Documentation for an online open data workflow for mechanical analysis of soft biomaterials

Alex Markopoulou Alexander Lake Anthony Rainey Djan Tanova
Kyle Watt Marina San Jose Pena Raphael Nekam

March 17, 2023

Abstract

The authors of this documentation developed an online open data workflow for mechanical analysis of soft biomaterials for the School of Engineering in the University of Glasgow. This tool will assist scientists in analysing the results from nano-indentation experiments, to understand the interaction between cells and materials.

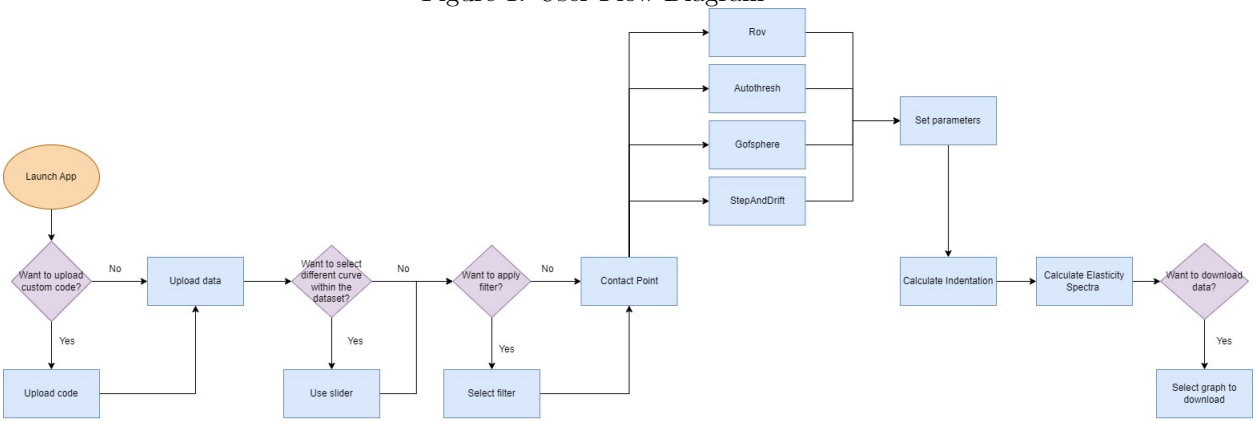
1 Overview

Once a data file has been uploaded to the app, the user can select a set of filters to apply on the data. Also, the contact point, indentation and the elasticity spectra can be calculated.

If wanted, the results from the calculations can be downloaded.

The following figure is a user flow diagram of a standard path a user will take.

Figure 1: User Flow Diagram



2 Installation

All the source code can be found in our group GitLab repository.

2.1 Install NodeJS

The executable can be downloaded through the website: see here. For Windows, we recommend installing the .mpi file and for MacOS, the .pkg. For Linux OS, Node.js has a guide for each Linux distribution: see here.

Run the executable to start the installation process.

After that is finished, you can run the following commands in the terminal to check that the installation has been successful.

To check the node version:

```
$ Node –v
```

To check the npm version:

```
$ npm –v
```

If you already have Node.js installed, you can update it with the following terminal command:

```
$ npm install –g npm@latest
```

2.2 Install Angular 15 CLI

You can use the Angular client to aid with the installation.

```
$ npm install –g @angular/cli@15
```

2.3 Install remaining dependencies

There is a package.json file that holds the dependencies used. To install them, navigate to the folder /ese2-main/nanoindentation-dashboard/ and run npm install.

```
$ cd /ese2–main/nanoindentation–dashboard/
```

2.4 Build application

To build and serve the application:

```
$ ng serve
```

The page can be visited on http://localhost:4200/.

2.5 Run Backend

The backend is built on Flask and is used for the AFMFormats library. The AFMFormats library is a Python library for reading atomic force microscopy (AFM) data file formats which can come in the form of a jpk-force-map. Jpk-force-map files are large data files usually created by machines in atomic force microscopy. These files can be very large and require an external library to parse. More about afmformats can be read here. To install these, a tutorial can be found here.

```
$ cd /ese2–main/
$ pip install flask flask–cors afmformats
$ python server.py
```

3 Usage

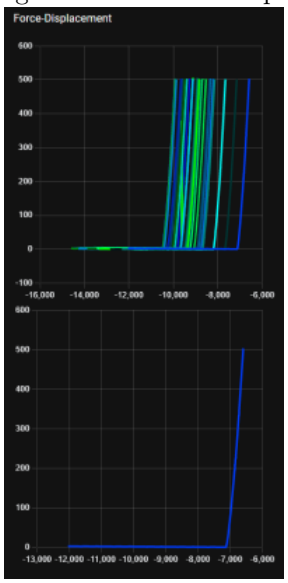
Visuals and Usage This is what you will see after building the application.

Figure 2: Website layout

3.1 Graphs

Three sets of graphs are displayed, each with a primary and secondary graph. The primary graph displays all of the uploaded curves, whereas the secondary graph displays one curve from the set, picked out by a slider on the sidebar.

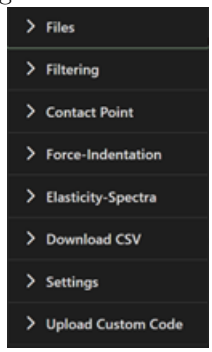
Figure 3: Curves Example



3.2 Sidebar

The interface is the sidebar will allow you to control the data shown in the graphs. Is made up of 8 parts.

Figure 4: Sidebar Tabs



3.3 Files

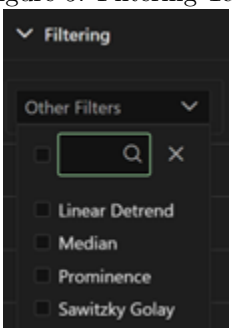
Files shows and gives the options to select which uploaded files to show in the graphs. There are also buttons to upload either text or jpk-force-map files which will automatically shown in the graphs as well as in the "Select file to use" box. There is a drop down menu that allows the user to activate filters. Multiple filters can be selected, and they will apply to all graphs.

3.4 Filtering

Filtering lets you search and select for the preset filters that can be applied to the uploaded datasets and viewed in the graphs.

Below this, there is a drop down menu that allows the user to activate filters. Multiple filters can be selected, and they will apply to all graphs.

Figure 5: Filtering Tab



3.5 Contact Point

The Contact Point tab defines which contact point calculation method wish to use. Contact point is the point at which the cantilever made contact with the surface. The Contact Point is needed in latter calculations such as Force-Indentation and Elasticity-Spectra

3.6 Force-Ind

The Force-Ind tab is used to control the second set of graphs - the Force-Indentation graphs. The feature is a drop down box which allows the user to select a spring constant from a range. There is also a toggle to choose to set "Zero Force"

3.7 Elasticity-Spectra

The Elasticity-Spectra tab is used to control the third set of graphs. The type of Tip geometry can be selected, The user can also choose the settings for the values for "Radius", "Win" and "Order" as well as toggle the option to use "Interpolation".

3.8 Download CSV

This is a drop-down feature which allows the user to download the datasets of either the "Force Displacement", "Force Indentation" or "Elasticity Spectra" and choose to either download the entire datasets or a single curve.

3.9 Settings

This is a drop-down feature which allows the user to download the settings they have selected for the displayed datasets as a text file or to upload their own desired settings

3.10 Upload Custom Code Tab

The Upload Custom Code Tab allows the user to upload and run there own custom filters and Contact Point equations on the data. Users select whether they are uploading a custom filter or Contact Point equation then they upload there code. Then the user will navigate to either the filters tab or Contact Point tab, depending on what they choose to upload, to select there custom code.

3.11 Curve Selection

This is a curve selection slider feature on the bottom bar which gives the user the option of easily selecting a single curve or looking through the single curves from the entire dataset with will automatically displayed on the bottom graphs.

3.12 Update Curves

When a change is made to any of the filters or selections then a button called "Update Curves" will appear allowing the graphs to apply any changes made to the displayed graphs

4 Custom Code

The interface allows to run custom python code for Filters and CPoint calculations. However, as the custom code has to interact with the javascript code of the interface, certain requirements have to met to do this successfully. This section is about how to add upload and use custom code, how the code must be structured and written, which python packages can be used etc.

4.1 Requirements

- **Correct file-type:** The "upload custom-code" function of the interface, supports the upload custom python code inside and common text file. This includes .txt, as well as .py files.
- **Correct function header and return values:** Every uploaded python file, must have a function following the header definition of

```
def calculate(x, y):
    ...
```

  and the return definition of

```
    ...
    return x,y
```

  This forms the starting point of every execution of the custom file. Hence, every other functions must be called from inside calculate.

  - **Input parameter x:** x is a list of all the x coordinates of the related curve. (E.g.: [1.2, 4.632, 3. 6.63]
  - **Input parameter y:** y is a list of all the y coordinates of the related curve. (E.g.: [1.2, 4.632, 3. 6.63]
  - **Return value x:** The required format depends on the type of custom code. If the code is for a filter, this must be the list of all manipulated x values of the related curve. (E.g.: [1.2, 4.632, 3. 6.63]). This also supports the return of numpy-arrays.
    If it is for the contact-point calculation, this should be a single numerical value (integer or floating point) representing the x-coordinate of the contact-point (E.g.: 3 or 6.24).
  - **Return value y:** The required format depends on the type of custom code. If the code is for a filter, this must be the list of all manipulated y values of the related curve. (E.g.: [1.2, 4.632, 3. 6.63]). This also supports the return of numpy-arrays.
    If it is for the contact-point calculation, this should be a single numerical value (integer or floating point) representing the y-coordinate of the contact-point (E.g.: 3 or 6.24).

- **Supported python packages and their usage:** The interface supports the usage of the numpy and the scipy python packages. To allow the usage in the custom code, the correct imports need to be done at the top of the custom code file, according to the needed packages. It follows the standard procedure used for python. Example for using both packages:

```
import numpy as np
import scipy as scipy
...
```

  The packages can then be used inside the python code as usual. Example of using numpy:

```
...
np.array(x)
...
```

If only certain methods of a package are needed, smaller imports can also be done such as:

```
from scipy.signal import medfilt
```

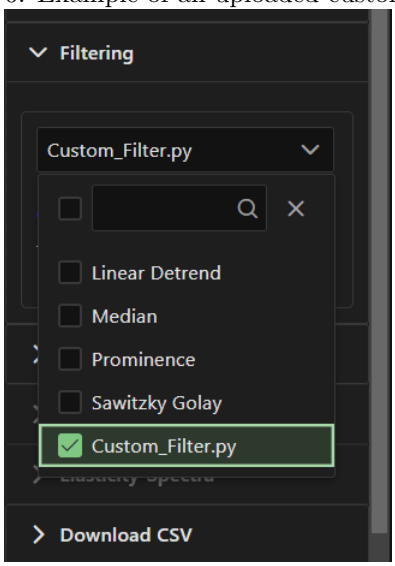
- **Correctly transforming the input parameters into numpy-arrays:** For most use cases of custom python code, users want to convert the value of the input parameters of x and y into numpy arrays, to allow easier use inside the python function. An easy approach to do that is using the following code snippet right after the definition of the calculate(x,y) function:

```
def calculate(x, y):
    x = [element for element in x]
    y = [element for element in y]
    x = np.array(x)
    y = np.array(y)
    ...
    YOUR CODE HERE
    ...
    return x,y
```

4.2 Upload and usage

- **Custom code can be uploaded in the custom-code tab:** First, the correct type of the custom-code must be selected. This can either be for Filter, or Contact-Point calculation. Pressing the "Upload"-Button, allows the selection of the file from the users machine.
- **Using already uploaded code:** The uploaded code, appears just as any predefined code in the related dropdown of the nanoindention-interface. The used label is the name of the uploaded file. Example of an uploaded filter:

Figure 6: Example of an uploaded custom filter



4.3 Examples

- **Code of a custom filter** This is an example of the python code of a possible custom filter that can be uploaded into the interface. This filter simply multiplies every y value of the curve by 10.

```
import numpy as np

def calculate(x, y):
  x = [element for element in x]
```

```
    y = [element for element in y]
    x = np.array(x)
    y = np.array(y)

    y = y * 10

    return x, y
```

- **Further examples** Even the actually used code for all processes run inside the nanoindentation-interface (filters, contact-point calculation, etc.) are written in python and follow the same guidance. The code for these can be found inside the repository at ./nanoindentation-dashboard/src/assets/Processes

5 Adding Pre-Set Code

Adding pre-set code to the application makes the process a permanent option for users. Pre-set processes are also able to use user inputs as parameters.

5.1 Defining a Process

For a new process to be recognised in the application, some metadata about the process is required. Within the *environment.ts* and *environment.prod.ts* files, there is a collection of *Process* objects called *WrittenProcesses* with the following definition:

```
export const WrittenProcesses: {
filters: Process[], cPoints: Process[], eModels: Process[],
fModels: Process[], internal: Process[], test: Process[] }
```

For a new process to be recognised in the application, a *Process* object representing the process needs to be declared and appended to the relevant array of processes of the same type.

- **Process Definition** A *Process* object can be declared using the following interface:

```
export interface Process
{
  id: string, // unique name to find place in filesystem
  name: string, // displayed name by the UI
  procType: EProcType, // type of the process, needed for correct display in UI, find place
  inputs?: Input[], //Required inputs by the process: array of custom input type object
  custom?: boolean, //whether the process is custom
  script?: string; //script of the process, loaded in when the process runs
}
```

- **EProcType Definition** The *EProcType* enum stores the value of what type of process the *Process* object is. The *EProcType* enum has the following definition:

```
export enum EProcType
{ //Types of processes
  FILTER = 'filter',
  CPOINT = 'cPoint',
  EMODELS = 'eModels',
  FMODELS = 'fModels',
  INTERNAL = 'internal',
  TEST = 'test'
}
```

5.2 Defining Inputs

Within *Process* objects that require user inputs for parameters, there is an array of *Input* objects that define the name, default value, and type of value the input is. The defined *Input* object is appended to the *inputs* array in the *Process* definition.

- **Input Definition** The following *Input* object definition can be used to add a user input into a *Process* object:

```
export interface Input
{
  name: string; //Name of input field
  selectedValue: any; //selected value of input field, set as default
  type: EInputFieldType; //type of input expected
}
```

- **EInputFieldType Definition**

  The *EInputFieldType* enum has the following definition:

```
export enum EInputFieldType
{ //Types of inputs
  BOOLEAN = 'boolean',
  NUMBER = 'number',
  GEOMETRY = 'geometry',
  DATAPOINT = 'datapoint'
}
```

5.3 Example Process Addition

The following *Process* object definition is used to add the *savgol* filter to the environment:

```
{
  id: 'savgol', name: 'Sawitzky Golay', procType: EProcType.FILTER, custom: false, inputs: [
    {name: 'Window', type: EInputFieldType.NUMBER, selectedValue: 25},
    {name: 'Order', type: EInputFieldType.NUMBER, selectedValue: 3}
  ]
}
```

6 Saving and importing settings

It is possible to export and re-upload all settings a user has applied to the dataset inside the nanoindentation-dashboard.

6.1 Structure of a Settings-File

Settings-Files are .txt files containing a JSON object representing all inputs made by a user. The file is named ”nanoindentation-settings.txt”. The JSON consists of the following fields:

- **availableProceses:** Contains all processes available. Including all custom code processes a user uploaded. This makes custom code reusable. The processes also contain every value selected by the user
- **selectedFilters:** This contains all filters selected by the user
- **selectedCPointProcess:** This field contains the Contact-Point process the user used for their manipulation of the data

  Here is an example of how exported settings look like:

Figure 7: Example of a nanoindentation-settings


```
{"availableProcesses":{"filters":[{"id":"linearDetrend","name":"Linear Detrend",
"procType":"filter","custom":false},{"id":"median","name":"Median","procType":
"filter","custom":false,"inputs":[{"name":"Window","type":"number","selectedValue"
:999}]},{"id":"prominence","name":"Prominence","procType":"filter","custom":false,
"inputs":[{"name":"Band [% preak pos]","type":"number","selectedValue":999},{"name"
:"Pro","type":"number","selectedValue":999},{"name":"Threshold","type":"number",
"selectedValue":999}]},{"id":"savgol","name":"Sawitzky Golay","procType":"filter"
,"custom":false,"inputs":[{"name":"Window","type":"number","selectedValue":999},{
"name":"Order","type":"number","selectedValue":999}]},{"id":"All_Times_Ten.py","name":
"All_Times_Ten.py","custom":true,"procType":"filter","script":"from scipy.signal
import savgol_filter\r\nimport numpy as np\r\n\r\ndef calculate(x, y):\r\n  x = [element
 for element in x]\r\n  y = [element * 10 for element in y]\r\n  x = np.array(x)
\r\n  y = np.array(y)\r\n\r\n  return x, y\r\n"}],"cPoints":[{"id":"autothresh",
"name":"Auto-Threshold","procType":"cPoint","custom":false,"inputs":[{"name":"Zero
Range","type":"number","selectedValue":999}]},{"id":"gofSphere","name":"GofSphere",
"procType":"cPoint","custom":false,"inputs":null},{"id":"rov","name":"Rov","procType":
"cPoint","custom":false,"inputs":null},{"id":"stepAndDrift","name":"Step and Drift",
"procType":"cPoint","custom":false,"inputs":[{"name":"Window","type":"number",
"selectedValue":101},{"name":"Threshold","type":"number","selectedValue":10},{"name":
"ThRatio","type":"number","selectedValue":25}]}],"eModels":[],"fModels":[],"internal":
[{"id":"calc_indentation","name":"Calculating Indentation","procType":"internal","custom"
:false,"inputs":[{"name":"CP","selectedValue":{"x":11123.014758,"y":2229.18},"type":
"datapoint"},{"name":"Spring constant","selectedValue":1,"type":"number"},{"name":
"Set Zero Force","selectedValue":true,"type":"boolean"}]},{"id":"calc_elspectra","name"
:"Calculating Elasticity Spectra","procType":"internal","custom":false,"inputs":[{"name"
:"Tip Geometry","selectedValue":{"name":"Sphere","value":"Sphere"},"type":"geometry"},
{"name":"Radius","selectedValue":1,"type":"number"},{"name":"Win","selectedValue":100,
"type":"number"},{"name":"Order","selectedValue":2,"type":"number"},{"name":"Use
Interpolation","selectedValue":false,"type":"boolean"}]}],"test":[]},"selectedFilters":
[{"id":"prominence","name":"Prominence","procType":"filter","custom":false,"inputs":
[{"name":"Band [% preak pos]","type":"number","selectedValue":999},{"name":"Pro","type":
"number","selectedValue":999},{"name":"Threshold","type":"number","selectedValue":999}]},
{"id":"All_Times_Ten.py","name":"All_Times_Ten.py","custom":true,"procType":"filter",
"script":"from scipy.signal import savgol_filter\r\nimport numpy as np\r\n\r\ndef
calculate(x, y):\r\n  x = [element for element in x]\r\n  y = [element * 10 for element
 in y]\r\n  x = np.array(x)\r\n  y = np.array(y)\r\n\r\n  return x, y\r\n"},{"id":"median"
,"name":"Median","procType":"filter","custom":false,"inputs":[{"name":"Window",
"type":"number","selectedValue":999}]},{"id":"linearDetrend","name":"Linear Detrend",
"procType":"filter","custom":false},{"id":"savgol","name":"Sawitzky Golay","procType":
"filter","custom":false,"inputs":[{"name":"Window","type":"number","selectedValue":999},
{"name":"Order","type":"number","selectedValue":999}]}],"selectedCPointProcess":{"id":
"autothresh","name":"Auto-Threshold","procType":"cPoint","custom":false,"inputs":[{"name"
:"Zero Range","type":"number","selectedValue":999}]}}
```


6.2 Exporting/Downloading settings

The settings can be downloaded, using the "Download Settings"-Button inside the "Settings"-Tab. This downloads a .txt-File, containing a JSON object, filled with all settings applied by the user

6.3 Importing/Uploading settings

A downloaded "Settings"-File can be uploaded using the "Upload Settings"-Button inside the "Settings"-Tab. This allows the selection of any file from the users machine.

7 Exporting and downloading datasets

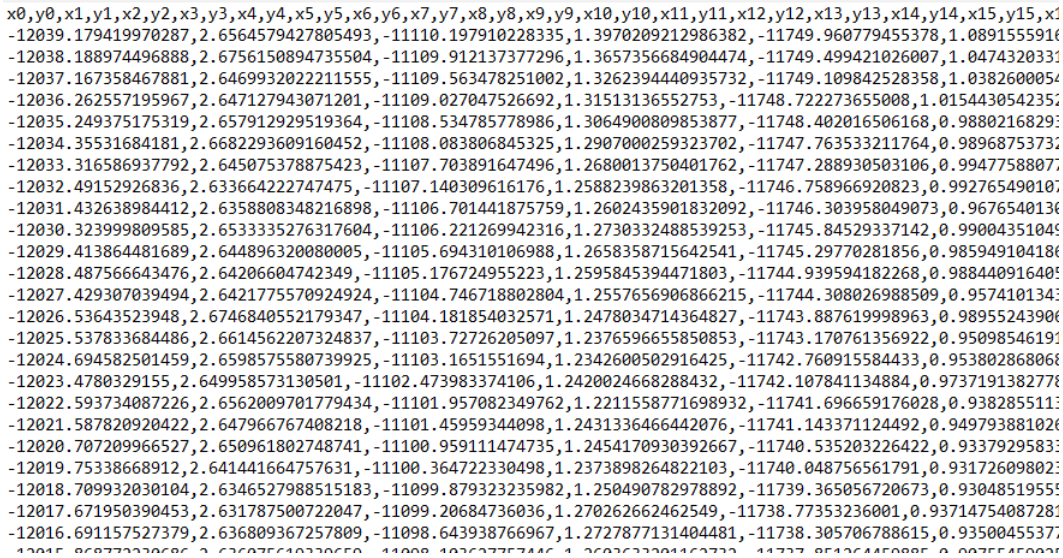
All data manipulation done in the interface can be exported using the "Download"-Button in the "Download-CSV"-Tab. Dropdowns can be used to select which data to download. This downloads a CSV.

7.1 Structure of the CSV

The csv is structured in rows and columns. Each column represents the x or y axis of one of the datasets/curves. The top row defines which axis it is, using x or y together with number starting from 0 to define which curve it is (E.g.: x0, y0, x1, y1, ...). All other rows then contain the numerical values of each axis, growing from top to bottom. As no all datasets are equally long, a dash ("-") is used to represent empty cells.

- **CSV delimiters:** Every new line in the exported file represents a new row. Any comma (",") represents a new column. An example is given here:

Figure 8: CSV as a Textfile example



```
x0,y0,x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,x7,y7,x8,y8,x9,y9,x10,y10,x11,y11,x12,y12,x13,y13,x14,y14,x15,y15,x1
-12039.179419970287,2.6564579427805493,-11110.197910228335,1.3970209212986382,-11749.960779455378,1.089155591
-12038.188974496888,2.6756150894735504,-11109.912137377296,1.3657356684904474,-11749.499421026007,1.047432033
-12037.167358467881,2.6469932022211555,-11109.563478251002,1.3262394440935732,-11749.109842528358,1.038260005
-12036.262557195967,2.647127943071201,-11109.027047526692,1.31513136552753,-11748.722273655008,1.015443054235
-12035.249375175319,2.657912929519364,-11108.534785778986,1.3064900809853877,-11748.402016506168,0.9880216829
-12034.35531684181,2.6682293609160452,-11108.083806845325,1.2907000259323702,-11747.763533211764,0.9896875373
-12033.316586937792,2.645075378875423,-11107.703891647496,1.2680013750401762,-11747.288930503106,0.9947758807
-12032.49152926836,2.633664222747475,-11107.140309616176,1.2588239863201358,-11746.758966920823,0.99276549010
-12031.432638984412,2.6358808348216898,-11106.701441875759,1.2602435901832092,-11746.303958049073,0.9676540136
-12030.323999809585,2.6533335276317604,-11106.221269942316,1.2730332488539253,-11745.84529337142,0.9900435104
-12029.413864481689,2.644896320080005,-11105.694310106988,1.2658358715642541,-11745.29770281856,0.985949104186
-12028.487566643476,2.64206604742349,-11105.176724955223,1.2595845394471803,-11744.939594182268,0.988440916405
-12027.429307039494,2.6421775570924924,-11104.746718802804,1.2557656906866215,-11744.308026988509,0.957410134
-12026.53643523948,2.6746840552179347,-11104.181854032571,1.2478034714364827,-11743.887619998963,0.9895524390
-12025.537833684486,2.6614562207324837,-11103.72726205097,1.2376596655850853,-11743.170761356922,0.950985461
-12024.694582501459,2.6598575580739925,-11103.1651551694,1.2342600502916425,-11742.760915584433,0.953802868068
-12023.4780329155,2.649958573130501,-11102.473983374106,1.2420024668288432,-11742.107841134884,0.973719138277
-12022.593734087226,2.6562009701779434,-11101.957082349762,1.2211558771698932,-11741.696659176028,0.938285511
-12021.587820920422,2.647966767408218,-11101.45959344098,1.2431336466442076,-11741.143371124492,0.949793881026
-12020.707209966527,2.650961802748741,-11100.959111474735,1.2454170930392667,-11740.535203226422,0.9337929583
-12019.75338668912,2.641441664757631,-11100.364722330498,1.2373898264822103,-11740.048756561791,0.931726098023
-12018.709932030104,2.6346527988515183,-11099.879323235982,1.250490782978892,-11739.365056720673,0.9304851955
-12017.671950390453,2.631787500722047,-11099.20684736036,1.270262662462549,-11738.77353236001,0.9371475408728
-12016.691157527379,2.636809367257809,-11098.643938766967,1.2727877131404481,-11738.305706788615,0.935045537
```

- **Importing CSV into MS Excel:** Most machines can open CSV-Files directly from the directory using a double click. Even the formatting is normally detected automatically. If that is not the case, the following steps can be used to import the CSV into MS Excel.
  - Open MS Excel
  - Go to the Tab: 'Data'
  - Select 'From text/CSV
  - If the delimiter is not detected automatically, select: 'Comma' as the Delimiter
  - Press 'Load' to import the CSV into Excel

  The Excel-Sheet of the imported CSV then looks like the following:

Figure 9: CSV imported into MS Excel

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	x0	y0	x1	y1	x2	y2	x3	y3	x4	y4	x5	y5	x6	y6	x7	y7	x8	y8
2	-12039.2	2.656458	-11110.2	1.397021	-11750	1.089156	-12313.5	1.120561	-12842.4	0.027824	-13332	1.713213	-13437.8	2.719923	-13275.7	3.010645	-13105.7	2.731938
3	-12038.2	2.675615	-11109.9	1.365736	-11749.5	1.047432	-12313.1	1.094787	-12841.9	0.053972	-13331.4	1.686391	-13437.2	2.709579	-13275	3.004239	-13105	2.689278
4	-12037.2	2.646993	-11109.6	1.326239	-11749.1	1.03826	-12312.7	1.108018	-12841.6	0.086065	-13330.7	1.705208	-13436.3	2.725693	-13274.5	2.980034	-13104.3	2.70856
5	-12036.3	2.647128	-11109	1.315131	-11748.7	1.015443	-12312.3	1.087853	-12841	0.071434	-13330	1.713782	-13435.7	2.709278	-13273.7	2.986717	-13103.6	2.68057
6	-12035.2	2.657913	-11108.5	1.30649	-11748.4	0.988022	-12311.8	1.083165	-12840.6	0.079491	-13329.2	1.716523	-13434.8	2.719718	-13272.9	2.995159	-13103	2.69622
7	-12034.4	2.668229	-11108.1	1.2907	-11747.8	0.989688	-12311.4	1.070962	-12840.3	0.128571	-13328.6	1.71833	-13434.2	2.71482	-13272.3	2.995773	-13102.2	2.672737
8	-12033.3	2.645075	-11107.7	1.268001	-11747.3	0.994776	-12310.9	1.077594	-12839.7	0.126367	-13327.8	1.708361	-13433.4	2.698843	-13271.4	2.993399	-13101.4	2.664024
9	-12032.5	2.633664	-11107.1	1.258824	-11746.8	0.992765	-12310.4	1.054626	-12839.2	0.129007	-13327.3	1.696964	-13432.7	2.675089	-13270.7	2.994905	-13100.8	2.642366
10	-12031.4	2.635881	-11106.7	1.260244	-11746.3	0.967654	-12309.9	1.064565	-12838.9	0.156963	-13326.7	1.681374	-13432	2.69129	-13270.1	3.008039	-13100	2.624589
11	-12030.3	2.653334	-11106.2	1.273033	-11745.8	0.990044	-12309.3	1.06145	-12838.1	0.161965	-13326.1	1.668225	-13431.3	2.68346	-13269.4	3.013398	-13099.2	2.625136
12	-12029.4	2.644896	-11105.7	1.265836	-11745.3	0.985949	-12309.1	1.053064	-12837.8	0.181302	-13325.5	1.618168	-13430.5	2.687908	-13268.6	3.02944	-13098.7	2.59859
13	-12028.5	2.642066	-11105.2	1.259585	-11744.9	0.988441	-12308.5	1.087119	-12837.2	0.133945	-13324.9	1.588312	-13429.9	2.717581	-13267.9	3.018188	-13098	2.592194
14	-12027.4	2.642178	-11104.7	1.255766	-11744.3	0.95741	-12308	1.086924	-12836.6	0.164661	-13324.2	1.57064	-13429.1	2.693	-13267.1	3.002483	-13097.3	2.581293
15	-12026.5	2.674684	-11104.2	1.247803	-11743.9	0.989552	-12307.4	1.083499	-12836.3	0.166828	-13323.6	1.554416	-13428.4	2.682307	-13266.5	3.035955	-13096.3	2.545514
16	-12025.5	2.661456	-11103.7	1.23766	-11743.2	0.950985	-12306.9	1.08339	-12835.7	0.176567	-13323.1	1.526939	-13427.8	2.704533	-13265.7	3.055844	-13095.7	2.594356
17	-12024.7	2.659858	-11103.2	1.23426	-11742.8	0.953803	-12306.5	1.101576	-12835.1	0.1836	-13322.6	1.51224	-13427	2.693255	-13265	3.035715	-13095.1	2.579855
18	-12023.5	2.649959	-11102.5	1.242002	-11742.1	0.973719	-12305.9	1.103728	-12834.6	0.176657	-13321.9	1.520694	-13426.3	2.700809	-13264.2	3.069381	-13094.4	2.578927
19	-12022.6	2.656201	-11102	1.221156	-11741.7	0.938286	-12305.3	1.124769	-12834	0.145956	-13321.4	1.522476	-13425.7	2.673232	-13263.5	3.083186	-13093.7	2.564087
20	-12021.6	2.647967	-11101.5	1.243134	-11741.1	0.949794	-12304.8	1.137882	-12833.6	0.143195	-13320.8	1.540918	-13424.8	2.694827	-13262.8	3.084298	-13092.9	2.548749
21	-12020.7	2.650962	-11101	1.245417	-11740.5	0.933793	-12304.2	1.132796	-12833.1	0.157053	-13320.1	1.525906	-13424.1	2.696831	-13262.1	3.096669	-13092.1	2.559898
22	-12019.8	2.641442	-11100.4	1.23739	-11740	0.931726	-12303.7	1.149105	-12832.3	0.138917	-13319.6	1.505703	-13423.3	2.703318	-13261.4	3.094451	-13091.6	2.572951
23	-12018.7	2.634653	-11099.9	1.250491	-11739.4	0.930485	-12303.1	1.17225	-12831.9	0.135838	-13319.1	1.499042	-13422.6	2.706314	-13260.7	3.100412	-13090.7	2.546488
24	-12017.7	2.631788	-11099.2	1.270263	-11738.8	0.937148	-12302.6	1.173921	-12831.4	0.142511	-13318.6	1.505643	-13421.9	2.70671	-13259.9	3.093981	-13090	2.540904
25	-12016.7	2.636809	-11098.6	1.272788	-11738.3	0.935005	-12302	1.184081	-12830.7	0.135718	-13318.1	1.489568	-13421.3	2.703333	-13259.3	3.091786	-13089.2	2.544496
26	-12015.9	2.636076	-11098.1	1.260363	-11737.9	0.907555	-12301.4	1.19658	-12830.1	0.144368	-13317.3	1.48598	-13420.5	2.742145	-13258.5	3.12698	-13088.5	2.54535
27	-12014.7	2.631893	-11097.6	1.280327	-11737.3	0.921542	-12300.9	1.186052	-12829.6	0.142766	-13316.8	1.478536	-13419.9	2.730635	-13258.1	3.136788	-13087.9	2.535094
28	-12013.9	2.638142	-11096.9	1.277832	-11736.7	0.914405	-12300.2	1.155569	-12829.1	0.136033	-13316.3	1.486866	-13419	2.749445	-13257	3.115298	-13087.2	2.536709
29	-12012.8	2.625337	-11096.1	1.299331	-11736.1	0.931516	-12299.6	1.195471	-12828.5	0.130425	-13315.7	1.471737	-13418.2	2.733226	-13256.5	3.14907	-13086.3	2.518908
30	-12011.9	2.620151	-11095.7	1.288998	-11735.5	0.91055	-12299	1.182038	-12827.8	0.113682	-13315.1	1.443208	-13417.6	2.722848	-13255.7	3.139784	-13085.7	2.506217
31	-12010.9	2.626985	-11095.2	1.298898	-11734.8	0.918984	-12298.5	1.15683	-12827.3	0.096996	-13314.5	1.429375	-13416.8	2.721634	-13254.9	3.128807	-13084.9	2.525896
32	-12009.8	2.609806	-11094.6	1.271251	-11734.3	0.923324	-12297.9	1.174969	-12826.7	0.110088	-13314	1.443477	-13416.2	2.752294	-13254.3	3.153469	-13084.3	2.545424
33	-12008.8	2.609718	-11094	1.26962	-11733.8	0.91514	-12297.2	1.150989	-12826.2	0.123587	-13313.4	1.428444	-13415.5	2.760198	-13253.6	3.170291	-13083.5	2.551834
34	-12007.9	2.609358	-11093.4	1.292323	-11733.1	0.92	-12296.6	1.123241	-12825.5	0.102165	-13313	1.420309	-13414.7	2.756544	-13252.8	3.174769	-13082.7	2.533054
35	-12006.9	2.610512	-11092.8	1.306101	-11732.4	0.916241	-12296.2	1.128288	-12825	0.099994	-13312.3	1.425416	-13414	2.75274	-13252.2	3.201741	-13082	2.520905
36	-12005.9	2.613297	-11092.3	1.329708	-11732	0.89409	-12295.4	1.081412	-12824.4	0.085014	-13311.8	1.402784	-13413.3	2.760647	-13251.3	3.203867	-13081.3	2.532755
37	-12005	2.620455	-11091.5	1.328999	-11731.3	0.903137	-12294.8	1.079376	-12823.8	0.094842	-13311.1	1.376953	-13412.6	2.743811	-13250.7	3.212551	-13080.5	2.55212
38	-12003.9	2.612283	-11091	1.336667	-11730.7	0.886243	-12294.3	1.074254	-12823.2	0.102869	-13310.5	1.384561	-13411.8	2.758147	-13249.8	3.215564	-13079.8	2.568129
39	-12002.9	2.637918	-11090.2	1.360007	-11730	0.882919	-12293.7	1.071232	-12822.5	0.076516	-13310	1.36843	-13411.1	2.764062	-13249.1	3.217481	-13079.2	2.542849
40	-12002.1	2.626802	-11089.7	1.340681	-11729.5	0.895199	-12293	1.073296	-12821.9	0.060996	-13309.6	1.380068	-13410.3	2.762925	-13248.5	3.210254	-13078.3	2.543261

8 Testing

Since Python is the main language known by scientists, we have decided to add a guide on how we implemented tests for an Angular app using TypeScript, so that our app can be expanded more easily.

8.1 Using Karma with TypeScript

The testing suite is built using Karma and Jasmine, the testing frameworks that come with Angular. Each Service and Component has its own unique testing file. It is in the form of:
Name.*Type*.spec.ts
For example, a component named "Component1" would have its testing file names as:
Component1.component.spec.ts

Command to run the testing suite:

```
$ ng test
```

8.2 Component

When testing a Component this is the format needed, inside the .spec.ts file:

```
//Test for Component1

import { Component1 } from './Component1.component';
import {ComponentFixture, TestBed} from '@angular/core/testing';
import {HttpClientModule} from '@angular/common/http';
//... add all other imports needed

describe('Component1', () => {
  beforeEach(async() => {
    TestBed.configureTestingModule({
```

```
      imports:[
        HttpClientModule
      ],
      providers:[
      Component1 //Include all services and components needed for testing
      ]
    }).compileComponents();
  });

  it('should create', () => {
    const fixture = TestBed.createComponent( Component1 );
    const component = fixture.componentInstance;
    fixture.detectChanges();
    expect(component).toBeTruthy();//test the function needed in this line
  });
});
```

8.3 Service

The structure to test a .service.ts file is following.

```
//Test for Service1

import { Service1 } from './Service1.service';
import {ComponentFixture, TestBed} from '@angular/core/testing';
import {HttpClientModule} from '@angular/common/http';
//... add all other imports needed

describe('Service1', () => {

  let service: Service1;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports:[
        HttpClientModule
      ],
      providers:[
      Service1 //Include all services and components needed for testing
      ]
    }).compileComponents();
  });

  it('should be created', () => {
    const service: Component1 = TestBed.get(Component1);
    expect(service).toBeTruthy(); //test the function needed in this line
  });
});
```

8.4 Promises

The structure to test promises is following.

```
// Test for Service1 in a method returning a promise

describe('Service1', () => {
```

```
  let service: Service1;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports:[
        HttpClientModule
      ],
      providers:[
      Service1 //Include all services and components needed for testing
     ]
    }).compileComponents();
  });

  it('Test', (done) => {
      const service: ProcessorService = TestBed.get(ProcessorService);
      let promise = service.getScript(testProcess);

      if (promise instanceof Promise<String>){

        promise.then((result) =>{
          expect(result).toEqual(testScript);
          done();
        })
      }
  });
});
```