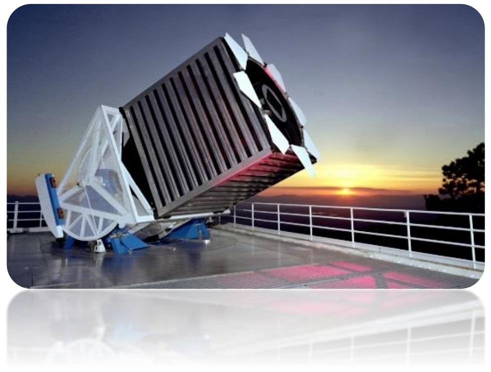
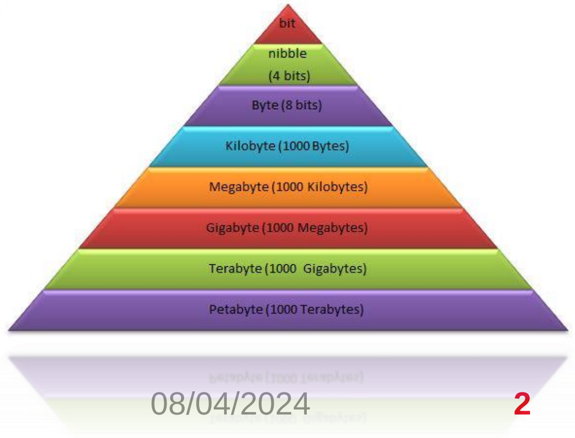
Scientific data management – HDF5

Clément Fisher

SHM & Artificial Intelligence Research Engineer

Introduction – Why ?

- **Data management for scientific research**
  - The amount of data created has exploded over time as storage and processing capacities have increased (Moore's Law).
    - CERN's Large Hadron Collider uses 150 million sensors generating around **25Pb/year** of filtered data.
    - The Sloan Digital Sky Survey generates **200Gb/night** of astronomical data.
    - The NASA Centre for Climate Simulation (NCCS) stores **32Pb** of climate observations and simulations.



bit
nibble (4 bits)
Byte (8 bits)
Kilobyte (1000 Bytes)
Megabyte (1000 Kilobytes)
Gigabyte (1000 Megabytes)
Terabyte (1000 Gigabytes)
Petabyte (1000 Terabytes)

Introduction – Why ?

- **Data management for scientific research**

  - **Several researchers** within a team can share the same resources (experiment or simulation), even if they were not involved in creating them.

  - Data acquired now may be useful **several years from now**

  - ➢ It is important to have clear databases containing all the information needed to use them (method of creation, parameters ....).

  - ➢ In 2016, a consortium of scientists and organizations defined a set of best practices for scientific data management, known as **FAIR**[1].

[1] M. D. Wilkinson *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Sci Data*, vol. 3, p. 160018, Mar. 2016, doi: 10.1038/sdata.2016.18.

Introduction – FAIR Principle[1]

Findable

- Metadata and data should be easy to find for both humans and computers.

Accessible

- The data are archived in long-term storage and can be made available using standard technical procedures.

Interoperable

- The data can be exchanged and used across different applications and systems — also in the future, for example, by using open file formats.

Reusable

- The data are well documented and curated and provide rich information about the context of data creation.

[1] https://www.go-fair.org/fair-principles/

Introduction – Use Case

- **Structural Health Monitoring** (SHM):
  - “The process of acquiring and analyzing data from **on-board sensors** to evaluate the health of a structure”[1]

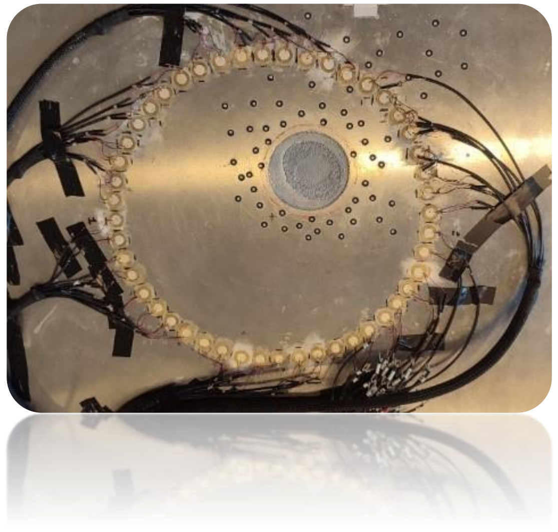
- **Guided wave-based SHM** detects anomalies through changes in the propagation of guided ultrasonic waves through structures.

- Due to the complexity of the analysis, there is a great deal of research into the use of **Artificial Intelligence**.

[1] M. D. Wilkinson *et al.*, “The FAIR Guiding Principles for scientific data management and stewardship,” *Sci Data*, vol. 3, p. 160018, Mar. 2016, doi: 10.1038/sdata.2016.18.

Use case – Guided Waves database

- Experimental campaign
  - To test the robustness of guided wave tomography as a function of temperature, an experimental study was carried out in the laboratory [1].
  - The complete acquisition procedure is repeated 4 times, for the pristine plate and then with 3 successive stages of corrosion ($\mathcal{D}$).
  - Measurements are taken at 8 different frequencies ($\mathcal{F}$, from 30kHz to 200kHz) and 8 different temperatures ($\mathcal{T}$, from -20°C to 50°C).
  - Finally, an acquisition consists of an FMC measurement of the 48 sensors ($\mathcal{S}_e, \mathcal{S}_r$), i.e. for all emitter-receiver pairs, for portions of 1ms ($\mathcal{t}$, 2000 points).



[1] T. Druet, A. Thomas, A. Recoquillay, Y. Gélébart, E. Martin, and B. Chapuis, "Guided Wave Tomography for Corrosion Monitoring under Varying Environmental Conditions," e-Journal of Nondestructive Testing, vol. 28, no. 7, Jul. 2023, doi: 10.58286/28283.

Use case – Guided Waves database

- There are several ways to save these acquisitions in Python (numpy), some of which are described below:

  - All acquisitions can be grouped together in a 6-dimensional matrix $[\mathcal{D}, \mathcal{F}, \mathcal{T}, \mathcal{S}_e, \mathcal{S}_r, \mathcal{t}]$ stored in a .npy file.

  - Since each state of the structure is often used separately, it is possible to separate the data into 4 matrices $[\mathcal{F}, \mathcal{T}, \mathcal{S}_e, \mathcal{S}_r, \mathcal{t}]$.
    - In a single .npz file to store values as in a dictionary.
    - In separate .npy files.

  - The same reasoning can be applied to frequency and temperature to manipulate the matrices $[\mathcal{S}_e, \mathcal{S}_r, \mathcal{t}]$, in this case the database is stored in 256 .npy files.

- In terms of disk space usage, all of the above options are similar, the database takes up around 8.8GB.

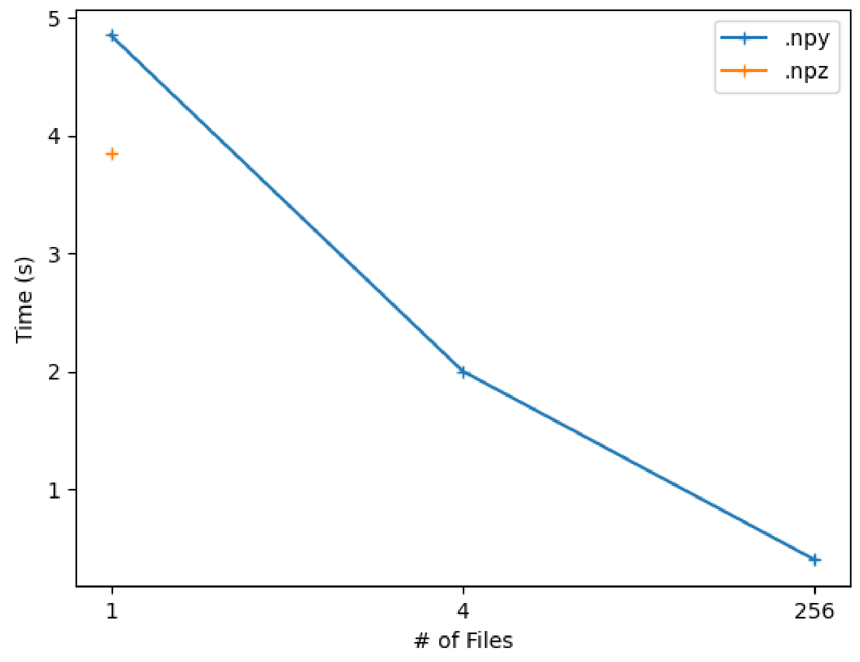
Use case – Guided Waves database

- For analysis purposes it is also important to consider the time taken **to read the data** (and sometimes the time taken to write it). To test the read time, a quick benchmark is performed, which consists of extracting the maximum value of a randomly selected signal (1000 iterations).

- The '**divide and conquer**' strategy has a big impact on reducing read times. As the file is read in its entirety to load the matrix, this was expected.

- However, this is limited by the smallest **portion of interest**. If more than one file is required for analysis, the time taken will increase.

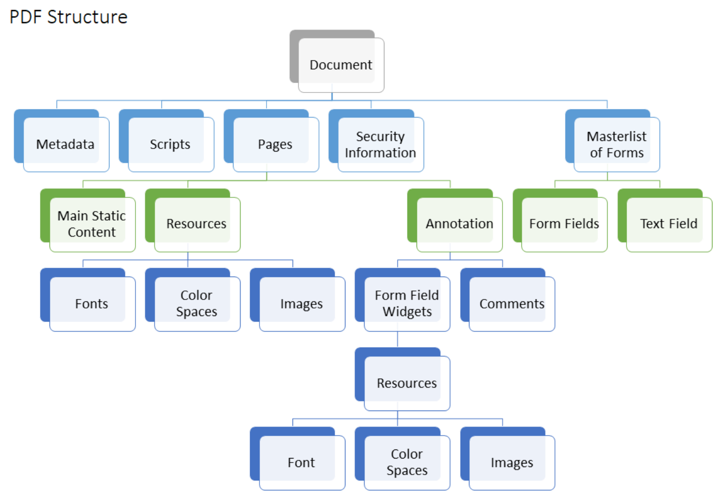
- What's more, in this case "only" 256 files are needed, but it's **inconceivable to have a larger number of subfiles**.

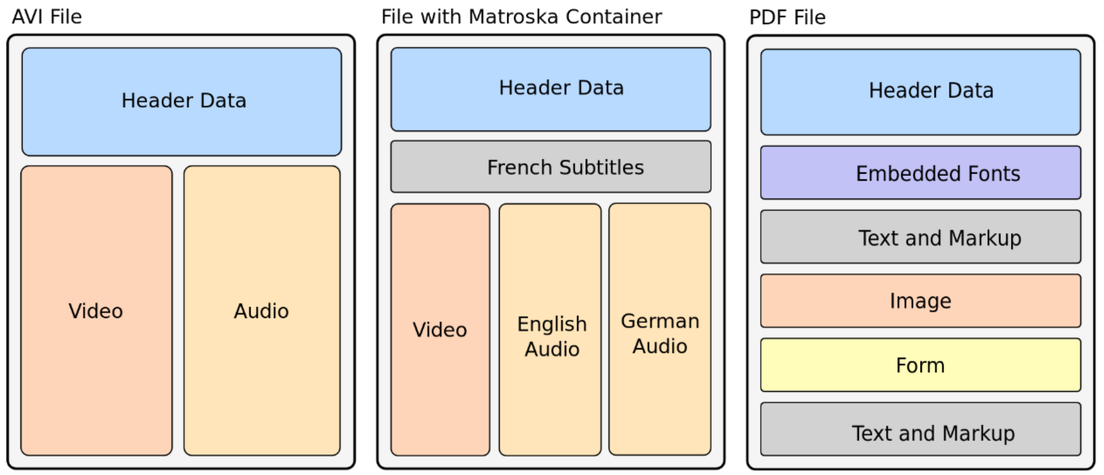


Container files

- In computing, the question of the number of files is a recurring one, and many formats have been created to solve this problem. All these formats are known as **container formats**, wrappers or metafiles.

- Many common file formats are container formats (WAV, PDF, SVG, AVI, ZIP….)





Examples of container formats - by Ernst Rohlicek jun.

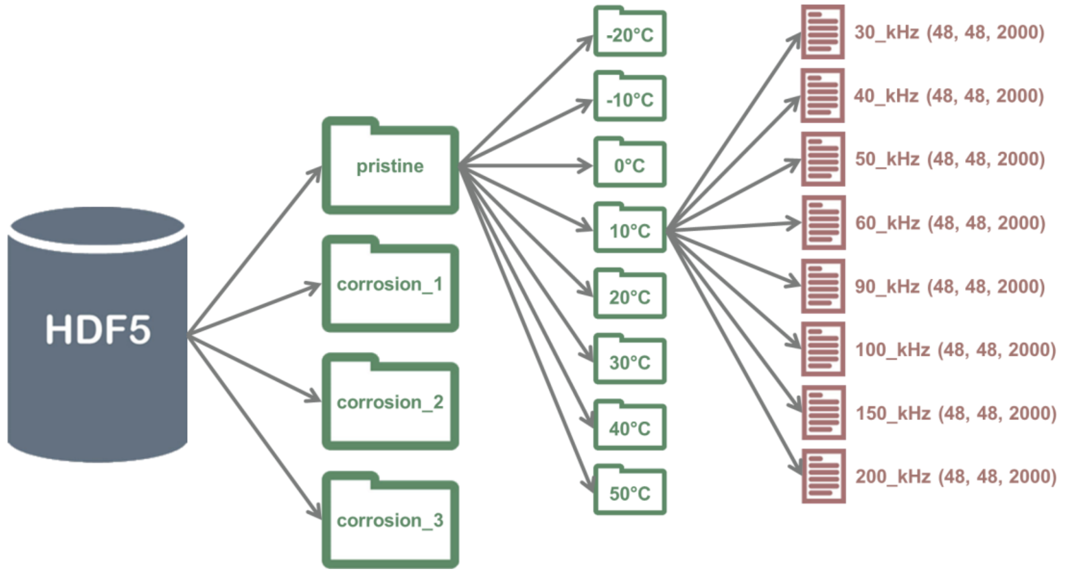
HDF/HDF5 Format

- Created in 1987 by the National Center for Supercomputing Applications (NCSA), the **Hierarchical Data Format** is a standard format in the scientific community.

- It is a "portable" format, **independent of platform, operating system and language** (C, C++, Java, Python, LabView, R, Julia…).

- This is a **container format**, in which linked **datasets** are grouped together in **group**. Each dataset can be considered as a different file.

- Each unit may also contain information called **metadata** for example means, prf, voltage, timestamp, defects position, sampling frequency….



HDF/HDF5 Format

- The guided wave database presented above can be stored in an hdf5 file. In our case, the smallest unit used is an acquisition at a given frequency, temperature and state.

- The metadatas linked to defects concern a large number of acquisitions, so this is the first group level. In this case, the temperature and frequency groups are interchangeable, so we chose temperature subgroups containing a $[\mathcal{S}_e, \mathcal{S}_r, t]$ dataset for each frequency.

- For the read test presented above, the hdf5 extracts the value in 0.02s per signal, which is **much faster** than the 256 .npy files (0.4s). Moreover, the file takes up 4.6GB of disk space, which is **2 times smaller** than the numpy files.

HDF/HDF5 Optimizations

- By default, datasets use 1D contiguous storage, i.e. all data is stored side by side on disk. Matrices are linearized using a row-major order (C-order).

- The HDF5 format allows the use of a different type of storage, called chunks, in which data is split into chunks that are stored separately in memory. Chunk is the reading unit, unlike the file in the default case or with .npy files.

- Chunks can use a format other than row-major, which can be useful depending on the algorithms applied later. In the example below, row-major chunk storage requires 4 chunks to be read, whereas column-major chunk storage allows only 1 chunk to be read.

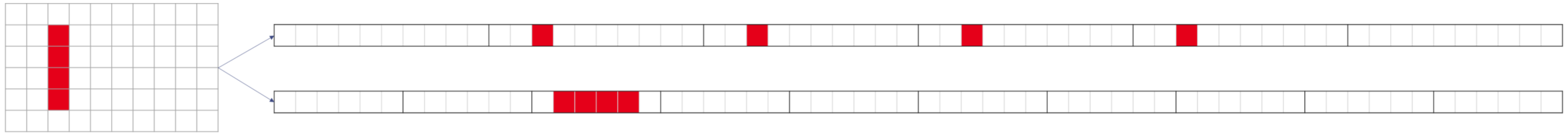


Figure 1: Row-major vs Column-major order

HDF/HDF5 Optimisations

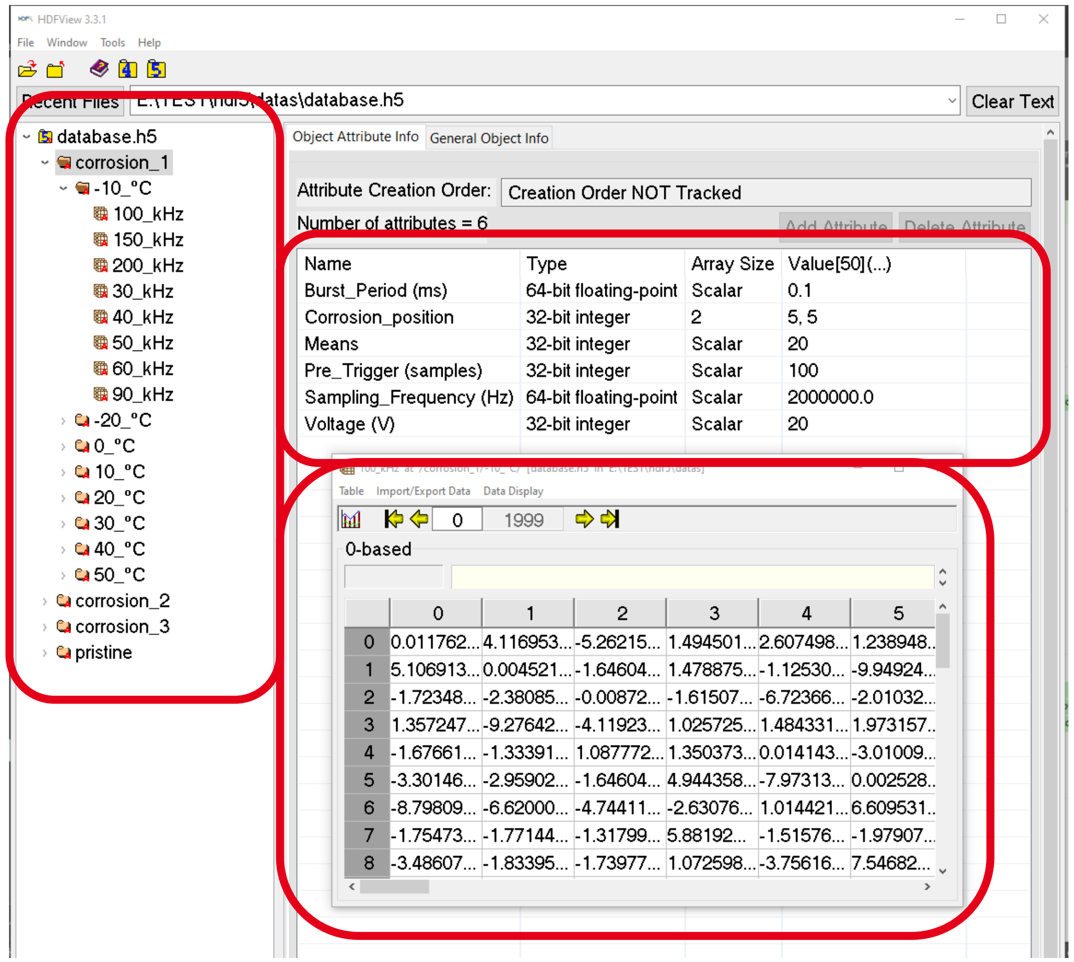
- It is also possible to **optimise file size**, using the hdf5 format to compress data. Compression does not change anything for the user in terms of read/write calls. On the other hand, compression can increase read/write time.

- You need to choose the type of compression (GZIP, LZF, SZIP) and the parameters, **making a compromise between compression rate and time**.

- For large files, it is also possible to **link multiple files** together.

HDF/HDF5 Viewer

- For rapid analysis, the HDF group provides viewing software HDF® View[1].
- The left panel shows the tree structure of groups and datasets.
- The right pane lists all the metadata of the selected object (group or dataset).
- Finally, the data can be displayed for rapid exploration.



```
database.h5
  corrosion_1
    -10_°C
      100_kHz
      150_kHz
      200_kHz
      30_kHz
      40_kHz
      50_kHz
      60_kHz
      90_kHz
    -20_°C
    0_°C
    10_°C
    20_°C
    30_°C
    40_°C
    50_°C
  corrosion_2
  corrosion_3
  pristine
```

Attribute Creation Order: Creation Order NOT Tracked

Number of attributes = 6

Name	Type	Array Size	Value[50](...)
Burst_Period (ms)	64-bit floating-point	Scalar	0.1
Corrosion_position	32-bit integer	2	5, 5
Means	32-bit integer	Scalar	20
Pre_Trigger (samples)	32-bit integer	Scalar	100
Sampling_Frequency (Hz)	64-bit floating-point	Scalar	2000000.0
Voltage (V)	32-bit integer	Scalar	20

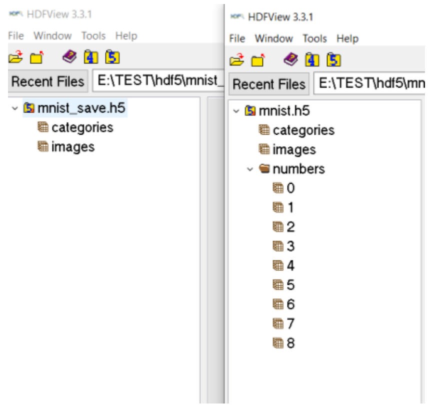
	0	1	2	3	4	5
0	0.011762...	4.116953...	-5.26215...	1.494501...	2.607498...	1.238948..
1	5.106913...	0.004521...	-1.64604...	1.478875...	-1.12530...	-9.94924...
2	-1.72348...	-2.38085...	-0.00872...	-1.61507...	-6.72366...	-2.01032...
3	1.357247...	-9.27642...	-4.11923...	1.025725...	1.484331...	1.973157..
4	-1.67661...	-1.33391...	1.087772...	1.350373...	0.014143...	-3.01009...
5	-3.30146...	-2.95902...	-1.64604...	4.944358...	-7.97313...	0.002528..
6	-8.79809...	-6.62000...	-4.74411...	-2.63076...	1.014421...	6.609531..
7	-1.75473...	-1.77144...	-1.31799...	5.88192...	-1.51576...	-1.97907...
8	-3.48607...	-1.83395...	-1.73977...	1.072598...	-3.75616...	7.54682...

[1] https://www.hdfgroup.org/downloads/hdfview/

HDF/HDF5 TP

- The provided file contains data from the MNIST database with two groups (*images* and *categories*). The aim is to create a group containing one dataset per number.

- The maximum argument of each row of *categories* dataset correspond to the number value.



```
mnist_save.h5
    categories
    images
```

```
mnist.h5
    categories
    images
    numbers
        0
        1
        2
        3
        4
        5
        6
        7
        8
```

HDF/HDF5 TP

```
import h5py
import numpy as np

# Open the file in append mode


    # Load Images and Categories



    # Extract number for each images


    # Create group Numbers


    # Loop on each number

        # Extract indices of images for number i


        # Create the dataset
```

HDF/HDF5 TP

```python
import h5py
import numpy as np

# Open the file in append mode
with h5py.File(r"./mnist.h5", 'a') as file:

    # Load Images and Categories
    images = file["images"]
    categories = file["categories"]

    # Extract number for each images
    values = np.argmax(categories, axis=1)

    # Create group Numbers
    numbers = file.create_group("numbers")

    # Loop on each number
    for i in range(10):
        # Extract indices of images for number i
        inds = np.where(values == i)

        # Create the dataset
        file["numbers"].create_dataset(f"{i}", data=images[inds])
```