4th year Introduction to Research Project

Modelling and sizing of Proton Exchange Membrane Fuel Cells (PEMFC)

Scientific report

(Paul Aubert) *Abroad during S8*
Félix Bonnes
Mathis De Malvin De Montazet
Justin Lombard

4th year Mechanical Engineering/ME
14th May 2023

Tutor: Ion Hazyuk

Abstract

Among all the technologies that currently exist, Proton Exchange Membrane Fuel Cell (PEMFC) has the best characteristics for transportation purposes. As PEMFCs are very complex systems, the architecture can be chosen and optimized for different working conditions depending on the usage and environment constraints. As PEMFCs are not yet widely used, very few models are currently available on sizing software. Mainly for educational purposes, we developed a PEMFC dimensioning script, to predict optimised performance and configuration of a stack, and a simulation model to provide a means of manipulating the different configurations of the entire system. We elaborated a complete analytical model from each component governing equations and electrochemical laws. The python implementation as well as the simulation model consisted in an ordered assembly of these sub-system equations. By making some simple assumptions, we were able to reduce the computational requirements of the model, making it suitable for simulating an entire fuel-cell stack as part of an energy system. The model has been tested and emulates experimental data well across the current density range. We have obtained a user-friendly simulation tool that allows for the exploration of the influence of each specific system component.

Keywords:

- PEMFC
- Pre-dimensioning tool
- Hydrogen fuel cell
- Hydrogen reduction with oxygen
- V-I diagram
- Electrochemistry
- Modelica library

Introduction

In the frame of a global energy transition, the development of new technologies in the field of power systems and electrification is mandatory. A very promising technology is the fuel cell. Fuel cells produce electricity with an easily storable fuel, like hydrogen, methanol, ect. Such a system is designed to be integrated in transport and to assist or even replace batteries which are not adapted for a massive electrification in this field. Fuel cell systems are complex and made of different sub-systems. The main sub-system is the fuel cell stack. A chemical reaction will take place in this component. To obtain the best efficiency, the reactants must be in very specific thermodynamic conditions. Then, a comprehensive system is an architecture composed by compressors, heat exchangers, turbines, pressure regulators, ect.

For the conception of such complex systems, we used a "V" conception cycle, as depicted in the figure below:

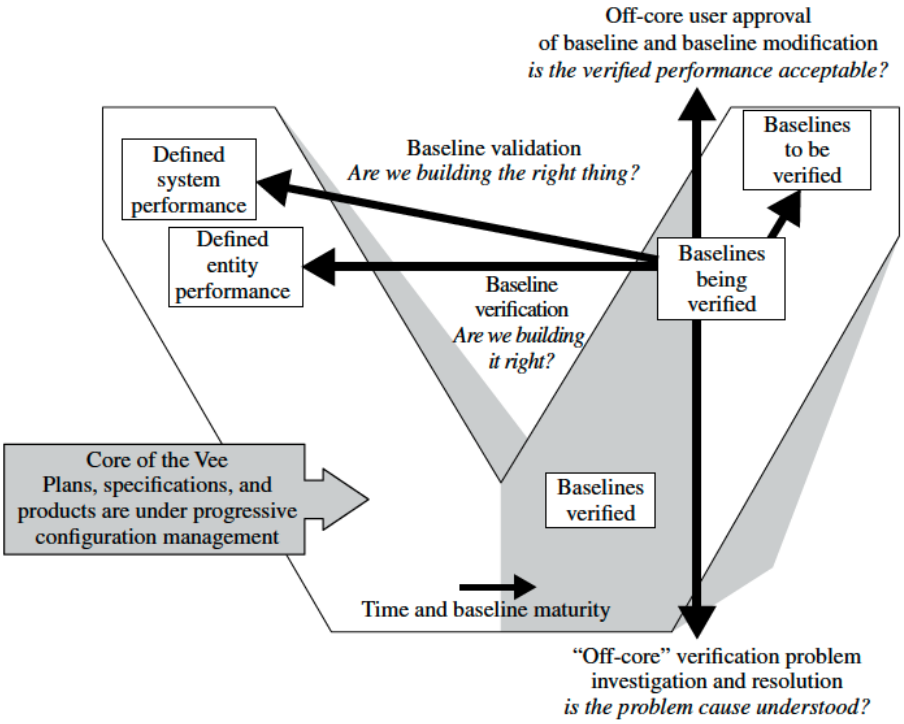

Fig.1: The "V" conception cycle [1]

On the left side of the "V", the conceptor aims to size a system which corresponds to the specifications of the customer, going more and more in the details, until reaching the bottom of the "V". Then, the conceptor needs to verify the designed system: does it reach the initial targeted performances? Do the different subsystems work well together? This second part of the conception cycle aims to perform different simulations on the system, to test it and to validate its specifications.

The purpose of our project is to build tools in order to carry out such "V" conception cycles, and make them accessible and understandable for educational cases of use.

This report describes the work done as part of this project led by Dr Ion Hazyuk at Institut Clement Ader. The project was divided in two distinct parts, led independently:

- Develop a sizing tool on Python for Proton Exchange Membrane Fuel Cell (PEMFC)
- Create a library of components on a multiphysical simulation software for the modelling and simulation of PEMFC systems

For the sizing tool, we will detail the implementation, in the code, of the equations found during the literature review. We will also provide the first results obtained, and discuss their relevance by comparing them with already existing systems. Concerning the simulation tool, we will present an overview of the components chosen or coded to build the library. We won't be able to provide results as we have not finished this aspect of the research project yet.

Sizing tool

1. Introduction to the sizing tool

As the two aspects of the project are led independently, this paper will firstly report the work done for the sizing tool before moving onto the simulation tool.

This tool aims to dimension the PEMFC stack only, it does not deal with the different subsystems needed to make the stack work well. Thus, we suppose here that the surrounding sub-systems are perfectly tuned to feed the PEMFC stack as it is designed in the sizing tool. The work achieved in this part led to a Python code implemented in *Jupyter Notebook* environment. We chose this option to add markdown text between our lines of code to physically explain and make it clearer.

1.1. Input and output parameters

This code addresses the left part of the "V" conception cycle, then the input parameters are the specifications we can find in a set of requirements from a customer, who builds a car or a plane for example. The output parameters are the geometrical characteristics of the stack, the operating characteristics of the designed stack and the theoretical fuel consumption. The parameters are detailed in the table below:

Input parameters		**Output parameters**	
Name	*Type/Unit*	*Name*	*Type/Unit*
Power mission profile (function of time)	Time and Power vectors [s], [W]	Number of cells in series	Scalar [-]
Required stack voltage	Scalar [V]	Cell area	Scalar [cm^2]
Hydrogen inlet pressure	Scalar [bar]	Stack mass	Scalar [kg]
Air inlet pressure	Scalar [bar]	**Specifications at mean power**	
Stack temperature	Scalar [°C]	Stack operating voltage	Scalar [V]
		Stack operating current	Scalar [A]
		Stack operating current density	Scalar [A/cm^2]
		Load of the stack	Scalar [% of max power]
		Mean efficiency	Scalar [%]
		H_2, O_2 consumption H_2O production	Scalars [kg/h]

Table 1: Input and output parameters

Notes:

- State-of-the-art PEMFC systems use the same hydrogen and air pressure to avoid internal stresses
- We assume the entire stack to be at the same temperature, as we use 0D model and equations

1.2. The different versions of the code

We have currently implemented 5 different versions. The current working version is 4.1.2 We will provide an overview of the 5 finished versions.

1.2.1. Version 1

The first version aimed at implementing the basis equations for plotting the V-I diagram.
The implemented equations were the following ones:

- Nernst equation
- Activation losses (bugging)
- Ohmic losses
- Mass transport losses (bugging)

1.2.2. Version 2

The current program improves the following points compared to V1:

- Better explanation of the comments
- Actualization of the mass transport losses constants
- Calculation of the stack power and efficiency function of the current density
- Better V-I plot including power and efficiency curves
- Calculation of the design points: maximum power and best efficiency

1.2.3. Version 3

The current version aims at improving/implementing the following parts compared to V2:

- Implement some volume and mass estimations for a state-of-the-art PEMFC
- Finalize the V-I diagram and validate it
- Propose another performance definition based on a cost function including a weighted stack mass
- Propose minimum volume and mass design points

1.2.4. Version 4

This 4th version aims at implementing a "mission profile" input:

- Add a mission profile input for power with a fixed voltage
- Add experimental data from manufacturers on the theoretical V-I curve
- Find an accurate value for mass density of a PEMFC stack

1.2.5. Version 4.1.1 (latest)

This latest version brings a few improvements:

- Implementation of a standard mission profile
- Split V-I diagrams to make them easier to read

In the next section, we will focus on version 4.1.1.

2. Code structure explanation

In this section, we will detail each section of the Python code. The full code is available in the appendix section.

2.1. General structure

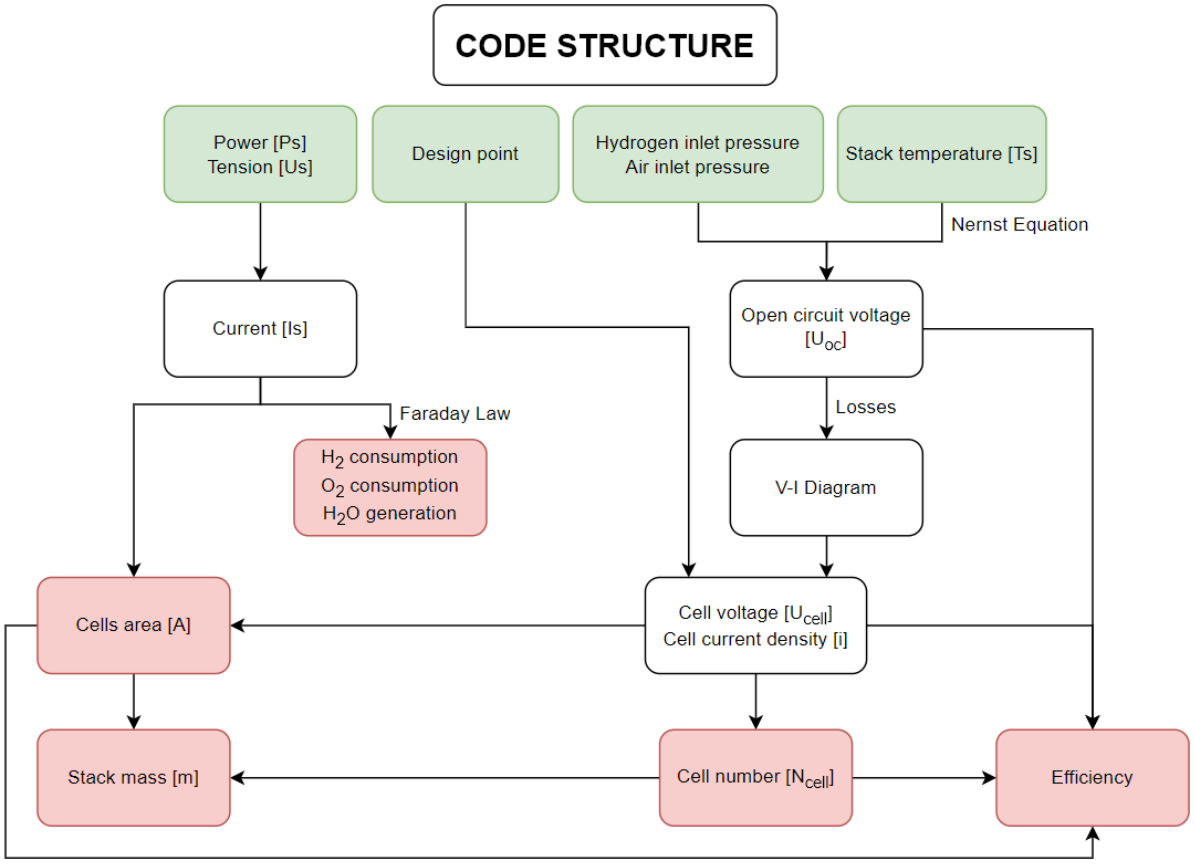

Fig.2: Flow chart of the sizing tool

We first designed a flowchart in order to set up the order of the code calculations. In fact, we preliminarily needed to check if there were no interdependencies. Therefore, as can be seen above, we placed in green the input parameters. It was needed to be able to calculate all the other values starting from them. In red, we can observe the resulting values we can use for sizing a PEMFC stack or controlling its quality.

The following parts of this chapter will attempt to clarify the content of this flow chart and the equations and methods used to go from one box to another.

2.2. Power mission profile

For the design of energy conversion systems, conceptors usually use a power mission profile, which is a representation over time of a typical mission for the designed system. This faithfully models the needs of a system and optimizes the design of the energy converter according to different criteria. Here is an extract of the mission profile presented in the *EU harmonised test protocols for PEMFC MEA testing in single cell configuration for automotive applications* [2] used in PEMFC pollution tests.

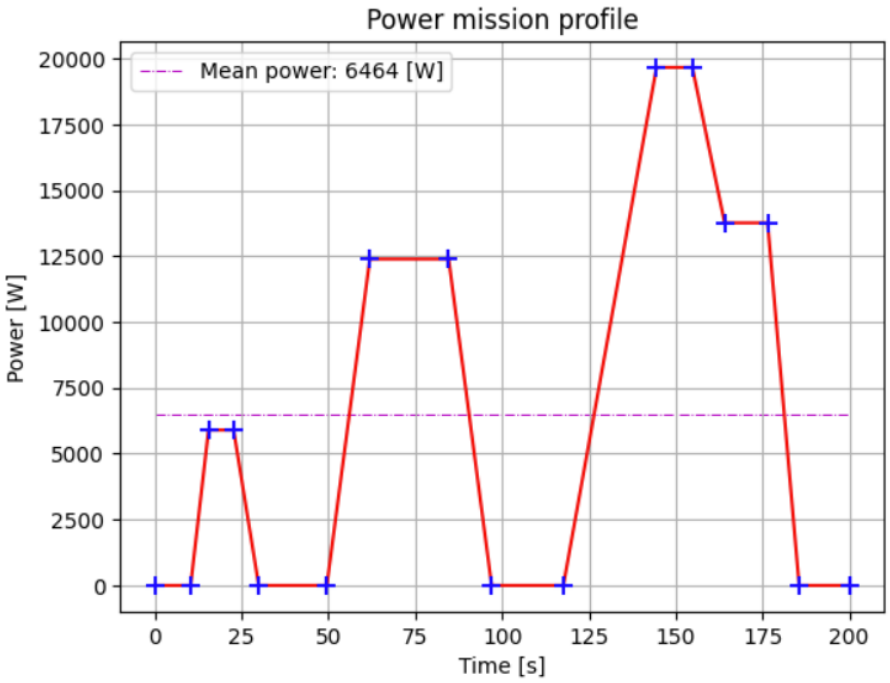

Fig.3: Extract of the EU harmonized mission profile

2.3. Performance function

To determine the design point, the user can define a performance function that will be maximized during the dimensioning process. This function can use the following variables:

- Efficiency of the stack
- Stack mass
- Stack power

The user can add coefficients to the variables, to obtain a performance function with this shape:

$$F_{perf} = \varepsilon - K_m \cdot m - K_p \cdot P$$

2.4. Equations and fixed parameters

We will now detail the equations used to model the behaviour of the PEMFC stack as well as the parameters taken from state-of-the-art literature.

2.4.1. Open circuit voltage (Nernst equation)

This is the optimal (theoretical maximum) voltage reachable computed with the Nernst equation. This voltage would be obtained without losses.

$$U_{OC} = E_{rev} + \frac{RT_s}{2F} \cdot ln\left[p_{H2} \cdot \sqrt{p_{O2}}\right]$$

$$E_{rev} = E_{rev}^{0} + \left(T_s - T_{ref}\right) \cdot \frac{\Delta S_0}{nF}$$

With:
- R: universal gas constant ($8.3144\ J \cdot mol^{-1} \cdot K^{-1}$)
- $\frac{\Delta S_0}{nF}$: standard state entropy change ($-0.85 \cdot 10^{-3}\ V \cdot K^{-1}$: [2] page 5)
- F: Faraday constant ($96485\ C \cdot mol^{-1}$)
- E_{rev}: reversible voltage (V)
- p_{H2}: hydrogen partial pressure (atm)
- p_{O2}: oxygen partial pressure (atm)
- E^0_{rev}: standard state reference potential ($1.229\ V$: [2] page 5)
- T_{ref}: standard state temperature ($298.15\ K$)

2.4.2. Activation losses

Some voltage difference from equilibrium, called overpotential, is needed to get the electrochemical reaction going. In a hydrogen/oxygen fuel cell, the oxygen reduction requires much higher overpotential. Thus, we will focus only on the cathode side. State-of-the-art cathodes are covered by platinum, we then give appropriate values for this case of study.

$$U_{act,c} = \frac{RT_s}{\alpha_c F} \cdot \log\left(i + i_{0,c}\right)$$

$$i_{0,c} = i_{0,c}^{ref} \cdot a_c \cdot L_c \cdot \left(\frac{p_{O2}}{p_{O2}^{ref}}\right)^{\gamma} \cdot \exp\left[-\frac{E_c}{RT_s} \cdot \left(1 - \frac{T_s}{T_{ref}}\right)\right]$$

With:
- α_c : transfer coefficient ($0.65\ [0.5 - 0.7]$: [3] abstract)
- $i_{0,c}$: exchange current density on cathode side ($A \cdot m^{-2}$)
- $i_{0,c}^{ref}$: reference exchange current density ($1 \cdot 10^{-3} A \cdot m^{-1}$: [4])
- a_c : catalyst specific area ($800\ [600 - 1000]\ cm^2 \cdot mg^{-1}$: [6] page 18)
- L_c : catalyst loading ($0.3\ [0.3 - 0.5]\ mg \cdot cm^{-2}$: [6] page 18)
- γ : pressure dependency coefficient (0.5: [6] page 18)
- E_c : activation energy for oxygen reduction ($66 \cdot 10^3\ J \cdot mol^{-1}$: [5] page 18)

2.4.3. Mass transport losses

This loss is due to a diffusion process. H_2 and O_2 must diffuse through the electrodes to react. The consumption of reactants on the catalyst surface results in the creation of a concentration gradient through the diffusion layer (in our case the thickness of the electrode). This consumption can reach a maximum value (when reactant concentration on the catalyst layer reaches 0) and corresponds to a limit current that a fuel cell will never be able to overtake. Similarly to the activation losses, we can neglect the mass transport losses on the anode (H^2) side because the limit current on the cathode side will be much lower than on the anode side. This is because air is used (rather than pure oxygen) which makes O^2 diffusion slower than H^2 [7].

$$U_m = \frac{RT_s}{nF} \cdot \ln\left(\frac{i_{lim}}{i_{lim} - i}\right)$$

$$i_{lim} = \frac{nFDC_{O2}}{\delta}$$

With: $\begin{cases} i_{lim} : \text{limit current density } (A \cdot m^{-2}) \\ D : \text{diffusion coefficient of cathode GDL } (5 \cdot 10^{-3}\ m^2 \cdot s^{-1}) \\ C_{O2} : \text{concentration of } O_2 \text{ on cathode side } (mol.m^{-3}) \\ \delta : \text{diffusion layer thickness } (100\ \in [100 - 400]\ \mu m\text{: [7]}) \end{cases}$ [7]

Note: The value of the diffusion coefficient was adjusted to get a value of i_{lim} close to 2 A/cm^2 as it is a common value for state-of-the-art PEMFCs and we were not able to find a value for D in literature. This value can be modified by the user if needed for its own applications.

2.4.4. Ohmic losses

Ohmic losses occur because of resistance to the flow of ions in the electrolyte and resistance to the flow of electrons through the electrically conductive fuel cell components. These losses can be simply expressed by Ohm's law:

$$U_\Omega = i \cdot R_i$$

$$R_i = R_{i,i} + R_{i,e} + R_{i,c}$$

With:
$\begin{cases} R_i\text{: total cell internal resistance } (0.14 \in\ [0.1 - 0.2] \cdot 10^{-3}\ \Omega \cdot m^{-2}\text{: [5] page 20}) \\ R_{i,i}\text{: ionic resistance } (\Omega \cdot m^{-2}) \\ R_{i,e}\text{: electronic resistance } (\Omega \cdot m^{-2}) \\ R_{i,c}\text{: contact resistance } (\Omega \cdot m^{-2}) \end{cases}$ [5] page 20$R_{i,i}$: ionic resistan
$m^{-2})R_{i,e}$: electronic resistance $(\Omega \cdot m^{-2})R_{i,c}$: contact resistance $(\Omega \cdot m^{-2})$

Electronic resistance is almost negligible. Ionic resistance mostly depends on the state of hydration of the polymer membrane. Contact resistance depends on the materials used for GDL and bipolar plates. Typical values for R_i in well-designed fuel cells are between <u>0.1 and 0.2 $\Omega.cm^{-2}$</u>.

There are other models available to compute this resistance more precisely in [9], [10], using models to evaluate the conductivity of the different components. We could also capture more precisely the impact of the membrane hydration on the ionic resistance if needed. For a pre-dimensioning study, the range above is sufficient.

2.5. V-I diagram

2.5.1. Expression

The V-I diagram is the most important data for a conceptor to design a PEMFC stack. It gives the curve relating the voltage of the stack with the current density in it. Similarly to a chemical battery, the voltage drops when the current density increases due to the losses detailed above.
The mathematical expression of the V-I curve is:

$$U_{cell} = U_{OC} - U_{act}(i) - U_m(i) - U_\Omega(i)$$

The shape of the curve obtained is shown on the figure below:

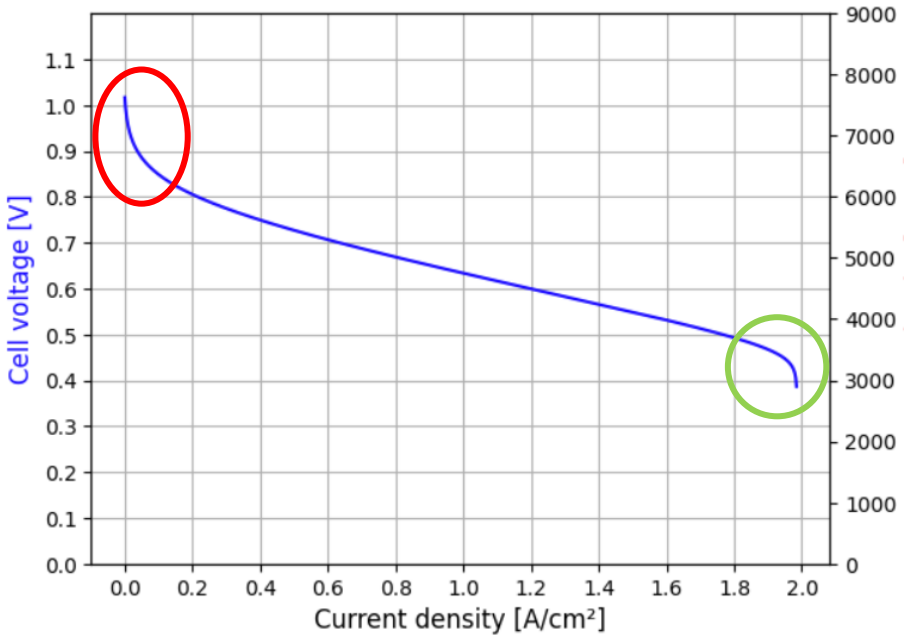

Fig.4: Shape of a V-I diagram

On this figure, we can notice 3 main zones:

- For low current densities (red enclosed area), we observe the effects of activation losses
- The linear middle area is due to ohmic losses
- For high current densities (green enclosed area), we observe a sharp drop due to diffusion limitations

A PEMFC stack operates necessarily on a point of this curve. The conceptor then aims to choose the best point to design its system.

2.5.2. Definition of the design points

In the code, we propose several design points. For their computation, we use the efficiency of the fuel cell, which is designed as follows:

$$\varepsilon = \frac{U_{cell}}{U_{OC}}$$

A first design point is obtained by maximizing the performance function defined in section 2.3.

We can also size the stack at the point giving the best ratio efficiency/mass, which is an alternative way to the performance function of comparing the efficiency and the mass.

The last implemented design point is the max power design point (which is also the lowest mass). This would result in a 100% load of the stack.

The distribution of the design points is shown on the figure below:

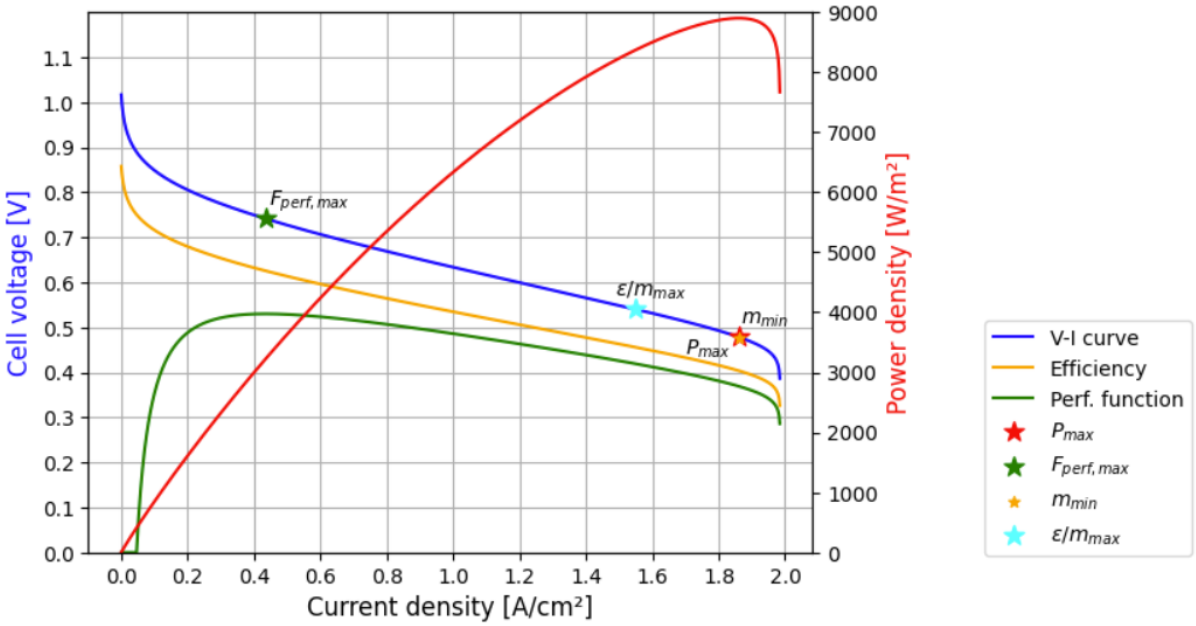

Fig.5: V-I diagram with design points

2.6. Mass estimation

In order to compute the design points, we need an estimation of the mass of the stack for each point on the V-I curve. We then studied the structure of a fuel cell, which is shown on the figures below:

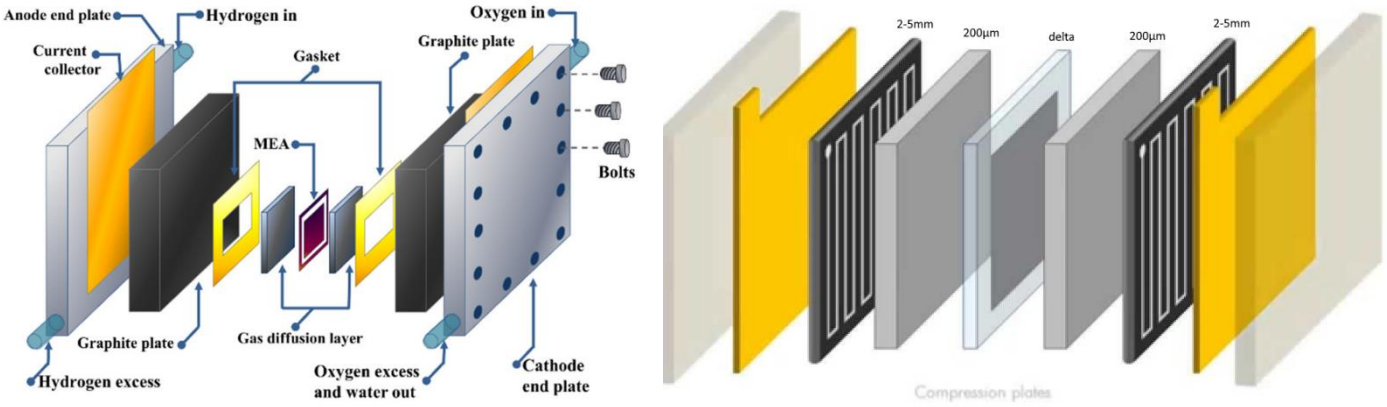

Fig.6.A-B: PEMFC structure and layers [11]

Component	Material	Thickness [m]	Density [kg/m3]
End plate	Aluminium	*0,0050*	*2800*
Bipolar plate	Graphite	*0,0020*	*1750*
Gasket	Silicon	*0,0002*	*2300*
GDL	Carbon Paper	*0,0001*	*600*
PEM	Nafion	*0,0001*	*1900*
GDL	Carbon Paper	0,0001	600
Gasket	Silicon	0,0002	2300
Bipolar plate	Graphite	0,002	1750
End plate	Aluminium	0,005	2800

We also summarized the thicknesses, materials and densities of each layer, shown in the table on the left. More information can be found in [12], [13]. The dimensions used to compute the mass are calculated using the methods detailed in the next section (cell area, number of cells).

Table 2: PEMFC materials and thicknesses

2.7. Computation of the results

2.7.1. Verification of the mission profile compatibility

The mission profile entered by the manufacturer represents an evolution of the power requested by the surrounding systems to the stack. Each power has to be reached, and this is why we must choose the sizing power for optimized operation of the PEMFC. For this, we start by considering the mean power regarding the complete profile. Then, we check whether we actually reach the maximum required power or not. If yes, then the stack operating at its chosen design point is already fulfilling the requirements, so the output values can be calculated. Unfortunately, most of the time the maximum power reachable is lower than the required one. In this case, we process iteratively:

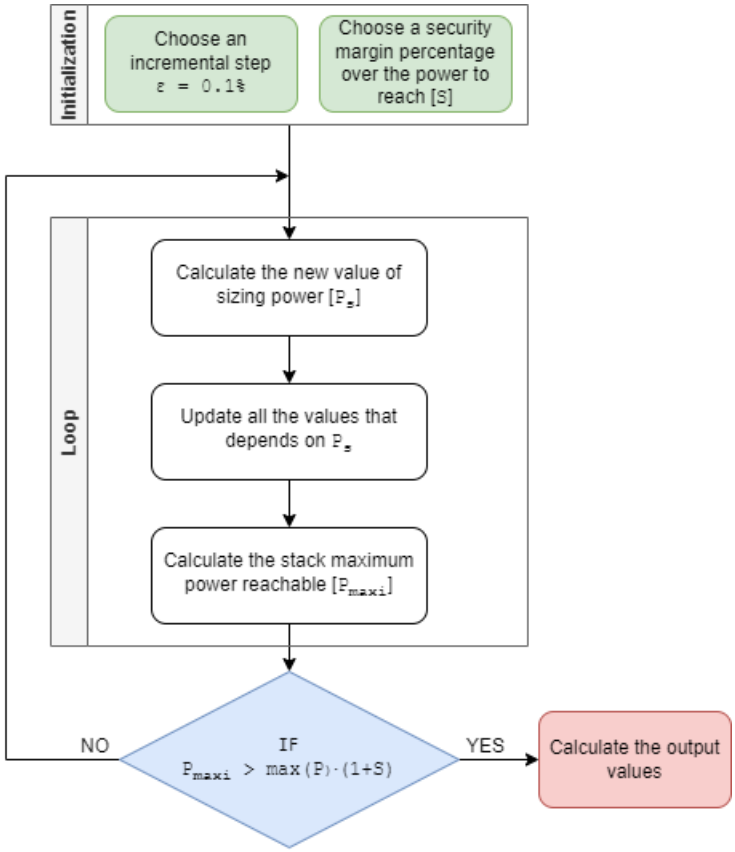

Fig.7: Structure of the code to optimize the sizing power

Using this process, the sizing power can be calculated as well as the output values. A real example will be detailed in section 3.

2.7.2. Outputs calculation

To conclude this part of the process explanation to size a PEM fuel cell, we need to calculate the output variables. These results are useful to a fuel cell manufacturer to optimize and fit its conception to the usage of their system.

The results are calculated as follows:

- Number of cells in series:

$$N_{cell} = \text{ceil}\left(\frac{U_s}{U}\right)$$

Where: $\begin{cases} U_s : \text{stack required voltage} \\ U : \text{cell voltage (from design point)} \end{cases}$

- Cell area:

$$A = \frac{I_s}{i}$$

Where: $\begin{cases} I_s : \text{stack required current} \\ i : \text{cell current (from design point)} \end{cases}$

- Total stack mass:

We now have our missing parameters to compute the stack mass. Note that the anode and cathode plates, also named "end plates", are not present in every cell. These two plates that compose the structure of the entire stack are only present in the beginning and at the very end of the stack. They include all the cells in series located in between them.

From the composition of a single cell and knowing the number of them (determined during sizing process), we can retrieve the complete composition of the stack. Finally, as we already calculated the cell area, we know the total volume of each material composing the PEM fuel cell. The global mass can then be estimated based on the volume and the density of all the materials.

$$m_{stack} = \sum_{\text{material}} V_{\text{material}} \cdot \mu_{\text{material}} \cdot n_{\text{material}}$$

Where: $\begin{cases} m_{stack} : \text{stack mass} \\ V_{\text{material}} = A \cdot t_{1\ \text{cell}} : \text{material volume in one cell} \\ \mu_{\text{material}} : \text{material density} \\ n_{\text{material}} : \text{number of occurrences of the material in the stack} \\ t_{1\ \text{cell}} : \text{thickness of the material plate in one cell} \end{cases}$

- Stack consumption and emission

$$\dot{N}_{cons}^{H_2} = N_{cell} \cdot \frac{\overline{I_s}}{n \cdot F} \cdot M_{H_2}$$

$$\dot{N}_{cons}^{O_2} = N_{cell} \cdot \frac{\overline{I_s}}{2n \cdot F} \cdot M_{O_2}$$

$$\dot{N}_{emiss}^{H_2O} = N_{cell} \cdot \frac{\overline{I_s}}{n \cdot F} \cdot M_{H_2O}$$

Where: $\begin{cases} \dot{N}_{cons}^{H_2} : \text{quantity of } H_2 \text{ consumed} \\ \dot{N}_{cons}^{O_2} : \text{quantity of } O_2 \text{ consumed} \\ \dot{N}_{emiss}^{H_2O} : \text{quantity of } H_2O \text{ emitted} \\ \overline{I_s} : \text{stack current as mean operating point} \\ N_{cell} : \text{number of cells} \\ n : \text{number of electrons exchanged in the redox reaction} \\ F = 96485\ [\text{C/mol}] : \text{Faraday's constant} \\ M_e : \text{molar mass of the specie } e \end{cases}$

- Stack load at operating point

At operating point, we can define the stack load as the ratio between the mean power of the mission profile and the maximum reachable power.

$$L_s = \frac{\overline{P_{\text{mission}}}}{P_{maxi}}$$

These are all the deciding parameters and information for the conception of a PEMFC stack that a manufacturer can manipulate depending on the usage and the chosen function of performance. This function has a great impact on how the stack is sized, and it is a useful tool when conceiving a fuel cell for very specific applications.

3. Study of a standard case

This section will focus on how we experienced with and tested developed sizing tool. This step is necessary to verify that the computed results are reasonable and fit with potential experimental data.

3.1. Standard pollution test

3.1.1. Fuel cell input parameters

PEMFC tends to become common in the transportation industry due to its high level efficiency, low operating temperature and high energy density. Therefore, the European Commission has tried to elaborate a harmonized test to qualify a vehicle-implemented PEM fuel cell. These operating conditions and initial parameters are found in [1].

As our sizing tool is now able to represent any configuration, we chose to test it with the standard pollution test presented by the European Commission. Here are the input parameters implemented to run the procedure:

The power mission profile is the one presented in *Fig.2*.
Stack voltage: $U_s = 800$ [V] (not mentioned in [1], but this value has been chosen for a common car example).
Air inlet pressure: 2.3 [bar]
Hydrogen inlet pressure: 2.3 [bar]

In our test, we will define the following performance function:

$$F_{perf} = \varepsilon - K_m \cdot m$$

Where: $\begin{cases} \varepsilon : \text{efficiency} \\ K_m : \text{mass impact coefficient} \\ m : \text{stack mass} \end{cases}$

We set the mass coefficient to $K_m = 10 \cdot 10^{-3}$. This value gives the mass a relatively average importance, typically for a road vehicle. In an aerospace application, we could increase the mass coefficient to have a better performance when the system weight is low. However, in a bus or boat application, the mass does not have a great of impact, so the K_m value can be decreased.

The following *Fig.8* shows the look of the performance function in our car pollution test.

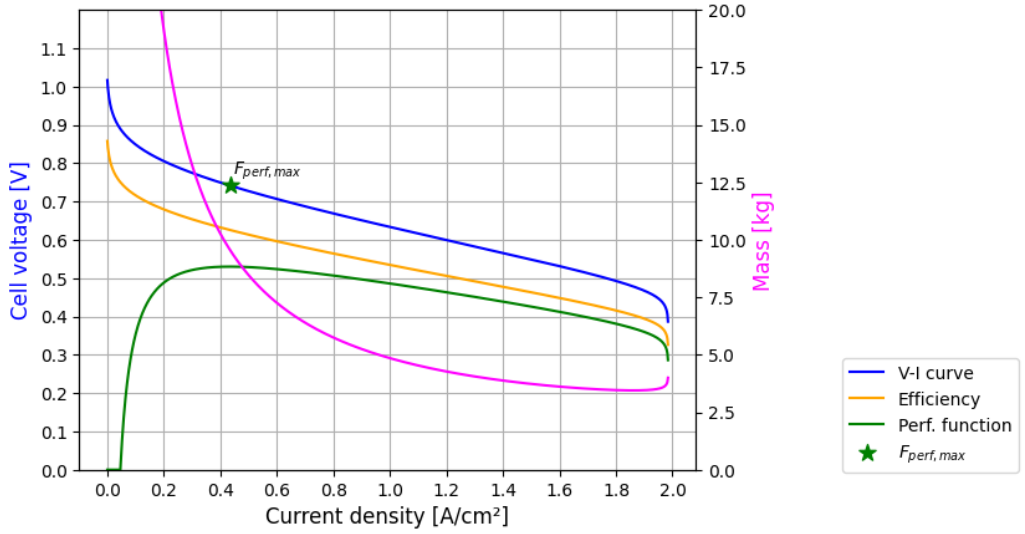

Fig.8: Performance function in the case of a car pollution test stack optimization

This function is supposed to give a performance overview of the PEMFC. It is used to choose a design point on the V-I curve.

3.1.2. PEM fuel cell sizing results

By running the optimization program, we have sized the fuel cell stack to the input requirements. In fact, by requesting the specific mission defined by all the parameters, we finally obtained the stack characteristics as a result. It can be seen on the following picture the initial power mission profile, from which the initial mean power was used at the first iteration to size the stack. In green, we can see the final sizing power to which the program converged. This sizing power allows the PEMFC to reach every power value of its mission, while remaining the most efficient at the operating point.

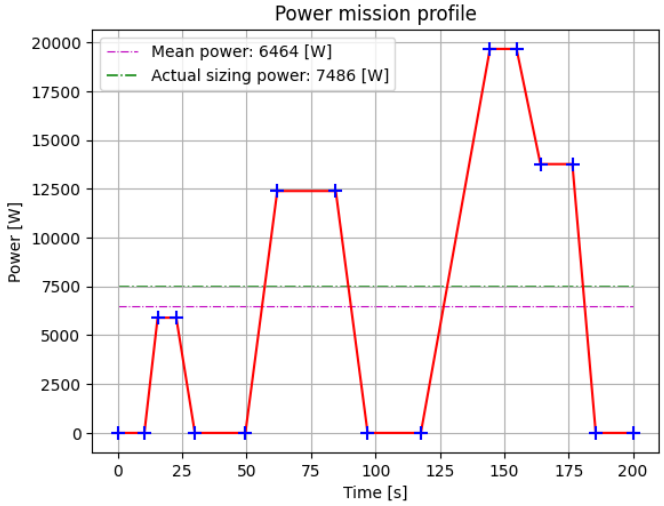

Fig.9: Power mission profile showing the sizing power after sizing process

The program is tuned to size the stack so that the maximum power to reach is 5% above the maximum power required by the power profile. Therefore, we can observe the following characteristics for the stack performance:

```
                  Sizing power:  7486.7  [W]
       Required maximum power: 19662.9  [W]
Stack maximum power available: 20700.0  [W]
```

Global specifications of the stack size to be optimized for the initial parameters:

```
Number of cells in series: 1080   [-]
               Cell area:   21.5 [cm²]
              Stack mass:   11.1 [kg]
```

Characteristics when operating at mean power:

```
        Stack operating voltage: 824.0  [V]
        Stack operating current:   8.0  [A]
Stack operating current density:   0.36 [A/cm²]
             Load of the stack:  31.23 [% of maximum power]
               Mean efficiency:  64.33 [%]
```

Stack fluid consumption and generation:

```
    H₂ consumed: 0.316 [kg/h]
    O₂ consumed: 2.530 [kg/h]
  H₂O generated: 2.847 [kg/h]
```

In the current configuration, the stack *only* consumes 0.313 [kg] of dihydrogen per operating hour. From this value, the autonomy of the vehicle can be calculated knowing the total hydrogen mass on board. For example, here, we can retrieve the range of the car designed to fit these test conditions:

The standard pollution test power mission profile is designed for a typical city car. Such a car holds around 6 [kg] of hydrogen and runs at a mean speed of 17.4 [km/h] in the standard pollution test specifications. Knowing the hydrogen consumption of 0.316 [kg/h], we can deduce the range of this typical car with optimized parameters calculated by the sizing script:

$$d = \frac{6}{0.316} \cdot 17.4 = 330 \text{ [km]}$$

This value can be compared to a typical electric car range, which would be around 200 to 400 [km] for city runs. The value calculated fits the general interval, which confirms the reasonability of the script.

3.2. Comparison with experimental data

Lastly, in order to check the correctness of our sizing tool, we can compare the V-I curve produced by our mathematical model to some experimental data. As we did not have any PEM fuel cell to do experiments, we found experimental data from the literature.

Fig.10 shows the V-I diagram extracted from the sizing results compared to three lots of experimental data from different sources with different initial parameters.

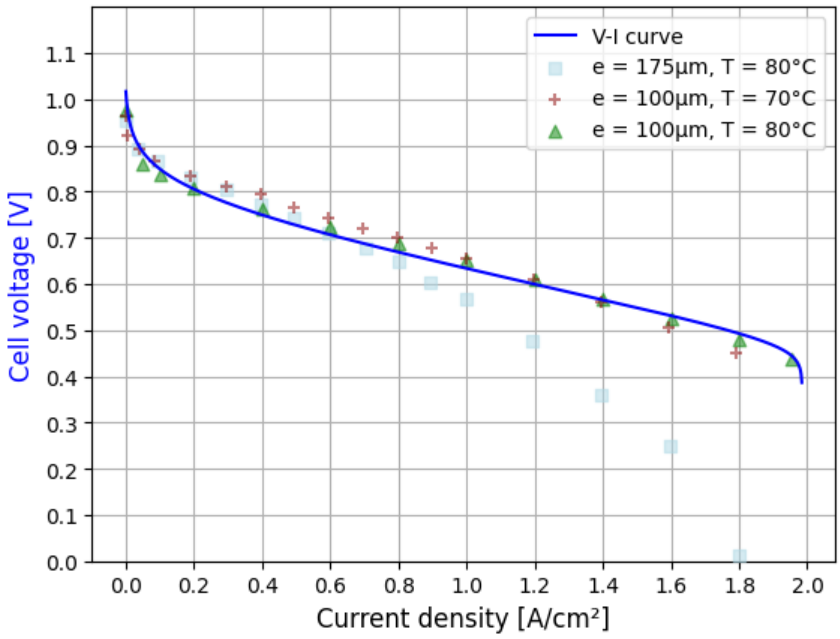

Fig.10: Comparison between theoretical V-I curve and experimental data

As we can see, the V-I curve we calculated correlates the best with the experimental data having the same initial parameters as our configuration. This *green triangle* data is extracted from [14] where the initial configuration was the following:

$T_s = 80$ °C
Membrane material: Nafion 117
$e_{nafion} = 100$ µm

The V-I curve sticks quite well to *brown crosses* data [15]. In fact, this data is extracted for experiments where the only modification regarding the data presented above is the operating temperature. We can observe that this small change has a negligeable impact in this order of magnitude.

Finally, we also plotted experimental data (*blue squares*) from a test [15] where the membrane was 175 µm thick, and it is obvious that this change has a great impact on the look of the V-I curve.

4. Review of the sizing script method and results

In this section, we explain the construction of the script we developed to size and optimize a proton exchange membrane fuel cell stack. This process of design can be requested by manufacturers to conceive PEMFC stacks and tailor them to their very specific application.

To optimize the characteristics of the stack, a few things are needed. The power mission profile or the mean, and maximum power as well as the stack working voltage are requested to fit the fuel cell to the operating conditions. After setting up a performance function that will determine the working point function of your specific application, it is only needed to set up the fluids inlet pressures, that depend on the surrounding compressor systems, and the stack temperature. The latter will be determined function of the stack environment, working conditions and neighbouring systems.

With these few input data, the optimizing process gives you the dimensions of the PEMFC stack and its mass. It also provides you the hydrogen and oxygen consumption rate per hour as well as the emission of water in kilograms per operating hour. The efficiency and the nominal stack load are also estimated.

Modelling tool

5. Introduction to the modelling tool

In the first part of this report, we discussed how we developed a sizing tool for the PEMFC only and we considered that the sub-systems around the PEMFC were perfectly tuned to allow optimal functioning of the PEMFC. In this section we will review the different subsystems that work around the PEMFC and how we started the modelling process using Dymola.

5.1. PEMFC subsystems

For the PEMFC to be functioning properly a series of components are necessary to condition and regulate the different parameters.

In the figure below we can see a typical PEMFC architecture used for transport applications.

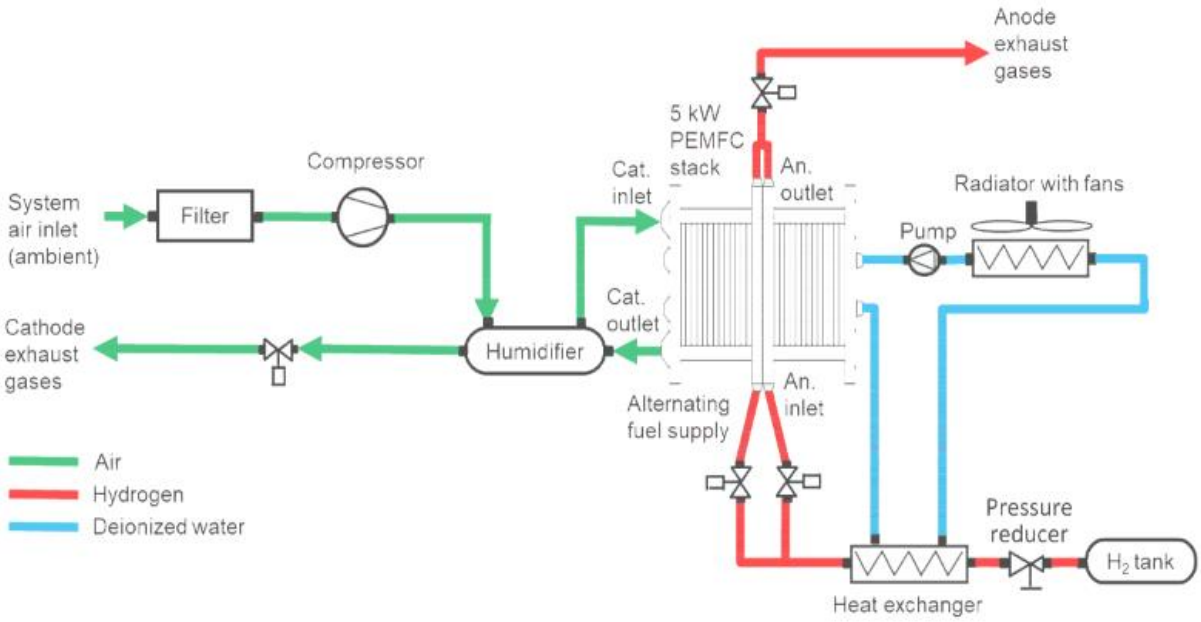

Fig.11: System layout - automobile

Here we can see three different circuits. In green the air circuit which consists of an air filter, a compressor, a humidifier, and a valve. These components are used to condition the ambient air to the appropriate pressure, temperature, and humidity before it enters the PEMFC. In red we can see the hydrogen circuit, which consists of a hydrogen source (the tank), a pressure valve and a heat exchanger. These components also condition the hydrogen by controlling the flow, phase, temperature, and pressure of the hydrogen before it enters the PEMFC. Finally in blue we have the temperature regulating circuit, which consists of deionized water, a radiator and fan and a heat exchanger. These components regulate the temperature of the PEMFC so that it can work in an optimized environment as well as heating up the hydrogen before it enters the PEMFC.

As you can see, for a PEMFC to work properly a lot of components need to work in harmony with one another to provide the perfect condition for peak performance.

5.2. Dymola

Modelling this type of architecture for simulation and test purposes is tedious and while it could be done using different types of softwares like Simulink, Matlab, and others, we have chosen to use a software called Dymola as it will give us more freedom in how we can configure the system down the road.

Dymola is a simulation software initially designed by Hilding Elmqvist in 1978 and stands for Dynamic Modelling Language. This software, now owned by Dassault System, is a simulation software tool used particularly for modelling and simulating complex dynamic systems. This software is based on the Modelica language which allows users to create complex dynamic models in the form of blocks and components. The Modelica language provides a set of libraries of pre-built components that can be used to create numerous models and simulations from simple mechanical systems to complex multi-domain applications. These components or blocks typically represent physical phenomena such as fluid flow, heat transfer, electrical circuit, and mechanical systems.

Once these blocks or components are coded and modelled, they can be assembled into different configurations based on the needs and wants of the user. Once the inputs and parameters are entered, Dymola then generates the equations that describe the behaviour of the system over time, which can then be analysed as tables and graphs.

Some of the benefits of Dymola and Open Modelica are that its innerworkings involve a combination of object-oriented modelling, numerical analysis, and Differential-Algebraic System Solver (DASSL) algorithms to solve these complex multi-domain applications. Another advantage of Dymola is that the components and model are coded using non directional equations. Meaning that the software itself will exploit and rearrange the given equations to fit the input parameters given, giving the users an infinity of possibilities to use different blocks and components in whichever way they need.

6. Phase one – Exploring and understanding the innerworkings of Dymola through the analysis of a pump system

Our first phase of the modelling process was to get acquainted with the software by using the available library to create models and run simulations. As later on we would be modelling components like compressors and valves which deal with fluids we decided to focus on fluidic components. We first started by looking at a pump system example.

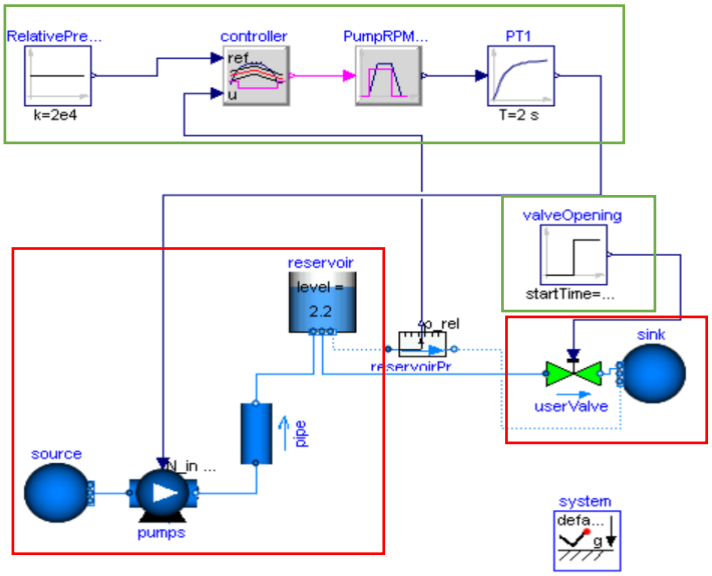

Fig.12: Dymola library example of a pump system

In the figure below you can see a pump system example taken directly from the Dymola library. This model consists of two different parts. The physical components in red and the control components in green.

6.1. The physical components

The physical components are as follows:

- Fluid sources
- Pump
- Pipe
- Reservoir
- Valve
- Sink

These components and blocks are the core of the model, as they contain equations that translate the physical properties of the fluid into equations. Each component has their own sets of equations that enable the software to understand how the fluid moves through the system and how its parameters such as its temperature, pressure, enthalpy, and phase change by going through the different blocks. One of the advantages of Dymola is that after a simulation a user is able to retrieve these sets of data as a function of time at each entering and exiting point of each of the different components.

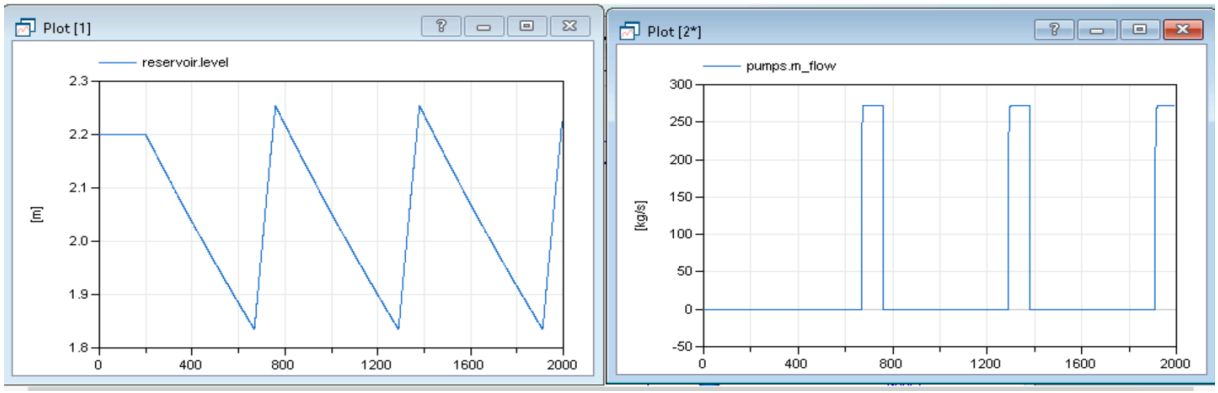

Fig.13: Data plots of a 2000 sec simulation of Dymola's pumping system

As we can see in these two plots after the simulation over a period of 2,000 seconds, when the simulation starts the water level in the reservoir is at 2.2 m high and the pump is off. However, when the simulation starts, and the reservoir level falls to below 1.83 m the pump turns on to refill the tank to about 2.22m high and then shuts off again. This analysis brought us to the next question of how we control the different components to make everything work together in harmony.

6.2. The control components

This is where we started to look at the second part of the model, the control part. The control part is made up of 6 components.

- Constant (Relative Pressure Set point)
- On/Off Controller
- Triggered trapezoid (Pump RPM Generator)
- First order block (PT1)
- Source Step (Valve Opening)
- Relative pressure sensor (Reservoir Pressure)

Like we mentioned, our first assumption about the control system was that the pump turned on once the reservoir level reached 1.83 m. However, when looking closely at the model we realised that the reservoir pressure sensor was the one giving the information to the control system to turn the pump on and off. To reinforce our thoughts, we plotted the reservoir pressure and the reservoir height and validated our thought process as the graphs are visually identical in relation to time.

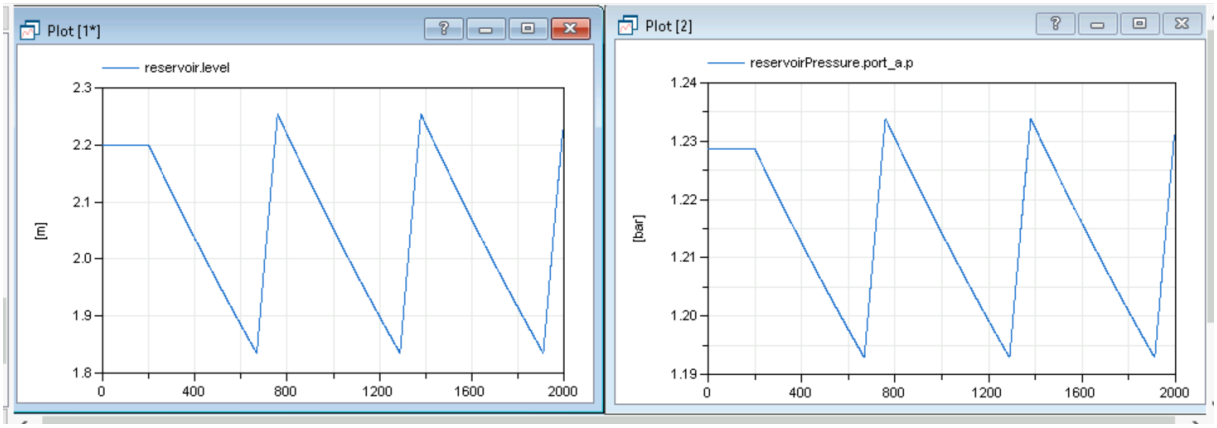

Fig.14: Data plots of a 2000 sec simulation of Dymola's pumping system

6.3. Creating our own test model

After getting acquainted with the pump system presented above, we then decided to try to create our own system to finalise our understanding of the general innerworkings of Dymola. For that we created the system below and did not bother to integrate a variable control system as it would not be necessary for the rest of our project.

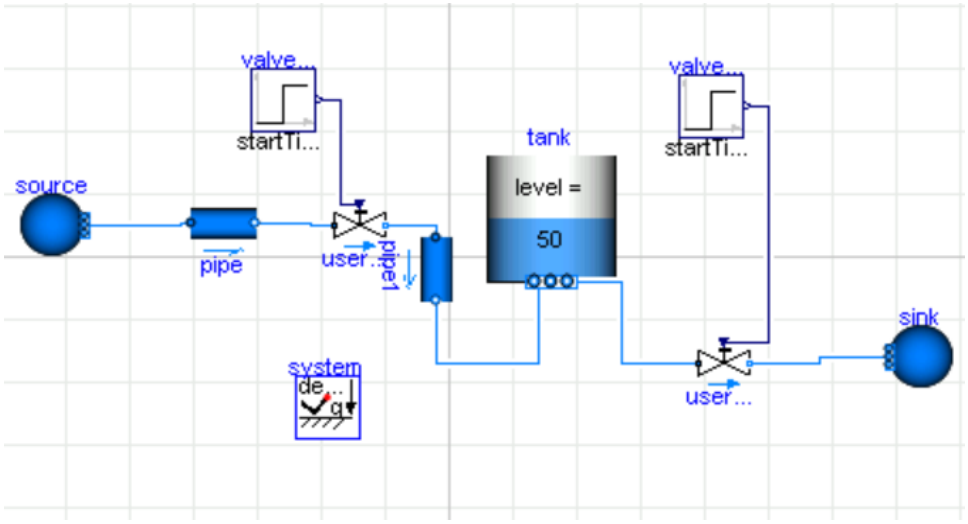

Fig.15: Simple fluid test system

Here the goal of our system was to familiarize ourselves with the components and the different parameters so there is no concrete physical purpose for our model.

Here our model is composed of:

- Source
- 2 pipes
- 2 valves
- Tank
- 2 Step control
- Sink

In our model we used to step control to delay the opening of the two valves. The first one opens at 200 secs and fills the tank till the second valve opens at t = 400sec and allows the tank to empty into the sink. Here we are not using a pump, but we are generating current through a pressure difference between the reservoir and the water source. However, as we know, when the water level rises the pressure at the bottom of the reservoir increases, therefore the difference in pressure between the reservoir and the source decreases, which causes the flow to decrease until the pressure difference is equal to zero. In our model we did not wait until the pressure equalized and therefore at t= 400 sec the second valve opens and drains the fluid into the sink. Once the valve is open the pressure difference is so great that the reservoir will empty fast. This rate will however decrease until the flow rate coming into the tank and the flow rate exiting the tank balance out and the reservoir level becomes constant. This was our goal, and this is what we observed when we plotted the reservoir level in function of time as show in the figure to the side.

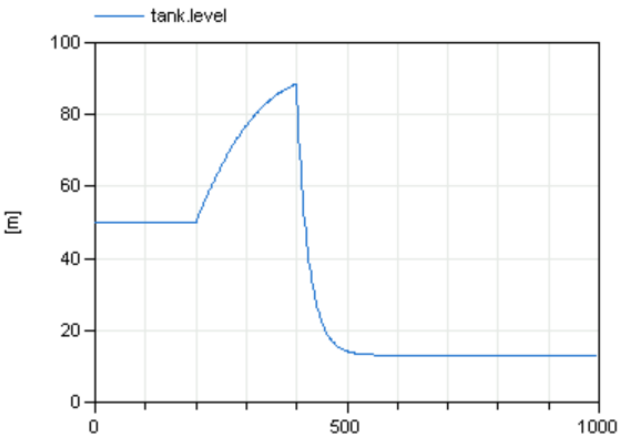

Fig.16: Plot of the tank level

7. Phase two – Creating a model of a linear valve

Our second phase was modelling our first components. Even though valves do exist in the original Dymola library, we chose to first model a linear van to ensure that we had a model to compare ours with in case we needed to debug or detect an error.

7.1. Understanding the structure of the Modelica coding Language

We first started by trying to understand the Modelica language and its structure. Through our work we discovered that the Modelica language is not very complex as it uses very simple syntax. What makes Dymola complicated is its use of different libraries throughout its code.

7.2. Dymola libraries and its building blocks

Dymola and the Modelica like previously mentioned is a modelling and simulation software tool that uses a component-oriented approach to model physical systems. These components can be created by coding, using the Modelica language, or by combining existing building blocks and bricks.

Libraries in Dymola are collections of pre-built components, and they contain a set of related models that are organized in packages. A component is made of multiple building blocks and bricks. These blocks and bricks are specific codes used to define different variables, parameters, and equations. What is tricky is that each block and brick contain other more fundamental bricks in a never-ending chain just like a Russian doll. This is advantageous as it removes the need to create a component from scratch and having to completely redefine and code a whole component. However, this can also be a very big disadvantage. During our project, this was the case. As our goal was to use as few of the components as possible to help us understand the inner working of Dymola, we had to search through the different levels of the bricks and blocks to understand what each of the blocks did and how we could code our components to our specific needs.

7.3. Modelica structure

The Modelica language is structured into 2 parts: parameters and the equations. However, for the sake of understanding exactly how everything worked together we further divided these three parts.

7.3.1. Introduction

Introduction is the part of the code where we first needed to import some essential libraries such as the constant library. As we were trying to limit the number of libraries and blocks in our code this is the only thing we used, but other more complex components will generally import or “extend” a block. By extending a block in a code the component will inherit all the brick’s properties and equations.

```
model LinearValve_Test_V2
  import Modelica.Constants.pi;
```

Fig.17: First lines of code our linear valve test

```
model ValveLinear "Valve for water/steam flows with linear pressure drop"
  extends Modelica.Fluid.Interfaces.PartialTwoPortTransport;
```

Fig.18: First lines of code of Dymola's linear valve

7.3.2. Parameters, Variables, Inputs

In the Modelica language parameters are used to define a variable that the user will be able to modify when a specific component is used. These parameters can be furthermore clarified by adding a specification such as the unit. This is useful as this unit will be seen in the parameter window when a user uses it and it will also be seen on the plots during a simulation. As you can see in the figure below, in the case of our linear valve, we chose to define two parameters that will modify the valve. The first is the valve flow coefficient and the second is the section area of the valve.

```
// Parameters
parameter Real Cv = 1.0 "Valve flow coefficient";
parameter Modelica.SIunits.Area A = 0.01 "Valve effective area [m2]";
```

Fig.19: The parameters of our linear valve test

In this section the user will also be able to define variables. These variables will be a series of non-fixed values that will be used in the equations later on but must be first defined here. In our case as shown in the figure below, we defined **"dp"** as the pressure drop across the valve in pascals and **"m_flow"** which is the flow rate through the valve in m3/s.

```
// Variables
Modelica.SIunits.PressureDifference dp "Pressure drop across valve [Pa]";
Modelica.SIunits.MassFlowRate m_flow "Flow rate through valve [m3/s]";
```

Fig.20: The variables of our linear valve test

Finally we must also define the inputs and interfaces of our components so that the software knows the connections to expect when integrated into a model. In our case, since we are modelling a linear valve with one entrance and one exit, we used the modelica brick **"Modelica.Fluid.Interfaces.FluidPort_a/b"** to indicate our input and outputs. Since we also want to control the opening and closing of said valve, we needed to specify that an input **"valve_opening"** is necessary.

```
//Block's Inputs
Modelica.Fluid.Interfaces.FluidPort_a port_a
  a ;
Modelica.Fluid.Interfaces.FluidPort_b port_b
  a ;
Modelica.Blocks.Interfaces.RealInput valve_opening(min=0, max=1)
  a ;
```

Fig.21: The inputs of our linear valve test

Once defined these inputs and outputs can be seen on the graphic display of the components as two circles for the two ports and one arrow for the valve opening input.

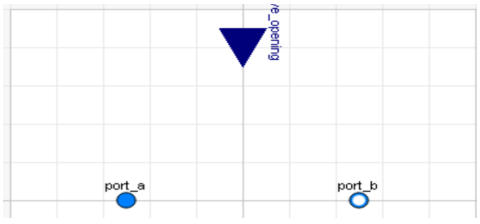

Fig.22: Graphic representation of our linear valve model

7.3.3. Equations

The equation part of the code is the most important and the core of the code. This is where the user can translate physical phenomenon as equations. With these equations the software can solve the system and find all the unknowns that will represent in function of time the model. Here again we can subdivide this part into two parts: governing equations and fundamental equations.

The governing equations are the equation that will enable our component to manipulate the variable as a physical component would.

```
equation
    // Pressure drop(problem with negative values added (abs))
    dp = abs(port_a.p - port_b.p);

    // Flow rate
    m_flow = Cv * A * sqrt(dp) * valve_opening;

    // Design direction of mass flow rate (PTPT)
    m_flow = port_a.m_flow;
```

Fig.23: The equations of our linear valve test

The fundamental equations are equations we had to implement as we did not use any of Dymola's bricks. In reality, these equations are automatically implemented when using the blocks but, in our case, we had to do it ourselves. The fundamental equations are equations that ensure that all rules of physics are respected such as mass balance, phase, the percentages of the different particles in a fluid.

```
//Fondemental Equations

    // Mass balance (no storage)
    port_a.m_flow + port_b.m_flow = 0;

    // Isenthalpic state transformation (no storage and no loss of energy)
    port_a.h_outflow = inStream(port_b.h_outflow);
    port_b.h_outflow = inStream(port_a.h_outflow);

    // Transport of substances
    port_a.Xi_outflow = inStream(port_b.Xi_outflow);
    port_b.Xi_outflow = inStream(port_a.Xi_outflow);

    port_a.C_outflow = inStream(port_b.C_outflow);
    port_b.C_outflow = inStream(port_a.C_outflow);

end LinearValve_Test_V2;
```

Fig.24: The fundamental equations of our linear valve test

8. Phase three – Incorporating a compressor model

After we became acquainted with the software and the modelling language our next task was to incorporate Mr. Hazyuk's own compressor in a model.

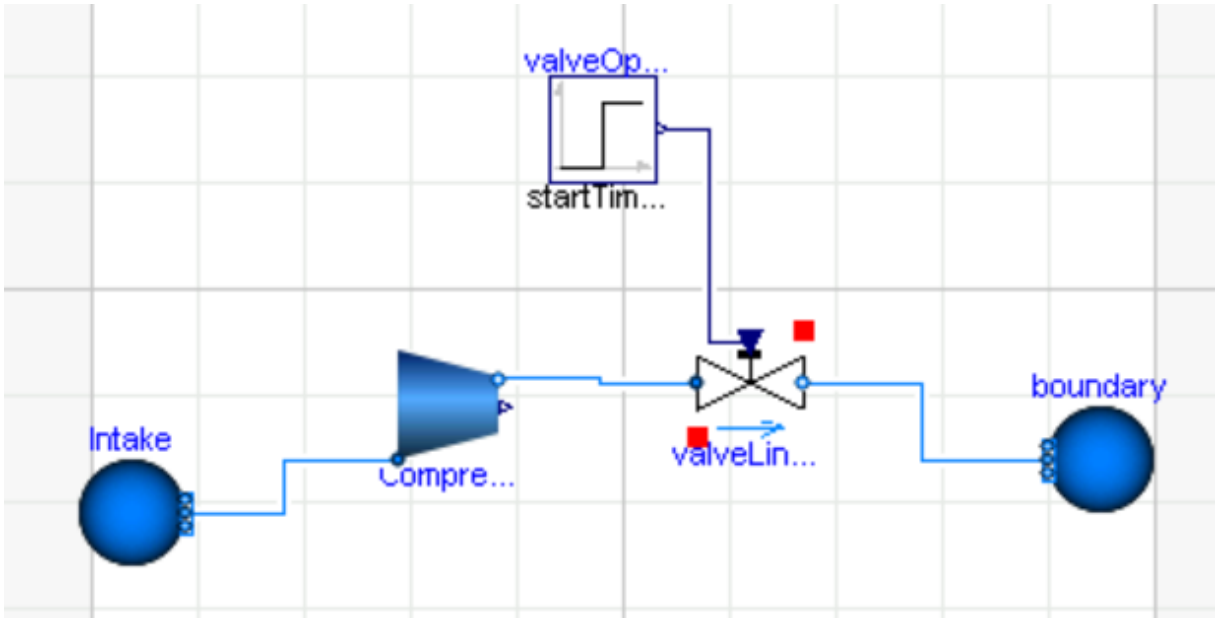

Fig.25: Compressor implementation into a system

The goal of this model was to compress a fluid and control its flow using a valve. However, after long hours of debugging and trying to figure out where the errors were, we were not able to run a successful simulation. This is due to the fact that in our model we have 64 scalar unknowns and only 63 equations. This is obviously unsolvable, and we tried to go through the different bricks of the components but to no avail. However, this experience helped us further understand how the software works and how to accurately tailor the libraries to our needs.

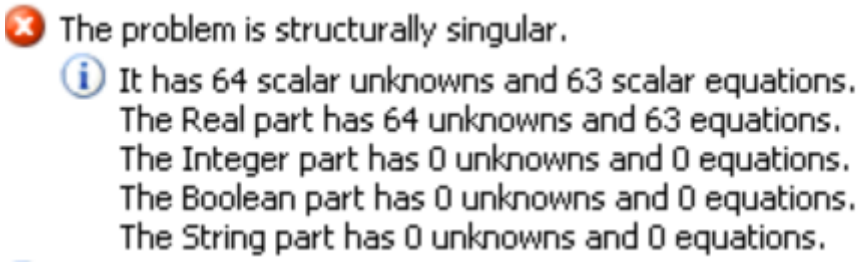

Fig.26: Error messages after simulation

9. Review of the Modelling Tool

Through this project and working with Dymola, we faced many challenges, but through it we gained a lot of useful knowledge. Understanding the innerworkings of Dymola had its learning curve but by digging into the Modelica language we now better understand its functionalities and capabilities. By studying Dymola's libraries and its models we learned how to efficiently organize and manage the components required for our future models. We also gained first-hand experience in creating and coding our own components, learning how to manipulate the different bricks and blocks that Dymola proposes to easily obtain a functioning model. This experience provided us with a deeper understanding of the component's behaviour, characteristics and interaction within the broader context of a simulation. We also learned how to translate physical laws into systems of equations to define the properties and dynamics of the components.

While we are very satisfied with the learning experience, we however remain slightly frustrated as we were not able to create a functioning model with Mr. Hazyuk's own library. We do believe that with more time we would have been able to finalize this model and move on to the creation of more components, but this is part of the process, and we hope to accomplish the rest of this project next semester.

Conclusion – Discussion

This research project was divided into two parts. In the first phase, during the first semester, we researched the ins and outs of a PEMFC. We identified the different architectures and analysed their different components to comprehend how each piece of this big puzzle interacted with each other to turn hydrogen and air into electricity. Through this phase we also had to dive into the world of physics to learn how technologies were implemented to create the perfect working environment to optimise this process. With all this knowledge we wrote our state of the art.

In our second phase, our goal was also divided into two parts. First to create a sizing tool that would be able to dimension a PEMF cell according to different parameters such as power consumption and the external environment. Through this process we created a python code enabling us to manipulate different governing equations and inputs to give us an optimised efficiency of the stack and the corresponding stack characteristics.

Finally, this project is not yet finished. This report comes too early compared to the final adjustments we could have made to the models. The Modelica component library is not yet elaborated, and some important evolutions of the sizing optimization remain outstanding.

Concerning the script, it currently works well and does fit with the requirements we stated at the beginning of the project, but we could have improved the following elements:

- Better power optimization based on the mission profile: the current optimization is based on the average value of the power mission profile. This does work, but since the efficiency of the stack varies along the V-I curve, it varies when changing the stack power supply, during the mission. Therefore, in a future version, it would be better to iteratively go through the mission profile to derive a function of several variables (including efficiency, for example) to find its extremum. Such an optimization would be more efficient for manufacturers.

- Derive the K_m value from a vehicle model: using a vehicle simulation model and a real complete mission profile, it would be feasible to derive the Km value used in the performance function from a mean calculation of the variation of power function of the variation of total mass (including hydrogen tank): $K_m = \overline{\frac{\partial P}{\partial m_{tot}}}$.

Appendix

```
# -*- coding: utf-8 -*-

"""
##### Sizing program for PEMFC #####

This code is a tool to size a PEMFC stack. This is based on governing equations derived analyticaly from physical laws. Those equations use parameters with physical meaning. The hypothesises and references used can be found in the attached litterature.

@authors: Mathis DE MONTAZET and Felix BONNES, INSA Toulouse, 2023

"""



### Instructions

# The desired PEMFC properties must be filled in the "Input parameters" section.

# The state of the art parameters used for calculation as well as their acceptable range can be consulted in "State-of-the-art parameters" section.

# The sources used are detailed in the "Sources" section of the report.



### Versions

# __Version 1 (old):__

# The first version aimed at implement the basis equations for plotting the V-I diagram.
# The implemented equations were the following ones:
# - Nernst equation
# - Activation losses (bugging)
# - Ohmic losses
# - Mass transport losses (bugging)

# __Version 2 (old):__

# The current program improves the following points compared to V1:
# - Better explanation of the comments
# - Actualisation of the mass transport losses contants
# - Calculation of the stack power and efficiency function of the current density
# - Better V-I plot including power and efficiency curves
# - Calculation of thedesign points: maximum power and best efficiency

# The following points show mistakes or are not fully understood:
# - Understanding of $i_{0,c}$
# - Activation offset seams wrong
# - $U_{act}$ definition is not sure
# - GDL diffusion coefficient cannot be found in the litterature

# __Version 3 (old):__

# The current version aims at improving/implementing the following parts compared to V2:
# - Implement some volume and mass estimations for a state of the art PEMFC
# - Finalize the V-I diagram and validate it
# - Propose another efficiency definition based on a cost function including a weighted stack mass
# - Propose minimum volume and mass design points

# __Version 4 (current and final):__

# This final version aims at implemeting a "mission profile" input:
# - Add a mission profile input for power with a fixed voltage
# - Add experimental data from manufacturers on the theorical V-I curve
# - Find an accurate value for mass density of a PEMFC stack

"""
BEGINNING OF THE PROGRAM
"""

### Import of librairies

import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import interp1d


### Input parameters

# `[t, P]` = Power mission profile\
# `Us` = $U_s$ is the required stack voltage, in [V] \
# `P_H2` = $P_{H_2}$ is the hydrogen inlet pressure, in [bar] \
# `P_air` = $P_{air}$ is the air inlet pressure, in [bar] \
# `Ts` = $T_s$ is the operating temperature of the stack, in [°C]. The usual temperature is about 80°C. *Play with it!*
```

```

## Power mission profile definition

# Possibility to import data from database
t = [0, 10.32, 15.48, 22.70, 29.93, 49.54, 61.92, 84.62, 97.0, 117.65, 144.48, 154.80, 164.09, 176.47, 185.55, 200] #[s] Time
F_av = 140e3/356 #[N] Typical car forward force
speed = [0, 0, 15, 15, 0, 0, 31.5, 31.5, 0, 0, 50, 50, 35, 35, 0, 0] #[m/s] Car speed
P = [speed[e] * F_av for e in range(len(t))] #[W]

# First iteration mean power calculation
Ps = np.mean(P) #[W] Mean power based on dataset input power (not interpolated power!)
Ps_init = Ps

fig, ax = plt.subplots(1)
plt.plot(t, P, 'r', zorder=90)
plt.scatter(t, P, c='b', marker='+', s=80, zorder=99)
plt.plot([min(t), max(t)], [Ps, Ps], 'm-.', lw=0.7, zorder=1, label=f"Mean power: {int(Ps)} [W]")
plt.grid()
plt.title("Power mission profile")
plt.xlabel("Time [s]")
plt.ylabel("Power [W]")
plt.legend(loc="upper left")
plt.show()

## Other parameters

Us = 800 #[V] Stack required voltage

# State-of-the-art PEMFCs use the same pressure for H2 and air to avoid internal stresses in the stack
P_H2 = 2.3 #[bar] Hydrogen inlet pressure
P_air = 2.3 #[bar] Air inlet pressure
Ts = 80 #[°C] Stack temperature

# If you want to export the results to a TXT file, change the following variable to `True`
print_results_to_txt = False


### Design point

# To determine the design point, the user can define a performance function that will be maximized during the dimensionning process. This function can use the following variables:

# - The `efficiency` of the stack
# - The stack `mass`
# - The stack `power`

## Performance function
def F_perf(eff, **Par):
    return eff - Par['K_m'] * Par['m'] # - Par['K_p'] * Par['P']
    # return eff / (Par['m']+Par['N_cell'])

K_m = 10 * 1e-3 # Mass influence parameter


### Calculations from input parameters

# T_s[K] = T_s[°C] + 273.15
Ts = Ts+273 #[K] Temperature of the stack in Kelvin


### Constants definition

R = 8.314 #[J/mol/K] Perfect gaz constant
F = 96485 #[C/mol] Faraday's constant
Tref = 293.15 #[K] Reference temperature
Pref = 1 #[bar] Reference pressure
x_O2 = 0.21 #[-] Molar fraction of O2 in air
n = 2 #[-] Number of electrons exchanged in redox reaction


### State of the art parameters

# __Open circuit voltage__
# This is the optimal (maximum) voltage reachable computed with the Nernst equation. It would be obtained without losses.

Erev_0 = 1.229 #[V] Standard state reference potential
delta_S0_nF = -0.85e-3 #[V/K] Standard state enthropy change

# __Activation losses__ \
# Some voltage difference from equilibrium, called overpotential, is needed to get the electrochemical reaction going. In a hydrogen/oxygen fuel cell, the oxygen reduction requires much higher overpotential. Thus, we will focus only on the cathode side. State of the art cathodes are covered by platinum, we then give appropriate values for this case of study.\
```

```
alpha_c = 0.65 #[-] Transfer coefficient for Oxygen reduction on Platinum [0.5 - 0.7]
i_0c_ref = 1e-3 #[A/m] Reference exchange current density for Oxygen reduction on Platinum
a_c = 800 #[cm²/mg] Catalyst specific area [600 - 1000]
L_c = 0.3 #[mg/cm²] Catalyst loading [0.3 - 0.5]
gamma_c = 0.5 #[-] Pressure dependency coefficient for cathode side
E_c = 66e3 #[J/mol] Activation energy for Oxygen reduction on Platinum

# __Mass transport losses__ \
# This loss is due to a diffusion process. $H_2$ and $O_2$ must diffuse through the electrodes to react.
D = 0.625e-6 #[m²/s] Diffusion coefficient of cathode GDL (carbon paper) [adjusted to have i_lim = 2 A/cm²]
delta = 100e-6 #[m] Diffusion layer thickness [100 - 400]

# __Ohmic losses__ \
# Ohmic losses occur because of resistance to the flow of ions int the electrolyte and resistance to the flow of electrons through the electrically conductive fuel cell components.
R_i = 0.11e-4 #[Ohm.m²] Total cell internal resistance [0.1 - 0.2]e-4


### V-I diagram

p_O2 = P_air*x_O2 #[bar] Partial pressure of O2

# Mass transport coefficients (cathode side only, as anode side losses are neglectable)
C_O2 = (p_O2*1e5)/(R*Ts)
i_lim = (n*F*D*C_O2)/delta

i = np.linspace(1, np.ceil(i_lim/1000)*1000, 1000) #[A/m²] Range of current density to study

## Nernst Equation
Erev = Erev_0 + (Ts - Tref)*delta_S0_nF #[V] Reversible voltage
U_oc = Erev + (R*Ts/(n*F))*np.log(P_H2*(p_O2)**0.5) #[V] Open circuit voltage

## Losses
# Activation losses (cathode side)
i_0c = i_0c_ref*a_c*L_c*(p_O2/(Pref*x_O2))**gamma_c*np.exp(-(E_c/(R*Ts))*(1-(Ts/Tref))) #[A/m²] Exchange current density
print(f"i_0c = {round(i_0c*1e-4, 3)} A/cm²")
U_act = []
for j in i:
    U_act.append((R*Ts/(alpha_c*F))*np.log(j+i_0c))

# Mass transport calculation (cathode side only, as anode side losses are neglectable)
print(f"i_lim = {round(i_lim*1e-4, 1)} A/cm²")
U_m = []
for j in i:
    if i_lim/(i_lim-j) >= 0:
        U_m.append((R*Ts/(n*F))*np.log(i_lim/(i_lim-j)))
    else:
        U_m.append(float('nan'))

# Ohmic losses
U_ohm = R_i * i


### Calculation of the resulting voltage

# Cell voltage
U_cell = U_oc - U_act - U_m - U_ohm

### Power density

Pd = [U_cell[e] * i[e] for e in range(len(i))] #[W/m²]


### Mass estimation

def mass(A, N_cell, e_GDL):
    component = ["End Plate", "Bipolar Plate", "Gasket", "GDL", "PEM", "GDL", "Gasket", "Bipolar Plate", "End Plate"] #[m²]
    thicknesses = [0.0050, 0.0020, 0.0002, e_GDL, 0.0001, e_GDL, 0.0002, 0.002, 0.005] #[m]
    densities = [2800, 1750, 2300, 600, 1900, 600, 2300, 1750, 2800] #[kg/m^3]
    occurences = [1, N_cell, N_cell, N_cell, N_cell, N_cell, N_cell, 1, 1] #[-]
    m = 0
    # length = 0
    for idx in range(len(thicknesses)):
        m += A * thicknesses[idx] * densities[idx] * occurences[idx]
        # length += thicknesses[idx] * occurences[idx]
    return m #, length

### Efficiency

# __Definition of the efficiency:__

# We define the efficiency of a fuel cell as the voltage density provided divided by the chemical power consumed
```

```
# $U_s$: volatge delivered by the stack<br>
# $U_{s, th}$: Theorical volatge deliverable by the chemical reaction

### Design points
# __The performance function:__\
# The performance function is defined by user above

# __The best trade-off between efficiency and mass:__\
# An other available design point is the best ratio efficiency over mass. This design point can be selected by user

# __The best power / lower mass:__\
# The last design point available is the best power, which is also the lower mass design point

## Efficiency calculation
Eff = [0 for idx in range(len(i))]
for idx in range(len(Eff)):
    Eff[idx] = U_cell[idx] / U_oc #[-]

Is = Ps / Us #[A] Intensity of the current provided by the stack
N_cell = [Us / U_cell[e] for e in range(len(i))]
A = [Is / i[e] for e in range(len(i))]
m = [mass(A[e], N_cell[e], delta) for e in range(len(i))] #[kg]

Fun_perf = [0 for idx in range(len(i))]
Eff_mass = [0 for idx in range(len(i))]
Pd_times_eff = [0 for idx in range(len(i))]

for idx in range(len(i)):
    Fun_perf[idx] = np.max([0, F_perf(Eff[idx], K_m=K_m, m=m[idx])])
    Eff_mass[idx] = np.max([0, Eff[idx] / m[idx]])
    Pd_times_eff[idx] = np.max([0, Pd[idx] * Eff[idx]])


### Calculation of the design points

# Optimum values
max_Pd = 0
best_Perf = -1e20
best_Eff_m = 0
min_m = 1e20
for idx in range(len(i)):
    if Pd[idx] > max_Pd:
        max_Pd = Pd[idx]
        idx_max_Pd = idx

    if Fun_perf[idx] > best_Perf:
        best_Perf = Fun_perf[idx]
        idx_best_Perf = idx

    if Eff_mass[idx] > best_Eff_m:
        best_Eff_m = Eff_mass[idx]
        idx_best_Eff_m = idx

    if m[idx] < min_m:
        min_m = m[idx]
        idx_min_m = idx

## Get experimental data
# To plot experimental data on our V-I diagram
import exp_data as exp


### V-I plot

x_scale = 1e-4 # Change from [A/m²] to [A/cm²]

max_power_VI = (i[idx_max_Pd]*x_scale, U_cell[idx_max_Pd]) # V-I coordinates of maximum power
best_perf_VI = (i[idx_best_Perf]*x_scale, U_cell[idx_best_Perf]) # V-I coordinates of best performance
best_eff_m_VI = (i[idx_best_Eff_m]*x_scale, U_cell[idx_best_Eff_m]) # V-I coordinates of best efficiency/mass
min_mass_VI = (i[idx_min_m]*x_scale, U_cell[idx_min_m]) # V-I coordinates of minimum mass

# Bi-axis graph
VIfig, ax = plt.subplots()
ax.plot(i*x_scale, U_cell, color="blue", zorder=99, label="V-I curve")
ax.plot(i*x_scale, Eff, color="orange", zorder=99, label="Efficiency")
ax.plot(i*x_scale, Fun_perf, color="green", zorder=99, label="Perf. function")
# ax.plot(i*x_scale, Eff_mass, color="cyan", label=f"$\epsilon$/mass") # Scale not adapted
ax.scatter(max_power_VI[0], max_power_VI[1], color="red", marker="*", s=100, zorder=100, label="$P_{{max}}$")
ax.text(max_power_VI[0]-max(i*x_scale)*0.01, max_power_VI[1]-max(U_cell)*0.001, "$P_{{max}}$", zorder=100, ha="right",
va="top")
ax.scatter(best_perf_VI[0], best_perf_VI[1], color="green", marker="*", s=100, zorder=100, label=f"$F_{{perf, max}}$")
ax.text(best_perf_VI[0]+max(i*x_scale)*0.005, best_perf_VI[1]+max(U_cell)*0.01, "$F_{{perf, max}}$", zorder=100,
ha="left", va="bottom")
ax.scatter(min_mass_VI[0], min_mass_VI[1], color="orange", marker="*", s=30, zorder=100, label=f"$m_{{min}}$")
```

```
ax.text(min_mass_VI[0]+max(i*x_scale)*0.002, min_mass_VI[1]+max(U_cell)*0.015, "$m_{{min}}$", zorder=101, ha="left", va="bottom")
ax.scatter(best_eff_m_VI[0], best_eff_m_VI[1], color="cyan", marker="*", s=100, zorder=100, label=f"$\epsilon / m_{{max}}$")
ax.text(best_eff_m_VI[0]-max(i*x_scale)*0.03, best_eff_m_VI[1]+max(U_cell)*0.015, "$\epsilon / m_{{max}}$", zorder=100, ha="left", va="bottom")

# Formatting
ax.set_xlabel("Current density [A/cm²]", fontsize = 12)
ax.set_ylabel("Cell voltage [V]", color="blue", fontsize=12)
ax.set_ylim([0, np.ceil((max(U_cell)+0.1)*10)/10])
ax.grid()
ax.set_yticks(np.arange(0, max(U_cell)+0.1, 0.1))
ax2=ax.twinx()
ax2.plot(i*x_scale, Pd, color="red")
# ax2.plot(i*x_scale, Pd_times_eff, color="darkred", label="Power dens. * eff")
# ax2.plot(i*x_scale, m, color="cyan")
# ax2.plot(i*x_scale, N_cell, color="magenta")
ax2.set_ylabel("Power density [W/m²]",color="red",fontsize=12)
ax2.set_ylim([0, np.ceil(max(Pd)/1000)*1000])
lgd = ax.legend(loc="lower center", bbox_to_anchor=(1.38, -0.025))
plt.xticks(np.arange(0, i[np.where(np.isnan(U_cell))][0]*x_scale+0.2, 0.2))
plt.show()

# Experimental data
Datafig = plt.figure()
plt.plot(i*x_scale, U_cell, color="blue", zorder=99, label="V-I curve")

plt.scatter(exp.nafion117_e175[:, 0], exp.nafion117_e175[:, 1], zorder=77, alpha=0.5, marker="s", c="lightblue", label="e = 175µm, T = 80°C")
plt.scatter(exp.nafion117_e100[:, 0], exp.nafion117_e100[:, 1], zorder=80, alpha=0.5, marker="+", c="darkred", label="e = 100µm, T = 70°C")
plt.scatter(exp.nafion_70deg_e100[:, 0], exp.nafion_70deg_e100[:, 1], zorder=79, alpha=0.5, marker="^", c="green", label="e = 100µm, T = 80°C")

# Formatting
plt.xlabel("Current density [A/cm²]", fontsize = 12)
plt.ylabel("Cell voltage [V]", color="blue", fontsize=12)
plt.ylim([0, np.ceil((max(U_cell)+0.1)*10)/10])
plt.grid()
plt.legend()
plt.yticks(np.arange(0, max(U_cell)+0.1, 0.1))
plt.xticks(np.arange(0, i[np.where(np.isnan(U_cell))][0]*x_scale+0.2, 0.2))
plt.show()


### Choice of design point

dp = best_perf_VI # Best performance
# dp = best_eff_m_VI # Best efficiency/mass
# dp = min_mass_VI # Also P_max


### Calculation of A and N_cell

def area(Is, i): # Is [A] / i [A/cm²]
    return Is / (i/x_scale) # A [m²]

def cell_number(Us, U):
    return np.ceil(Us/U)


### Test for reaching the maximum power

A_dp = area(Is, dp[0])
N_cell_dp = cell_number(Us, dp[1])
m_dp = mass(A_dp, N_cell_dp, delta)
P_maxi = max_Pd*A_dp*N_cell_dp

power_margin = 5 #[%] Margin percentage over the maximum power to reach

print(f"             Max power margin percentage: {power_margin:5} [%]")
print(f"                    Current sizing power: {int(Ps):5} [W]")
print(f"           Stack maximum power available: {int(P_maxi):5} [W]")
print(f"Required maximum power (margin included): {int(np.max(P) * (1 + power_margin/100)):5} [W]\n")

if P_maxi > np.max(P) * (1 + power_margin/100):
    ok = True
    print("[+] The current design is compatible with the power mission profile.")
else:
    ok = False
    print("[-] The current design is too low to satisfy the maximum power required by the mission profile.")


### Determination of the best design point compatible with the mission profile

```

```
iter = 0
while ok == False:
    # Choice of a new Ps
    eps_percent = 0.1 #[%]
    eps_Ps = eps_percent/100 * (np.max(P) - np.min(P)) #[W] Step of additional power
    Ps = Ps + eps_Ps #[W] Calculation of the new Ps value

    # Update all the values that are function of Ps
    Is = Ps / Us #[A] Current provided by the stack
    A = [area(Is, i[e]*x_scale) for e in range(len(i))]
    m = [mass(A[e], N_cell[e], delta) for e in range(len(i))] #[kg]
    # print(m)

    Fun_perf = [0 for idx in range(len(i))]
    Eff_mass = [0 for idx in range(len(i))]

    for idx in range(len(i)):
        Fun_perf[idx] = F_perf(Eff[idx], K_m=K_m, m=m[idx])
        Eff_mass[idx] = Eff[idx] / m[idx]

    max_Pd = 0
    best_Perf = -1e20
    best_Eff_m = 0
    min_m = 1e20
    for idx in range(len(i)):
        if Pd[idx] > max_Pd:
            max_Pd = Pd[idx]
            idx_max_Pd = idx

        if Fun_perf[idx] > best_Perf:
            best_Perf = Fun_perf[idx]
            idx_best_Perf = idx

        if Eff_mass[idx] > best_Eff_m:
            best_Eff_m = Eff_mass[idx]
            idx_best_Eff_m = idx

        if m[idx] < min_m:
            min_m = m[idx]
            idx_min_m = idx
    max_power_VI = (i[idx_max_Pd]*x_scale, U_cell[idx_max_Pd]) # V-I coordinates of maximum power

    # Calculate the stack maximum power available
    i_max_power = max_power_VI[0]
    U_cell_max_power = max_power_VI[1]

    A_dp = area(Is, dp[0])
    N_cell_dp = cell_number(Us, dp[1])
    P_maxi = max_Pd*A_dp*N_cell_dp

    iter += 1

    if P_maxi > np.max(P) * (1 + power_margin/100):
        ok = True # Value of Ps is OK
    else:
        ok = False # Value of Ps is still too low, new iteration ...

print(f"         Number of iterations: {int(iter):8}")
print(f"        Sizing power increase: {round((1-Ps_init/Ps)*100, 1):8} [%]")
print(f"                Sizing power: {round(Ps, 1):8} [W]")
print(f"Stack maximum power available: {round(P_maxi, 1):8} [W]")
if np.max(P) * (1 + power_margin/100) != np.max(P): print(f"       Maximum power to reach: {round(np.max(P) * (1 + power_margin/100), 1):8} [W]")
print(f"       Required maximum power: {round(np.max(P), 1):8} [W]")

missionfig, ax = plt.subplots(1)
plt.plot(t, P, 'r', zorder=90)
plt.scatter(t, P, c='b', marker='+', s=80, zorder=99)
plt.plot([min(t), max(t)], [Ps_init, Ps_init], 'm-.', lw=0.8, zorder=1, label=f"Mean power: {int(Ps_init)} [W]")
plt.plot([min(t), max(t)], [Ps, Ps], 'g-.', lw=1.1, zorder=1, label=f"Actual sizing power: {int(Ps)} [W]")
plt.grid()
plt.title("Power mission profile")
plt.xlabel("Time [s]")
plt.ylabel("Power [W]")
plt.legend()
plt.show()


### Required specifications for the stack

# Global specifications
N_cell = cell_number(Us, dp[1])
A = area(Is, dp[0])
m = mass(A, N_cell, delta)

```

```python
# Specifications at mean power: we determine the working point on V-I diagram
power = Ps
gap = power - Ps_init + 1
idx = np.where(i == i[np.abs(i-dp[0]*1e4).argmin()])[0][0]
while abs(power - Ps_init) < gap:
    idx = idx - 1
    gap = abs(power - Ps_init)
    power = i[idx]*A*U_cell[idx]*N_cell
i_mean = i[idx]
Is_mean = i_mean * A
Us_mean = Ps_init/Is_mean
eff_mean = ((Us_mean/N_cell)/U_oc)*100

print("Global specifications:")
print(f"      Number of cells in series: {round(N_cell):6} [-]")
print(f"                      Cell area: {round(A*1e4, 1):6} [cm²]")
print(f"                     Stack mass: {round(m, 1):6} [kg]\n")
print("Specifications at mean power:")
print(f"        Stack operating voltage: {round(Us_mean, 0):6} [V]")
print(f"        Stack operating current: {round(Is_mean, 0):6} [A]")
print(f"Stack operating current density: {round(i_mean*1e-4, 2):6} [A/cm²]")
print(f"              Load of the stack: {round((Ps_init/P_maxi)*100, 2):6} [% of max power]")
print(f"                Mean efficiency: {round(eff_mean, 2):6} [%]")

### H_2 and O_2 consumption, H_2_O generation

# Knowing the equation of the chemical reaction in the cell and the Faraday's law, we can easily find H2 and O2 consumption.
mol_mass_H2 = 0.002 #[kg/mol] H2 molar mass
mol_mass_O2 = 0.032 #[kg/mol] O2 molar mass
mol_mass_H2O = 0.018 #[kg/mol] H2O molar mass

M_H2_cons = N_cell * (Is_mean / (n*F)) * mol_mass_H2 * 3600
M_O2_cons = N_cell * (Is_mean / (2*n*F)) * mol_mass_O2 * 3600
M_H2O_gen = N_cell * (Is_mean / (n*F)) * mol_mass_H2O * 3600

print(f"      H_2 consumed: {round(M_H2_cons, 3):5} [kg/h]")
print(f"      O_2 consumed: {round(M_O2_cons, 3):5} [kg/h]")
print(f"   H_2_O generated: {round(M_H2O_gen, 3):5} [kg/h]")


### Export the results

import datetime
date = datetime.date.today()
txtdate = date.strftime("%B %d, %Y")
time = datetime.datetime.now().strftime("%Hh%M")

if print_results_to_txt == True:
    with open(f"exports/Sizing_results - {date} - {time}.txt", "w", encoding="utf8") as export_file:
        export_file.write("RESULTS OF THE PEMFC SIZING PROCESS\n\n")
        export_file.write(f"Results of {txtdate} - {time}\n\n\n")

        export_file.write(" > Input parameters\n")
        export_file.write(f"\t               Mean power:   Ps = {int(Ps):5} [W]\n")
        export_file.write(f"\t      Max power available: Pmax = {int(P_maxi):5} [W]\n")
        export_file.write(f"\t        Operating voltage:   Us = {Us:5} [V]\n")
        export_file.write(f"\t  Hydrogen inlet pressure: P_H2 = {P_H2:5} [bar]\n")
        export_file.write(f"\t    Oxygen inlet pressure: P_O2 = {P_air:5} [bar]\n")
        export_file.write(f"\t    Operating temperature:   Ts = {Ts-273:5} [°C]\n\n")

        export_file.write(" > Results\n")
        export_file.write("   > Global specifications\n")
        export_file.write(f"\tNumber of cells in series: N_cell = {round(N_cell):5}\n")
        export_file.write(f"\t                Cell area:      A = {round(A*1e4, 1):5} [cm²]\n")
        export_file.write(f"\t               Stack mass:      m = {round(m, 1):5} [kg]\n\n")
        export_file.write("   > Specifications at mean power\n")
        export_file.write(f"\t        Stack operating voltage: Us_mean = {round(Us_mean, 0):5} [V]\n")
        export_file.write(f"\t        Stack operating current: Is_mean = {round(Is_mean, 0):5} [A]\n")
        export_file.write(f"\tStack operating current density:  i_mean = {round(i_mean*1e-4, 2):5} [A/cm²]\n")
        export_file.write(f"\t              Load of the stack:      Ls = {round((Ps_init/P_maxi)*100, 2):5} [% of max power]\n")
        export_file.write(f"\t                Mean efficiency:     eff = {round(eff_mean, 2):5} [%]\n\n")
        export_file.write("   > Fuel cell consumption and emission\n")
        export_file.write(f"\tHydrogen consumed: H2_cons = {round(M_H2_cons, 3):5} [kg/h]\n")
        export_file.write(f"\t  Oxygen consumed: O2_cons = {round(M_O2_cons, 3):5} [kg/h]\n")
        export_file.write(f"\t    Water emitted: H2O_gen = {round(M_H2O_gen, 3):5} [kg/h]\n")

    VIfig.savefig(f"exports/V-I diagram - {date} - {time}", dpi=300, bbox_extra_artists=lgd)
    missionfig.savefig(f"exports/Mission Profile - {date} - {time}", dpi=300)
```

Bibliographical references

[1] "System Lifecycle Process Models: Vee - SEBoK." https://sebokwiki.org/wiki/System_Lifecycle_Process_Models:_Vee (accessed May 08, 2023).

[2] G. Tsotridis, A. Pilenga, M. G. De, and T. Malkow, "EU HARMONISED TEST PROTOCOLS FOR PEMFC MEA TESTING IN SINGLE CELL CONFIGURATION FOR AUTOMOTIVE APPLICATIONS," *JRC Publications Repository*, Jan. 27, 2016. https://publications.jrc.ec.europa.eu/repository/handle/JRC99115 (accessed May 02, 2023).

[3] J. C. Amphlett, R. M. Baumert, R. F. Mann, B. A. Peppley, P. R. Roberge, and T. J. Harris, "Performance Modeling of the Ballard Mark IV Solid Polymer Electrolyte Fuel Cell: I . Mechanistic Model Development," *J. Electrochem. Soc.*, vol. 142, no. 1, pp. 1–8, Jan. 1995, doi: 10.1149/1.2043866.

[4] R. Maurya, R. Das, A. K. Tripathi, and M. Neergat, "Relationship between the electron-transfer coefficients of the oxygen reduction reaction estimated from the Gibbs free energy of activation and the Butler–Volmer equation," *Phys. Chem. Chem. Phys.*, vol. 25, no. 1, pp. 700–707, Dec. 2022, doi: 10.1039/D2CP04331A.

[5] F. Barbir, "Chapter Three - Fuel Cell Electrochemistry," in *PEM Fuel Cells (Second Edition)*, F. Barbir, Ed., Boston: Academic Press, 2013, pp. 33–72. doi: 10.1016/B978-0-12-387710-9.00003-5.

[6] F. Barbir, "Fuel Cell Basic Chemistry, Electrochemistry and Thermodynamics," in *Mini-Micro Fuel Cells*, S. Kakaç, A. Pramuanjaroenkij, and L. Vasiliev, Eds., in NATO Science for Peace and Security Series C: Environmental Security. Dordrecht: Springer Netherlands, 2008, pp. 13–26. doi: 10.1007/978-1-4020-8295-5_2.

[7] R. O'Hayre, S.-W. Cha, W. Colella, and F. B. Prinz, *Fuel Cell Fundamentals*. John Wiley & Sons, 2016.

[8] R. Omrani, "Chapter 5 - Gas diffusion layer for proton exchange membrane fuel cells," in *PEM Fuel Cells*, G. Kaur, Ed., Elsevier, 2022, pp. 91–122. doi: 10.1016/B978-0-12-823708-3.00017-1.

[9] M. Schröder, F. Becker, J. Kallo, and C. Gentner, "Optimal operating conditions of PEM fuel cells in commercial aircraft," *Int. J. Hydrog. Energy*, vol. 46, no. 66, pp. 33218–33240, Sep. 2021, doi: 10.1016/j.ijhydene.2021.07.099.

[10] Z. Abdin, C. J. Webb, and E. MacA. Gray, "PEM fuel cell model and simulation in Matlab–Simulink based on physical parameters," *Energy*, vol. 116, pp. 1131–1144, Dec. 2016, doi: 10.1016/j.energy.2016.10.033.

[11] M. Tellez, J. Escorihuela, O. Solorza-Feria, and V. Compañ, "Proton Exchange Membrane Fuel Cells (PEMFCs): Advances and Challenges," *Polymers*, vol. 13, p. 3064, Sep. 2021, doi: 10.3390/polym13183064.

[12] M. Mori, R. Stropnik, M. Sekavčnik, and A. Lotrič, "Criticality and Life-Cycle Assessment of Materials Used in Fuel-Cell and Hydrogen Technologies," *Sustainability*, vol. 13, p. 3565, Mar. 2021, doi: 10.3390/su13063565.

[13] A. A. Ebrahimzadeh, I. Khazaee, and A. Fasihfar, "Numerical investigation of dimensions and arrangement of obstacle on the performance of PEM fuel cell," *Heliyon*, vol. 4, no. 11, p. e00974, Nov. 2018, doi: 10.1016/j.heliyon.2018.e00974.

[14] F. Huang, D. Qiu, S. Lan, P. Yi, and L. Peng, "Performance evaluation of commercial-size proton exchange membrane fuel cell stacks considering air flow distribution in the manifold," *Energy Convers. Manag.*, vol. 203, p. 112256, Jan. 2020, doi: 10.1016/j.enconman.2019.112256.

[15] C. Lamy and J.-M. Leger, "Les piles à combustible : application au véhicule électrique," *J. Phys. IV*, vol. 04, no. C1, pp. C1-253-C1-281, Jan. 1994, doi: 10.1051/jp4:1994119.