DOI: 10.2478/fas-2021-0003
FATIGUE OF AIRCRAFT STRUCTURES
Volume 2021: Issue 13, pp. 17-30

LOAD SPECTRUM ANALYSIS WITH OPEN SOURCE SOFTWARE – AN APPLICATION EXAMPLE

Marek S. Łukasiewicz 0000-0002-3034-1709

Warsaw University of Technology, Faculty of Power and Aeronautical Engineering
Nowowiejska 24, 00-665 Warsaw, Poland

marek@lukasiewicz.tech

Abstract

Processing of digital experimental data has become a key part of virtually every research project. As sensors get both more diverse and cheaper, the amount of information to be handled greatly increases as well. Especially fatigue failure modelling requires by its nature large numbers of samples to be processed, and visualised. The presented paper is based on analysis of load data gathered in flight on an unmanned aircraft. A few versions of an analysis program were developed and considered for the use case. Each implementation included ingesting the data files, creating transfer arrays and the "rain flow counting" algorithm. For the sake of the ease of use and functionality, the version based on Python programming language was selected for presentation. Short development iteration time of this approach allowed gaining new insights by tweaking parameters to better represent actual acquired data. Both the results and the software itself can be easily viewed in a web browser and run with modifications without the need to install any software locally. The developed software is meant as a demonstration of capabilities of open source computation tools dedicated to aerospace and mechanical engineering research, where they remain relatively unpopular.

Keywords: load spectrum, rainflow counting, data visualisation
Article Category: Research Article

INTRODUCTION

During the Fatigue and Aircraft Diagnostic Systems course, students are taught several calculation methods suited for estimation and extrapolation of fatigue cycles acting upon a structure. The main goal of the exercises done during the course is to get students acquainted with basic procedures and phenomena of aircraft fatigue. The examples given are typically simple enough to allow manual solutions with a pen and paper approach. Because of repeating the same actions many times, it seems that automating the task with software should be the preferred method of solving these problems.

This paper provides an example of how software tools used in the rapidly growing fields of data science and machine learning can be utilised to create visually appealing interactive solutions that do not require extensive programming experience to implement. Additionally, over the last years, there has been a sudden increase in remote teaching at all levels, including higher education. This has in turn produced a demand for teaching aids that can be easily shared online. It is hoped that this paper can be a useful reference for implementing a similar tool for classroom use and/or research.

METHODS AND TOOLS

All presented algorithms operate on one dimensional data. The real flight data consists of a time series of load factor observed on "PW-ZOOM" fixed wing unmanned aerial vehicle during a photogrammetry flight [1]. Take-off and a number of test manoeuvres were performed with manual remote control, after which the aircraft was switched to automatic operation. At the moment of engaging the on-board flight control unit, a sudden change in input to control surfaces caused a momentary increase in structural load, which was twice larger than any other value measured. In the Figure 1 this moment of flight was marked with red circles in the map and load factor plot.

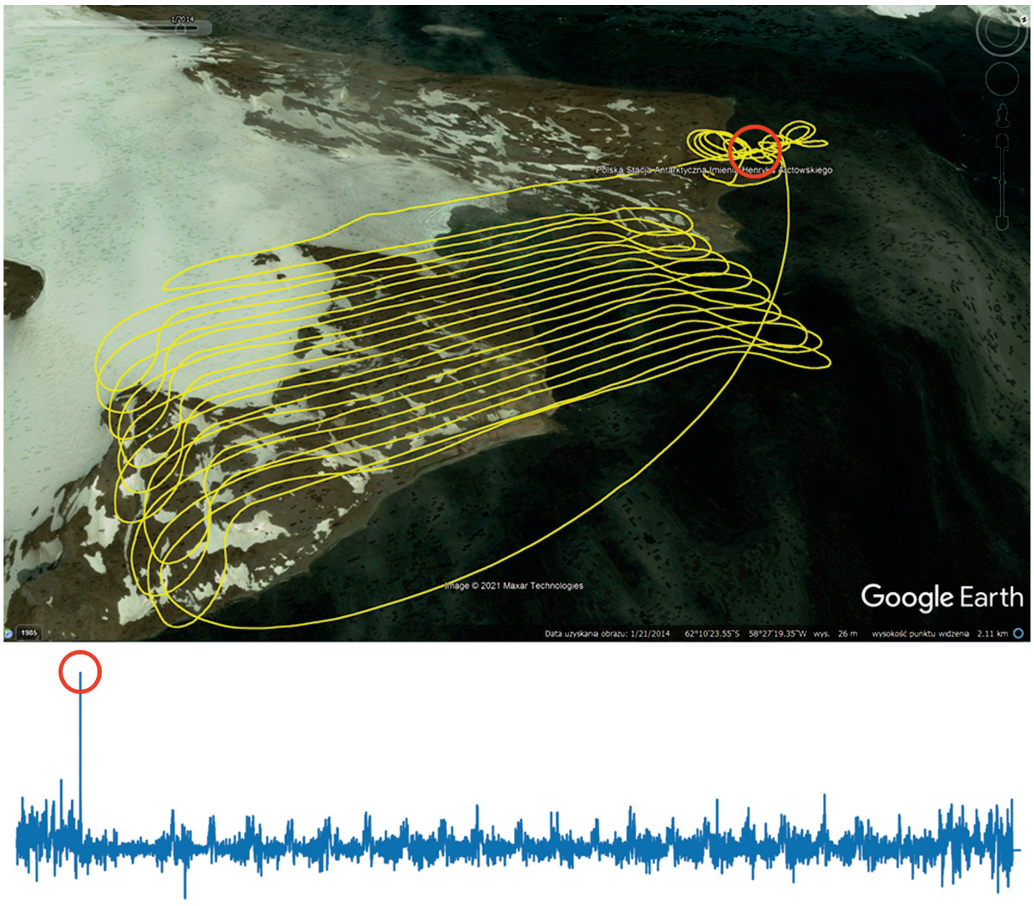

Polska Stacja Antarktyczna imienia Henryka Arctowskiego
Image © 2021 Maxar Technologies
Google Earth

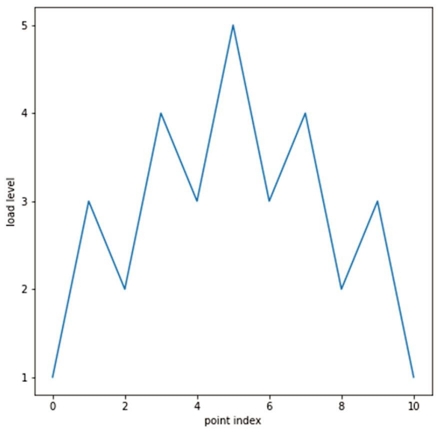
Figure 1. Trajectory of the analysed flight of PW-ZOOM UAV and the resulting load factor plot.

The method used follows the established algorithm [2] of quantization of a continuous signal to permit easier processing in further steps. As a result of that, an integer load level number is assigned to every load factor value. The time data is discarded, since only the number of cycles is considered in fatigue estimation, not their individual temporal characteristics. The sequence of integers obtained this way will typically have many repeated values representing a constant load which does not contribute to material fatigue. In the next filtering step, all consecutive repetitions of a load level are removed along with intermediate numbers between local extrema. In the resulting sequence, there are only alternating positive and negative changes in load level.

Every interval between two values in the load level sequence corresponds to a decrease or increase in the measured load. Each instance of a specific load level changing to another one is then counted and aggregated in a Transfer Array. This time-independent representation allows researchers to reason about and prognose loads much more easily than when using individual measurements.

Rain flow counting

For some load signals sequences, their direct transfer array representation as described above may not model accurately the character of increments caused by the acting loads. Suppose a signal is processed which is a sum of a slow oscillation with a large amplitude, and a faster oscillation with a smaller amplitude, but still spanning multiple load levels. Such a set of movement components is typical of many real mechanical systems. In the resulting transfer array, only many small load-transfers will appear, corresponding to the more frequent changes. A simple example of such a case is presented in Figure 2.



	1	2	3	4	5
5	0	0	1	0	X
4	0	1	1	X	0
3	1	1	X	1	1
2	0	X	1	1	0
1	X	0	1	0	0

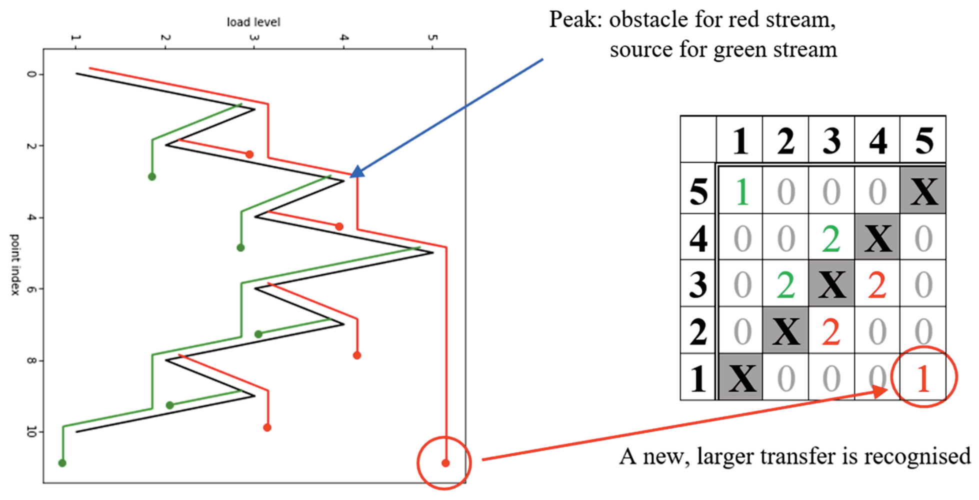
Figure 2. Example load history with corresponding transfer array.

The original, large amplitude component can be retrieved using the rain flow counting algorithm proposed by [3]. In this method, each transfer will be represented by a “rain stream” flowing along from earlier to later data points. Local maxima are “peaks”, while local minima are “valleys”. Naturally, a bigger maximum value corresponds to a “taller peak”, and a minimum of a smaller value to a “deeper valley”.

With this nomenclature defined, the specific set of rules used in this paper is as follows:

- Every valley creates a stream that flows until it ends, before the next stream is simulated.
- The stream flows along the graph of the local extremes chain until the closest peak and then "falls":
    - If the end of the data is reached, the stream is terminated;
    - If the next valley is deeper than the one from which the stream started, the stream is terminated;
    - If the next peak is not taller than the one it is falling from, it is ignored;
    - If the next peak is taller than the one it is falling from, the graph is intercepted at the value of the peak the stream falls from, and then flows along this line;
- If the stream collides with a trace of a previous stream, it is terminated.
- The whole algorithm is repeated for streams originating from each peak, flowing towards valleys.

As seen on the following plot, this relatively simple set of rules prioritises larger changes in the value, even if they are not monotonic. The total number of transfers is preserved. To convey the idea of rain streams flowing, the plot is oriented so that the streams "fall" towards the bottom of page, as shown in Figure 3. The first set, starting from valleys is drawn in red, and the set starting from peaks in green. The corresponding entries in the half-cycle array, use the same colour. To save space, all other plots throughout this paper are presented in the typical orientation, with the sample index on the horizontal axis.


load level
1
2
3
4
5
point index
0
2
4
6
8
10
Peak: obstacle for red stream, source for green stream
A new, larger transfer is recognised

	1	2	3	4	5
5	1	0	0	0	X
4	0	0	2	X	0
3	0	2	X	2	0
2	0	X	2	0	0
1	X	0	0	0	1

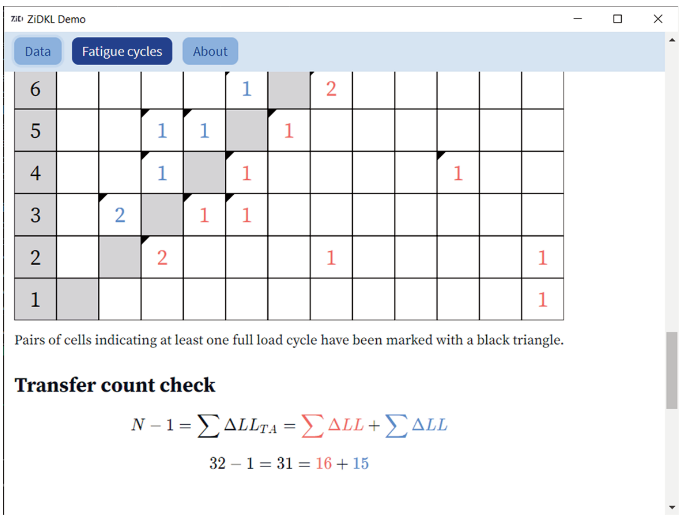
Figure 3. Example of rain flow counting result.

Comparison of software solutions

Currently many tools are available to facilitate scientific computation and processing of data. The following sections aim to compare the tools considered for this application, together with their perceived advantages and disadvantages. It should be noted that this evaluation is based solely on the subjective opinion. However, this opinion was formed only after using each of the mentioned tools for processing and visualisation of experimental results.

There are commercial software packages dedicated to fatigue analysis. However, they enforce specific workflow and analysis methods. Since they are made for professional users, it is harder to use them for teaching purposes. Because of this use case, a custom solution would be preferred.

The closest match for the commercial packages would be a solution developed with a basic window interface library in a compiled language, e.g. C++ or C#. The main issue with this approach is that such applications typically require months of work and extensive software engineering experience. Following a popular [4] trend in software development, using web technologies (i.e. JavaScript, HTML, CSS) is a viable choice for any application with a graphical user interface. This approach allows most extensive customization of appearance. It requires however previous acquaintance with a large number of libraries and solutions.



6					1		2					
5			1	1		1						
4			1		1					1		
3		2		1	1							
2			2				1					1
1												1

Pairs of cells indicating at least one full load cycle have been marked with a black triangle.

Transfer count check

$$N-1=\sum \Delta LL_{TA}=\sum \Delta LL+\sum \Delta LL$$

$$32-1=31=16+15$$

Figure 4. Screenshot of the browser-based version of the analysis program.

For mechanical engineering researchers, MATLAB remains the preferred choice with wide adoption in academia, and many years of experience for most tutors. Using the educational license, it is easy to forget that this software has a price that is prohibitively high for many companies, leaving some graduates without the tools they were trained to use.

Recently, using Python programming language organised into Jupyter [5] notebooks becoming increasingly popular choice for presenting machine learning and data science papers. Because the components used for creating the visualisation are open source software, the software can be easily run by anyone interested. Moreover, it can be integrated with third-party services, (e.g. like [6]), allowing a variety of sharing, collaboration and workflow options.

IMPLEMENTATION

Given the qualities described in the preceding section, the Jupyter-Python environment was selected for solution development. Apart from the standard library, two additional packages were utilised. The numpy package allows for performant computations on n-dimensional matrices of numbers [7]. The matplotlib package provides plotting capabilities [8] with much customizability and good integration with both Jupyter and numpy. These libraries are often used in notebooks, and as such are preinstalled in many Jupyter environments.

The program was prepared in two versions. The *simple.ipynb* notebook makes use of fewer features and is extensively annotated to serve as an example of an amount of code needed for the classroom presentation. Conversely, the *advanced.ipynb* is intended for processing of actual flight data. The biggest differences are formatting, handling larger sets of data, and file input/output. For instructions how to access and run both versions, see the appendix.

Original data received from an external program parsing flight logs was in an unusual format, with some information missing. To decouple this specific case from the rest of the application, a very simple Python script was used to convert the data file into a typical *.csv* format.

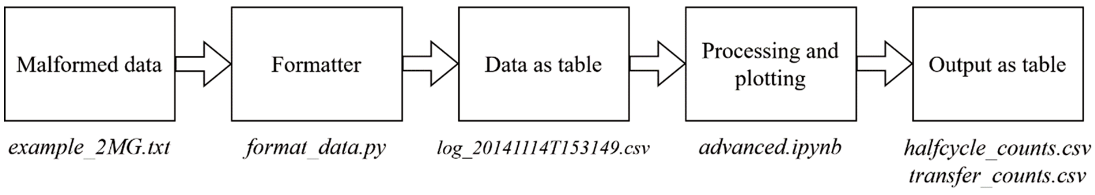


Figure 5. Data flow block diagram.

The development was accomplished using Visual Studio Code, which is an open source text editor with Jupyter notebook integration among other features. Backups and versioning were accomplished using Git server hosted by Github.

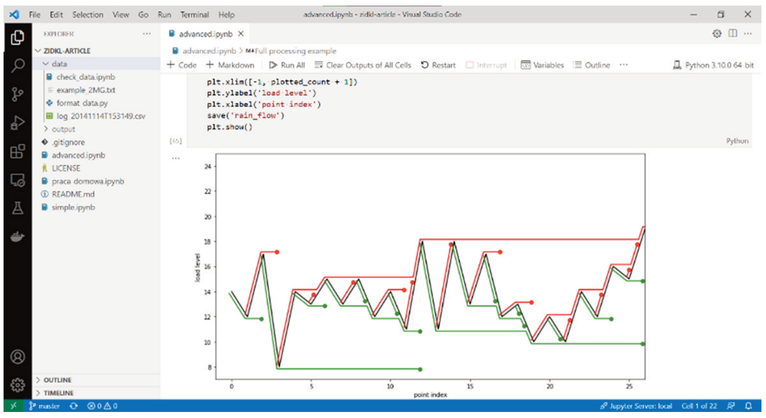

```
plt.xlim([-1, plotted_count + 1])
plt.ylabel('load level')
plt.xlabel('point index')
save('rain_flow')
plt.show()
```

Figure 6. Screenshot of local development environment.

RESULTS

As described above, the original data was ingested into the software from a text file. The first step was analysis of the received sequence. For the analysed flight, almost all of the load factor values measured were in a very narrow region of flight envelope, as can be seen in histogram presented in Figure 7. The outliers on both ends of the range are shown in Table 1.

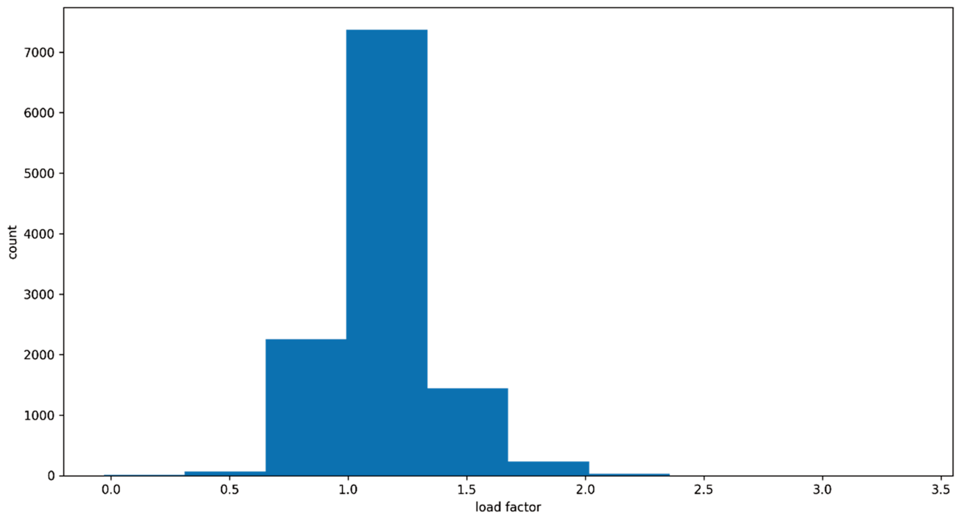


Figure 7. Histogram of load factor values in the analysed data.

Table 1. Extreme values for the analysed data.

min	Sorted load factor values												max
-0.03	0.1	0.21	0.23	0.24	0.3	0.33	…	2.23	2.3	2.37	2.58	2.63	3.38	5.03

Because of that distribution, it was decided to use a custom assignment of load levels to load factor, instead of the typical assignment based on maximal certified range of loads. For this aircraft, this would be LL=31 for n_z=6 and LL=3 for n_z=-3. This change leads to a more diverse sequence of load levels, and helps to better illustrate capabilities of the tool. The mapping can be easily adjusted by modifying member variables of the *Extents* class. In case any value falls outside the given boundaries, which may be true for some outliers, the assigned load level is either the minimal LL=1 or maximal LL=32, whichever is closer.

A simple iterative algorithm was used to remove the duplicates and samples that were not local maxima in the sequence. This step reduced the length of the sequence from 11419 points to 4995. On the following plots, the first 25 points are shown. As expected, the filtered sequence consists of alternating increases and decreases in load level. Figure 8 shows the samples after quantizing the signal to load levels, and the resulting sequence of local extremes.

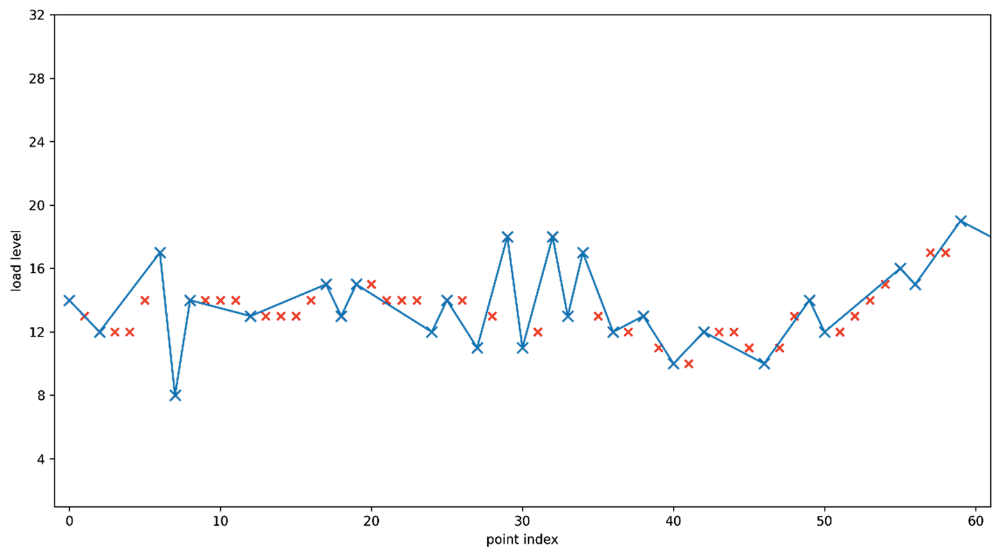


Figure 8. Filtering load level sequence for local extrema, illustrating selected and rejected samples.

Subsequently, all differences between consecutive values were summed to produce the transfer array. The results correspond to the received data, with the highest counts of repetitions close to the identity diagonal. Comparing the typical level assignment presented in Table 2 with the one based on data distribution in Table 3, one can see that the standard transfer array is very sparse for the same sequence.

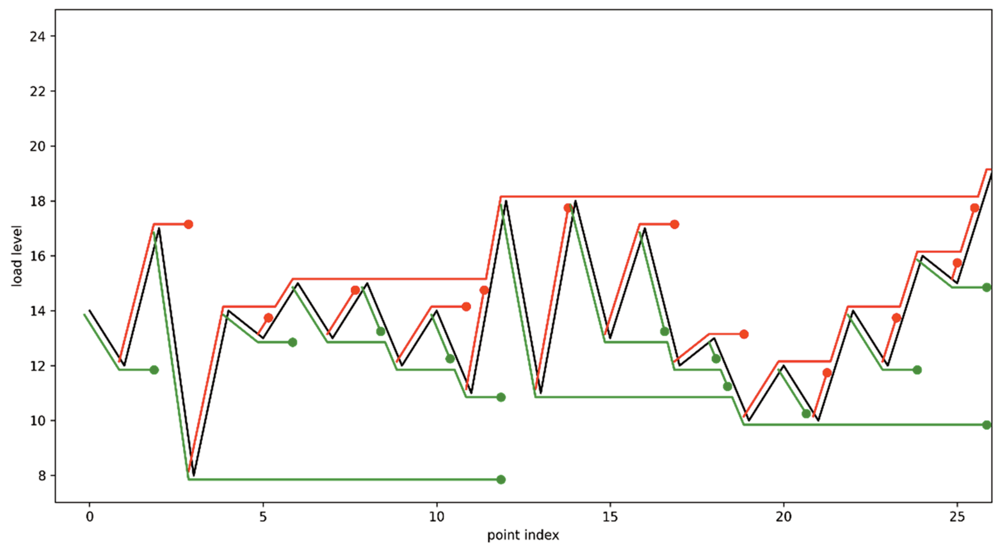


Figure 9. Visualisation of rain flow counting for the first 25 samples in filtered data.

Table 2. Transfer array for standard level assignment, 2868 load transfers in total.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	10	4	2	X	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	1	0	0	0	0	0	0	1	0	2	27	220	122	X	2	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	5	11	110	776	X	122	3	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	2	24	102	X	787	211	10	1	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	X	107	97	36	1	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	11	19	13	2	2	1	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	1	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3. Transfer array for level assignment based on observed values,
4994 load transfers in total.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0
28	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	2	0	0	1	1	0	0	0	0	0	0	0	1	0	0	X	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	2	0	1	0	X	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	4	2	1	X	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	1	0	0	0	0	0	2	3	1	7	2	3	4	1	1	X	2	1	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	1	0	0	0	0	2	1	10	3	4	5	8	6	1	X	2	0	1	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	1	0	3	4	0	3	7	7	13	6	9	8	X	4	2	1	1	0	0	0	0	0	0	0	0	0	0
17	0	1	1	0	0	0	0	3	6	6	12	12	12	20	17	20	X	8	8	2	2	0	0	1	0	0	0	0	0	0	0	0
16	0	0	0	0	1	3	0	7	6	8	20	30	23	19	23	X	18	11	6	2	4	1	1	0	0	0	0	0	0	0	0	0
15	1	0	0	1	0	1	4	8	14	25	46	43	31	33	X	18	12	10	7	5	1	1	0	1	0	0	0	0	0	0	0	0
14	0	0	1	0	0	2	4	9	29	51	102	98	56	X	31	23	23	7	4	6	4	0	1	0	0	0	0	0	0	0	0	0
13	0	0	0	0	1	3	8	20	59	139	229	163	X	50	30	20	20	8	2	2	2	1	0	0	0	0	0	0	0	0	0	0
12	0	0	0	1	0	4	8	30	73	188	379	X	176	97	48	25	11	3	5	2	1	0	2	0	0	0	0	0	0	0	0	0
11	0	0	0	1	1	2	8	22	45	89	X	363	249	89	46	24	11	9	3	1	0	0	1	0	0	0	0	1	0	0	0	0
10	0	0	0	0	3	2	3	10	28	X	78	202	117	71	21	7	6	3	1	1	2	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	6	X	25	52	77	53	26	15	10	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	X	5	15	27	28	16	10	7	9	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	X	1	0	3	5	8	7	6	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	X	0	0	0	2	5	4	2	1	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	X	0	0	0	1	0	1	1	0	0	4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	X	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	X	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	X	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The rain flow counting algorithm was applied to both sets of data. The actual implementation contains some additional logic to allow the stream trajectories to be displayed along the sequence plot. However, the key information, i.e. the starting and ending load level is as defined. In Table 4 and Table 5, the transfers that do not correspond to a full load cycle are marked with square brackets.

Table 4. Half-cycle transfer array from rain flow counting for standard assignment of levels, 2868 load transfers in total.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	[1]	0	0	0	0	0	0	1	2	9	3	0	0	2	X	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	3	8	50	186	126	X	2	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	70	822	X	125	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	118	X	823	186	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	X	118	69	51	3	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	11	10	10	8	9	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	3	3	1	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	[1]	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5. Half-cycle transfer array from rain flow counting for level assignment based on observed data, 4994 load transfers in total.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
32	[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0
28	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0
24	0	[1]	0	[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0
23	0	0	0	1	1	0	0	[1]	2	0	0	0	0	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0	0	0
22	0	0	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	X	0	0	0	0	0	0	0	0	0	0
21	0	0	1	0	1	3	2	1	1	0	1	1	0	1	1	0	2	1	1	2	X	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	1	1	1	1	3	1	0	3	1	4	0	2	2	3	2	X	2	1	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	3	0	4	4	1	2	4	2	2	3	3	3	6	4	X	2	1	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	2	4	3	3	5	4	5	2	6	6	10	11	X	4	3	1	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	2	0	10	3	5	11	10	9	16	16	28	X	11	6	2	2	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	6	5	20	9	3	10	20	15	23	29	X	28	10	3	2	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	3	9	23	29	20	28	21	28	46	X	29	16	6	3	0	1	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	4	20	55	73	49	73	78	X	46	23	16	6	3	4	1	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	1	0	6	42	141	198	234	X	78	28	15	9	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	2	6	42	141	492	X	234	72	21	20	11	5	2	3	1	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	2	12	36	118	X	492	198	49	28	10	11	4	4	0	1	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	1	7	38	X	118	141	141	73	20	3	5	5	2	1	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	6	X	38	37	42	42	55	29	8	3	3	1	3	1	0	2	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	X	6	7	12	6	6	20	23	21	9	3	4	1	1	0	0	[1]	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	X	1	0	1	2	2	0	4	9	5	0	4	4	1	2	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	X	1	0	0	0	0	0	1	0	3	6	2	2	0	1	3	1	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	1	1	1	0	0	0	0	1	0	0	0	0
4	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	[1]
3	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
2	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	[1]	0	0	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	[1]	0	0	0	0	0	0	0	0

The following plot uses a logarithmic colormap to show the difference that the rain flow counting makes for each cell of the transfer array. As stated in the introduction, the purpose of this algorithm is to extract the larger oscillation components from data. In the following graph, one can see that many transfers corresponding to moderate changes were changed to sets of smaller and larger changes. Both the transfer count and the maximal values were preserved unchanged. However, the extreme values are now reached from farther states, corresponding to "a rain stream dripping" from consecutive peaks. According to many models, including the classic Miner-Palmgren hypothesis [9], it is the cycles with larger amplitudes that contribute the most to fatigue failure of a component.

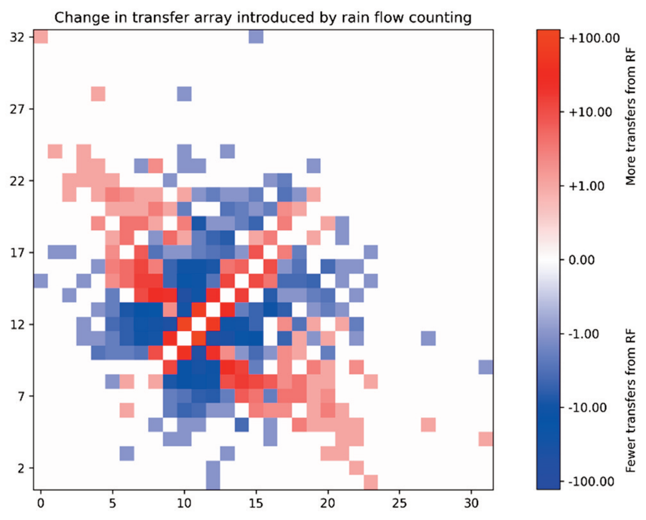


Figure 10. Logarithmic heatmap of change introduced by rain flow counting.

CONCLUSIONS

The selected approach of using Python notebooks has proven viable for fatigue load analysis. Having a transparent implementation available allowed researchers to customize load level assignment and obtain much better resolution of the results. Only the first steps of processing were added at this point, but the software will be extended with more visualisation and extrapolation options using the same functionalities of the libraries. By developing own solution with didactics in mind, intermediate steps and values could be highlighted to aid teaching the methods used.

Compared to other approaches to developing this application, notebooks were by far the easiest to use thanks to availability of matplotlib plotting library, customisability of Python scripting, and short iteration time provided by Jupyter. The second fastest in terms of development time version was the web application. An objective, but still crude comparison is based on the fact that the web version has approximately 609 thousand characters in its source code. Meanwhile, the notebook version has only 11.4 thousand characters (fewer than half the number this paper contains), while having more features implemented. Both counts only include the code written by the author, without libraries.

Acknowledgements

The author wishes to thank Assoc. Prof. Mirosław Rodzewicz, PhD. Eng. of Aircraft Design Division, Faculty of Power and Aeronautical Engineering for providing the flight data, with permitting its release with a public license and additionally offering helpful advice on writing this paper.

REFERENCES

[1] Korczak-Abshire, M., Zmarz, A., Rodzewicz, M. et al., (2019). Study of fauna population changes on Penguin Island and Turret Point Oasis (King George Island, Antarctica) using an unmanned aerial vehicle, *Polar Biol.*, vol. 42, no. 1, pp. 217–224, doi: 10.1007/s00300-018-2379-1.

[2] Kossira, H. and Reinke, W. (1986). “Entwicklung eines Belastungskollektivs fur Segel- und Motorflugzeuge,” IFL-IB, TU Braunschweig, no. 86-O5.

[3] Matsuishi, M. and Endo, T. (1968). Fatigue of metals subjected to varying stress, *Japan Soc. Mech. Eng. Fukuoka*, Japan, vol. 68, no. 2, pp. 37–40.

[4] Stack Exchange Inc, (2021). *Stack Overflow Developer Survey 2021*. Retrieved December 1, 2021, from https://insights.stackoverflow.com/survey/2021#technology.

[5] Kluyver, T., Ragan-Kelley, B., Pérez, F. et al., (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows, Loizides, Fernando and Scmidt, Birgit (eds.) in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, IOS Press, pp. 87–90, doi: 10.3233/978-1-61499-649-1-87.

[6] Mordvintsev, A., Randazzo, E., Niklasson, E. and Levin, M. (2020). Growing Neural Cellular Automata, *Distill*, vol. 5, no. 2, doi: 10.23915/distill.00023.

[7] Harris, C.R., Millman, K.J., van der Walt, S.J. et al., (2020). Array programming with NumPy, *Nature*, vol. 585, no. 7825, pp. 357–362, doi: 10.1038/s41586-020-2649-2.

[8] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment, *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 90–95, doi: 10.1109/MCSE.2007.55.

[9] Miner, M.A. (1945). Cumulative Damage in Fatigue, *J. Appl. Mech.*, vol. 12, no. 3, pp. A159–A164, doi: 10.1115/1.4009458.

APPENDIX

The reader is invited to inspect the code and experiment on their own with an interactive version, by visiting one of the following pages. Note that read-only display will be immediately available after webpage load, but a short wait will be needed before the server starts the environment for the interactive examples.

- Simple version, read-only:
  https://nbviewer.org/github/Maarrk/zidkl-article/blob/master/simple.ipynb
- Simple version, interactive:
  https://mybinder.org/v2/gh/Maarrk/zidkl-article/master?filepath=simple.ipynb
- Advanced version, read-only:
  https://nbviewer.org/github/Maarrk/zidkl-article/blob/master/advanced.ipynb
- Advanced version, interactive:
  https://mybinder.org/v2/gh/Maarrk/zidkl-article/master?filepath=advanced.ipynb
- Full repository contents: https://github.com/Maarrk/zidkl-article