TECHNISCHE
UNIVERSITÄT
DARMSTADT

Modelling the heating performance of magnetic nanoparticles for hyperthermia applications

User Manual v2.0.0

Caroline Karina Chandra

April 19, 2021

Caroline Karina Chandra
Matriculation Number: 2507556
Course of Studies: M. Sc. Materials Science

User Manual v2.0.0 (April 19, 2021)
Topic: Modelling the heating performance of magnetic nanoparticles for hyperthermia applications
Submission: (v1.0.0) March 29, 2021
(v2.0.0) April 19, 2021

Supervisor: Dr. Imants Dirba

Prof. Dr. Oliver Gutfleisch
Functional Materials
Alarich-Weiss-Str. 16
64287 Darmstadt

Table of Content

1 What's new in v2.0.0

##

Added:
- changelog.ipynb
- etc.ipynb
- material_properties.csv
- medium_properties.csv
- User_Manual_v2.pdf

Deleted:
- **libraries.ipynb**
- User_Manual_v1.pdf

##

etc.ipynb
- save unit conversion
- save constants
- save DropDown option for other files
- imported in allcodes.ipynb, allcodes3d.ipynb, expsim.ipynb

changelog.ipynb
- track changes made for every available version

allcodes.ipynb
- include the numpy array completelist directly as global variable in the file
- field, frequency, and surfactant layer thickness are in allcodes.ipynb
- **adding option to exclude Brownian relaxation from the calculation**
- add extra note in the export function accordingly
- **import material_properties.csv and medium_properties.csv directly in the file**
- progress bars for importing .csv files (actual loading time for calculating completelist)
completelist numpy array
- has the shape of (84, 95)
  palltaun = 55-74
  pmaxalltaun = 75-94

material_properties.csv
- contains databank for materials only

medium_properties.csv
- contains databank for mediums only

##

2 Installation

This version of User Manual is valid for v1.0, last update: March 28, 2021.

The simulation is run on JupyterLab Framework as ipython notebook. To be able to operate on JupyterLab, Anaconda Navigator must be installed. Moreover, multiple python libraries must be installed as well.

2.1 Anaconda Navigator and python libraries

Anaconda can be installed from `https://www.anaconda.com/distribution/`. The latest as well as older versions are available for download through the link. How to properly install necessary files is explained in the following steps:

1. Open “Anaconda Navigator”.
2. Create virtual programming environment, go to “Environments” >> “Create” >> give a name, for example “scientcomp” >> tick Python packages
3. Go to “Home”, install Jupyterlab
4. Start “Anaconda Prompt” to install the necessary python libraries.
5. Go to the newly created environment with `conda activate scientcomp`
6. Install pip to load libraries using `conda install pip`
7. Following libraries are to be loaded:

```
numpy==1.18.2
pandas==1.0.3
pyomo==5.6.9
ffmpeg==1.4
matplotlib==3.2.1
scipy==1.4.1
numba==0.49.0
conda install -c conda-forge ipopt glpk
conda install pytorch
conda install -c conda-forge ipywidgets
```

8. Install matplotlib widget on JupyterLab using following code:

```
conda install -c conda-forge nodejs
pip install ipympl
pip install --upgrade jupyterlab
jupyter labextension install @jupyter-widgets/jupyterlab-manager
jupyter labextension install jupyter-matplotlib
jupyter nbextension enable --py widgetsnbextension
```

9. Anaconda Navigator is installed and JupyterLab is ready to use. If this is not the case, refer to Chapter 4: Error

2.2 JupyterLab

JupyterLab is the next-generation web-based user interface for Project Jupyter. JupyterLab will eventually replace the classic Jupyter Notebook. Throughout this transition, the same notebook document format will be supported by both the classic Notebook and JupyterLab, Figure 1.

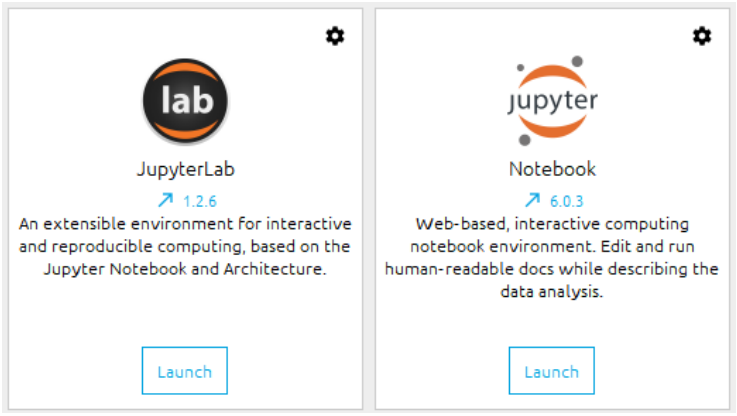

Figure 1: JupyterLab and Jupyter Notebook from Anaconda Navigator.

2.3 Python libraries import

Loading python libraries via pip in Anaconda Prompt allows the libraries to be imported in the ipython notebook `.ipynb`. Following imports are necessary:

```
import pandas as pd
import numpy as np
import time
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.widgets import CheckButtons
from matplotlib.offsetbox import AnchoredText
from matplotlib.patches import Patch
from matplotlib import cm
from matplotlib.lines import Line2D
from mpl_toolkits.mplot3d import Axes3D
%matplotlib widget
import ipywidgets as wg
from ipywidgets import HBox, VBox, Checkbox, ToggleButton
from IPython.display import display
```

To import another `.py` files, the command is `%run  filename`, while another `.ipynb` files, the command ist `%run filename.ipynb`

If a library is already imported in one particular file, and that particular file as a whole is imported to a new file, the import need not to be repeated in the new file. The magic command `%` takes the import from the imported file as well.

The purpose of `import library as something` with shorter name has no technical reason other than to simply shorten the name and minimizing the typing effort. As an example, instead of writing `matplotlib.pyplot.plot(x, y)`, only `plt.plot(x, y)` is necessary.

The command `from library import something` has the same purpose. Instead of importing the whole library, only that particular command(s) is imported. For example, if the command `display` is needed, by writing `from IPython.display import display`, no additional prefix is necessary anymore: `display(...)` instead of `IPython.display.display(...)`.

2.3.1 pandas

pandas is imported in `libraries.ipynb`. All databases are stored as pandas data frame. Pandas allows easy organizing the database thanks to its row and column naming ability (which is not the case for numpy array). [1] To call the intended cell, the following command notation is required:

`name_of_dataframe['column_name']['row_name']`.

As an example, unit conversion for all files which are also stored as a pandas data frame, Figure 2.

Unit conversion

```
unit_data = [['kA/m', 1e3, 'A/m'],\
             ['kJ/m3', 1e3, 'J/m3'],\
             ['Ns/m2', 1, 'Ns/m2'],\
             ['nm', 1e-9, 'm'],\
             ['G', 1e-4 / vacuumpermeability, 'A/m'],\
             ['kHz', 1e3, 'Hz'],
             ['g/cm3', 1e3, 'kg/m3']]
unit = pd.DataFrame(unit_data, columns = ['Unit', 'Factor', 'SI unit'], index = ['Saturation magnetization',\
                                                                                 'Anisotropy constant',\
                                                                                 'Viscosity',\
                                                                                 'Surfactant layer thickness',\
                                                                                 'Field',\
                                                                                 'Frequency',\
                                                                                 'Density'])
print(unit)
```

```
                          Unit       Factor SI unit
Saturation magnetization   kA/m  1.000000e+03    A/m
Anisotropy constant       kJ/m3  1.000000e+03    J/m3
Viscosity                Ns/m2  1.000000e+00   Ns/m2
Surfactant layer thickness   nm  1.000000e-09      m
Field                         G  7.957748e+01    A/m
Frequency                   kHz  1.000000e+03     Hz
Density                   g/cm3  1.000000e+03   kg/m3
```

Figure 2: unit conversion stored as pandas data frame and its output.

In this case, the command `unit['Factor']['Anisotropy constant']` will give out the value 1e3 or 1000 if printed.

2.3.2 NumPy

NumPy is imported in `libraries.ipynb`. NumPy is an easy-to-use numerical computing tool which offers large collection of high-level mathematical functions, linear algebra, N-dimensional arrays and matrices, etc. The concept of NumPy array is the standard of array computing today. The array `completelist` is stored as NumPy array. The export function also depends on NumPy and pandas. [2]

2.3.3 Matplotlib

Matplotlib is a numerical mathematical extension of NumPy for plotting library. The extension `matplotlib.pyplot` has a collection of function that makes matplotlib works like MATLAB. One known drawback of plotting with matplotlib is the 3D plotting function. Matplotlib does not have an actual 3D engine. For example, the import `mpl_toolkits.mplot3d` takes a scatter point and projects them to what it would look like on a 2D plot from a particular camera position (angle view). This is a little hack from Matplotlib to get some 3D functionality. However, this causes the `zorder` argument to be lost. Regardless of what is in front, the object that is plotted last will override the one plotted before (if viewed from a particular point-of-view). The `zorder` argument can be set manually, either by adding `item.set_zorder(number)` as a command or writing the argument `zorder=number` directly as a parameter in the `plt.plot(...)`. Smaller number (integer, starts from 1) corresponds to further position from viewer point-of-view. [3]

2.3.4 ipywidgets

In order to make the file to be user-friendly, it hast to be interactive. User must be able to give (easy and understandable) input into the program to plot without having to write an actual python code. `ipywidgets` makes it possible [4]. `ipywidgets` are interactive HTML widgets for Jupyter notebooks (JupyterLab) and the IPython kernel [5].

2.3.5 time

Python provides time library without having to install extra modules via pip. The command `time.time()` returns the number of seconds passed since January 1, 1970, 00:00:00 at UTC in second. The command `time.strftime("%Y%m%d_%H%M%S")` returns real-time in `string` with the format YYYYMMDD_hhmmss (for example, March 22, 2021, 8:15 p.m. would be 20210322_201500). This function is used to name the exported file. Naming the file with real-time stamp avoid any of the exported file to be overwritten. [6]

3 Using the files

3.1 Calculation

3.1.1 Relaxation times

The formula to calculate Néel relaxation time [7]:

$$\tau_N = \tau_0 \left(\frac{K_u V_M}{k_B T}\right)^{-1/2} e^{\frac{K_u V_M}{k_B T}} \tag{1}$$

With V_M the volume of sphere, Equation (1) can be rewritten into:

$$\tau_N = \tau_0 \left(\frac{K_u\, \pi\, d_p^3}{k_B T}\right)^{-1/2} e^{\frac{K_u\, \pi\, d_p^3}{k_B T}} \tag{2}$$

$$\tau_N = \frac{\tau_0}{\sqrt{\frac{K_u\, \pi\, d_p^3}{6\, k_B T}}} \exp\left(\frac{K_u\, \pi\, d_p^3}{6\, k_B T}\right) \tag{3}$$

Equation (3) is the final formula used in the calculation of τ_N. The parameters are as follows:

τ_N : Néel relaxation time $[s]$

τ_0 : constant factor $(10^{-9}\ s)$

K_u : anisotropy constant $[J/m^3]$

d_p : particle diameter $[m]$

k_B : Boltzmann constant $[m^2 kg/s^2 K^1]$

T : temperature (300 K)

As d_p is present in the exponential function, this term could rapidly go into infinity. As it goes to the infinity, instead of any number, python returns `inf`, which can't be used for further calculation. The threshold value for a number being non-infinity in python is 1.8e308. Any number larger than this is identified as infinity. The information at number this large is irrelevant to this work and therefore, the calculation is a waste of memory. To reduce the number of iterations, a limit can be set. The limit is arbitrary, in this work, the limit is set to 1e50 for the exponential factor.

$$\exp\left(\frac{K_u\, \pi\, d_p^3}{6\, k_B T}\right) = 1e50 \tag{4}$$

$$\ln\left(exp\left(\frac{K_u\, \pi\, d_p^3}{6\, k_B T}\right)\right) = \ln(1e50) \tag{5}$$

$$\frac{K_u\, \pi\, d_p^3}{6\, k_B T} = \ln(1e50) \tag{6}$$

$$d_p^3 = \frac{6\, k_B T}{K_u\, \pi} \ln(1e50) \tag{7}$$

$$d_p(limit) = \left(\frac{6\, k_B T\, \ln(1e50)}{K_u\, \pi}\right)^{\frac{1}{3}} \tag{8}$$

This calculation in Equation (8) is done simply to define a limit when making an array of d_p.

The formula to calculate Brownian relaxation time [8]:

$$\tau_B = \frac{\pi\, \eta\, V_H}{k_B T} \tag{9}$$

With V_H being the hydrodynamic volume (volume of sphere nanoparticle and fluid coating), it can be rewritten as follow:

$$\tau_B = \frac{\pi\, \eta\, \left(d_p + d_s\right)^3}{2\, k_B T} \tag{10}$$

The parameter d_s is the surfactant layer thickness of the particle and η is viscosity of the fluid $[Ns/m^2]$. Equation (10) is the final formula used for calculating τ_B.

The total relaxation time is the reciprocal combination of both, Equation (11).

$$\frac{1}{\tau_{total}} = \frac{1}{\tau_N} + \frac{1}{\tau_B} \tag{11}$$

3.1.2 Power loss density and SAR

The formula for power loss density is [9]:

$$P = \frac{\pi\, \mu_0^2\, M_s^2\, V_M H^2}{3\, k_B T} \frac{\omega\tau}{1 + (\omega\tau)^2}\, f \tag{12}$$

The parameters are as follow:

P : powerloss density [W/m^3]

μ_0 : vacuum permeability [N/A^2]

M_s : saturation magnetization $[A/m]$

H : field $[A/m]$

f : frequency $[Hz]$

ω : angular frequency $[Hz]$

The maximum is reached when $\omega\tau = 1$:

$$\frac{d\, \frac{(\omega\tau)}{1 + (\omega\tau)^2}}{d\, (\omega\tau)} = 0 \tag{13}$$

Let $x = (\omega\tau)$:

$$\frac{d\, \frac{x}{1 + x^2}}{d\, x} = 0 \tag{14}$$

$$-\frac{x^2 - 1}{(x^2 + 1)^2} = 0 \tag{15}$$

$$x = 1 \quad \rightarrow \quad (\omega\tau) = 1 \tag{16}$$

Specific absorption rate (SAR $[W/g]$) is powerloss density normalized by the density of the nanoparticle ρ $[kg/m^3]$:

$$SAR = \frac{P}{\rho} \tag{17}$$

The final formula for calculating SAR within this work is written in Equation (18):

$$SAR = \frac{\pi^3 \mu_0^2 M_s^2 d_p^3 H^2 f^2 \tau}{9 k_B T \rho (1 + (2\pi f\tau)^2)} \tag{18}$$

3.1.3 Determining optimum particle diameter

As described in Equation (16), a maximum is reached when $(\omega\tau) = 1$. By expanding this term, the particle diameter can be determined.

$$2\pi f\tau = 1 \tag{19}$$

For high viscosity fluid, the influence of τ_B is completely negligible, thus, leaving τ_N to take the role of relaxation time. Combining Equation (19) and Equation (3) leaves:

$$2\pi f \frac{\tau_0}{\sqrt{\frac{K_u \pi d_p^3}{6 k_B T}}} \exp\left(\frac{K_u \pi d_p^3}{6 k_B T}\right) = 1 \tag{20}$$

Let $x = \frac{K_u \pi}{6 k_B T}$:

$$2\pi f \frac{\tau_0}{\sqrt{x d_p^3}} \exp\left(x d_p^3\right) = 1 \tag{21}$$

$$\exp\left(x d_p^3\right) = \frac{\left(x d_p^3\right)^{0.5}}{2\pi f \tau_0} \tag{22}$$

$$x d_p^3 = \ln\left(\frac{\left(x d_p^3\right)^{0.5}}{2\pi f \tau_0}\right) \tag{23}$$

$$x d_p^3 = \ln\left(\left(x d_p^3\right)^{0.5}\right) - \ln(2\pi f \tau_0) \tag{24}$$

$$x d_p^3 - \ln\left(\left(x d_p^3\right)^{0.5}\right) = \ln(2\pi f \tau_0) \tag{25}$$

$$x d_p^3 - \ln(x^{0.5}) + \ln(d_p^{1.5}) = \ln(2\pi f \tau_0) \tag{26}$$

$$x d_p^3 + \ln\left(d_p^{1.5}\right) = \ln(2\pi f \tau_0) + \ln(x^{0.5}) \tag{27}$$

$$\frac{K_u \pi}{6 k_B T} d_p^3 + \ln\left(d_p^{1.5}\right) = \ln(2\pi f \tau_0) + \ln\left(\left(\frac{K_u \pi}{6 k_B T}\right)^{0.5}\right) \tag{28}$$

This leaves the equation unable to be solved analytically. However, Equation (20) can be solved numerically when one goes to the iteration of d_p for each f. Depending on the iteration step, the process

can be massively shortened by using larger step and going over the value of 1 and then go back with finer step. A pseudo-code for this goes as follow:

```
for j in range(len(f)): # iteration of f
    for i in range(len(dp)): # iteration of dp, step size = 1 nm
        calculate variable wt # as in Equation (20), dp iterated by 1 nm
        if wt > 1.0:
            for k in range(100): # iteration of dp, step size = 0.01 nm
                calculate variable wt # dp going from .99 to .01 nm
                if wt == 1.0:
                    break # stop iteration and save the found dp
            break # no need to check further dp value with step size of 1 nm
```

3.1.4 SAR at optimum particle diameter

The highest SAR that can be achieved mathematically by a material is at the highest available frequency and field. Because at the optimum d_p the product of $\omega\tau = 1$, the optimizable term becomes a constant, Equation (35):

$$\frac{\omega\tau}{1+(\omega\tau)^2} = \frac{1}{1+1^2} = 0.5 \tag{29}$$

This allows the SAR formula to be simplified to Equation (30):

$$P = \frac{\pi\,\mu_0^2\,M_s^2\,V_M}{3\,k_B T}\frac{1}{2} f\,H^2 \tag{30}$$

And further simplification leads to Equation (31):

$$P = \frac{\pi^2\,\mu_0^2\,M_s^2\,d_p^3}{36\,k_B T} f\,H^2 \tag{31}$$

Increasing field, however, has higher impact on the SAR value than frequency, for two reason:

1. Field is to the second power and is independent from all other parameter, while frequency increases the SAR only linearly.
2. Increasing frequency reduces optimum d_p slightly, which is to the third power.

3.1.5 Comparing the heating performance of multiple materials

For comparison purpose, constants (and parameters that can be kept constant) can be omitted, leaving only M_s, d_p, f, τ, and ρ. Field is neglectable since it is an independent parameter. For each material, the SAR with arbitrary unit can be calculated, Equation (32).

$$SAR_{compare} = \frac{M_s^2\,d_p^3\,f^2\,\tau}{\rho\,(1+(2\pi f\tau)^2)} \tag{32}$$

The particle diameter d_p is the optimum d_p for each frequency calculated according to Chapter 3.1.3

3.1.6 Atkinson (and Brezovich) limit

Atkinson limit is a biological limit that was based on the patient withstanding of the treatment for more than one hour without major discomfort. The heat generated by eddy current is proportional to the square of the product of Hf (and the square of the radial distance). This means that unless the product of Hf is maintained below a certain limit, (healthy) body tissues might overheat. The Atkinson limit is proposed to be 4.85e8 A/ms. [10] With typical used frequency of $f = 100\ kHz$ [11].

$$Hf = 4.85e8 \frac{A}{ms} \tag{33}$$

$$H = \frac{4.85e8\ \frac{A}{ms}}{1e5\ \ 1/s} = 4.85e3\ \frac{A}{m} \approx 61\ G \tag{34}$$

Researchers are currently discussing about this limit being more like a recommended value according to statistical data which depends highly from person to person, rather than a critical absolute value.

3.2 Functionalities

3.2.1 Navigation bar matplotlib widget

Matplotlib widget (the import `%matplotlib widget`) offers user interaction with built-in navigation bar with various function, including download plot, Figure 3.

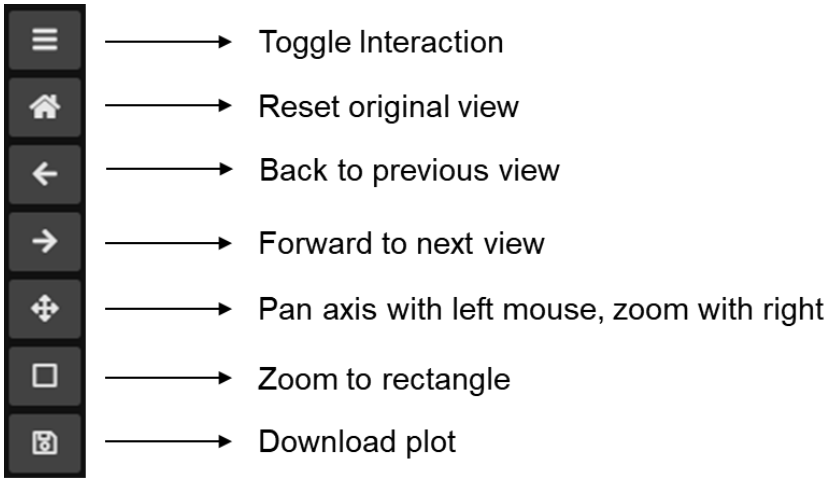

Figure 3: navigation bar of matplotlib widget.

3.2.2 Export data

Under every plot, with exception of multi plot 2D $(d_p - \tau)$ and 3D compare $\left(f - SAR_{max} - d_p\right)$, there is an export button. The function of this button is self-explanatory, to export the data (from plot above it) to a .csv file, Figure 4.

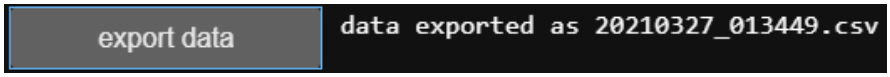

Figure 4: clicked export button.

When the export button is clicked, a .csv file will be generated. The file name has the format of current time, YYYYMMDD_hhmmss. The printed output next to the button serves only as an assurance for the user that the button is clicked. The exported .csv files consist of various axis values and a header at the top of the column. The export function is written on `libraries.ipynb` using NumPy and pandas, Figure 5.

```
def export(X,Y, Material, Medium, Axes, Header):
    ExportButton = wg.ToggleButton(value=False, description='export data', tooltip='export data as csv file')
    outputexport = wg.Output()
    display(HBox([ExportButton, outputexport]))
    def exportbutton(change):
        with outputexport:

            data = np.asarray(np.hstack((X, Y)))
            timestr = time.strftime("%Y%m%d_%H%M%S")
            print(f'data exported as {timestr}.csv')
            pd_data = pd.DataFrame(data)
            pd_data.to_csv(f'{timestr}.csv', header=Header, index=False)

            f = open("list_of_csv.txt", "a")
            f.write(f"{timestr}.csv: Material={Material}, Medium={Medium}, Plot={Axes}\n")
            f.close()
    ExportButton.observe(exportbutton, names='value')
```

Figure 5: export function in `libraries.ipynb`.

3.3 Files

Some files are user-friendly, and some are not:

Non user-friendly:

- `libraries.ipynb`
- `allcodes.ipynb`
- `allcodes3d.ipynb`
- `expsim.ipynb`

User-friendly:

- `userfriendly_plotter.ipynb`
- `userfriendly_quickplot.ipynb`
- `experiment_simulation.ipynb`
- `list_of_csv.txt`

To use any of the files, user must open JupyterLab from Anaconda Navigator:

1. Open “Anaconda Navigator”.
2. Lauch “JupyterLab”. This will trigger your default browser to be opened.
3. The current active directory can be seen on the left side, go to intended directory (where the files are saved).
4. Double click to open desired file.
5. Select “Run” from top left toolbar.
6. Select “Run All Cells” from the dropdown menu.
7. Wait a few second until the default home view is shown.

3.3.1 `libraries.ipynb`

Database are written in this file. Material database contains M_s, K_u and ρ. Medium contains η. There are four field values stored, they are 100, 200, 300, 350 Gauss. The frequencies stored are 98, 158, 204, 313, 402 kHz. Those are the values, which are commonly used in the lab during an actual experiment. If user intends to use another value of field and or frequency, user can either change it directly in the `libraries` (for continuous use in `userfriendly_plotter`) or doing it manually each time in `userfriendly_quickplot` or `experiment_simulation`.

In `libraries.ipynb`, completelist is an array stored as global variable written as a NumPy array with `np.shape(completelist)` equals to (length_of_medium * length_of_material, 55). The array stores all material with each medium in every row. The pseudo-code determining the row order goes as follow:

```
for i in range(length_of_medium):
    for j in range(length_of_material):
        append
```

Each column contains:

Material	= 0	d_p_all	= 6
M_s	= 1	$\tau_N, \tau_B, \tau_{total}$	= 7-9
K_u	= 2	SAR_all	= 10-29
ρ	= 3	SARmax_all	= 30-49
Medium	= 4	d_p_max	= 50-54
η	= 5		

The order of the row goes as follows:

0	Material_0	→	Medium_0
1	Material_1	→	Medium_0
2	Material_2	→	Medium_0
⋮	⋮	→	Medium_0
⋮	Material_n	→	Medium_0
⋮	Material_0	→	Medium_1
⋮	Material_1	→	Medium_1
⋮	Material_2	→	Medium_1
⋮	⋮	→	Medium_1
⋮	Material_n	→	Medium_1

Example:

```
return(completelist[2,3]) # calls row 2, col 3 (density)
>> rho # rho of Material_2 in Medium_0 (whichever the medium doesn't matter in this case)
```

The file `libraries.ipynb` is imported in all other files. `allcodes` is completely dependent of it, while `allcodes3d` and `expsim` are partially dependent (unit conversions, global variable). No user output is to be expected from `libraries.ipynb`.

The saved values in material and medium databases are theoretical values (from literature). Saturation magnetization M_s saved are the value measured (theoretical) at $H = 2\,T$ or $H = 2000\,G$. The purpose of deliberately having all saturation magnetization at $H = 2\,T$, despite aware that the values are obviously off, are comparability between materials and practical reason, since these values are easier to find in the literature than, for example, the saturation magnetization of a certain material at $H = 0.35\,T$.

3.3.2 `allcodes.ipynb` and `userfriendly_plotter.ipynb`

The code used to run `userfriendly_plotter.ipynb` is stored in `allcodes.ipynb`.

The default home view looks as shown in Figure 6.

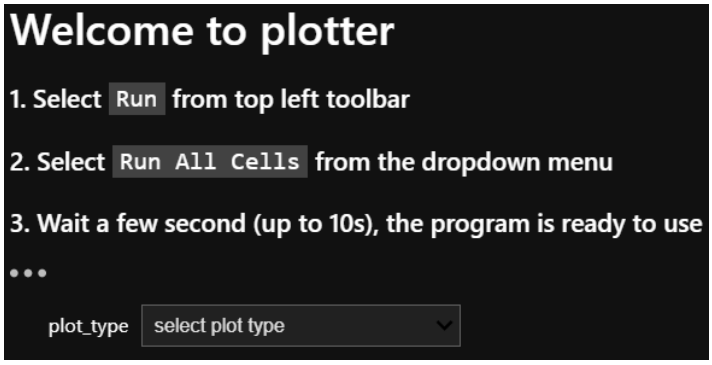

Figure 6: default home view of `userfriendly_plotter.ipynb` after "Run All Cells". Hidden ⋯ is `%run allcodes.ipynb`.

The file can plot following:

- single plot 2D:
  - Particle diameter – Relaxation times $(d_p - \tau)$
  - Particle diameter – SAR $(d_p - SAR)$
  - Frequency – maximum SAR $(f - SAR_{max})$
  - Field – maximum SAR $(H - SAR_{max})$
- multi plot 2D:
  - Particle diameter – Relaxation times $(d_p - \tau)$
  - Saturation magnetization – maximum SAR $(M_s - SAR_{max})$
- 3D/4D plot:
  - Frequency – Field – Particle diameter – maximum SAR $(f - H - d_p - SAR_{max})$
  - Frequency – SAR – Particle diameter $(f - SAR_{max} - d_p)$

The purpose of `allcodes` (and `userfriendly_plotter`) is for comparison between material in their respective theoretical state and ideal condition. All values returned by `allcodes` are incomparable to experimental results, since MFH is not carried at $H = 2\,T$. Moreover, the assumption of linear response

is not valid anymore at field that high. Single and multi plot 2D take in `completelist` precalculated values (from `libraries.ipynb`).

In the single plot 2D where SARs are plotted, all in `libraries` listed frequencies and fields are used for calculation at once. To avoid overcrowding the graph, there are toggle buttons that serve as activation button to show/hide the plot, Figure 7.

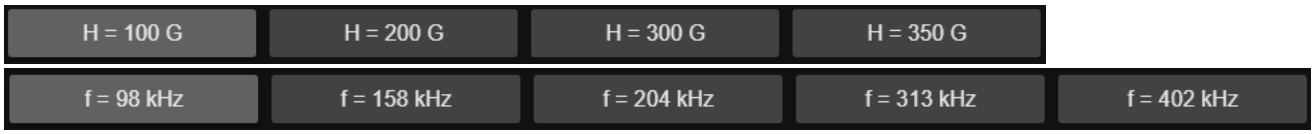

Figure 7: ToggleButton for show/hide respective plot.

In Figure 7, the toggle buttons at the top with the field are available at $(d_p - SAR)$ and $(f - SAR_{max})$, while the one at the bottom with the frequency are available at $(H - SAR_{max})$. Activated toggle button means "show" and deactivated means "hide". In the plot $(f - SAR_{max})$ and $(H - SAR_{max})$, the SARs are calculated at optimum d_p (written above the x-axis or in the legend).

In the multi plot 2D $(d_p - \tau)$ only τ_{total} would be plotted. The functions of the buttons are written and are self-explanatory. The "Remove" function only works once for last plotted line.

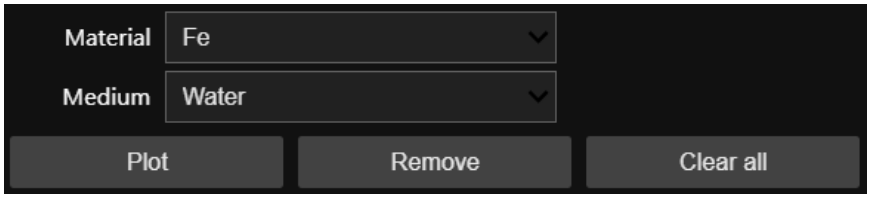

Figure 8: buttons found in multi plot 2D $(d_p - \tau)$.

No export function is offered here since the data is not complete anyway. For exporting user could refer to single plot $(d_p - \tau)$ for each desired material-medium combination.

In multi plot 2D $(M_s - SAR_{max})$, the SAR is all calculated at their respective optimum particle diameter (written in legend) at $f = 402\ kHz$ and $H = 350\ G$, Figure 9.

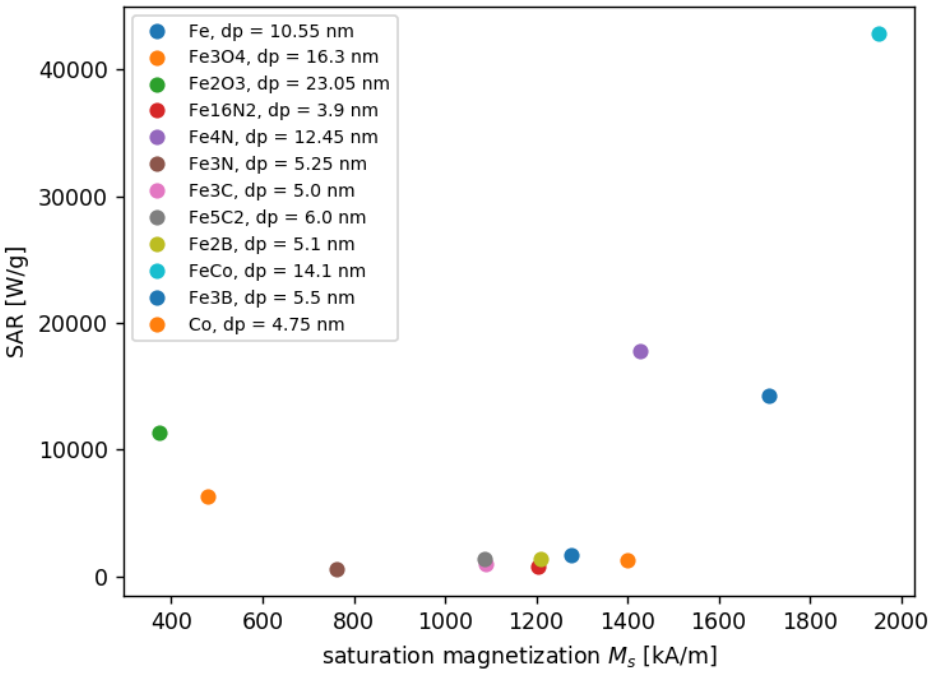

Figure 9: multi plot 2D ($M_s - SAR_{max}$) at at $f = 402\ kHz$ and $H = 350\ G$.

This plot basically compares the heating performance of all materials saved in `libraries` without further user input. The values for f and H are chosen that way, because mathematically the SARs are at their highest at those values (refer to Chapter 3.1.4).

The 4D plot takes material specific parameters from `completelist` as well, but doing the calculation independently from `libraries.ipynb`. Frequency and field are made-up values, while particle diameter and SAR are calculated within the file themselves. Particle diameter is calculated based on Equation (20). SAR is calculated based on Equation (18). SAR information is shown by the color, thus, making the plot 4D. These are the steps:

1. Load the material specific parameters as chosen from the dropdown list from `libraries`.
2. Create an array for frequency.
3. Prepare an empty array for saving optimum d_p for each frequency.
4. Calculate the optimum d_p for one frequency:
    a. Iterate with a step size of 1 for d_p, increasing from 1, 2, 3, …. The number of iterations can be set to 100, this does not matter, as the loop will be broken before the maximum number of iterations is reached.
    b. Set an if according to Equation (20) to break the loop once 1 is exceeded, the maximum value, Equation (35).

$$2\pi f \frac{\tau_0}{\sqrt{\frac{K_u\ \pi\ d_p^3[i]}{6\ k_B T}}} \exp\left(\frac{K_u\ \pi\ d_p^3[i]}{6\ k_B T}\right) > 1 \tag{35}$$

c. Iterate with a step size of 0.01 for d_p, decreasing from $(d_p[i])$ to $(d_p[i]-1)$. The number of iterations is 100, this means x.99, x.98, x.97, …, x.01 with x being $(d_p[i])$.
d. Set an if that satisfy Equation (20), break the loop once it is satisfied.
e. The d_p that satisfy Equation (20) is the optimum particle diameter for one particular value of frequency. Append this value to the prepared array.

5. Repeat step 4 to cover all frequency. For each frequency, there is one optimum particle diameter. After this point, the shape of the frequency array and the d_p array must be the same.
6. Create an array for field.
7. Via nested list comprehension, create a new array for frequency, field, and particle diameter. Each array should have the shape of `(len(f) * len(H), 1)`.
    a. New H-array : `np.array([i for i in H for r in range(len(f))])`
    b. New f-array : `np.array([i for i in f] * len(H))`
    c. New dp-array : `np.array([i for i in dp] * len(H))`
8. Calculate SAR using Equation (18).
9. Plot scatter point using `ax.scatter(x, y, z, c=color)`.
10. Show colorbar using `cbar = fig.colorbar(img)`.
11. 4D scatter plot is ready.

Step 7, 8, and 9 make sure, that the the command `ax.scatter` must only be called once. This is a much faster method than doing a simple for-loop, in which `ax.scatter` is called len(H) times.

1. Iterate through H-array `(H[i])`.
2. Calculate SAR using Equation (18) for each `H[i]`. At this point, SAR-array has the shape of (Nx1) as frequency- and dp-array.
3. Plot scatter point with x = array(f), y = float(H), z = array(dp), c = array(SAR) using `ax.scatter` directly for each `(H[i])`.

On a Core i5 8th Gen laptop with 8 GB RAM, the average runtime (statistical average of 10 runs) if using nested list comprehension is 1.38 s, while a simple for-loop needs in average 6.87 s. While it is clearly more efficient to use nested list comprehension method, user could opt for simple for-loop. One reason why one would want to opt for a slower method is that by using for-loop, there is an option to have a progress bar shown on the screen with an actual percentage of iterated H-array. The progress bar does not have any technical purpose, its sole purpose is for user's sanity check if using older computers or using a larger dataset (where the waiting time is not a mere 1.38 s or 6.87 s). Without progress bar, user can't be sure, if the code is actually running or the user is waiting in eternity since the code is not running. Progress bar requires a continuously changing number for its percentage (which can be perfectly done in for-loop). Nested list comprehension calculates all at once and plot the data as a whole, which leaves

no continuously changing number. For this reason, progress bar is only available for plotting using for-loop method.

As shown in Equation (32), many parameters can be neglected for the sole purpose of comparison between material. A 3D graph with frequency and SAR in the x- and y-axis and particle diameter as color-map can be plotted. The functionality is similar to $(d_p - \tau)$ 2D multi plot, Figure 10.

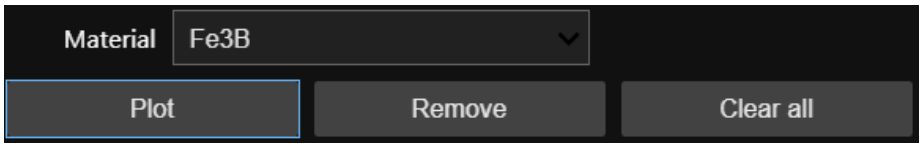

Figure 10: functionalities offered in $(f - SAR - d_p)$ 3D graph, high viscosity medium is assumed.

This plot focus on the heating performance of the material, independent of the applied field. The unit of the SAR is arbitrary. The color bar shows no absolute values, only min and max. The dp min and max of each material is different and is written on the legend. The color map of each set of scatter point (each material) is normalized (0 - 1).

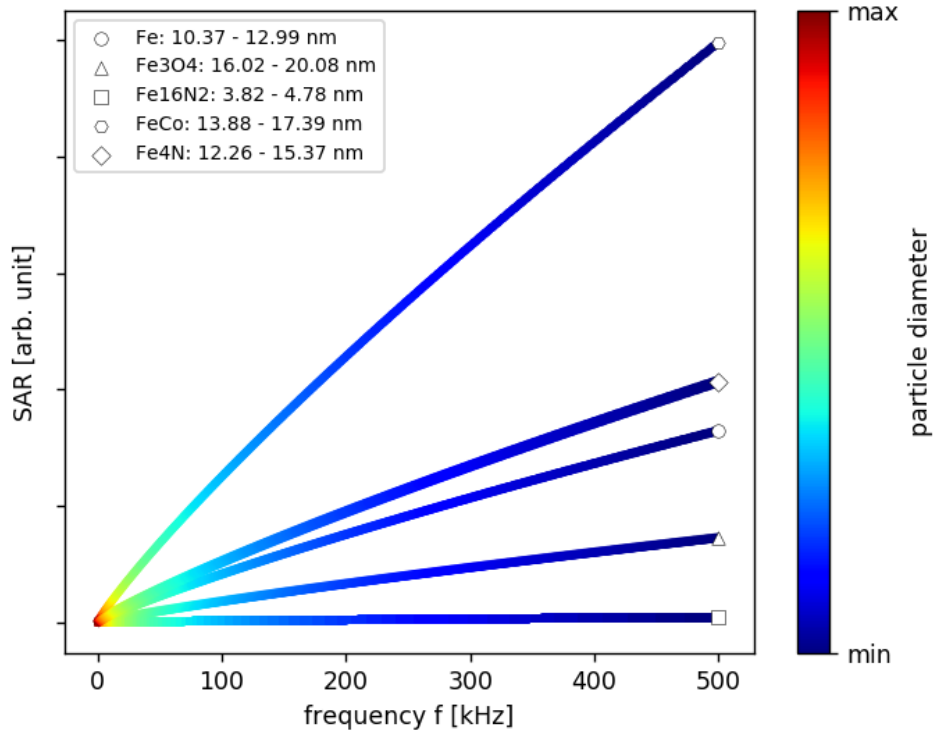

Figure 11: sample $(f - SAR - d_p)$ plot containing multiple materials.

3.3.3 `allcodes3d.ipynb` and `userfriendly_quickplot.ipynb`

These files are partly dependent of `libraries.ipynb` as they take unit conversion and global variables from libraries. The default home view of `userfriendly_quickplot.ipynb` looks as shown in Figure 12.

Figure 12: default home view of userfriendly_quickplot.ipynb after "Run All Cells". Hidden ⋯ is `%run allcodes3d.ipynb`.

Depends on the type of graph, various user input is required. The values shown by default correspond to the theoretical values of pure iron and water as medium. Certain data type of user input and the unit are predetermined as follow:

Material	: `str`	K_u $[kJ/m^3]$	: `float`	ρ $[g/cm^3]$	: `float`	M_s [kA/m]	: `float`
Medium	: `str`	d_s $[nm]$	: `float`	η $[Ns/m^2]$	: `float`		
f $[kHz]$	: `int`	H $[G]$	: `int`	d_p $[nm]$	: `float`		

Following graphs can be plotted using these files:

- 2D:
    - Particle diameter – Relaxation times $(d_p - \tau)$
    - Particle diameter – SAR $(d_p - SAR)$
    - Frequency – SAR $(f - SAR)$
    - Field – SAR $(H - SAR)$
- 3D:
    - Frequency – Field – SAR $(f - H - SAR)$
    - Field – Frequency – SAR $(H - f - SAR)$
    - Frequency – Particle diameter – SAR $(f - d_p - SAR)$
    - Particle diameter – Frequency – SAR $(d_p - f - SAR)$
    - Field – Particle diameter – SAR $(H - d_p - SAR)$
    - Particle diameter – Field – SAR $(d_p - H - SAR)$

In these files, the x- and y-axis are chosen separately, but not independently. Starting with the dimension, user input must be given in top-down order. Without selecting the x-axis, the options in the y-axis (and z-axis accordingly) are deactivated, stating that user needs to select the x-axis first. The options shown in the y-axis (and z-axis) depend on the selected option in the x-axis. The way to do this is by building python closures (or a nested function). Following is an example with pseudo code for a 3D axis:

```
def choosex(x):
      do something
      def choosey(y): # optiony is dependent to chosen optionx
            do something
            def choosez(z): # optionz is dependent to chosen optiony
                  do something
            wg.interact(choosez, z = optionz) # optionz is a list with str
      wg.interact(choosey, y = optiony) # optiony is a list with str
wg.interact(choosex, x = optionx) # optionx is a list with str
```

For 2D axis it works accordingly. The innermost function has access to the variable in the outer function, but not the other way around. `wg.interact()` is a function from `ipywidgets` library for interactive HTML widgets for Jupyter.

When user plot 2D graph of *anything – SAR*, there will be 3 lines, each line represents the SAR calculated from *only* τ_N, *only* τ_B, and τ_{total} respectively. As the medium only influences τ_B and the influence of τ_B is only at low viscosity medium, some user would want to have a function to differentiate the SAR calculation from their heat generation mechanism.

The 3D plot assumes high viscosity medium, thus, ignoring the influence of τ_B in the SAR calculation. This means less user input, as Medium, viscosity and surfactant layer thickness are left out. Material, ρ, K_u, and M_s are the standard user input and depending on the plot chosen, further user input of f, H, or d_p is required. For d_p as user input, the values will be checked directly, and if it is too large (exponential factor exceeded 1e50), user will get a warning stating exactly this and will be told the highest available d_p that is still within the limit of 1e50. Following are the steps done in the `allcodes3d` to create the 3D graphs:

1. Create arrays for respective x- and y-axes:
    a. Frequency f : start = 1, end = desired arbitrary limit of frequency.
    b. Field H : start = 1, end = desired arbitrary limit of field.
    c. Particle diameter d_p : start = 1, end = from Equation (8) calculated $d_p(limit)$
2. Create mesh grid from the arrays using `X, Y = np.meshgrid(X, Y)`.
3. Calculate SAR.
4. Plot the 3D graph using `ax.plot_surface(...)`.

Making mesh grid from two arrays with the command `X, Y = np.meshgrid(X, Y)` makes the shape of both arrays the same to `(len(X), len(Y))`. This forms the x-y-plane for the plot. The SAR values are calculated at once as one big array. Mesh grid is required for surface plotting.

3.3.4 `expsim.ipynb` and `experiment_simulation.ipynb`

These files requires user input for plotting instead of looking up the values in `libraries.ipynb`. Only unit conversion and global variables are taken from libraries. Same as `userfriendly_quickplot`, these files also require user input. The data types for the user input are the same as explained in the previous chapter (Chapter 3.3.3). In `experiment_simulation.ipynb`, the default home view looks as in Figure 13.

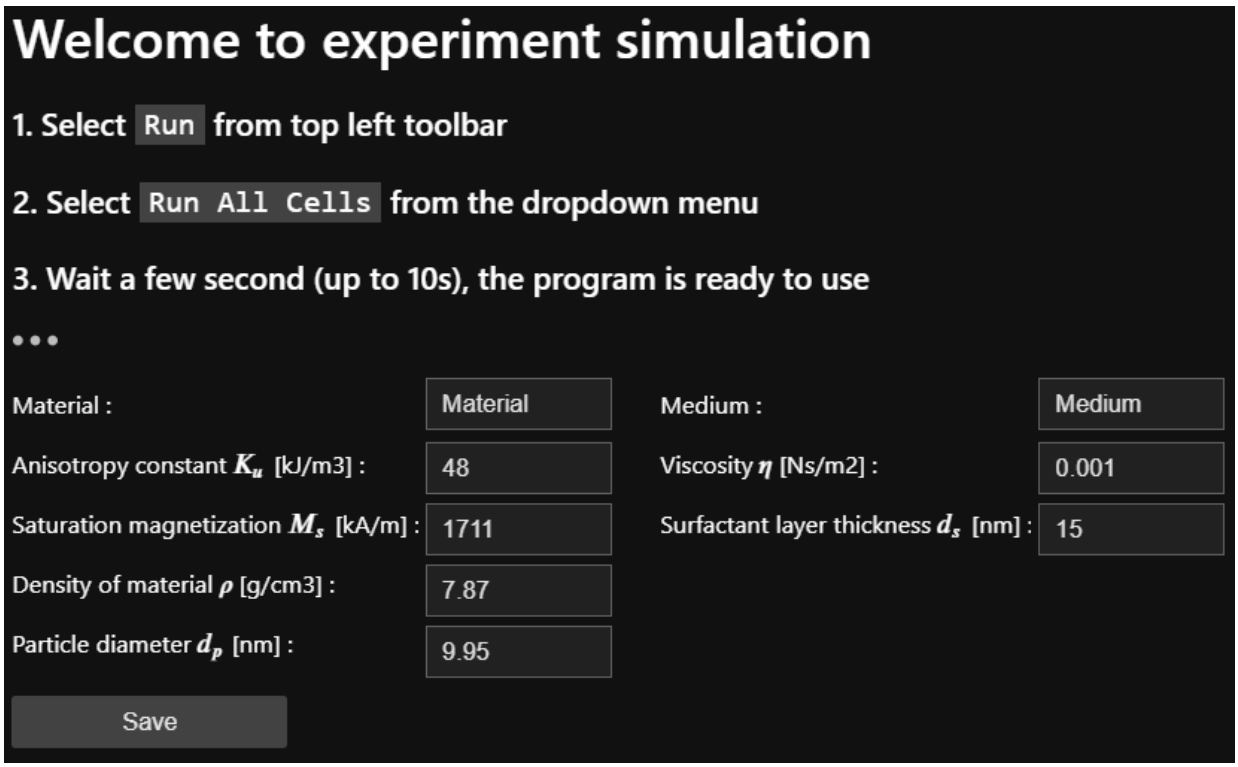

Figure 13: default home view of experiment_simulation.ipynb after "Run All Cells". Hidden ⋯ is `%run expsim.ipynb`.

In these files, following graphs can be plotted:

- Frequency – SAR $(f - SAR)$
- Field – SAR $(H - SAR)$
- Frequency - Field – SAR $(f - H - SAR)$

These are the graphs that are relevant to an actual experiments (other graphs are useful, but irrelevant for comparison with actual experimental results, for example $(d_p - \tau)$ and $(d_p - SAR)$, as in real-life, not all d_p are available.

Under each plot in $(f - SAR)$ and $(H - SAR)$, there is an IntSlider. Both IntSliders start from 10 and end at 1000 (kHz for frequency and G for Field), Figure 14.

Figure 14: (left) IntSlider for frequency and (right) IntSlider for field.

User can plot 5 frequencies or 5 field accordingly at once. All 5 input fields can be filled, but do not have to. By leaving the input field with default value of 0, the program will recognize it as empty and therefore will be ignored, Figure 15.

Figure 15: input fields of field and frequency.

There is a biological limit at how much field and or frequency can be applied before eddy current is triggered and therefore, heating the surrounding organ, causing major discomfort for patient when the treatment last for more than one hour [10]. According to Equation (33), if the product of Hf exceed the Atkinson Limit, there will be a change in marker from o to x, Figure 16.

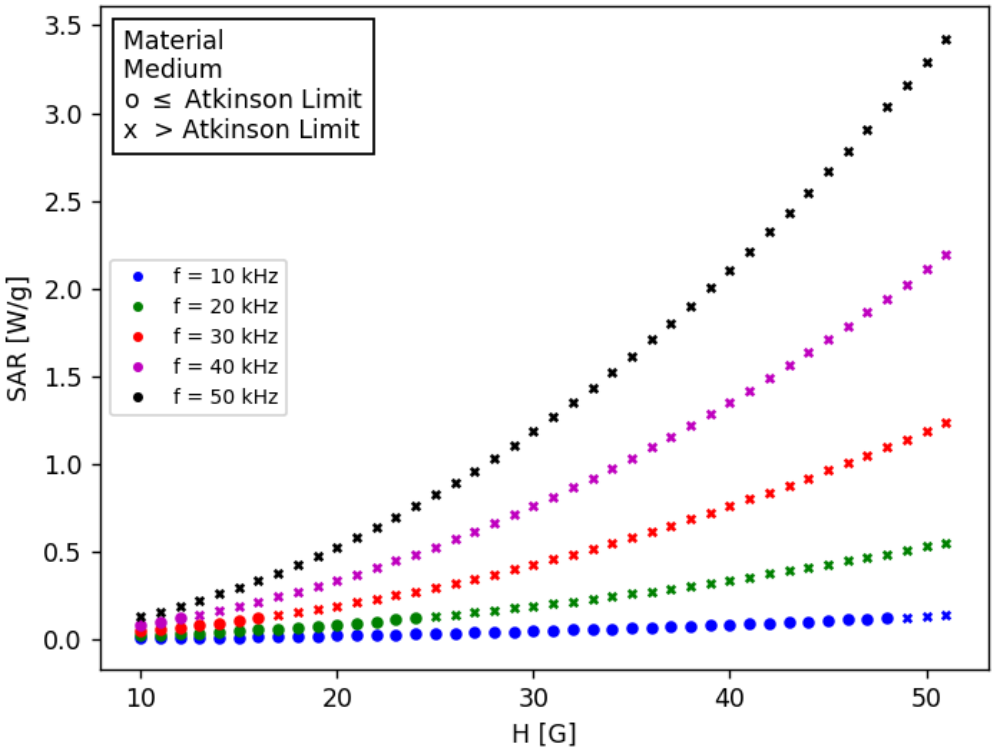

Figure 16: $(H - SAR)$ plot with various frequencies.

The export function for $(H - SAR)$ and $(f - SAR)$ will give a full range of H and f respectively, independent of the value in the IntSlider.

The $(f - H - SAR)$ 3D plot in these files is similar to the 3D $(f - H - SAR)$ plot from `userfriendly_quickplot`. The difference is that in `experiment_simulation`, the influence of medium is taken into account, while in `userfriendly_quickplot`, high viscosity medium is assumed, thus excluding the heat generation mechanism from Brownian relaxation. In `experiment_simulation`, the influence of medium cannot be neglected, because during an actual experiment, water and hexane are often used. These fluids have low viscosities.

3.3.5 list_of_csv.txt

This simple text file helps user to organize their exported .csv data. As the number of exported files are growing, it becomes more difficult to organize and recognize which data is what graph from which file

in which condition. In order to avoid confusion, a text file keeps track of all exported files (write an extra row in list_of_csv.txt, every time the export button is clicked), Figure 17.

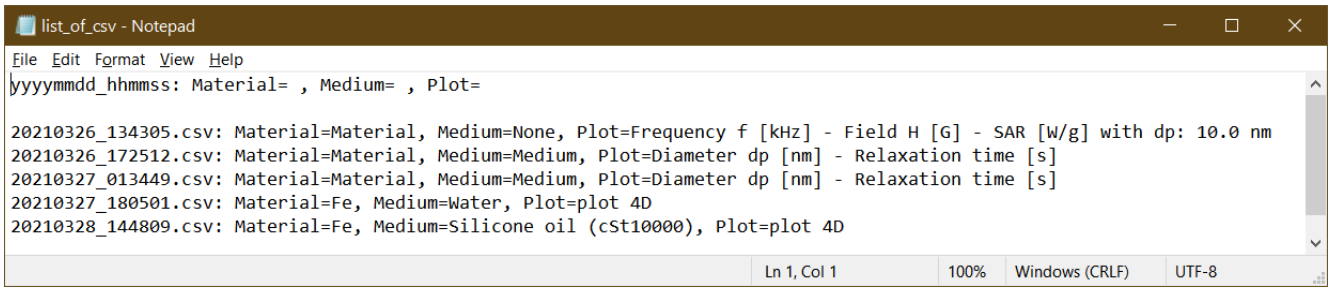

Figure 17: simple list to keep track of exported files.

The first row shows the format how each row will be saved. When `Medium=None` is written, means that the medium is neglected.

4 Errors and Warning

4.1 name '...' is not defined

At some cases user could get this error, Figure 18.

```
NameError: name 'pd' is not defined
```

Figure 18: variable or import not found.

This type of error can only occur in the non-user-friendly file. When an import is not found, it indicates that either the python libraries is not yet imported, or the kernel was interrupted and has to be rerun. If variable goes missing, the kernel is either interrupted or the variable does not exist in the above code. In this case, it is useful to check if there are any typos.

4.2 RuntimeWarning

RuntimeWarning is not an actual error that stops the program from running. It is an indication (warning) from matplotlib, that currently more than 20 figures are opened at once, Figure 19.

```
C:\Users\karin\anaconda3\lib\site-packages\ipykernel_launcher.py:25: RuntimeWarning: More than
20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.
igure`) are retained until explicitly closed and may consume too much memory. (To control this
warning, see the rcParam `figure.max_open_warning`).
```

Figure 19: matplotlib RuntimeWarning.

The warning serves only as a reminder, that there are too many opened figures. Whether it might cause problem depends on the RAM of user's computer. This warning can as well as be ignored if user think that no problem would occur and that the computer could handle it. If this is the case, the warning can be omitted once and for all with the following command written anywhere in the command cell:

```
plt.rcParams.update({'figure.max_open_warning': 0})
```

All figures are closed and the Figure number is reset when `import matplotlib.pyplot as plt` is called. This can be done by simply calling "Run All Cells" from the toolbar menu (in user-friendly files) or explicitly rerun the cell that contains the said import (in non-user-friendly files).

```
C:\Users\karin\anaconda3\lib\site-packages\ipykernel_launcher.py:6: RuntimeWarning:
overflow encountered in exp
```

Figure 20: RuntimeWarning infinity.

Figure 20 shows what happen if the result of any calculation reached infinity (threshold value set by python, 1.8e308), and this is encountered in the exponential function. This problem mostly occurs if user chose particle diameter that are too large. Choose a smaller particle diameter and the warning will be gone.

4.3 IOPub data rate exceeded

Loop is used for iterating over a sequence of command. By doing too many iterations, the limit of the data rate used could be exceeded, thus, creating an error, Figure 21.

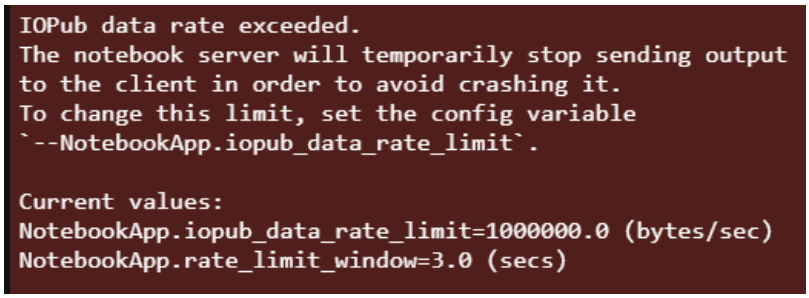

Figure 21: IOPub data rate exceeded.

Different from RuntimeWarning, this is an error that stop the program from running. User can decide, if the data rate limit is to be increased, or the iteration steps is to be decreased. To increase the data rate limit, following steps are to be done:

1. Open Anaconda Prompt.
2. Navigate to intended directory.
3. Type: `jupyter notebook --NotebookApp.iopub_data_rate_limit=1.0e10` and press enter
4. Close all related program (JupyterLab browser, Anaconda Navigator, Anaconda Prompt, …)
5. Open program again.

4.4 ModuleNotFoundError

Installation of modules and libraries via pip installer could fail for various reason, e.g. no permission, wrong version, etc. This could be easily missed, because anaconda prompt is doing a continuous installation without needing user input. The error report will be visible once the cell in the JupyterLab is run, Figure 22.

Figure 22: `ipyml` is not properly installed via pip installer.

In this case, the installation of `ipyml` should be done directly in the JupyterLab. Anywhere in the cell the following commands are to be called:

```
import sys
!{sys.executable} -m pip install --user ipympl
```

Another module that is common to be missed during the installation via pip is `nodejs`. In this case, `nodejs` can be directly installed to the user computer. The installer can be downloaded from `nodejs` official website: `https://nodejs.org/en/download/`

After installing `ipyml` and `nodejs`, the computer(!) must be restarted to finish installation. Restarting the JupyterLab or the Anaconda Navigator is not enough.

5 Outlook

5.1 Potential future improvement

5.1.1 Technical improvement

- The unit of each user input is predetermined. An option to choose the unit could be added as a dropdown list next to the input field.
- When calculating SAR, τ_B is sometimes ignored, sometimes not. Instead of having a fixed predetermined calculation step, a checkbox could be added to address this issue.

5.1.2 Calculation (approximation) improvement

- The parameter τ_0 is an approximated constant and is set to be 10^{-9} s. Instead of forcing user to have a fixed constant value of τ_0 for all calculation, this can be added as a user input.
- Magnetic volume fraction can be added as a parameter.
- Hysteresis loss is assumed to be absent for superparamagnetic nanoparticle, while in reality, this is not the case. Hysteresis loss could be calculated separately and at the end added to the calculation as a correction factor.
- Langevin equation in the Brownian motion is ignored. This issue could be addressed to improve the calculation.
- The density ρ always assumes the density of the magnetic nanoparticle, neglecting the fact, that the fluid actually has a part in the density, thus deviating from solely the density of the nanoparticle. Together with magnetic volume fraction, the real density can be calculated.

5.2 Unsolved technical problems

- The buttons available for single plot 2D $(f - SAR_{max})$ and $(H - SAR_{max})$, Figure 7, is generated manually. While it is possible to change the value from `libraries`, the number of field and or frequency cannot the increased or decreased.
- There is no export function for 2D multi-plot $(d_p - \tau)$ and 3D compare $(f - SAR - d_p)$ plot. The values are plotted without being saved. Because no values are saved, there are no data that can be exported.
- The database is saved as an `.ipynb` files. This is a little bit impractical and non-universal way of storing database. A common filetype to write and store database is .csv. A problem that arises while importing .csv files into `.ipynb` files is the local directory of the .csv files themselves. As the directory of the original .csv files must be given when importing, this is not possible without using cloud-based file. TU Darmstadt GitLab instance could be the solution of this problem.

6 Reference

[1] https://pandas.pydata.org/ (retrieved: March 29, 2021)

[2] https://numpy.org/ (retrieved: March 29, 2021)

[3] https://matplotlib.org/ (retrieved: March 29, 2021)

[4] https://anaconda.org/anaconda/ipywidgets (retrieved: March 29, 2021)

[5] https://ipywidgets.readthedocs.io/en/stable/ (retrieved: March 29, 2021)

[6] https://docs.python.org/3/library/time.html (retrieved: March 29, 2021)

[7] W.F. Brown, *J. Appl. Phys.,* **1959**, 30, 130

[8] F.H. MacDougall, J. Frenkel., J. *Phys. Colloid Chem*., **1947**, 51, 1032.

[9] J. Carrey, B. Mehdaoui, M. Respaud, *Journal of Applied Physics*, **2011**, *109*, 83921

[10] W. J. Atkinson, I. A. Brezovich, D. P. Chakraborty, *IEEE Transactions on Biomedical Engineering*, **1984**, 31, 70.

[11] A. Jordan, K. Maier-Hauff, et. al., *Onkologe,* **2007**, 13, 894.

Table of Figures

List of abbreviation

d_p	particle diameter
d_s	surfactant layer thickness
η	viscosity
f	frequency
H	field
ipynb	ipython notebook
k_B	Boltzmann constant
K_u	anisotropy constant
MFH	Magnetic Fluid Hyperthermia
M_s	saturation magnetization
P	power loss density
pip	preferred installer program / pip installs packages / pip installs python
ρ	density
SAR	Specific Absorption Rate
τ	relaxation time
T	temperature