Working with HDF5-Based Structured Data Files: .MAT, .H5, and .NC

In neuroscience, dealing with complex and large datasets is common. Learning to use .MAT, .H5, and .NC file formats is very helpful. The .MAT format is great for MATLAB users, allowing easy data handling. The .H5 format, or HDF5, manages large and complex data efficiently, which is often needed in neuroscience research. The .NC format, especially useful for multidimensional data, builds onto H5 and is ideal for handling varied neuroscience data sets with ease. This notebook will guide you through practical exercises to understand how to use these formats effectively in your research.

Reading and Writing .MAT Files

The .mat file, often used in MATLAB, uses HDF5 technology inside. This means it is good for storing MATLAB data like arrays and matrices, as well as being able to handle big and complex data well. This format is very helpful for MATLAB users because it makes working with data easy. But, it's important to know that .mat is best within MATLAB and might not be as useful for sharing data with different software.

Code	Description
`save("ex.mat")`	save all variables in the workspace to file **`ex.mat`**
`save("folder/ex.mat")`	save all variables in the workspace to **`ex.mat`** in a separate directory named **`folder`**
`save("ex.mat", "var1")`	save variable **`var1`** to file **`ex.mat`**
`save("ex.mat", "var1", "var2")`	save variables **`var1`** and **`var2`** to file **`ex.mat`**
`save("ex.mat", "-struct", "dset")`	save the struct **`dset`** to a file
----------------------------------	----------------------------------
`whos("-file", "ex.mat")`	query the contents of the file **`ex.mat`**
----------------------------------	----------------------------------
`clear dset`	remove the variable **`dset`** from the Workspace
`load("ex.mat", "var1")`	load variable **`var1`** from file **`ex.mat`**
`load("ex.mat")`	load all variables contained in **`ex.mat`**
`load("folder/ex.mat")`	load all variables contained in **`ex.mat`**
`dset = load("ex.mat")`	assign the contents of **`ex.mat`** to a variable named **`dset`**

Create a directory for storing exercise data

```
mkdir("exdata")
```

```
Warning: Directory already exists.
```

Example:

Data:

```
x = 1:6
```

```
x = 1×6
     1     2     3     4     5     6
```

Writing Code: Save the variable **`x`** into the file **`example.mat`** and use **`whos`** to verify it was saved correctly.

```
save("exdata/example.mat", "x")
whos("-file", "exdata/example.mat")
```

```
  Name      Size            Bytes  Class     Attributes

  x         1x6                48  double
```

Reading Code: Load the data back into a Matlab struct called `dset`.

```
dset = load("exdata/example.mat")
```

```
dset = struct with fields:
    x: [1 2 3 4 5 6]
```

ex1.mat

Data:

```
x = int32(1:100);
y = double(5:.2:50);
```

Writing Code: Save both **`x`** and **`y`** into the file **`ex1.mat`** and use **`whos`** to verify it was saved correctly.

```
save("exdata/ex1.mat", "y", "x")
whos("-file", "exdata/ex1.mat")
```

```
  Name      Size             Bytes  Class     Attributes

  x         1x100              400  int32
  y         1x226             1808  double
```

Reading Code: Load all the variables from the file back into a Matlab struct called **`dset.`**

```
dset = load("exdata/ex1.mat")
```

```
dset = struct with fields:
    x: [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 33 34 35 36
    y: [5 5.2000 5.4000 5.6000 5.8000 6 6.2000 6.4000 6.6000 6.8000 7 7.2000 7.4000 7.6000 7.8000 8 8.2000
```

Reading Code: Load only the variable **x** back into a Matlab struct called **`dset`**.

```
dset = load("exdata/ex1.mat", "x")
```

```
dset = struct with fields:
    x: [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 33 34 35 36
```

Reading Code: Load only the variable **y** back into a Matlab struct called **`dset`**.

```
dset = load("exdata/ex1.mat", "y")
```

```
dset = struct with fields:
    y: [5 5.2000 5.4000 5.6000 5.8000 6 6.2000 6.4000 6.6000 6.8000 7 7.2000 7.4000 7.6000 7.8000 8 8.2000
```

ex2.mat

Data:

```
dset1 = struct(first=1:5, second=magic(5), third=["hi", "world"])
```

```
dset1 = struct with fields:
     first: [1 2 3 4 5]
    second: [5×5 double]
     third: ["hi"    "world"]
```

Writing Code: Save all the variables inside the struct **`dset1`** to the file **`ex3.mat`** and use **`whos`** to verify it was saved correctly (you should see all three variables in the file).

```
save("exdata/ex2.mat", "-struct", "dset1")
whos("-file", "exdata/ex2.mat")
```

```
  Name        Size            Bytes  Class     Attributes

  first       1x5                40  double
  second      5x5               200  double
  third        -                204  string
```

Reading Code: Load all the variables from the file back into a Matlab struct called **`dset`**.

```
dset = load("exdata/ex2.mat")
```

```
dset = struct with fields:
     first: [1 2 3 4 5]
    second: [5×5 double]
     third: ["hi"    "world"]
```

Reading Code: Load only the variables **`first`** and **`third`** back into a Matlab struct called **`dset`**.

```
dset = load("exdata/ex2.mat", "first", "third")
```

```
dset = struct with fields:
    first: [1 2 3 4 5]
    third: ["hi"    "world"]
```

Reading and Writing .H5 Files

HDF5 is a format that can store a lot of different types of data. It is chosen by many fields because it can handle large and varied data sets. HDF5 is good at organizing data in complex ways and has features like compressing data and checking for errors. HDF5 is compatible with many tools and platforms, facilitating global scientific collaboration, and with a suite of features like data compression and efficient I/O operations, HDF5 is practical for large data sets.

Code	Description
`h5create("data.h5", "/x", size(x), "Datatype", class(x))`	Define a variable and pre-allocate space for it in the file, as well as defining what type of data it will be (numeric, string, etc)
`h5write("data.h5", "/x", x)`	Write the data to the variable in the file.
------------------	-------------
h5disp("data.h5")	Validate the data by printing its contents.
------------------	----------------------
`x = h5read("data.h5", "/x")`	Read in a variable from the file.

Example:

Data:

```
x = [1 2 3 4 5]
```

```
x = 1×5
     1     2     3     4     5
```

Writing Code: Save the variable **`x`** into the file **`example.h5`** and use **`h5disp()`** to verify it was saved correctly.

```
filename = "exdata/example.h5"
```

```
filename = 
"exdata/example.h5"
```

```
% start fresh: delete the file if it already exists
if exist(filename)
    delete(filename)
end

h5create(filename, "/x", size(x))
h5write(filename, "/x", x)

h5disp(filename)
```

```
HDF5 example.h5 
Group '/' 
```

```
    Dataset 'x'
        Size:  1x5
        MaxSize:  1x5
        Datatype:   H5T_IEEE_F64LE (double)
        ChunkSize:  []
        Filters:  none
        FillValue:  0.000000
```

Reading Code: Load the data back into the variable **x**.

```
x = h5read(filename, "/x")
```

```
x = 1×5
     1     2     3     4     5
```

Reading Code: Load the data back into a Matlab struct called `dset`.

```
dset = struct();
dset.x = h5read(filename, "/x")
```

```
dset = struct with fields:
    x: [1 2 3 4 5]
```

ex1.h5

Data:

```
x = [1 2 3];
y = ["a" "b"];
```

Writing Code: Save the variables **`x`** `and` **`y`** into the file **`ex1.h5`** and use **`h5disp()`** to verify it was saved correctly.

```
filename = "exdata/ex1.h5";
if exist(filename)
    delete(filename)
end

h5create(filename, "/x", size(x), "Datatype", class(x))
h5write(filename, "/x", x);

h5create(filename, "/y", size(y), "Datatype", class(y))
h5write(filename, "/y", y)

h5disp(filename)
```

```
HDF5 ex1.h5
Group '/'
    Dataset 'x'
        Size:  1x3
        MaxSize:  1x3
        Datatype:   H5T_IEEE_F64LE (double)
        ChunkSize:  []
        Filters:  none
        FillValue:  0.000000
```

```
    Dataset 'y'
        Size:  1x2
        MaxSize:  1x2
        Datatype:   H5T_STRING
            String Length: variable
            Padding: H5T_STR_NULLTERM
            Character Set: H5T_CSET_UTF8
            Character Type: H5T_C_S1
        ChunkSize:  []
        Filters:  none
        FillValue:  ''
```

Reading Code: Load the **x** variable back into the variable **x**

```
x = h5read(filename, "/x");
```

Reading Code: Load the **x** variable back into the variable **y**

```
y = h5read(filename, "/y");
```

Reading Code: Load the data back into to a Matlab struct called **`dset`**.

```
dset = struct();
dset.x = h5read(filename, "/x");
dset.y = h5read(filename, "/y");
dset
```

```
dset = struct with fields:
    x: [1 2 3]
    y: ["a"    "b"]
```

Reading and Writing .NC Files

NetCDF, especially NetCDF-4, is based on HDF5. It takes the good points of HDF5 and adds more features for scientific data. They are very good for sharing and managing data in science and environment studies. Because NetCDF uses HDF5, it works well on many different software platforms. It combines HDF5's ability to manage data with special features for science, making it a very good choice for complex scientific data. For example:

1. Self-Describing: NetCDF files contain their own data descriptions, making it easier to describe how your experiment is structured.
2. Community Standards: NetCDF supports common standards like CF conventions, ensuring consistent data usage across the scientific community.
3. Extension of HDF5: Building on HDF5, NetCDF-4 offers enhanced capabilities for managing scientific data.

Code
`nccreate(filename, "time", "Dimensions", {"time", length(x)}, 'Datatype', class(x), 'Format', 'netcdf4')`

nccreate(filename, "voltage", "Dimensions", {"time"}, 'Datatype', class(x), 'Format', 'netcdf4')
nccreate(filename, "voltage", "Dimensions", {"time", "channel"}, 'Datatype', class(x), 'Format', 'netcdf4')
ncwrite(filename, "x", x)
----
ncdisp(filename)
----
x = ncread(filename, "x")

Example

Data:

```
time = [1 2 3 4 5];
temperature = [3 4 5 4 3];
```

Writing Code: Save the variable **`temperature`** into the file **`example.nc`**, labeling it with the dimension **`time`**, and use **`ncdisp()`** to verify it was saved correctly.

```
filename = "exdata/example.nc";
if exist(filename)
    delete(filename)
end

nccreate(filename, "Time", "Dimensions", {"Time", length(time)}, 'Datatype',
class(time), 'Format', 'netcdf4')
ncwrite(filename, "Time", time)

nccreate(filename, "Temperature", "Dimensions", {"Time"}, 'Datatype',
class(temperature), 'Format', 'netcdf4')
ncwrite(filename, "Temperature", temperature)

ncdisp(filename)
```

```
Source:
         /home/ben/ibots/iBOTS-Tools/workshops/matlab-workshop-1/plan/day3/exdata/example.nc
Format:
         netcdf4
Dimensions:
         Time = 5
Variables:
```

```
    Time
           Size:       5x1
           Dimensions: Time
           Datatype:   double
    Temperature
           Size:       5x1
           Dimensions: Time
           Datatype:   double
```

Reading Code: Reload the variable temperature by reading it from the file.

```
temperature = ncread(filename, "Temperature")
```

```
temperature = 5×1
     3
     4
     5
     4
     3
```

ex1.nc

Data: Note that the voltage's size is **`channel`** x **`time`** (every voltage value can be described by given time point for a given channel)

```
time = [.2 .4 .6 .8 1.0];
channel = ["a", "b"]
```

```
channel = 1×2 string
"a"         "b"
```

```
voltage = [
     3   4   5   4   3;
    10  11  12  11  10
];
```

Writing Code: Save the variable voltage into the file **`example.nc`**, labeling it with the dimension **`time`** and **`channel`**, and use **`ncdisp()`** to verify it was saved correctly.

```
filename = "exdata/ex1.nc";
if exist(filename)
    delete(filename)
end

nccreate(filename, "Time", "Dimensions", {"Time", length(time)}, 'Datatype',
class(time), 'Format', 'netcdf4')
ncwrite(filename, "Time", time)

nccreate(filename, "Channel", "Dimensions", {"Channel", length(channel)},
'Datatype', class(channel), 'Format', 'netcdf4')
ncwrite(filename, "Channel", channel)

nccreate(filename, "Voltage", "Dimensions", {"Channel", "Time"})
ncwrite(filename, "Voltage", voltage)
```

```
ncdisp(filename)
```

```
Source:
           /home/ben/ibots/iBOTS-Tools/workshops/matlab-workshop-1/plan/day3/exdata/ex1.nc
Format:
           netcdf4
Dimensions:
           Time    = 5
           Channel = 2
Variables:
    Time
           Size:       5x1
           Dimensions: Time
           Datatype:   double
    Channel
           Size:       2x1
           Dimensions: Channel
           Datatype:   string
    Voltage
           Size:       2x5
           Dimensions: Channel,Time
           Datatype:   double
```

Reading Code: Reload the **`time`** variable into Matlab from the file.

```
time = ncread(filename, "Time")
```

```
time = 5×1
    0.2000
    0.4000
    0.6000
    0.8000
    1.0000
```

Reading Code: Reload the **`voltage`** variable into Matlab from the file.

```
voltage = ncread(filename, "Voltage")
```

```
voltage = 2×5
     3     4     5     4     3
    10    11    12    11    10
```

Reading Code: Reload the all the data into a Matlab struct called **`dset`** from the file.

```
dset = struct();
dset.time = ncread(filename, "Time");
dset.channel = ncread(filename, "Channel");
dset.voltage = ncread(filename, "Voltage");
dset
```

```
dset = struct with fields:
       time: [5×1 double]
    channel: [2×1 string]
    voltage: [2×5 double]
```

Working with .NC Files more easily, using the EasyNC Package

EasyNC has been developed to provide a more user-friendly alternative to MATLAB's built-in functions for handling NetCDF files, which can be complex for most experimental data, which have many variables.
The EasyNC interface simplifies the process: the `easyNC.Writer` and `easyNC.Reader` classes allow for straightforward writing and reading of NetCDF data. Users can quickly write data from MATLAB structures to NetCDF files with `write_from_struct`, and conveniently read data back into MATLAB as either structures or tables using `read2struct` and `read2table`. EasyNC will also automatically load the dimension data, making it easier to focus on the main variables for an analysis. Let's try it out!

Code	Description
`f = easyNC.Writer("ex.nc")`	Create an easyNC.Writer ob
`f.write_from_struct(dset, "temperature", {"day", "time"})` **`f.write_from_struct(dset, "sunset", {"day"})`**	Write the variable "temperatu from the corresponding field Include another variable "sur
-----------------	-------------
`f.print_info()`	Validate the file by printing th
-------------------	---------------
`f = easyNC.Reader("ex.nc")`	Create an easyNC.Reader o
`dset = f.read2struct()`	Read all the data from the fil
`t = f.read2table()`	Read all the data from the fil
`t = f.read2table({'temperature'})`	Read just the `temperature` `time.`

Load software for working with NetCDF files

```
easync_location =
fullfile(fileparts(matlab.desktop.editor.getActiveFilename), "src/easyNC");
if ~exist(easync_location)
    gitclone("https://github.com/ibehave-ibots/easyNC", easync_location)
end
```

```
ans =
  GitRepository with properties:

      WorkingFolder: "/home/ben/ibots/iBOTS-Tools/workshops/matlab-workshop-1/plan/day3/src/easyNC"
          GitFolder: "/home/ben/ibots/iBOTS-Tools/workshops/matlab-workshop-1/plan/day3/src/easyNC/.git"
      CurrentBranch: [1×1 GitBranch]  (main)
         LastCommit: [1×1 GitCommit]  (80285d5)
      ModifiedFiles: [0×1 string]
     UntrackedFiles: [0×1 string]
             IsBare: 0
          IsShallow: 0
         IsDetached: 0
         IsWorktree: 0
```

```
addpath(easync_location)
```

Example

Data:

```
dset = struct( ...
    time = [.2 .4 .6 .8 1.0], ...
    channel = ["a", "b"], ...
    voltage = [ ...
         3  4  5  4  3; ...
        10 11 12 11 10 ...
    ], ...
    volume = [1.2 2.3 3.4 4.5 5.6] ...
)
```

```
dset = struct with fields:
       time: [0.2000 0.4000 0.6000 0.8000 1]
    channel: ["a"    "b"]
    voltage: [2×5 double]
     volume: [1.2000 2.3000 3.4000 4.5000 5.6000]
```

Writing Code: Save both the voltage and volume variables into the file.

```
mkdir("exdata")
```

```
Warning: Directory already exists.
```

```
filename = "exdata/example2.nc"
```

```
filename = 
"exdata/example2.nc"
```

```
f = easyNC.Writer(filename)
```

```
f = 
  Writer with properties:

    filename: "exdata/example2.nc"
```

```
f.write_from_struct(dset, "voltage", {"channel", "time"})
```

```
ans = 1×2
     2     5
```

```
f.write_from_struct(dset, "volume", "time")
```

```
ans = 1×2
     1     5
```

Reading Code: Reload all the data into a Matlab struct called `dset1`

```
easyNC.Reader(filename).read2struct()
```

```
ans = struct with fields:
    channel: [2×1 string]
       time: [5×1 double]
```

```
     voltage: [2×5 double]
      volume: [5×1 double]
```

Reading Code: Reload the "voltage" data and its associated dimensions (will happen automatically into a Matlab struct called `dset2`

```
easyNC.Reader(filename).read2struct("voltage")
```

```
ans = struct with fields:
    channel: [2×1 string]
       time: [5×1 double]
    voltage: [2×5 double]
```

Reading Code: Load the data into a table.

```
easyNC.Reader(filename).read2table()
```

ans = 10×4 table

	channel	time	voltage	volume
1	"a"	0.2000	3	1.2000
2	"b"	0.2000	10	1.2000
3	"a"	0.4000	4	2.3000
4	"b"	0.4000	11	2.3000
5	"a"	0.6000	5	3.4000
6	"b"	0.6000	12	3.4000
7	"a"	0.8000	4	4.5000
8	"b"	0.8000	11	4.5000
9	"a"	1	3	5.6000
10	"b"	1	10	5.6000

ex2.nc

Data

```
dset = struct( ...
    day = [1, 2, 3, 4, 5], ...
    sensor = ["X", "Y", "Z"], ...
    temperature = [ ...
         20 21 22 23 24; ...
         25 26 27 28 29; ...
         30 31 32 33 34 ...
    ], ...
    humidity = [45 50 55 60 65] ...
)
```

```
dset = struct with fields:
            day: [1 2 3 4 5]
         sensor: ["X"    "Y"    "Z"]
```

```
     temperature: [3×5 double]
        humidity: [45 50 55 60 65]
```

Writing Code: Save both the temperature and humidity data from above into the file.

```
filename = "exdata/ex3.nc"
```

```
filename = 
"exdata/ex3.nc"
```

```
f = easyNC.Writer(filename)
```

```
f = 
  Writer with properties:

    filename: "exdata/ex3.nc"
```

```
f.write_from_struct(dset, "temperature", {"sensor", "day"})
```

```
ans = 1×2
     3     5
```

```
f.write_from_struct(dset, "humidity", "day")
```

```
ans = 1×2
     1     5
```

Reading Code: Reload all the data into a Matlab struct called **dset1**

```
dset1 = easyNC.Reader(filename).read2struct()
```

```
dset1 = struct with fields:
         sensor: [3×1 string]
            day: [5×1 double]
    temperature: [3×5 double]
       humidity: [5×1 double]
```

Reading Code: Reload the "temperature" data and its associated dimensions (will happen automatically) into a Matlab struct called **dset2**

```
dset2 = easyNC.Reader(filename).read2struct("temperature")
```

```
dset2 = struct with fields:
         sensor: [3×1 string]
            day: [5×1 double]
    temperature: [3×5 double]
```

Reading Code: Load the data into a table.

```
table = easyNC.Reader(filename).read2table()
```

```
table = 15×4 table
```

	sensor	day	temperature	humidity
1	"X"	1	20	45
2	"Y"	1	25	45
3	"Z"	1	30	45
4	"X"	2	21	50
5	"Y"	2	26	50
6	"Z"	2	31	50
7	"X"	3	22	55
8	"Y"	3	27	55
9	"Z"	3	32	55
10	"X"	4	23	60
11	"Y"	4	28	60
12	"Z"	4	33	60
13	"X"	5	24	65
14	"Y"	5	29	65

⋮