functionalities that HydroLang offer for data retrieval, cleaning, and sorting.

The specifications of the environment used the case study are the following:

* Model: ASUS GL702VSK.
* Processor: Inter Core i7-7700 HQ CPU @ 2.80 GHz.
* RAM: 12 GB, 2400 MHz.
* GPU: NVIDIA GeForce GTX 1070 GDDR5 @8 GB.
* Hard drive: 1049GB.
* Code editor: Visual Studio Code versión 1.47.3.
* Web browser: Google Chrome version 84.0.4147.105.

## Library usage

HydroLang is called using the onload feature within HTML. The source of the hydro.js file must be specified as well as any other external libraries for styling. Afterward, a new instance of HydroLang is created on the script of the document. Since a map layer will be loaded, the function caller for the framework is asynchronous and must be specified. A new variable is created named hydro, instantiating the library.

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <link
5. **rel**="stylesheet"
6. **href**="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
7. />
8. <title>Hydrolang Use Case</title>
9. <script
10. **type**="module"
11. **onload**="initHydrolang()"
12. **src**="./hydrolang/hydro.js"
13. ></script>
14. </head>
15. <body>
16. <script>
17. async function initHydrolang() {
18. hydro1 = new Hydrolang();

Snippet 21. Example of HydroLang library on HTML file.

## Loading a map for visualization

A new map was loaded to the HTML webpage to visualize the locations prompted from each station. To achieve this, the map module was used by calling the functions using the new framework instance. The map can also be drawn one the HTML goes live.

1. var map = {
2. maptype: "osm"
3. };
4. await hydro1.map.loader(map);
6. var mapconfig = {
7. maptype: "osm",
8. lat: "40",
9. lon: "-100",
10. zoom: "13",
11. layertype: {
12. type: "tile",
13. name: "OpenStreetMap",
14. },
15. };
17. //Map renderer.
18. hydro1.map.renderMap(mapconfig);

Snippet 22. Adding a map to screen.

## Data

NOAA has been selected as data source because the source can provide finer resolution precipitation of up to 15 minutes from several stations across the US. It is important to mention that the data source requires a token to the header of the request, which can be obtained by requesting one at the API web page. The request should include a limit for the number of items that it should contain; if it is not given, then the default number is 25. The quality of the data relies on the total coverage within a specific station, the frequency of the readings, and the number of gaps. Three different endpoints were used to select a station that fulfills the data quality requirements (NOAA, 2020). For rainfall information, two different values can be obtained from a query:

* QPCP: the amount of precipitation recorded at a station in a given interval in hundreds of inches of tenths of millimeters, depending on the units selected by the user.
* QGAG: volume of precipitation that is calculated by weight on a station in a given interval in hundreds of inches or tenths of millimeters, depending on the units selected by the user. This value was added as metric until 1992.

For the purpose of the case study, the requests sent only acquired QPCP data in metric system, precipitation data of 15 min. Data coverage—between 0 and 1, 1 being the best—is a variable included in the query that helps to sort the stations with best coverage. For the data analysis module, a function for handling retrieved data is also passed to the body of the HTML, used for showing the retrieved data to console. From here, the retrieval function was called, passing as arguments the created object and the handling data function.

1. dataRetrievalParams1= {
2. source: "noaa",
3. dataType: "availablestations",
4. type: "json",
5. arguments: {
6. datasetid: "PRECIP\_15",
7. locationid: "PRECIP\_15",
8. sortfield: "datacoverage",
9. sortorder: "desc",
10. limit: 1000,
11. },
12. token: tokennoa,
13. };
15. function handleWaterData(data) {
16. console.log(data);}
18. hydro1.data.retrieve(dataRetrievalParams1, handleWaterData);

Snippet 23. Example commands for downloading data.

By saving the file and opening it in the browser, the downloaded data was prompted to console a soon as it is retrieved. There are more than 3000 stations that provide 15-minute data within the US and most of them have coverage of around 25%.

As part of the quality, full years of data that have been recorded by a specific station are being considered. The stations that have the best coverages have been queried yearly from the minimum available date until the maximum date (most stations have data until 2014). For example, if a station has data from 1974 through 2014, then the query starts from 01/1974 through 12/1974. The number of values from each query was quantified and classified based on the expected number of features (4-hourly data, 24 hours, 365 days = 35,040). The years with the highest percentages of data are then used for processing.

For the case study, the station of Altavista, Virginia was selected because it can provide the most amount of data on a yearly basis. The data was retrieved for years 1984 to 1987. The selection was based on the amount of data that was retrieved in those years. The station was added to the existing map by adding it as a new layer using the Layers function on the map module.

1. var marker = {
2. maptype: "osm",
3. layertype: {
4. type: "marker",
5. markertype: "marker",
6. coord: [37.116667, -79.283333],
7. name: "Altavista, VA"
8. },
9. };
11. hydro1.map.Layers(marker)

Snippet 24. Adding the station into the map.

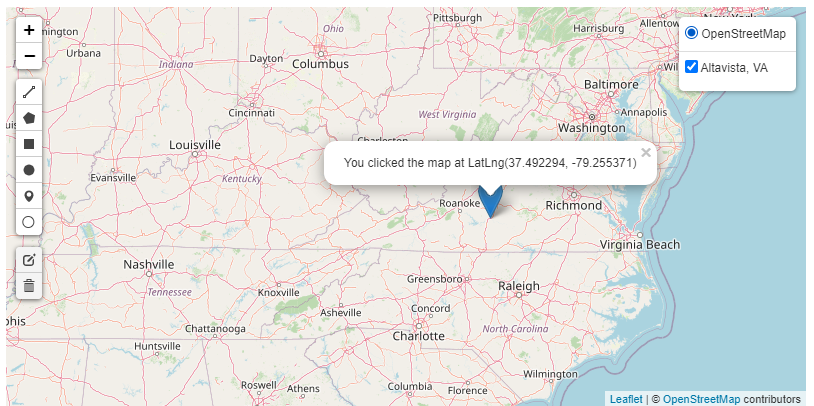


Figure 12. Station location.

Data that is available per time request is given on the metadata object on screen; the results of the request were saved in a temporary object for further manipulation. Onwards, all the manipulations of the files or functions that are called from the library have been done in the console of the browser; the objects created in the HTML file are available on console.

### Data analysis

Each request was handled separately per year. The max number of items that can be retrieved per request is 1000 and the years for the analysis have more than 1000 items, thus several requests were sent per year of study. Each yeah was dealt with separately; for retrieving data for a new year the request object changed the dates and the page was reloaded.

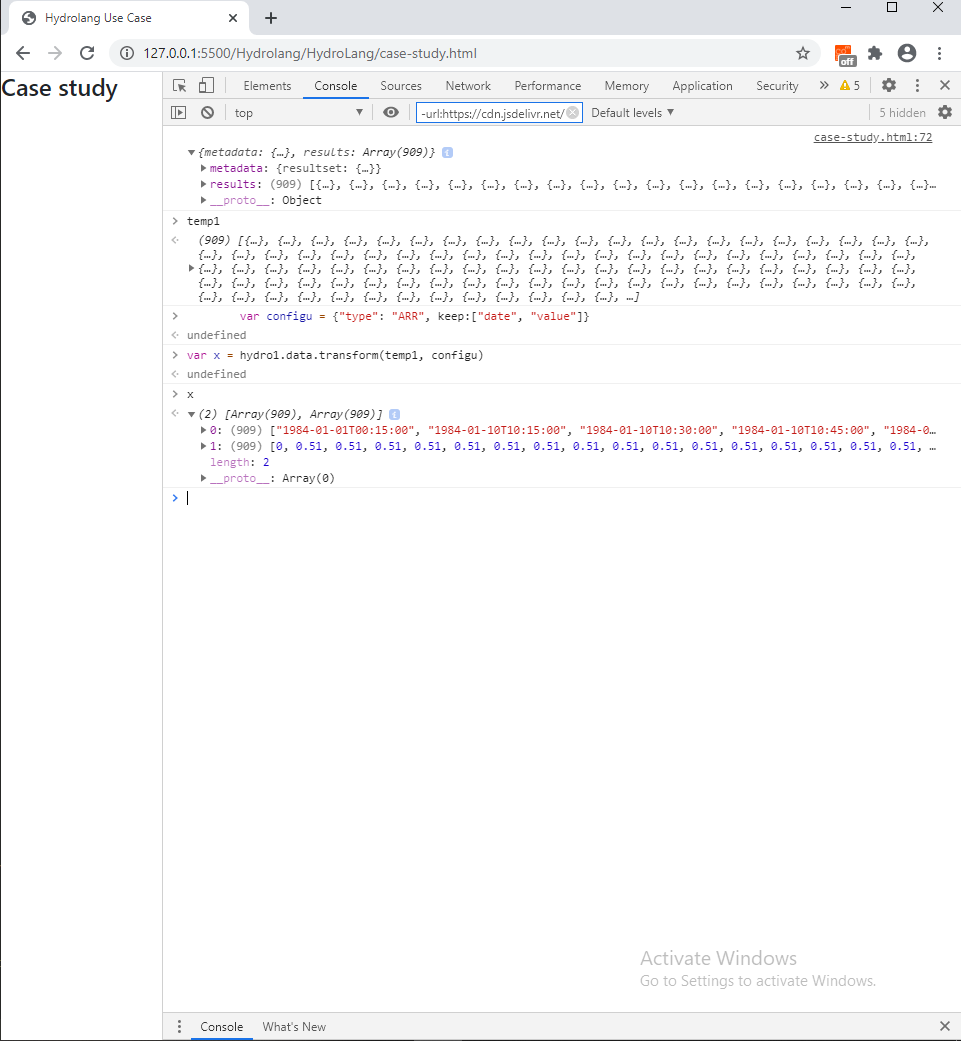


Figure 13. Response from console after first request.

After downloading all the data needed, it was cleaned up using the tools on the data module. The only fields that are required from the downloaded files are date and value. For this, a configuration object was created to be given as an argument to the transformation function. The final object, named x, will have all the data from the year.

1. var configu = {"type": "ARR", keep:["date", "value"]}
3. var xa = hydro1.data.transform(temp1, configu)
5. var xo = hydro1.data.transform(temp2, configu)

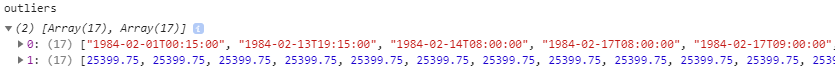
Snippet 25. Using the transformation function.

For an overview of the data, the function of basic statistics, frequency, and outliers have been used. The results of the variables can be seen in console. Moreover, the basicstats array can be rendered to screen using the table function on the visualization module

1. var final = hydro1.analyze.stats.push(xa, xo)
3. var stats = hydro1.analyze.stats.basicstats(final[1])
5. var freq = hydro1.analyze.stats.frequency(final[1])
7. var outliers = hydro1.analyze.stats.normoutliers(xo)
9. hydro1.visualize.styles({data: stats, draw: "table", config: {div: "statstable"}})
11. var clean = hydro1.analyze.stats.outremove(final)

Snippet 26. Code for calling statistics for the data.

By studying the latter variable, it is clear that the data contains outliers. The results of the variable outliers show that there are 17 outliers that are needed to be removed. For this, the outremove function was used, having as arguments the data and the name “normalized”. The data without outliers was named cleaned and the functions were run once again on the cleaned data.





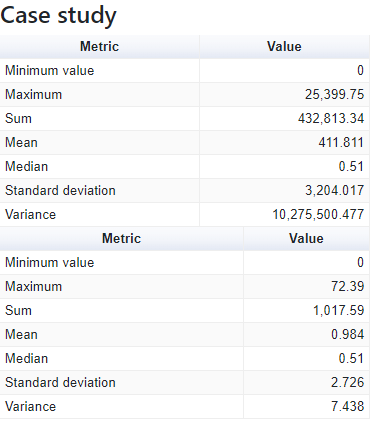


Figure 14 Example of statistics of raw data vs cleaned data.

Once the data has been cleaned from outliers, it was verified for gaps pertaining to both time and/or data.

1. var timegaps = hydro1.analyze.stats.timegaps(clean, 15)
3. var datagap = hydro1.analyze.stats.datagaps(clean)

Snippet 27. Function calls for time and data gaps identifiers.

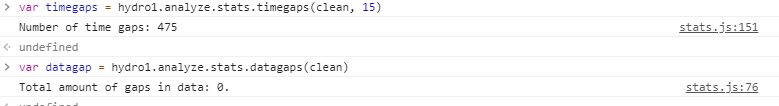


Figure 15. Identification of time gaps and data gaps.

After cleaning the data and identifying the gaps, the data was downloaded using the download function in the data module. CSV format was used so that further manipulation of it could be done using Excel. To see the rest of the data manipulations, please refer to the case study 1 annex.

1. hydro1.data.download(clean,{"type": "CSV"})

Snippet 28. Downloading the data.

# Case study 2: Derivation of a flooding hydrograph

# Discussions

So far, the framework has become a robust library that allows for the manipulation of environmental data in different formats. The flexibility of using a standardized programming language, open-source libraries, and no need for installation provides an important step for scalability and upgrades features, in the hopes that the library can grow by becoming a community-based framework with collaborations from research institutions or individuals with expertise.

One main feature considered when developing HydroLang was the importance of state-of-the-art technology. This meant the usage of various features of modern Javascript specifications as well as the incorporation of libraries such as TensorFlow, D3, Leaflet. Hydrolang can be customized and extended by interested parties to be suited for specific use cases, development environments, project requirements, etc. The language provides a single generalized solution for hydrology-related applications, but the functions can be extended/modified simply by following the unique software design. The underlying technologies such as the external libraries can also be changed by removing/adding new ones.

## Challenges and limitations

Manipulation of data is an important feature that has been taken close attention to. For instance, the data module has been developed to obtain, transform, download, and upload data; but it still misses more in-depth features like adding and handling other formats, the number of data sources, and endpoints for each source, interconnectivity with other modules like maps, among others. There is still room for improvement, but the biggest unavoidable constraint is data availability. When researching for sources, many times data was required to be paid or limited by the location of the request. Another example is that open-source API’s that can provide geoJSON format is very limited and can only give the user part of a query.

The analyze module can be considered the strongest in the number of available functions, but the weakest in terms of complexity. The stats component provides basic statistical cleaning but lacks on probabilistic features. A clear example of this is the gap filler function, which considers only the averages between the upper and the lower data numbers from the one that is missing; but this is just one of the many approaches developed on the literature for these situations. The hydro module contains very basic tools for precipitation and runoff analyses. All the functions have been used as a starting point so that the user is able to comprehend the data but cannot be considered as the only solution to a problem. Each feature is but a mere example of the vast number of procedures available and because of this, there is little room for flexibility in terms of arguments and results. Finally, the idea behind including a neural network module comes from all the uses that the field of artificial intelligence has on hydrology/hydraulics. The component includes only sequential recursive models, that once again, are limited to certain applications and is not the only approach that can be used for hydrology.

So far, the framework can be downloaded as a complete library to be used on local storage. Nonetheless, most of the functions require an internet connection to work because of the usage of external libraries. A migration towards back-end development can sole this which would give the library still more computational capability for performing more special cases. The variations that CPU performances on the client-side can result in abnormal runtimes, especially when large chunks of data are analyzed.

# Conclusions

The development of HydroLang has been focused on data. The analytical features within each module have been tailored to adapt to different data formats that are either retrieved from external sources or uploaded from local storage. Moreover, considering that the capabilities of web browsers nowadays are not limited by the computational power required for running complex algorithms, the possibilities for new features are limitless. The existing features can still be improved, and new features should be added to give a sense of a more complete program.

## Future work

Having in mind the idea that the framework can grow through the scientific community, all the existing features will be revised and improved were possible. HydroLang will be open-source in the future, allowing the scientific community to contribute to the framework for a more comprehensive and consensual solutions in software development, hydrological and statistical functionalities, and visualization. The framework can be ported to sever-side JavaScript environments (e.g. nodejs) of the purpose of having more computational power available for more specialized cases. Having the library on the server-side allows for analyses of huge and entangled data feasible.

Specifically per module: the data module can be improved in two different ways: the number of sources (and endpoints) and the types of data that can be retrieved. So far, the 8 sources have a total of 36 endpoints, and each can be modified to obtain different type of information by passing new arguments to each request. Nonetheless, the module is limited by spatial aggregation, quality of data and locations. The types of data that the module can handle can also be improved by adding new data handling situations. The upload function will be upgraded to deal with other types of data such as shapefiles, ASCII, etc. This can be then used on other modules. New functions will be added to the hydro component accounting for other runoff modules. The stats component will include new features for analyzing trends and new ways of analyzing data. The NN component’s functions can be expanded accounting for other applicable features (i.e. activation functions, model approaches).

Finally, the map module will be modified to handle other type of georeferenced formats, specially coming from GIS software. The layers created on an existing map, whether using the drawing tool or the Layers function, cannot be saved. An option will be passed to the function so that this can change.

# References

AEMET, 2020. *AEMET Open Data.* [Online]   
Available at: https://opendata.aemet.es/dist/index.html?#!/observacion-convencional/Datos\_de\_observaci%C3%B3n\_Tiempo\_actual\_1  
[Accessed June 2020].

Agafonking, V., 2020. *Leaflet: an open source Javascript library for mobile friendly interactive maps.* [Online]   
Available at: https://leafletjs.com/  
[Accessed June 2020].

Alcantara Souffront, M. A. et al., October 2019. Hydrologic Modeling as a Service (HMaaS): A New Approach to Address Hydroinformatic Challenges in Developing Countries. *Front. Environ. Sci..*

Ames, D., 2018. *HydroShare.* [Online]   
Available at: https://www.hydroshare.org/resource/76ed0c1d065f440486416a979ff6488e  
[Accessed 2020].

Ames, D. P. et al., November 2012. HydroDesktop: web services-based software for hydrologic data discovery, download, visualization and analysis. *Environmental Modelling & Software,* Volume 37, pp. 146-156.

Beven, K., 2012. *Rrainfall-Runoff Mdelling: The Primer,* s.l.: Wiley-Blackwell.

Bostock, M., 2020. *Data-Driven Documents D3.* [Online]   
Available at: https://d3js.org/  
[Accessed May 2020].

Brendel, C. E., Dymond, R. L. & Aguilar, M. F., July 2019. An interactive web app for retrieval, visualization and analysis of hydrologic and meteorological time series data. *Environmental Modelling & Software,* Volume 117, pp. 14-28.

Chow, V. T., Maidment, D. R. & Mays, L. W., 1988. *Applied Hydrology.* 1 ed. USA: McGraw-Hill.

Conner, L. G., Ames, D. P. & Gill, R. A., November 2013. HydroSever Lite as an open source solution for archiving and sharing environmental data for independent university labs. *Ecological Informatics,* Volume 18, pp. 171-177.

Dawson, C., Abrahart, R. & See, L., July 2007. HydroTest: A web-based toolbox of evaluation metrics for the standardised assessment of hydrological forecasts. *Environmenal Modelling & Software,* 22(7), pp. 1034-1052.

Delipetrev, B., Jonoski, A. & Solomatine, D. P., 2014. Development of a web application for water resources based on open source software. *Computers & Geosciences,* Volume 62, pp. 35-42.

Devi, G., Ganasri, B. & Dwarakish, G., 2015. A Review on Hydrological Models. *Aquatic Procedia,* Volume 4, pp. 1001-1007.

Documentation.js, 2020. *Documentation.js readme.* [Online]   
Available at: https://github.com/documentationjs/documentation  
[Accessed July 2020].

Engelschall, R. S., 2017. *ECMAScript 6 - New Features: Overview and Comparison.* [Online]   
Available at: http://es6-features.org/#ClassInheritance  
[Accessed 24 May 2020].

Environmental Agency UK, 2020. *Environment Agency Real Time flood-monitoring API.* [Online]   
Available at: https://environment.data.gov.uk/flood-monitoring/doc/reference  
[Accessed May 2020].

FEMA, 2020. *OpenFEMA API Documentation.* [Online]   
Available at: https://www.fema.gov/openfema-api-documentation  
[Accessed May 2020].

Fielding, R. T., University of California,. *Architectural Styles and the Design of Network-based Software Architectures.* Irvinie: Dissertation.

Foglia, L., Borsi, I. & Rossetto, R., April 2015. FREEWAT: FREE and open source software tools for Water Resource Management. *Rendiconti Online Societa Geologica Italiana,* Volume 35, pp. 252-255.

Georgy, A., 2016. *LHMP: lumped hydrological modelling playground.* [Online]   
Available at: https://zenodo.org/record/59501#.XzPm1jVS-Uk  
[Accessed April 2020].

Google Maps Platform, 2020. *Maps Javascript API.* [Online]   
Available at: https://developers.google.com/maps/documentation/javascript/tutorial  
[Accessed June 2020].

Google, 2020. *Google Visualization API.* [Online]   
Available at: https://developers.google.com/chart/interactive/docs/reference  
[Accessed May 2020].

Heistermann, M., Jacobi, S. & Pfaff, T., February 2013. An open source library for processing weather radar data (wradlib). *Hydrology and Earth System Sciences,* Volume 17, pp. 863-871.

JQuery, 2020. *JQuery API.* [Online]   
Available at: https://api.jquery.com/  
[Accessed 24 May 2020].

Klemes, V., 1982. *Empirical and Causal Models in Hydrology,* s.l.: National Hydrology Research Institute, Enviroment Canada.

Knoesen, D. & Smithers, J., 2009. The development and assessment of a daily rainfall disaggregation model for Sotuh Africa. *Hydrological Sciences Journal,* 54(2), pp. 217-233.

Koutsoyiannis, D., 2003. *Rainfall dissaggregation methods: theory and applications.* Rome, Workshop on Statistical and Mathematical Methods for Hydrological Analysis.

Kraft, P., Vache, K. B., Frede, H.-G. & Breuer, L., June 2011. CMF: A Hydrological Programming Language Extension for Integrated Catchment Models. *Environmental Modelling & Software,* 26(6), pp. 828-830.

Ma, Y. et al., 2019. *Moving DeepLearning into Web Browsers: How Far Can We Go?.* Beijing, China: Peking University.

MeteoNetwork API, 2020. *MeteoNetwork API.* [Online]   
Available at: https://api.meteonetwork.it/documentation.html  
[Accessed May 2020].

METEOSTAT, 2020. *About Meteostat.* [Online]   
Available at: https://meteostat.net/en/about  
[Accessed May 2020].

MetPy Developers, 2019. *MetPy Documentation.* [Online]   
Available at: https://unidata.github.io/MetPy/latest/index.html  
[Accessed April 2020].

Mozilla, 2020. *What is Javascript?.* [Online]   
Available at: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\_steps/What\_is\_JavaScript  
[Accessed May 2020].

Muller, H. & Haberlandt, U., 2018. Temporal rainfall disaggregation using a multiplicative cascade model for spatial application in urban hydrology. *Journal of Hydrology,* Volume 556, pp. 847-864.

Nadareishvili, I. M. R., Matt, M. & Amundsen, M., 2016. *Mircoservice Architecture: Aligning principles, practices and culture.* 1 ed. USA: O'Reily Media.

NOAA, 2020. *Climate Data Online: Web Services Documentation.* [Online]   
Available at: https://www.ncdc.noaa.gov/cdo-web/webservices/v2#data  
[Accessed June 2020].

NOAA, 2020. *REST Services Usage.* [Online]   
Available at: https://www7.ncdc.noaa.gov/rest/  
[Accessed May 2020].

Punchatz, C., 2017. *How JavaScript Became the Dominant Language of the Web.* [Online]   
Available at: https://www.lform.com/blog/post/how-javascript-became-the-dominant-language-of-the-web  
[Accessed May 2020].

PyMT Devs, 2016. *Pymt: the python modeling toolkit.* [Online]   
Available at: https://github.com/csdms/pymt  
[Accessed April 2020].

Raghunath, H., 2006. Chapter 5: Hydrographs. In: *Hydrology: Principles, Analysis, Design.* Delhi: New Age International Limited, pp. 117-157.

Raseman, W. J., J. J. & Kasprzyk, J. R., June 2019. Parasol: an open source, interactive parallel corodinates library for multi-objective decision making. *Environmental Modelling & Software,* Volume 116, pp. 153-163.

Roberge, M., 2016. *Hydrofunctions.* [Online]   
Available at: https://github.com/mroberge/hydrofunctions  
[Accessed April 2020].

Roberge, M. C., McGuire, M. P. & Jie, L., 2017. HydroCloud: A Web Application for Exploring Stream Gage Data. *Journal of Open Research Software,* 5(1), p. 18.

Rouse, M., 2020. *RESTful API.* [Online]   
Available at: https://searchapparchitecture.techtarget.com/definition/RESTful-API  
[Accessed May 2020].

Santos, L., Thirel, G. & Perrin, C., 2018. Continous state-space representation of a bucket-type rainfall-runoff model: a case study with GR4 model using state-space GR4. *Geoscientific Model Development,* Volume 11, pp. 1591-1605.

Schellekens, J. et al., 2019. *openstreams/ wflow 2019.1.* [Online]   
Available at: https://zenodo.org/record/3528154#.XzPgZTVS-Uk  
[Accessed May 2020].

Sharify, S. & Hosseini, M., 2011. Methodology for Identifying the Best Equations for Estimating the Time of Concentration of Watersheds in a Particular Region. *Journal of Irrigiation and Drainage Engineering,* 137(11).

Shaw, E., 1998. Unit hydrograph method in UK flood studies. *Encyclopedia of Hydrorlogy and Lakes,* 225(6), pp. 4020-4497.

Sitterson, J. et al., 2017. *An Overview of Rainfall-Runoff Model Types,* Athens, Georgia: EPA.

Swain, N. R. et al., November 2016. A new open source platform for lowering the barrier for environmental web app development. *Environmental Modelling & Software,* Volume 85, pp. 11-26.

Swain, N. R. et al., May 2015. A review of oepnsource software solutons for developing water resources web applications. *Environmental Modelling & Software,* 67(C).

TechnoMerge, 2020. *Types of API: How API Are Classified?.* [Online]   
Available at: https://www.technomerger.com/2020/03/types-of-apis.html  
[Accessed May 2020].

Tensorflow, 2020. *Tensorflow.js.* [Online]   
Available at: https://www.tensorflow.org/js  
[Accessed July July].

Tyralia, C. & Schumann, A. H., 2013. *HydPy-ein interaktiv nutzbares Framework zur Erstellung und Anwendung hydrologischer Modelle.* Klobenz: Ruhr-Universitet Bochum.

Ulmo, 2020. *Ulmo Documentation.* [Online]   
Available at: https://ulmo.readthedocs.io/en/latest/  
[Accessed May 2020].

USDA, 2007. Chapter 16: Hydrographs. In: USDA, ed. *Hydrologyy National Engineering Handbook.* s.l.:USDA.

USDA, 2015. Chapter 15: Time of Concentration. In: USDA, ed. *Hydrology National Engineering Handbook.* s.l.:National Resources Conservatioon Service.

Vaze, J. et al., 2011. Conceptual Rainfall-Runoff Model Perfromance with Different Spatial Rainfall Inputs. *Journal of Hydrometeorology,* 12(5), pp. 1100-1112.

Vemuri, V., 1993. *Main problems and issues in neural networks application.* New Zealand, IEEE.

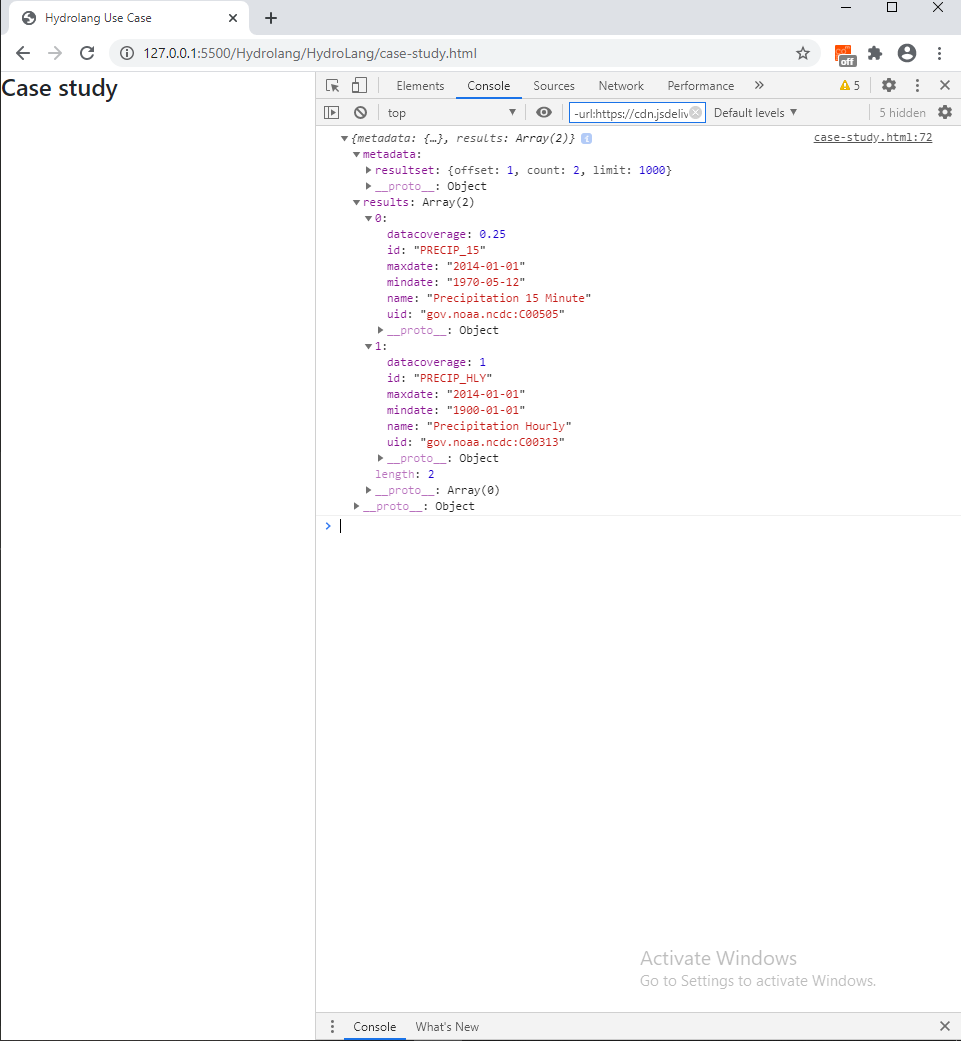
Xu, C., 2002. *Hydrologic Models.* 2 ed. Upssala: Uppssa;a University Department of Earth Sciences Hydrology.

# Appendix

## Case study 1

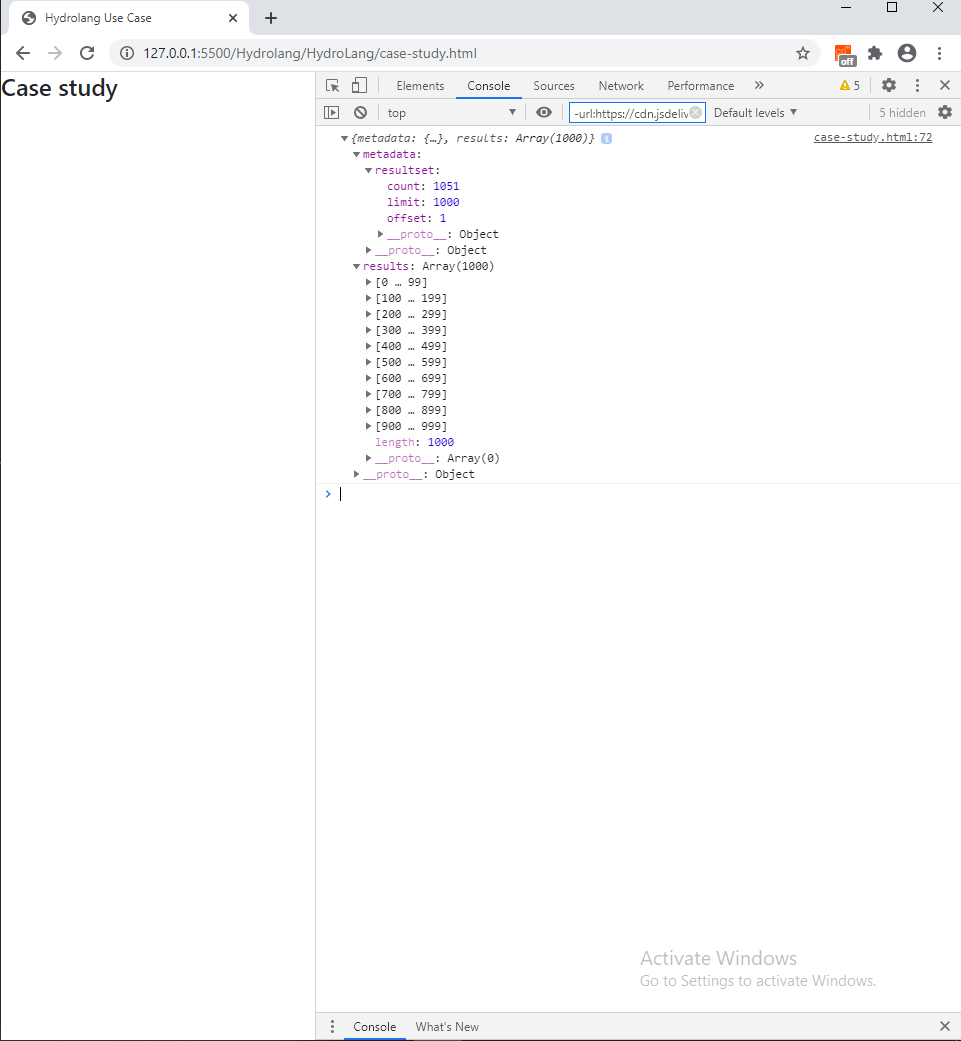
Objects used for data retrieval of the US site

1. dataRetrievalParams2 = {
2. source: "noaa",
3. dataType: "datasets",
4. type: "json",
5. arguments: {
6. stationid: "COOP:440166"
7. datasetid: "PRECIP\_15",
8. limit: 1000,
9. },
10. token: tokennoa1,
11. };



Data from 1984

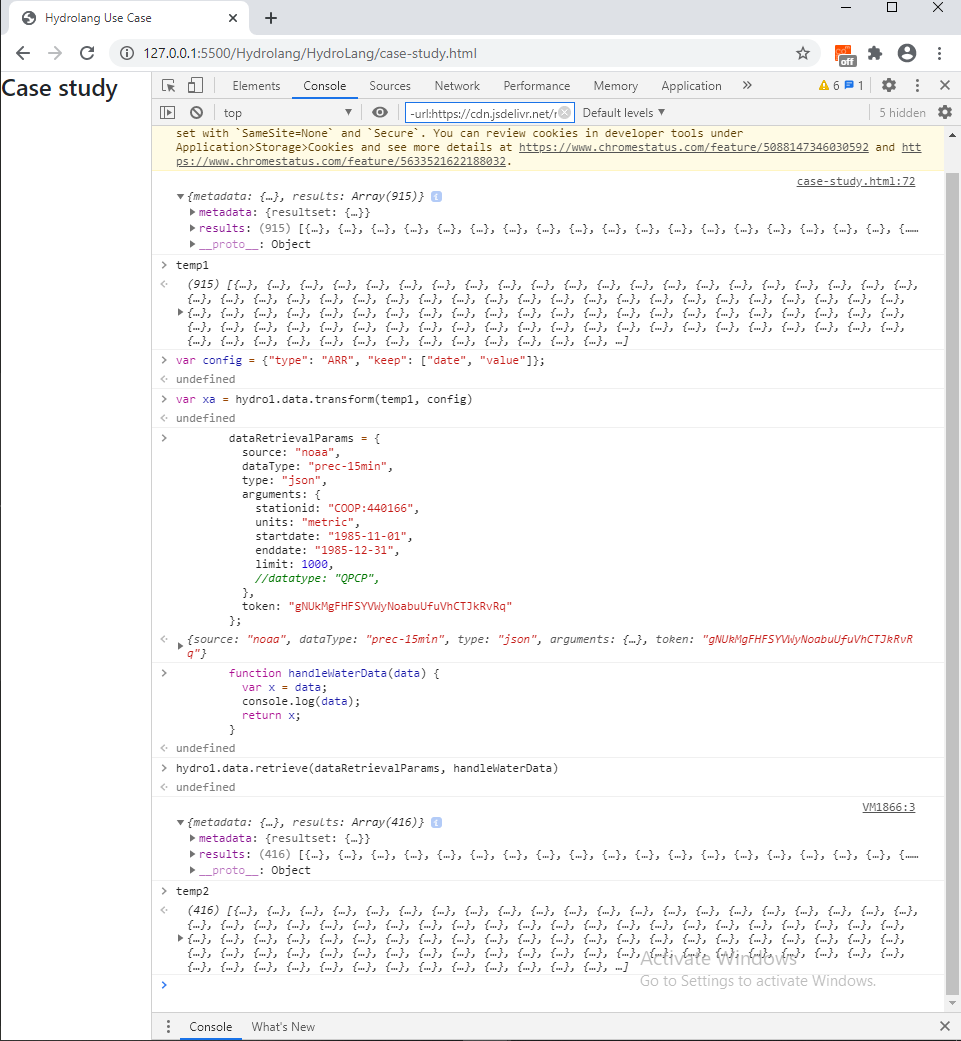
1. dataRetrievalParams3 = {
2. source: "noaa",
3. dataType: "prec-15min",
4. type: "json",
5. arguments: {
6. stationid: "COOP:440166"
7. units: "metric",
8. startdate: "1984-01-01",
9. enddate: "1984-12-31",
10. limit: 1000,
11. datatype: "QPCP",
12. },
13. token: tokennoa1,
14. };

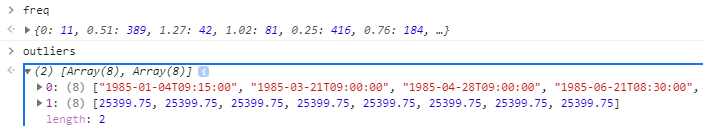


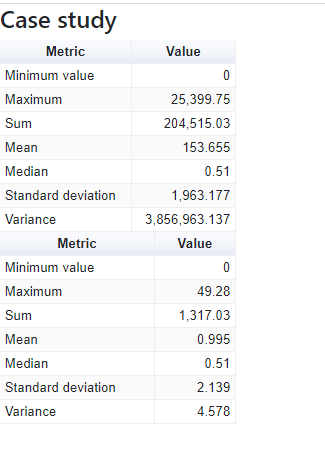
Object requests

1. //Data for 1985
2. dataRetrievalParams1 = {
3. source: "noaa",
4. dataType: "prec-15min",
5. type: "json",
6. arguments: {
7. stationid: "COOP:440166"
8. units: "metric",
9. startdate: "1985-01-01",
10. enddate: "1985-10-31",
11. limit: 1000,
12. datatype: "QPCP",
13. },
14. token: tokennoa1,
15. };

1. dataRetrievalParams2 = {
2. source: "noaa",
3. dataType: "prec-15min",
4. type: "json",
5. arguments: {
6. stationid: "COOP:440166"
7. units: "metric",
8. startdate: "1985-11-01",
9. enddate: "1985-12-31",
10. limit: 1000,
11. datatype: "QPCP",
12. },
13. token: tokennoa1,
14. };

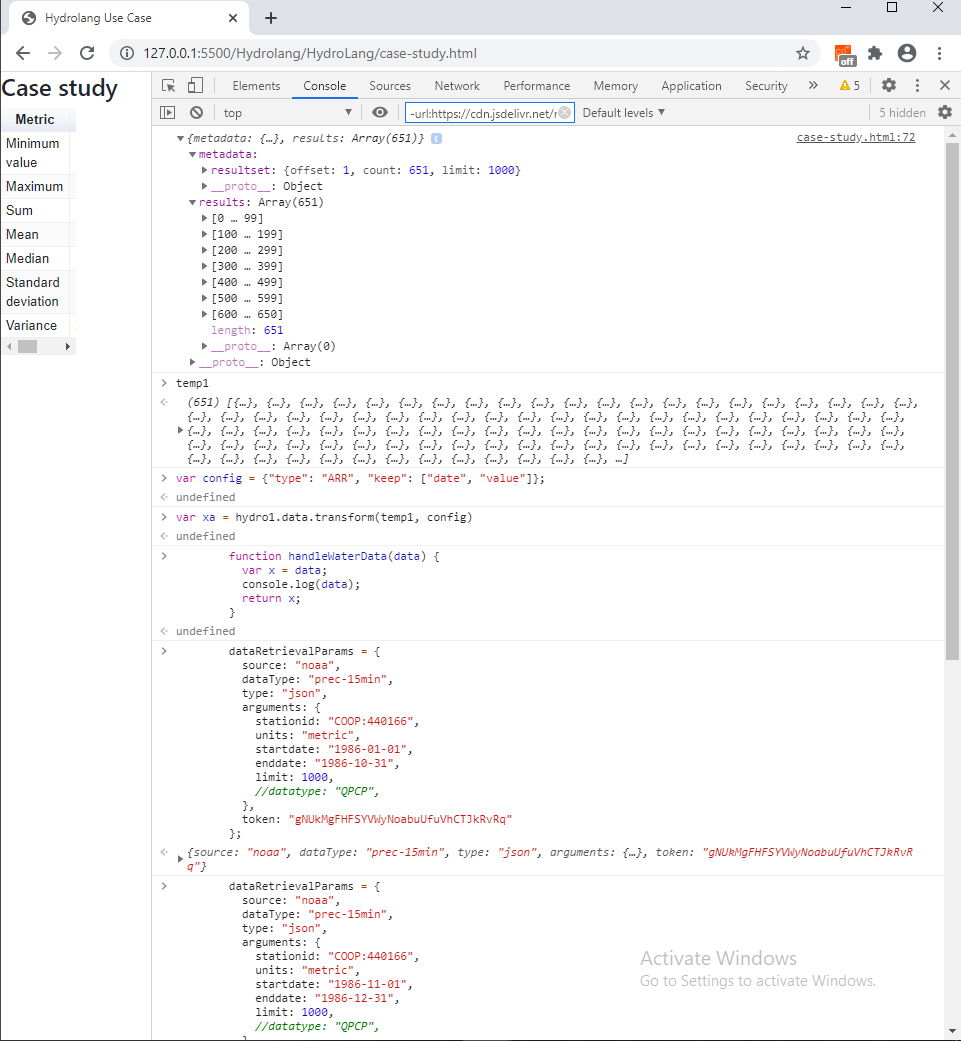




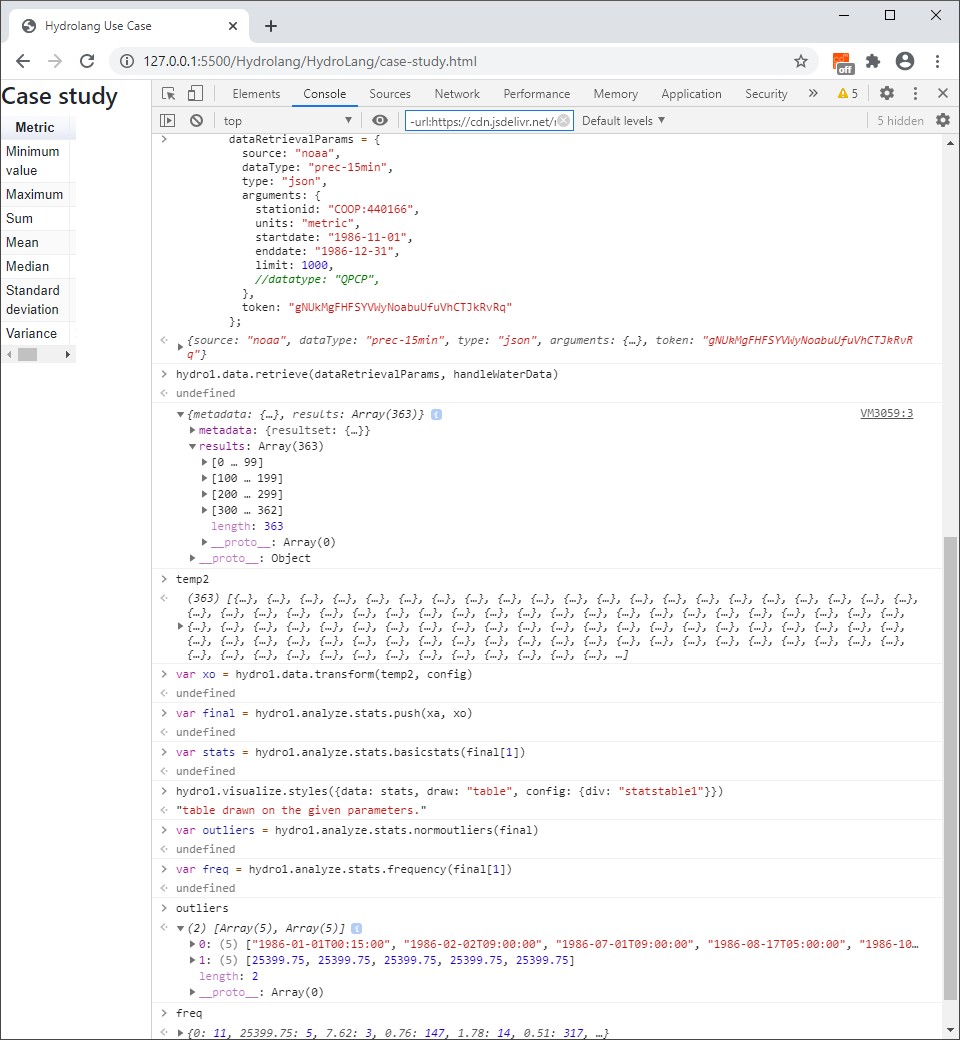


Raw versus cleaned data.

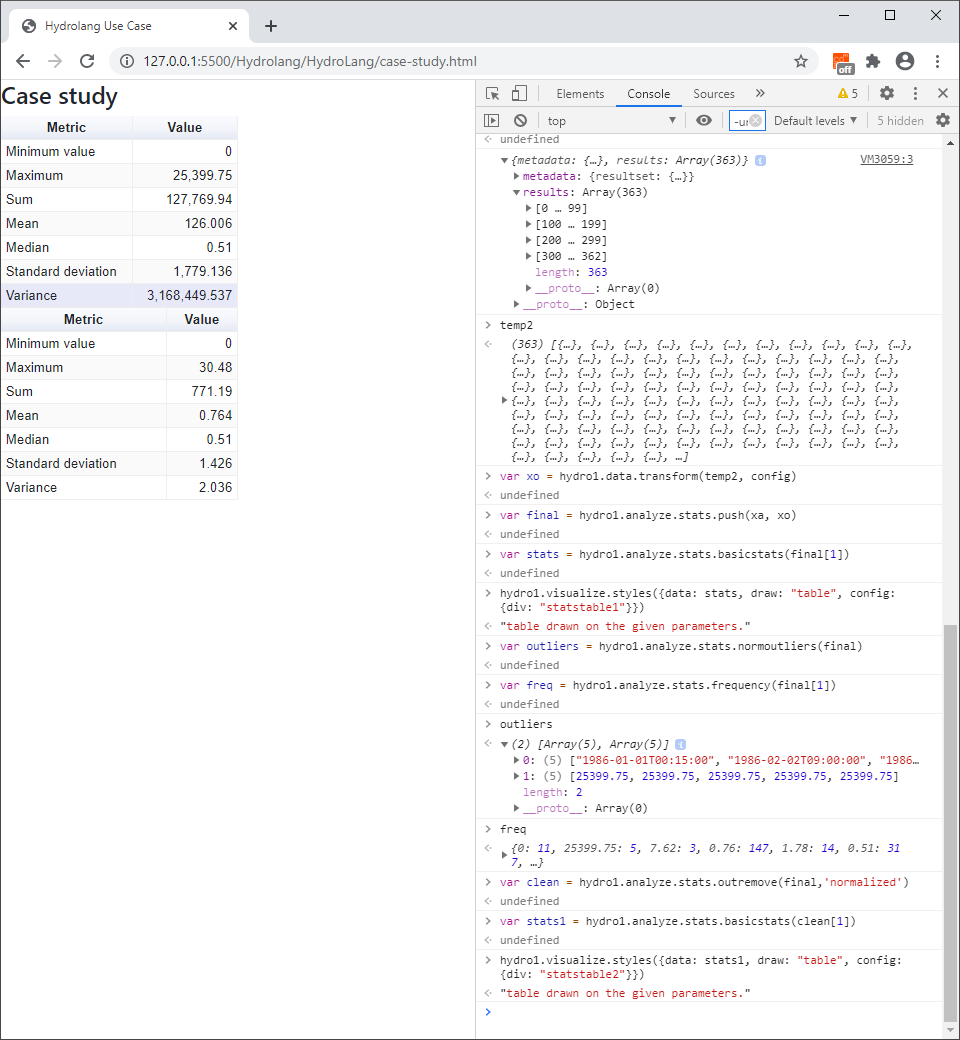
1. //Data for 1986
2. dataRetrievalParams1 = {
3. source: "noaa",
4. dataType: "prec-15min",
5. type: "json",
6. arguments: {
7. stationid: "COOP:440166"
8. units: "metric",
9. startdate: "1986-01-01",
10. enddate: "1986-10-31",
11. limit: 1000,
12. datatype: "QPCP",
13. },
14. token: tokennoa1,
15. };
16. dataRetrievalParams2 = {
17. source: "noaa",
18. dataType: "prec-15min",
19. type: "json",
20. arguments: {
21. stationid: "COOP:440166"
22. units: "metric",
23. startdate: "1986-11-01",
24. enddate: "1987-12-31",
25. limit: 1000,
26. datatype: "QPCP",
27. },
28. token: tokennoa1,
29. }



Response from the first request.



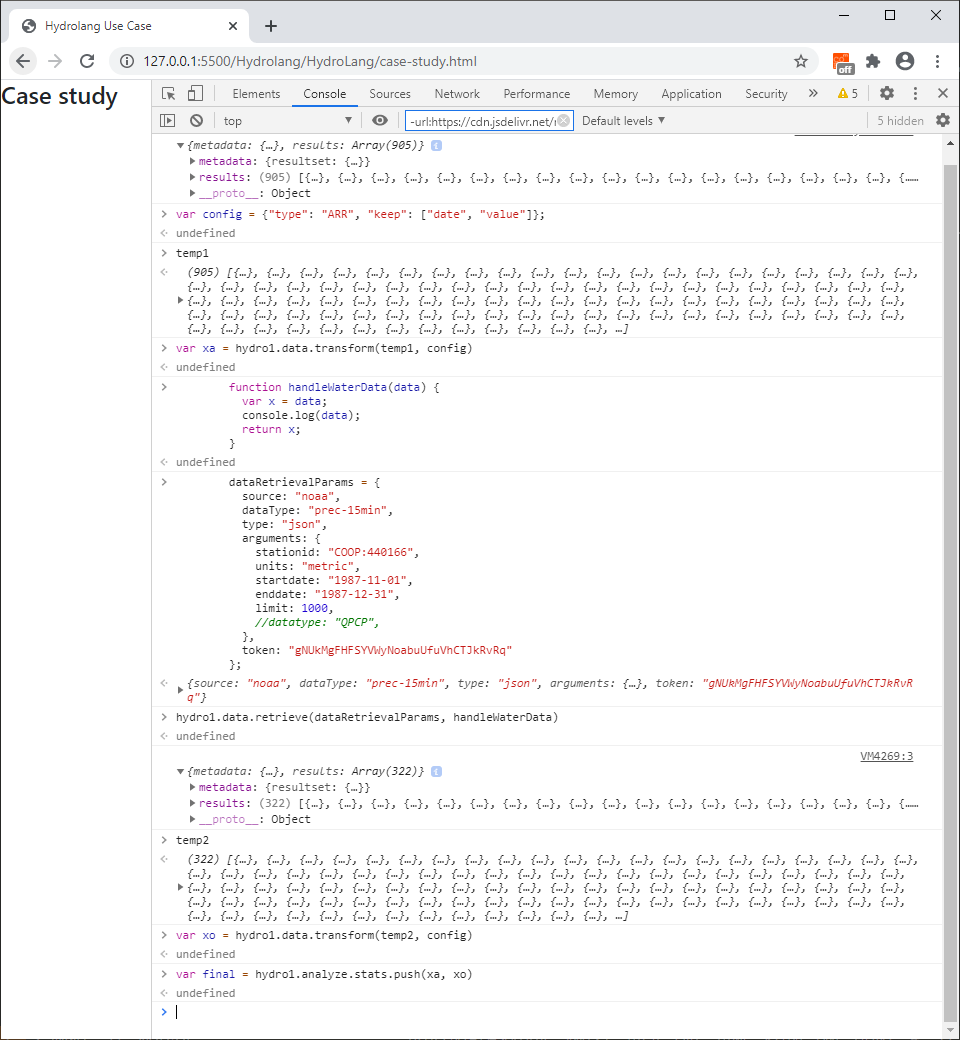
Second request, join two requests, identify outliers and frequency.



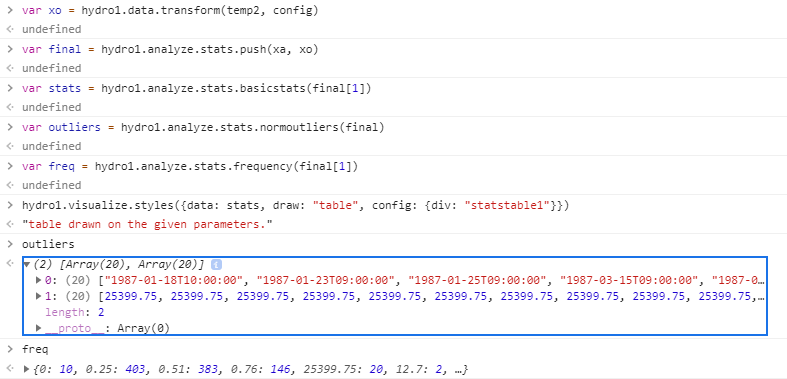
Raw data versus cleaned data.

1. //Data for 1987
2. dataRetrievalParams1 = {
3. source: "noaa",
4. dataType: "prec-15min",
5. type: "json",
6. arguments: {
7. stationid: "COOP:440166"
8. units: "metric",
9. startdate: "1987-01-01",
10. enddate: "1987-10-31",
11. limit: 1000,
12. datatype: "QPCP",
13. },
14. token: tokennoa1,
15. };

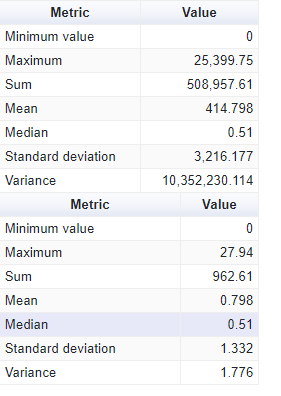
1. dataRetrievalParams2 = {
2. source: "noaa",
3. dataType: "prec-15min",
4. type: "json",
5. arguments: {
6. stationid: "COOP:440166"
7. units: "metric",
8. startdate: "1987-11-01",
9. enddate: "1987-12-31",
10. limit: 1000,
11. datatype: "QPCP",
12. },
13. token: tokennoa1,
14. };



Response for both requests and joining the two sets of data.



Outliers and frequency for raw data.



## Usages

If an object has been established as argument, the names inside the description represent the keys of the object. If it is an array instead, then the type of data will be specified. Time series can be both represented in the following formats:

1. var timeseries1: [["date1", "date2", "date3"], [1,2,3]]
2. var timeseries2: [[0.5,1,1.5],[1,2,3]]

 If an argument example mentions: “If time series, either in date string or number”, one of the two array types from above are expected. For more detailed explanations or examples on usage, please reference to the repository.

\* helper functions.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Analyze | *hydro* | Totalprec | arr - array of arrays | var arr = n\*array | var x = hydro[n].analyze.hydro.totalprec(arr) |
|  |  | arithmetic | arr - array of numbers. | var arr = array | var x = hydro[n].analyze.hydro.arithmetic(arr) |
|  |  | thiessen | Object - precipitation and areas. | var obj = {rainfall: 1darray(numbers), areas: 1darray(numbers)} | var x = hydro[n].analyze.hydro.thiessen(obj) |
|  |  | bucketmodel | Object - precipitation, baseflow, evaporation as object, landuse as object, infiltration\* | var obj = {rainfall: 1d or 2darray(time series or just rainfall event. If time series, either in date string or number for date.), baseflow: number, evaporation:{data: 1d or 2darray(time series or just rainfall event. If time series, either in date string or number for date.)} landuse: {agriculture: number, barerock: number, grassland: number, forest: number, moorland: number}, infiltration: number } | var x = hydro[n].analyze.hydro.bucketmodel(obj) |
|  |  | floodhydro | Object - precipitation, unithydro, cn, stormduration, timestep | var obj = {rainfall: 2d array(time series either in date string or number for date), unithydro: 2d array(timeseries in number), cn: number, stormduration: number, timestep: number} | var x = hydro[n].analyze.hydro.floodhydro(obj) |
|  |  | dimunithydro | Object - distribution as object, timestep, numhours | var obj = {distribution: {type: string, PRF: number}, timestep: number, numhours: number} | var x = hydro[n].analyze.hydro.dimunithydro(obj) |
|  |  | ground1d | Object - length, k, nodes, w0, w1, hL, q0, qL | var obj = {length: number, k: number, w0: number, w1: number, hL: number, q0: number, qL: number} | var x = hydro[n].analyze.hydro.ground1d(obj) |
|  |  | \*move | arr - array of numbers, from - number, to - number | var arr = array of numbers, from - number, to - number | hydro[n].analyze.hydro.move(arr, from, to) |
|  |  | \*matrix | m - number, n - number, d - number | m - number, n - number, d - number | var x = hydro[n].analyze.hydro.matrix(m, n, d) |
|  |  | \*equationsystemsolver | matrix - m\*array, vec\_left - 1d array, vec\_right - 1d array | matrix - m\*array, vec\_left - 1d array, vec\_right - 1d array | hydro[n].analyze.hydro.equationsystemsolver(m, n, d) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Analyze | *stats* | copydata | data - multidim array or object. | var data = data (timeseries or data) | var x = hydro[n].analyze.stats.copydata(data) |
|  |  | onearray | data - array with data | var data = 2darray (timeseries) | var x = hydro[n].analyze.stats.onearray(data) |
|  |  | datagaps | arr - array with data | var arr = arr (1d or 2darray. Timeseries in date string or number) | var x = hydro[n].analyze.stats.datagaps(arr) |
|  |  | gapremoval | arr - array with data | var arr = arr (1d or 2darray. Timeseries in date string or number for date). | var x = hydro[n].analyze.stats.gapremoval(arr) |
|  |  | sum | arr - array with numbers | var arr = 1darr(values) | var x = hydro[n].analyze.stats.sum(arr) |
|  |  | mean | arr - array with numbers | var arr = 1darr(values) | var x = hydro[n].analyze.stats.mean(arr) |
|  |  | stddev | arr - array with numbers | var arr = 1darr(values) | var x = hydro[n].analyze.stats.stddev(arr) |
|  |  | sumsqrd | arr - array with numbers | var arr = 1darr(values) | var x = hydro[n].analyze.stats.sumsqrd(arr) |
|  |  | min | arr - array with numbers | var arr = 1darr(values) | var x = hydro[n].analyze.stats.min(arr) |
|  |  | max | arr - array with numbers | var arr = 1darr(values) | var x = hydro[n].analyze.stats.max(arr) |
|  |  | unique | arr - array with numbers | var arr = 1darr(values) | var x = hydro[n].analyze.stats.unique(arr) |
|  |  | standardize | arr - array with numbers | var arr = 1darr(values) | var x = hydro[n].analyze.stats.standardize(arr) |
|  |  | quantile | arr - array with numbers, q - number | var arr = 1darr(values), var q = number | var x = hydro[n].analyze.stats.quantile(arr, q) |
|  |  | cleaner | arr - array with data | var arr = 1d array (data) | var x = hydro[n].analyze.stats.cleaner(arr) |
|  |  | outremove | arr - array with data, type - string, p1 - number, p2 - number | var arr = 1d or 2d array (Timeseries or just rainfall event. If time series, then date string or number for date.) var type = string (normalized or interquartile). | var x= hydro[n].analyze.stats.outremove(arr, type,p1, p2) |
|  |  | itemfilter | arr1 - array with data to be cleaned, arr2 - array with data to be removed. | var arr1, arr2 - 1d array of dates (number or string) or values | var x = hydro[n].analyze.stats.itemfilter(arr1, arr2) |
|  |  | correlation | Object - q1, q2 | var obj = {q1: 1darray (data), q2: 1darray (data)} | var x = hydro[n].analyze.stats.correlation(obj) |
|  |  | variance | arr - array with numbers | var arr = 1darr(values) | var x = hydro[n].analyze.stats.variance(arr) |
|  |  | frequency | arr - array with numbers or strings | var arr = 1darray (numbers or strings) | var x = hydro[n].analyze.stats.frequency(arr) |
|  |  | interoutliers | arr - array with data, q1 - number, q2 - number | var arr = 1d or 2d array (Timeseries or just rainfall event. If time series, then date string or number for date.) var q1 = number, q2 = number | var x = hydro[n].analyze.stats.interoutliers(arr, q1, q2) |
|  |  | normoutliers | arr - array with data, low - number, high - number | var arr = 1d or 2d array (Timeseries or just rainfall event. If time series, then date string or number for date), var low= number, var high = number | var x = hydro[n].analyze.stats.normoutliers(arr, low, high) |
|  |  | fastfourier | arr - array with numbers | var arr = 1darray (numbers) | var x = hydro[n].analyze.stats.fastfourier(arr) |
|  |  | basicstats | arr - array with numbers | var arr = 1darray (numbers) | var x = hydro[n].analyze.stats.basicstats(arr) |
|  |  | \*joinarray | arr - array with numbers or strings | var arr = 1darray (numbers or strings) | var x = hydro[n].analyze.stats.joinarray(arr) |
|  |  | \*numerise | arr - array with strings | var arr = 1darray (strings) | var x = hydro[n].analyze.stats.numerise(arr) |
|  |  | \*flatenise | Object - Columns, graphdata | var obj = {Columns: 1darray (strings), graphdata: 1darray(numbers)} | var x = hydro[n].analyze.stats.flatenise(obj) |
|  |  | \*dateparser | arr - array with strings | var arr = 1darray (strings) | var x = hydro[n].analyze.stats.dateparser(arr) |
|  |  | \*arrchange | arr - array (numbers or strings) | var arr = n\*m array | var x = hydro[n].analyze.stats.arrchange(arr) |
|  |  | \*push | arr1 - array which data will be pushed, arr2 - array with data to be pushed | arr1 = 1darray (numbers or strings), arr2 = 1darray (numbers or strings) | var x = hydro[n].analyze.hydro.push(arr1, arr2) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Analyze | *NN* | createModel | numinputs - number, numneurons - number, numoutputs - number | numinputs = number, numneurons = number, numoutputs = number (consistent with numinputs) | hydro[n].analyze.nn.createModel(numinputs, numneurons, numoutputs) |
|  |  | convertToTensor | arr1 - array with numbers, arr2 - array with numbers | arr1 = 1darray (numbers), arr2 = 1darray (numbers) | hydro[n].analyze.nn.convertToTensor(arr1, arr2) |
|  |  | trainModel | model - Object, inputs: 1darray, outputs: 1darray | model = Object (tensorflow), inputs = tensorflow arrays, outputs = tensorflow arrays | hydro[n].analyze.nn.trainModel(model, inputs, outputs) |
|  |  | prediction | model - Object, inputData: 1darray, normalizedData: Object | model = Object (tensorflow), inputData = tensorflow arrays, normalizedData = tensorflow Object | hydro[n].analyze.nn.prediction(model, inputData, normalizedData) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Data | *data* | retrieve | Object - source, dataType, type, arguments as object, token (if required), proxy (if required) | var obj = {source: string, dataTyep: string, arguments: {options: strings}, token (if required): string, proxy (if required): string} | hydro[n].data.retrieve(obj, callback) |
|  |  | transform | data - Object with data, Object - configuration options | data = Object (retrieved data or JS arrays), var obj = {type: string, keep : 1d array} | var x = hydro[n].data.transform(data, obj) |
|  |  | upload | type - string | type - string (either CSV or JSON) | var x = hydro[n].data.upload(type) |
|  |  | download | data - Object with data, Object - configuration options | data = Object (retrieved data or JS arrays), var obj = {type: string, keep : 1d array} | hydro[n].data.download(data, obj) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Visualize | *visualize* | chart | Object - chartType, data, divID, options | var obj = {chartType: string (column, line, scatter, histogram, timeline), divID: string, data: 1d or 2d array, options: {}} | hydro[n].visualize.chart(obj) |
|  |  | table | Object - chartType, data, divID, options | var obj = {chartType: string (table), divID: string, data: 1d array, dataType: 1d array (same size as data), options: {}} | hydro[n].visualize.table(obj) |
|  |  | styles | Object - chartType, data, divID, options as object | var obj = {chartType: string (charts or table), data: 1d or 2darray depending on chartType, divID: string, options: {title: string}} | hydro[n].visualize.styles(obj) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Map | *maps* | loader | Object - map | var obj = {maptype: string} | hydro[n].map.loader(obj) |
|  |  | Layers | Object - maptype, layertype as object | var obj2 = {maptype: string, layertype: {type: string, markertype: string, coord: 1d array, name: string}} var obj3 = {maptype: string, layertype: {type: string}} | hydro[n].map.Layers(obj) |
|  |  | renderMap | Object - maptype, lat, lon, zoom, layertype as object | var obj = {maptype: string, lat: number, lon: number, zoom, number, layertype: {type: string, name: string}} | hydro[n].map.renderMap(obj) |