Combination of simulation and optimization for gas storage filling

Department Project : IMI (IMIP2)

Group 3: Project submitted by Air Liquide

Students
Pierre-Louis Franco
Emilien Perrin
Charles Vielzeuf

Supervisory Engineers
Fouad Ammouri
Moulay-Driss Elalaouifaris
Karim Menshawy

Supervisor:
Fouad Ammouri

May 15, 2024

Contents

List of symbols

This table presents the symbols that we used in this report, and the notations used in the code.

Table 1: list of symbols

Symbol	Definition	Unit	Code
τ	Time	s	t
τ_f	Duration of the refuelling	s	refill_time
V	Volume of the tank	m^3	V
S_i	Internal surface of the tank's walls	m^2	S_int
S_e	External surface of the tank's walls	m^2	S_ext
T_{inlet}	Pre-cooled temperature of the hydrogen	K	T_inlet
T	Temperature of the hydrogen inside the tank	K	T
$T_{\max}$	Threshold of the tank's temperature	K	Tmax
c_p	Specific heat capacity of the gas at constant pressure	$\mathrm{J\,kg^{-1}\,K^{-1}}$	cp
T_w	Temperature of the tank's walls	K	Tw
$c_{p,w}$	Specific heat capacity of the wall	$\mathrm{J\,kg^{-1}\,K^{-1}}$	cp_wall
m_w	Mass of the wall	kg	m_wall
P	Pressure of the hydrogen	Pa	P
P_0	Initial pressure	Pa	Pgas_initial
P_{var}	Pressure slope	$\mathrm{Pa\,s^{-1}}$	pressure_variation
m	Mass of hydrogen in the tank	kg	m
n	Molar quantity	mol	n
R	Ideal gas constant	$\mathrm{J\,mol^{-1}\,K^{-1}}$	R
Z	Compressibility factor of the hydrogen	None	Z
k_g	Heat exchange coefficient between the wall and the gas	$\mathrm{W\,m^{-1}\,K^{-1}}$	k_int
k_a	Heat exchange between the wall and the ambient air	$\mathrm{W\,m^{-1}\,K^{-1}}$	k_ext
h_e	Specific enthalpy of incoming gas	$\mathrm{J\,kg^{-1}}$	h_inlet
h	Specific enthalpy of the gas	$\mathrm{J\,kg^{-1}}$	h
β	Thermal expansion coefficient of the gas	$\mathrm{K^{-1}}$	beta

I) Introduction

1) Motivation

Hydrogen vehicles represent a crucial part of the strategy to reduce greenhouse gas emissions linked to transport. However, to contribute to such a strategy the technology needs to be optimal for the users. In order for the user to not be bothered by this type of vehicle, the refueling time of the tank must be in the same range as that of a thermal vehicle. With this constrain, a problem arises: during hydrogen refueling, the high pressure (700 bara) in the dispensers -caused by the speed at which we wish to refuel- induces a significant rise of the tank's temperature. The tank's walls being made of composite materials which cannot withstand temperatures above 85°C. For these reasons, dispensers must be equipped with a pre-cooling system. [2]

However, pre-cooling is costly and its use must be determined for each particular case as the tank is not at the same temperature in every car and at every time. Thus, having pre-cooling at constant temperature is not optimal. To give some orders of magnitude, with hydrogen costing approximately 15 euros per kilogram and providing 100 kilometers of travel, the cooling process accounts for 10% of the total energy cost, with logistics (including transport and storage) constituting half of the expenditure.

Currently, Air Liquide uses tabulated data given input parameters such as refuelling time, initial temperature and pressure ramp, which are not optimal. Our objective is to provide real-time injection temperature profiles by optimizing the temperature profile parameters to reduce energy consumption within a goal of 2-3 % of precision.

Figure 1: Hydrogen fueled car

2) Modeling the problem

We consider the problem of filling a hydrogen tank of volume V in an environment with constant ambient temperature throughout the filling process. To simplify the problem, we have made a few assumptions:

1. Modeling 0D-1D: The gas is assumed to be homogeneous (same pressure/same temperature) throughout the tank (0-Dimension), and we consider that the temperature can evolve through the thickness of the tank (1-Dimension). [1]

2. Filling is carried out using a constant pressure ramp throughout the filling process, i.e. chosen so that the tank is full in 300s.

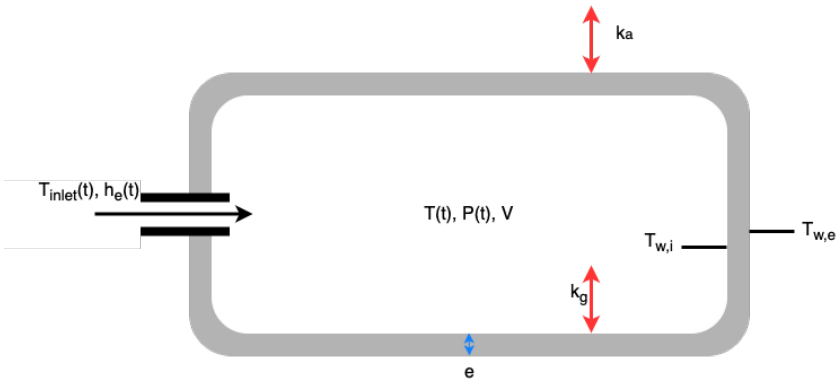


Figure 2: Schematic drawing of the tank

The tank is represented by a cylinder as shown in Figure 2.
We define $T_w(x)$ as the temperature along the wall thickness with edge conditions :

$$\begin{cases} T_w(0,t) = T(t) \\ T_w(e,t) = T_{amb} \end{cases} \tag{1}$$

1 - Pressure

Let P_{var} be the constant pressure ramp inside the tank. The evolution of pressure $P(t)$ follows the following equation :

$$P(t) = P_0 + P_{var} \cdot t \tag{2}$$

2 - Gas temperature

Let $T_{inlet}(t)$ be the pre-cooled temperature at which the hydrogen is injected in the tank. From the first law of thermodynamics we can extract the following equation for the temperature T :

$$m \cdot \frac{dh}{dt} = V \cdot \beta \cdot T \cdot \frac{dP}{dt} + k_g \cdot S_i \cdot (T_{w,i} - T) + \frac{dm}{dt} \cdot (h_e - h) \tag{3}$$

Where, $h_e := H(T_{inlet}, P(t))$.
As stated, previously, the temperature must not exceed the critical value of T_{max} allowed by the tank's composite materials.

3 - Wall temperature and interface

The exchange of energy between the interior of the tank and the ambient air is balanced by the wall's temperature. This thermal dynamic is described by the following heat balance equation :

$$m_w \cdot c_{p,w} \cdot \frac{dT_w}{dt} = k_g \cdot S_i \cdot (T - T_{w,i}) + k_a \cdot S_e \cdot (T_{amb} - T_{w,e}) \tag{4}$$

4 - Mass

The gas is modeled as a real gas with the following equation :

$$P \cdot V \cdot M = m \cdot R \cdot Z(T,P) \cdot T \tag{5}$$

Where, $Z(T, P)$ is the compressibility factor of the gas.

5 - Energy cost

The electric energy consumed during refuelling is the energy required to cool the gas during the entire time of refueling. Thus, we can compute this energy with the following formula :

$$E = \int_0^{t_f} (h_e - h_{inlet}) \cdot \frac{dm}{dt} dt \tag{6}$$

These equations enable us to calculate the changes in T, Tw, m, P, and E during refueling. In summary, the issue of the tank's internal temperature can be formulated as a DAE problem (Differential-Algebraic system of Equations).

$$\begin{cases} P(t) = P_0 + P_{var} \cdot t & \text{Differentiable variable} \\ m \cdot \frac{dT}{dt} = V \cdot \beta \cdot T \cdot \frac{dP}{dt} + k_g \cdot S_i \cdot (T_{w,i} - T) + \frac{dm}{dt} \cdot (h_e - h) & \text{Differentiable variable} \\ m_w \cdot c_{p,w} \cdot \frac{dT_w}{dt} = k_g \cdot S_i \cdot (T - T_{w,i}) + k_a \cdot S_e \cdot (T_{amb} - T_{w,e}) & \text{Algebraic} \\ P \cdot V \cdot M = m \cdot R \cdot Z(T, P) \cdot T & \text{Algebraic} \\ E = \int_0^{t_f} (h_e - h_{inlet}) \cdot \frac{dm}{dt} dt & \text{Differentiable variable} \end{cases}$$

3) From physics to coding

The first step before attempting any profile optimisation is to solve the above problem given a certain temperature profile. In order to do this, we use a differential equation solver in Python (Assimulo). In concrete terms, we use a class which is initialised with all the initial conditions of the problem and a function giving the pre-cooling temperature of the gas at the time of filling.

Using a constant temperature profile gives the following results:

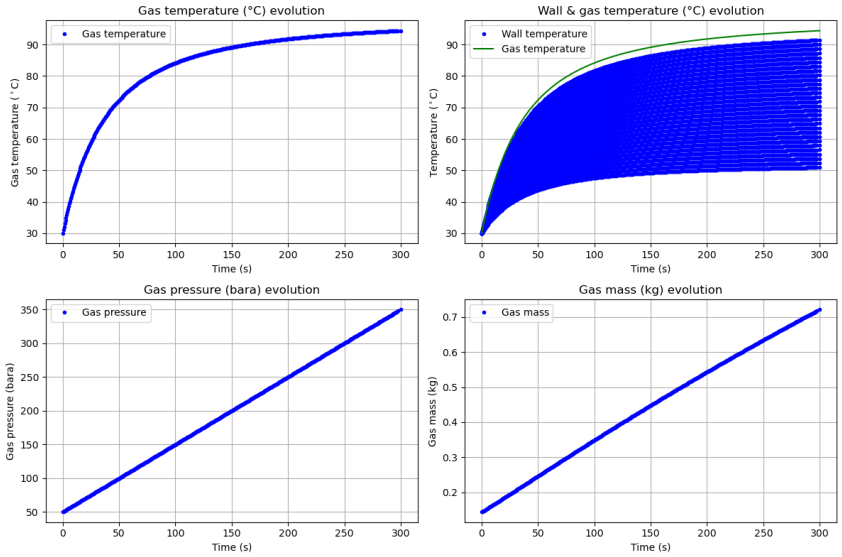


Figure 3: Physical quantities evolutions in the tank

We can see that with the initial conditions we chose, injecting gas at ambiant temperature leads the temperature inside the tank to exceed the critical threshold of 85°C. The challenge now is to find an injection temperature profile that does not exceed this temperature and that is the least expensive.

A naive solution is to use a constant temperature profile and find the pre-cooling temperature at which the gas does not exceed 85°C during filling.

However one of the constraints of the problem is that, initially, the injected gas must be at ambient temperature and cooled (or not) over time. Thus these solutions are not acceptable for our problem.

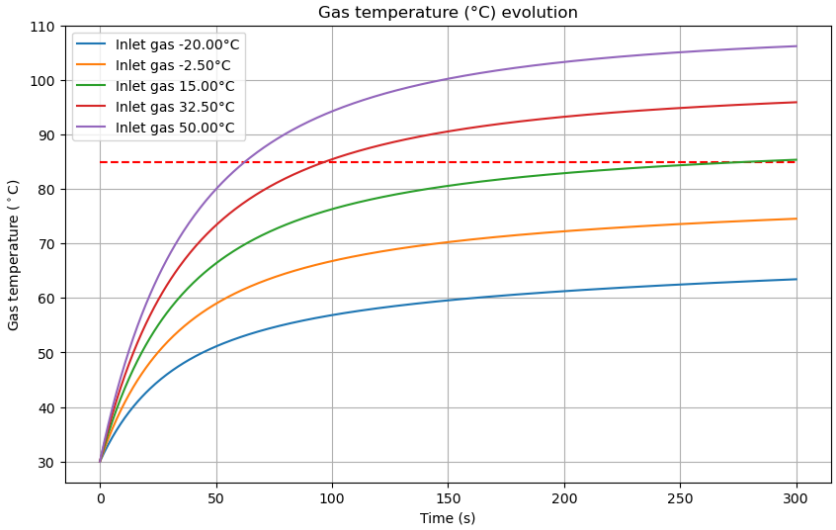


Figure 4: Gas temperature evolution according to different T_{inlet}

II) Finding the optimum injection temperature profile

We will now try to find a pre-cooling temperature profile for the injected gas that will allow the gas in the tank to avoid exceeding the threshold temperature while consuming the least energy. Our aim is the following:

Given a maximal temperature T_{max} the gas is allowed to reach, what inlet temperature profile must we input so that the gas temperature doesn't reach this maximum ? Under this condition, which profile is the cheapest in energy ?

1) Optimisation problem

2.1.1 Mathematical formulation

The problem we are facing can be expressed as a constrained optimisation problem such as :

$$\begin{aligned} \min_{u \in X} \quad & f(u,p) \\ \text{s.t.} \quad & g_i(u,p) \leq 0, \forall i \end{aligned} \tag{7}$$

This is the standard format used by Python optimisation solvers.
The input is then a temperature profile $T_{inlet} = T_{inlet}(t)$.

In order to reduce the size of the problem (input space) and to get a decent real time computational speed, we have chosen to discretise the temperature $(T_{inlet,i})_{1 \leq i \leq n}$ and interpolate between those points.

As stated above, the main constraint is that the maximum temperature reached in the tank during filling must not exceed 85°C.

$$\sup_{t \in [0,t_f]} T(t) \leq T_{max}$$

We need to add a second constraint on the cooling/heating speed. The chiller cannot heat or cool too quickly, which leads to the following constraint.

$$\sup_{t \in [0,t_f]} \frac{dT_{inlet}}{dt} \leq 2K.s^{-1}$$

Finally, we obtain the following optimisation problem.

$$\begin{aligned} \min_{(T_i)_{1 \leq i \leq n} \in [T_{inlet}^{min}, T_{inlet}^{max}]} \quad & \int_0^{t_f} (h_e - h_{in}) \cdot \frac{dm}{dt} dt \\ \text{s.t.} \quad & \sup_{t \in [0,t_f]} T(t) \leq T_{max} \\ & \sup_{t \in [0,t_f]} \frac{dT_{inlet}}{dt} \leq 2K.s^{-1} \end{aligned} \tag{8}$$

It is important to note that this optimisation problem does not follow a standard optimisation problem's form as some of the constraints relate to the solution of a differential equation and not to the inputs directly.

2.1.2 Tricks to tackle the problem

Optimisation solvers do not allow constraints to be included directly, other than on the inputs, so we have to get round this limitation by modifying the objective function.

The idea is to penalise the objective function (energy) as soon as the constraint on the maximum gas temperature is violated. Our problem can then be reformulated as the following :

$$\min_{(T_i)_{1\leq i\leq n}\in[T_{inlet}^{min},T_{inlet}^{max}]} \quad \int_0^{t_f}(h_e-h_{in})\cdot\frac{dm}{dt}dt + \mathbb{I}_{\sup_{t\in[0,t_f]}T(t)\leq T_{max}} + \mathbb{I}_{\sup_{t\in[0,t_f]}\frac{dT_{inlet}}{dt}\leq 2} \quad (9)$$

Where,

$$\mathbb{I}_A = \begin{cases} 0 & if\ x\in A \\ +\infty & otherwise \end{cases}$$

However, for computational purpose, we can't allow the function to take infinite values so we penalised the objective function as the following equation shows :

$$f(T_{inlet},\dots) = \begin{cases} E & \text{if } \sup_{t\in[0,t_f]}T(t)\leq T_{max} \\ E_{big} & \text{otherwise.} \end{cases}$$

Where E_{big} is a finite values five times greater than the order of magnitude of E.

2.1.3 Problems inherent to this method

The first problem is that the computation time here is essentially limited by the differential equation solver, since for any non-admissible profile it is necessary to solve the system of equations before realising that the constraints have been violated.

The second problem is that initialisation is crucial. If the profile given to initialise the solver is non-admissible, then the solver can try out several profiles (all non-admissible) and conclude that the solution to the optimisation problem is E_{big} because the objective function systematically returns the same value.

A final problem is that the solver has difficulty 'understanding' where to look for the optimal profile. Because the function is not smooth (due to penalisation), it does not know in which direction to go to find the optimal profile, which can make gradient methods ineffective.

On way to reduce a lot those problems is to consider, instead of a flat value when non admissible, a huge value that tend to decrease with "more admissibility"; taking for instance in consideration the final temperature and maximal variation when generating E_{big} (see annexe on tests on methods).

2) Optimisation tools

2.2.1 Scipy (NAME!)

Before diving in the core of the subject we had to do a preliminary work, which is that of benchmarking. Indeed, when working on such projects, it is necessary to choose precisely the tools that will be used afterwards because it is too costly to realise when everything

has been done that the tool is not adapted exactly as you had hoped it would. We started by comparing different optimization solvers based on different criteria :

- Compatibility with Assimulo
- Speed
- Maturity

In order to solve the physical equations of the tank, AirLiquide chose to use Assimulo. Thus, it was essential for the solver we chose to be compatible with it as the function we wanted to optimize was computed using Assimulo. Then we looked in the speed of these python libraries because as stated before, the computing must be done in real time. Finally, a criteria we didn't initially think about was the maturity because the solver needs to be devoided of bugs and it has more adaptability in the case of future changes and evolutions.

Libraries	**Pros**	**Cons**
PyMZN	Fast	Wrapper: might not be adapted
PyOMO	Good compatibility	Might take a long time to run
GEKKO	Fast and easy to use	Not mature enough
Scipy	Easy to use, good adaptability, mature	Might be a bit slow

Table 2: Pros and cons of different optimisation solvers

By comparing these four libraries we decided to choose **Scipy** even though it might sometimes be slower than other solvers because it is the most mature solver and that was the main constraint that we were given after the compatibility with **Assimulo**.

2.2.2 Combination of the optimisation solver and the resolution solver

We defined an objective function with strong penalty on constraints (on the output of the simulation when regarding final temperature in gas tank) and on the input made by the solver (by checking that the injection profile discrete derivative was below the given threshold).
As we are solving a DAE problem, when optimising, we must compute a solution of this problem at each iteration given a new inlet temperature profile in order for the optimisation solver to know in which "direction" to optimise, to verify the constraints and when to stop.
In order to store and to visualize the evolution of the temperature profiles and of the energy along the iterations, we decided to create an optimizer class.
Here is a flowchart detailing the architecture of the combination of these two solvers :

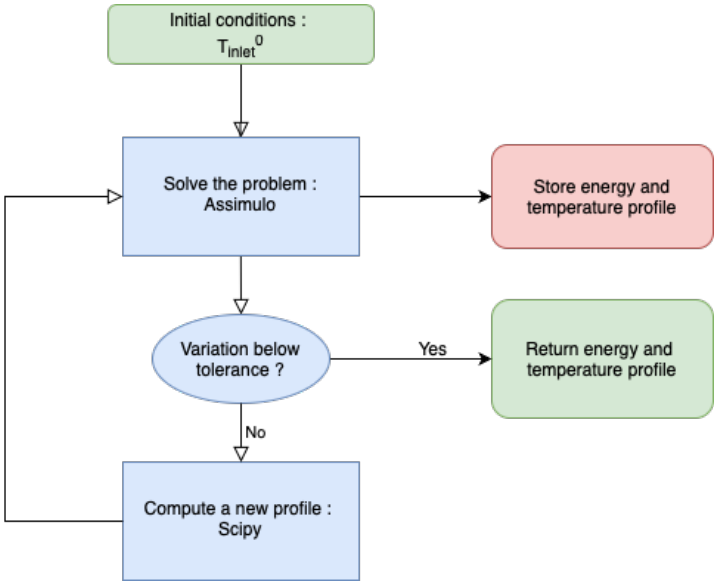
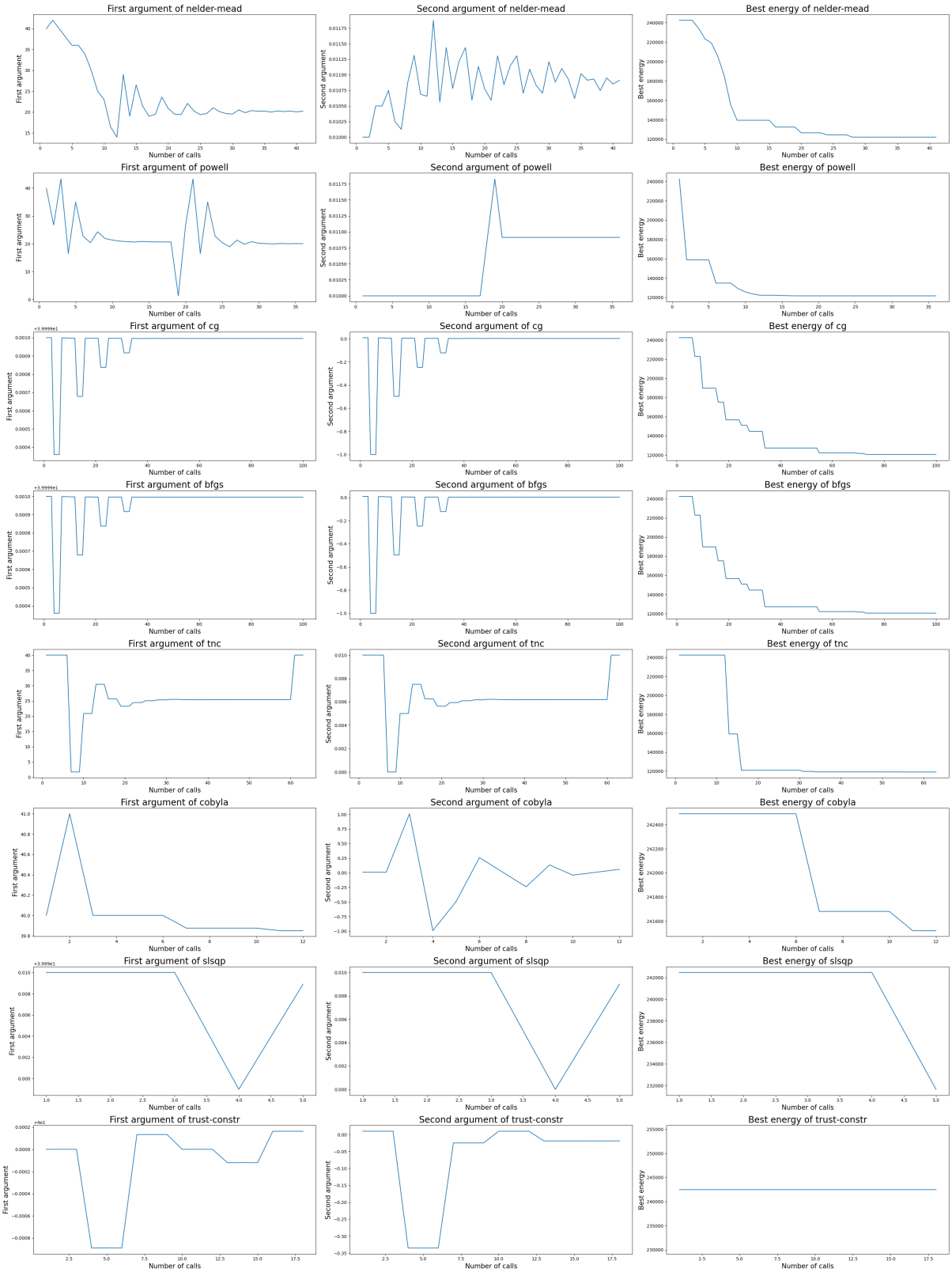


Figure 5: Flowchart of the optimizer class

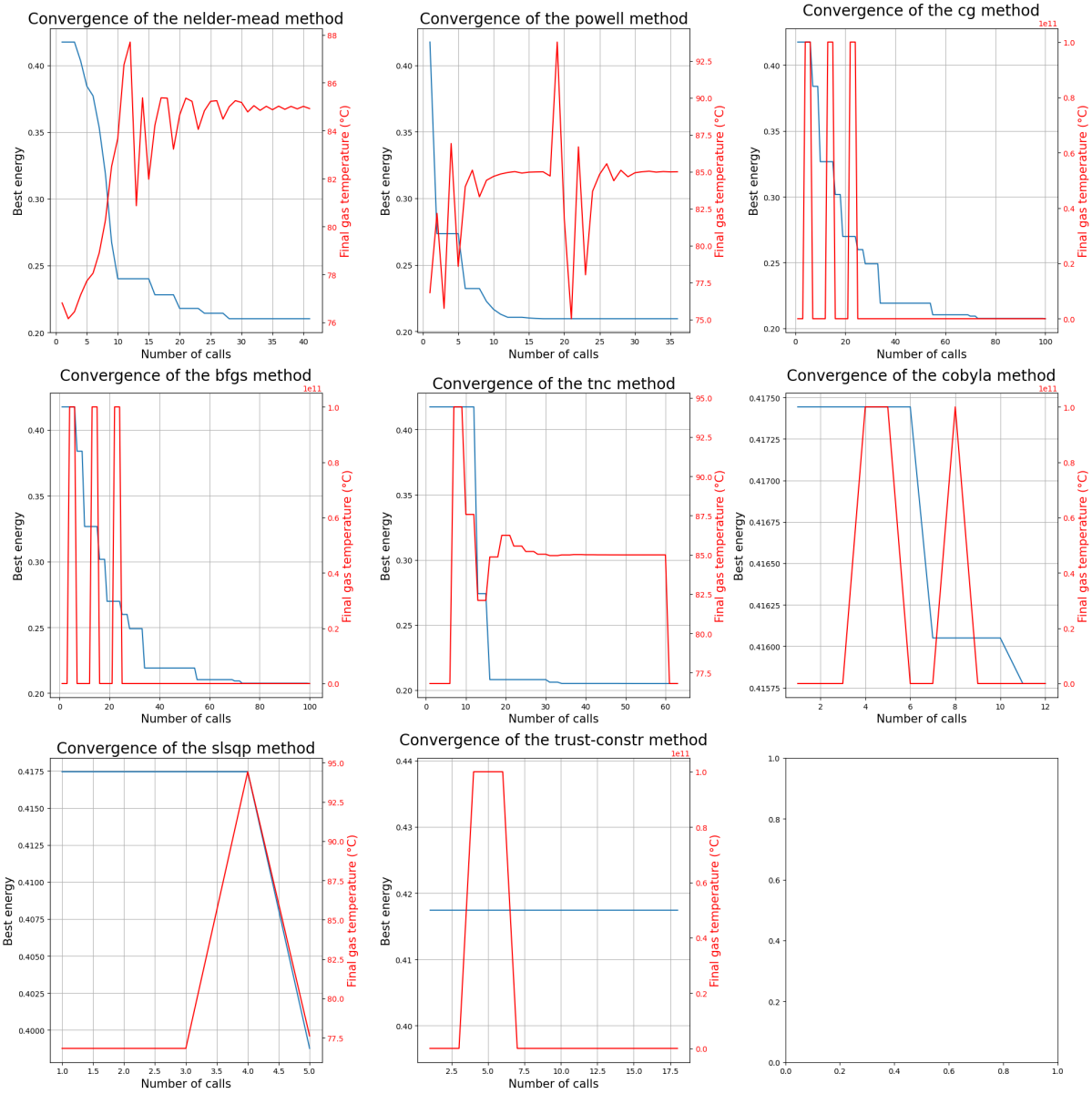
2.2.3 Optimisation methods

We performed **tests on profiles with two parameters** a and b in the form of a first-order equation: $T(t) = a \cdot \exp(-b.t) - a + T_0$, with a given refuelling duration of 5 minutes, a maximum injection gas temperature of 85°, an ambient temperature of 30°, and a variation of 2K.s^{-1}.

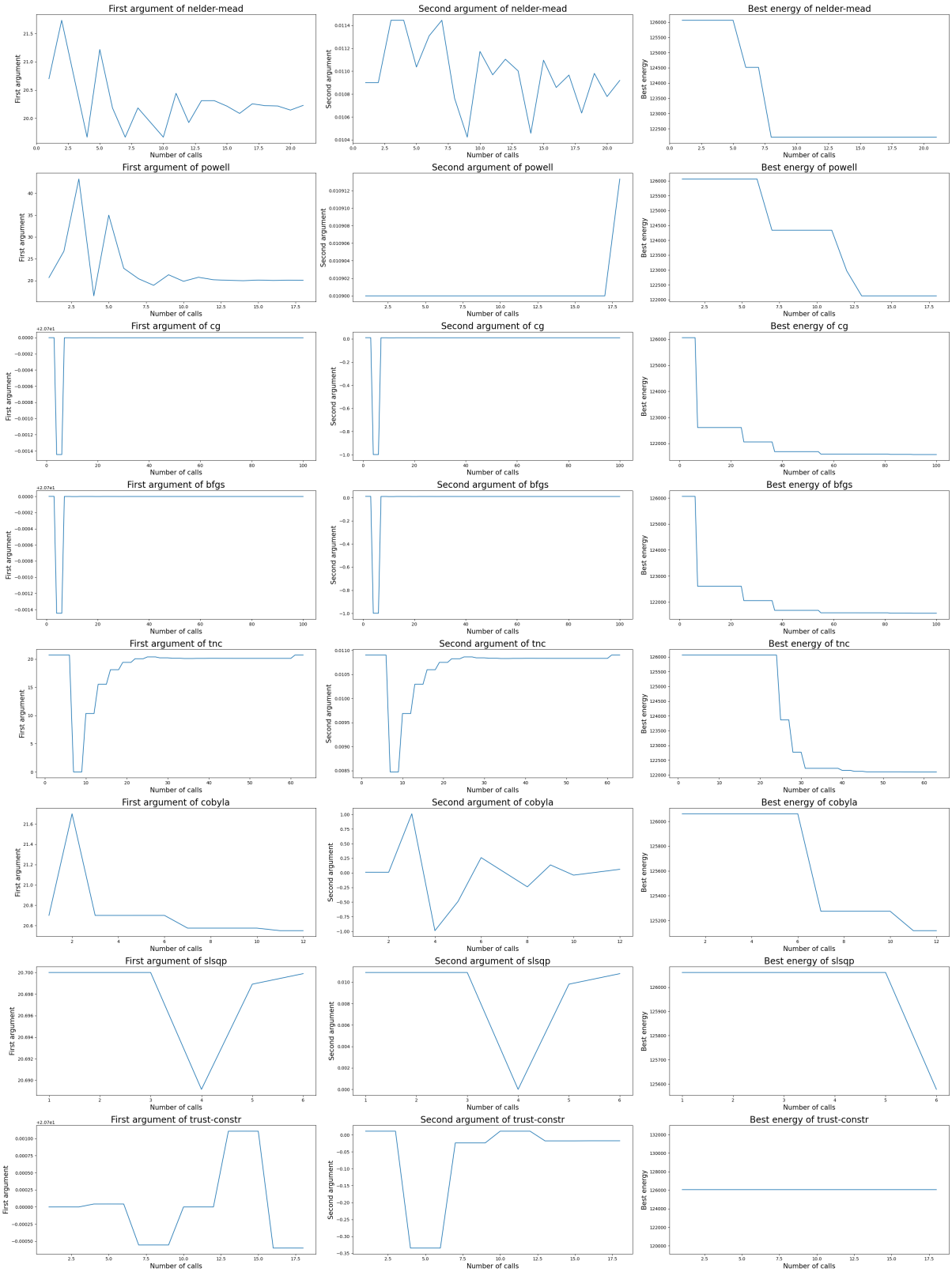
A first remark is that, when initialising to a "naive solution", (taking for instance (a, b) = (40.0; 0.01) for an optimal solution around (20; 0.01), the number of calls needed to start converging can be very high, and some methods don't even converge to the optimal profile, as we can see on the graphs below. For each row, we plot for a given method a and b and the current best energy after a given number of calls (best profiles should be at around 120000 J, which is not the case for all methods).



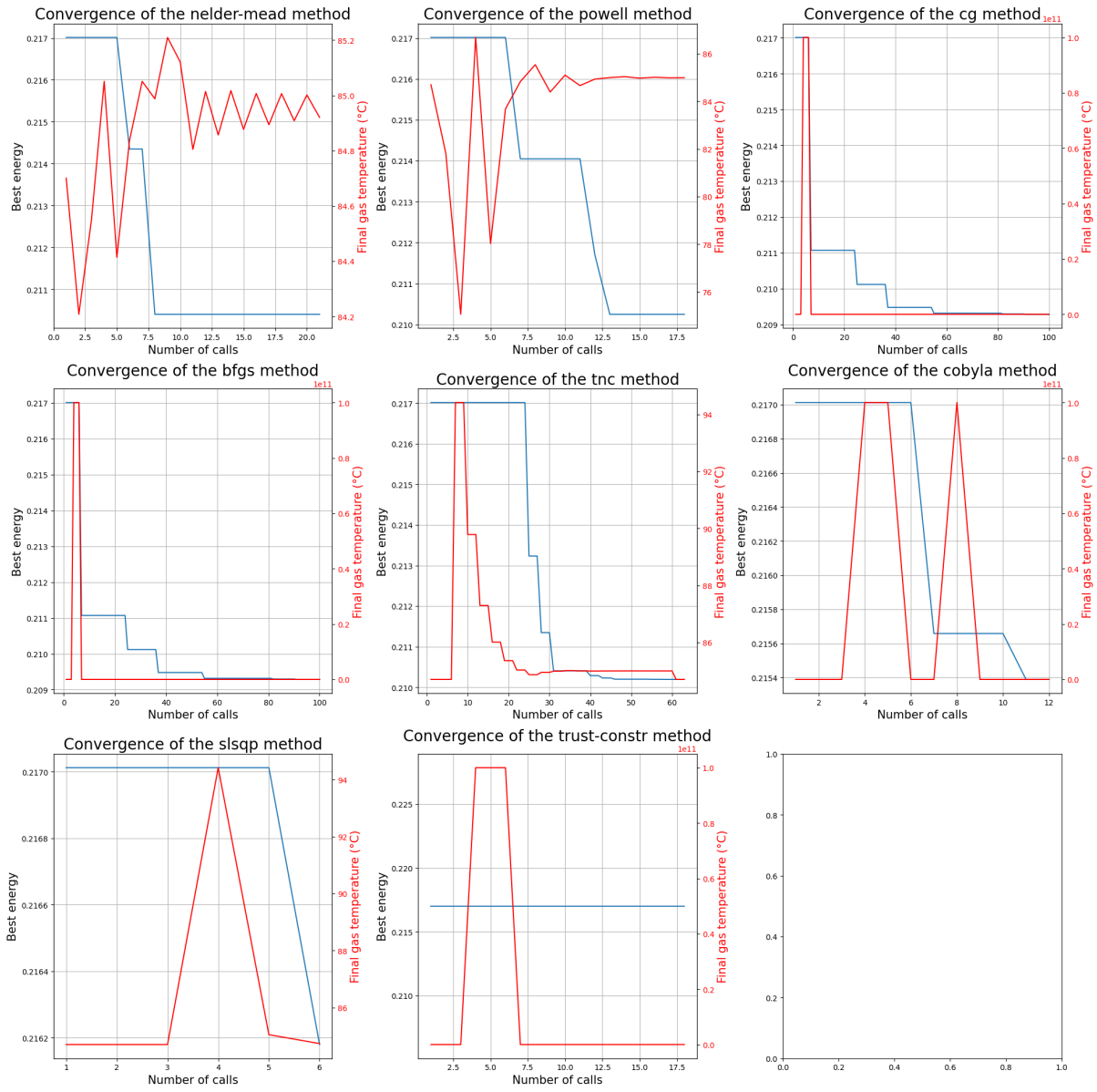
To have a better intuition of the convergence, we also decided to plot the final temperature, that should tend to 85°.



To keep a low level of calls in order to achieve the real time objective, **finding a good initialisation is crucial**. The same code with an initialisation closer to the optimal value (a, b) = (20; 0.0109) , leads to better results. In fact, it is like "skipping" calls on many methods.



As before, we also plot the final temperature to have a better vision on the convergence.



Peaks observed on the temperature are due to incoherent values in the IDA solver: the parameters tested are not leading to a solution through *assimulo* (either the system does not have solution or the physical domains of some parameters are not respected while using *coolprop*).

The conclusion on the different parameters used to tune the optimization to make it more efficient may vary from a situation to another but the tendencies stays the same with other initial parameters such as refuelling duration, a maximum injection gas temperature, ambient temperature, and maximum variation in the injected temperature profile. In the general case, the best is to find a good initialisation given the parametric model (maybe using tabulated data of the profile parameters for given physical inputs), and then to fix a tolerance in the objective function according to the needs (around 2 percents) and fixing the method to **Powell** in the case of two parameters. When testing on ten parameters (considering a linear interpolation on ten points) Powell is pretty slow as it optimizes more or less parameter per parameter. So in that case **conjugate gradient or BFGS** are more appropriated. When choosing a "good" parametric model the problem

is indeed efficiently solved by those methods, and in the general case, **Nelder-Mead** is always a good option not to fall in a local minimum if any, it will only cost a few more iterations that may be worth it.

2.2.4 Optimum injection profiles

As the objective function is nonlinear due to the penalisation and after reviewing the different methods allowed by scipy, we decided to choose the Nelder-Mead method. At first, the optimization algorithm didn't converge because it ran for more than 300 iterations which is the default maximal number of iterations when using a scipy algorithm.
However, we saw that a "trend" was obtained after 20 iterations and that the objective function only varied of a few kiloJoules afterwards. We then decided to put a tolerance of 2kJ and the algorithm returned an optimal solution after 22 iterations in the particular case we worked on. This is particularly interesting because the optimisation resulted was computed in approximately 10 seconds for an injected temperature profile given by the interpolation with a cubic function on 3 points.

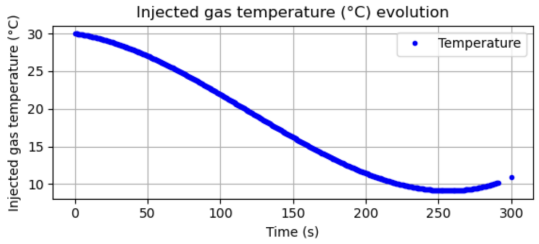


Figure 6: Optimized T_{inlet} profile

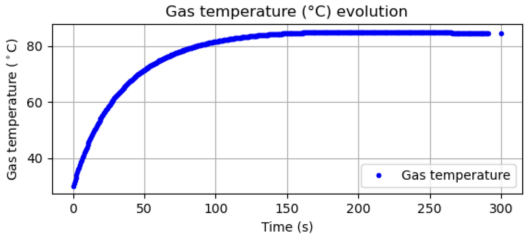


Figure 7: Gas temperature evolution

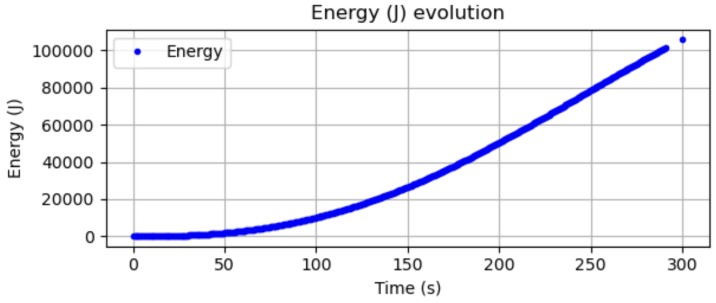


Figure 8: Optimized energy consumption

This figure is obtained for an ambient temperature $T_{amb} = 30°C$. One can have the intuition that an optimal injected temperature profile must lead to a maximal gas temperature of 85°C in the tank. This is the case with the profile shown above. We can also see that the energy consumed to obtain such a profile amounts to 110kJ.
In order to assess the gain in energy with this profile we must compare it to non optimal refilling. One could think of two "naïve" non-optimal injected gas temperature profiles. The first case is when one uses the pre-cooling system at maximal capacity. That is with a profile of the form :

$$T_{inlet}(t) = max(233.15, T_{amb} - 2t)$$

equation)

Implementing this in the dispenser does not require any computation and thus making it interesting when thinking about user experience, however, the cost is great as the following figure shows.

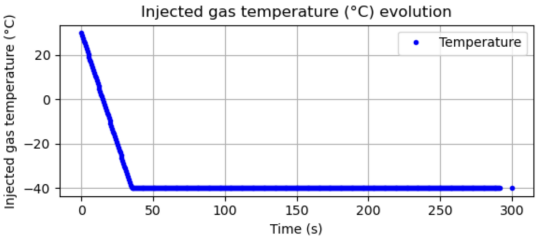


Figure 9: T_{inlet} profile at maximal capacity

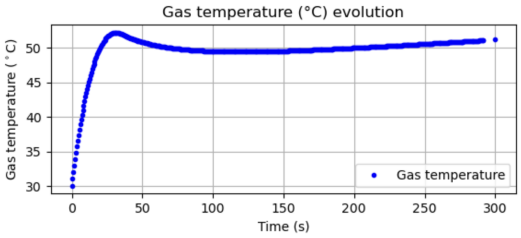


Figure 10: Gas temperature evolution at maximal capacity

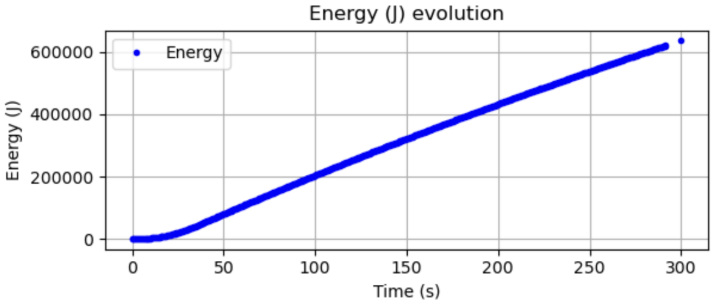


Figure 11: Energy consumption at maximal capacity

As we can see, the amount of energy consumed when doing such an operation is almost 6 times greater with a value of 640kJ. Moreover, we can observe that the gas temperature in the tank does not follow an optimal profile as the maximal temperature reached is 52°C. This means that such a method does not benefit fully from the physical characteristics of the tank walls' composition.

Another naive method requiring more computations would be to find the constant inlet temperature allowing the maximal temperature in the tank to remain under the maximal threshold of 85°C. In the case of an ambient temperature of 30°C, we must inject the gas at the constant temperature of 15°C. That is with a profile of the form :

$$T_{inlet}(t) = max(288, 15, T_{amb} - 2t) \tag{11}$$

This method is very effective as it consumes a lot less energy than with the first naive method and it still respects the constraints of $T_{max} \leq 85$°C.

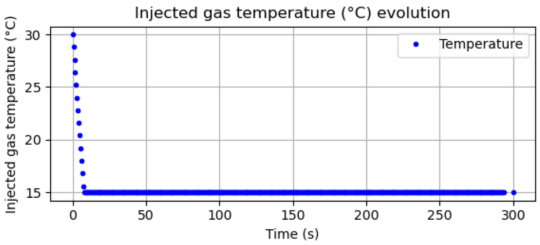


Figure 12: T_{inlet} profile at maximal capacity

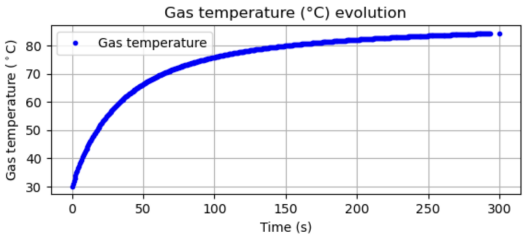


Figure 13: Gas temperature evolution at maximal capacity

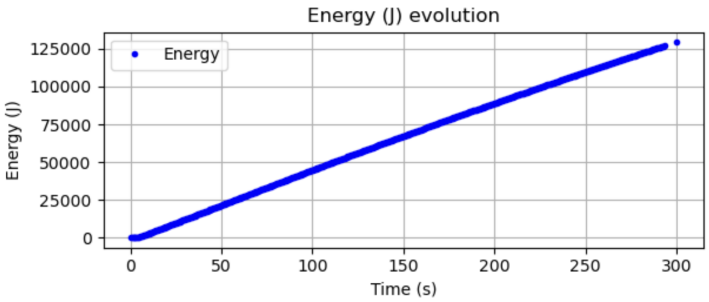


Figure 14: Energy consumption at maximal capacity

We can see that the energy consumed amounts to 130kJ which is only 1.18 times greater than the energy consumed when optimizing. We may wonder if it really is interesting to optimize as it takes some time to compute an optimized injected temperature profile while this method is immediate and only requires to store the values of the constant inlet temperature according to the ambient temperature.

Globally, we see that using the pre-cooling system is clearly non optimal and that less energy is consumed when using an optimized profile. However with the fairly low consumption for the injection at quasi-constant temperature, a study between the economical gain of optimizing and the time consumed for computing the optimized profile must be done by AirLiquide to determine if optimizing is worth it.

III) Conclusion and perspectives

1) Conclusion

Working in the industry is very different in many ways that doing theory in university. Such projects must always start by doing a state of the art of already existing solutions in other relevant areas and also by doing benchmarks of the tool to be used in the upcoming processes. Another different aspect is that we are not searching for the exact solution to the problem but an approached solution respecting different criteria such as a rather fast computational time in our case.
We managed to create an optimizing class using scipy that allowed us to obtain an optimal profile in a shorter time than what we first expected, however, this code could be improve by a lot because there are still too many iterations to meet AirLiquide's specifications especially since a much faster way of pre-cooling with a quasi-constant temperature costs only a bit more in energy.
In order to improve the speed of convergence we could reduce the number of parameters which we are optimizing. A way of doing this could be using parametric profiles rather than interpolated arrays of temperatures. This method is advantageous because it would also allow us to understand the evolution of these parameters as a function of the ambient temperature. And thus, given an ambient temperature, we could have a picture of the parameters of these functions by interpolating the different parameters.
Another time effective approach would be to store profiles for certain ambient temperatures. That would allow the optimizer to be initialized with a profile of interest and thus to find an optimal solution in fewer iterations.

Bibliography

[1] T. Bourgeois et al. "Optimization of hydrogen vehicle refuelling requirements". In: *International Journal of Hydrogen Energy* 42.19 (2017). Special Issue on The 21st World Hydrogen Energy Conference (WHEC 2016), 13-16 June 2016, Zaragoza, Spain, pp. 13789–13809.

[2] T. Bourgeois et al. "The temperature evolution in compressed gas filling processes: A review". In: *International Journal of Hydrogen Energy* 43.4 (2018), pp. 2268–2292.

Annex

Tests on Methods

The *'scipy.optimize()'* function offers various optimization methods tailored for different scenarios, here is a brief overview of methods that seems to work the best, as we will see below:

'Scipy.optimize()' method overview

- *'Nelder-Mead' method*: A **gradient-free algorithm** that suits non-smooth or noisy objective functions but might be slow for high-dimensional problems (which is not the case here as we do not have a lot of decision variables).
- *'Powell' method*: Performs well for unconstrained optimization with discontinuous objectives, exploring directions via **successive one-dimensional minimization**.
- *'CG' (Conjugate Gradient)*: Suits large-scale unconstrained problems by **iteratively updating search directions conjugate to previous ones**.
- *'BFGS' (Broyden–Fletcher–Goldfarb–Shanno)* and *'L-BFGS-B'*: Work well for smooth, large-scale problems, **approximating the inverse Hessian matrix to find descent directions efficiently**.
- *'Newton-CG'*: Applies **Newton's method with conjugate gradients** for large-scale unconstrained optimization.

Here is an example of how those optimization methods operate. Consider the simple convex objective function defined by $f(x_1, x_2) = x_1^2 + x_2^2$, with the initial condition given by $x_0 = (1.5, 0.5)$.

The graph plots the calls made on each argument as well as the objective function value. On this example, the best performing method in terms of objective value out of call number is the **SLSQP which combines the speed of gradient-based approaches with the robustness of least squares fitting** to efficiently handle constraints. The same test is then performed on an objective function of the form $f(x_1, x_2) = x_1^2 + x_2^2$ on $[-1, 1] * [-0.5, 0.3]$ and a huge value elsewhere, starting from $x_0 = (-0.1, 0.1)$ (otherwise the optimisation stops after a few calls if x_0 is in the flat region). It is still SLSQP performing the best. Taking instead of a flat region something like $E_{big} * f(x_1, x_2)$ solves the convergence problem for all of the methods. When considering a function that has

many local minimums it won't be the SLSQP method that converges the best but methods that does not converges "too soon" and explores other areas such as **Nelder-Mead**.

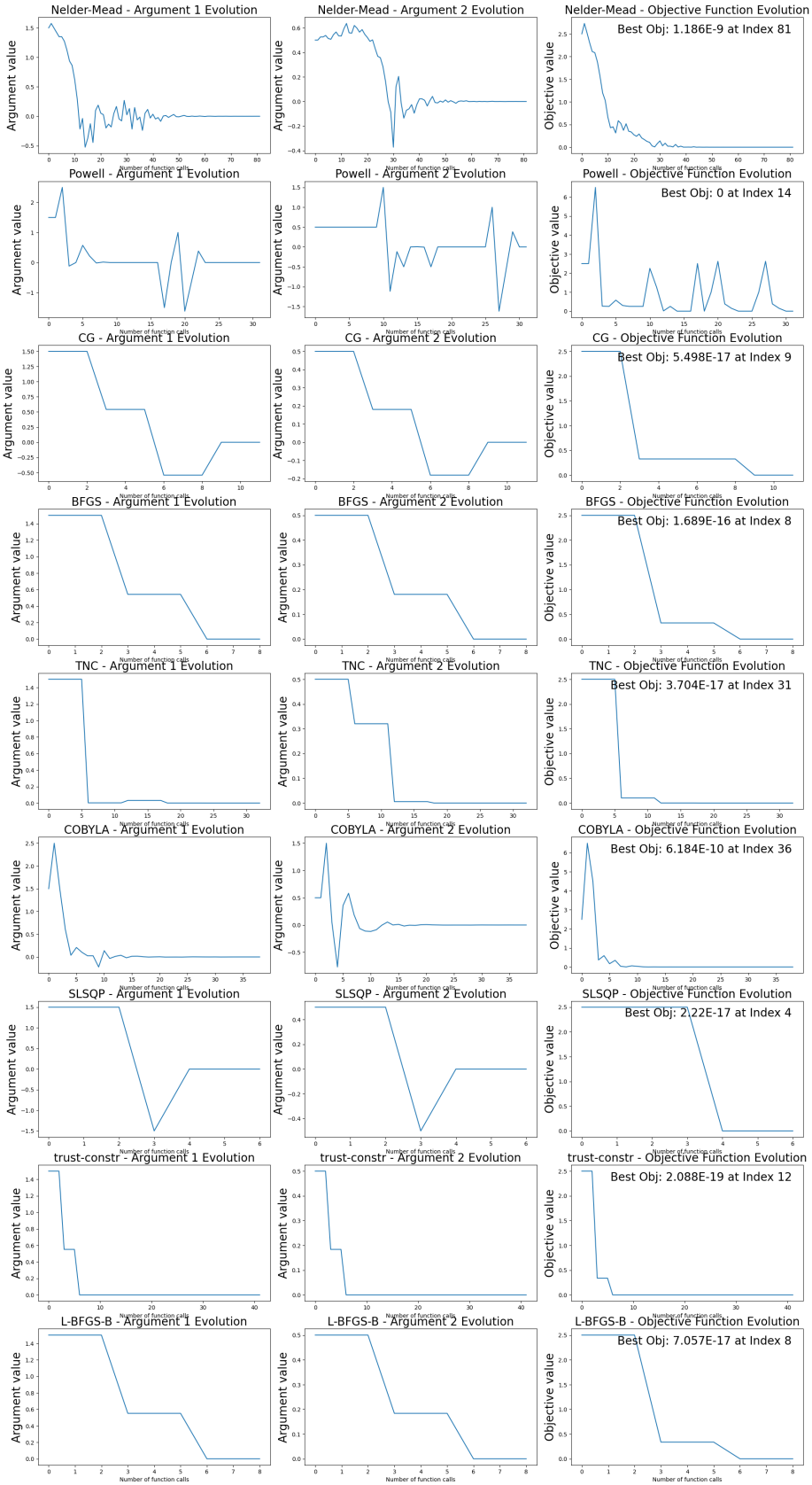


Figure 15: Calls and objective function Values on a 2D convexe function

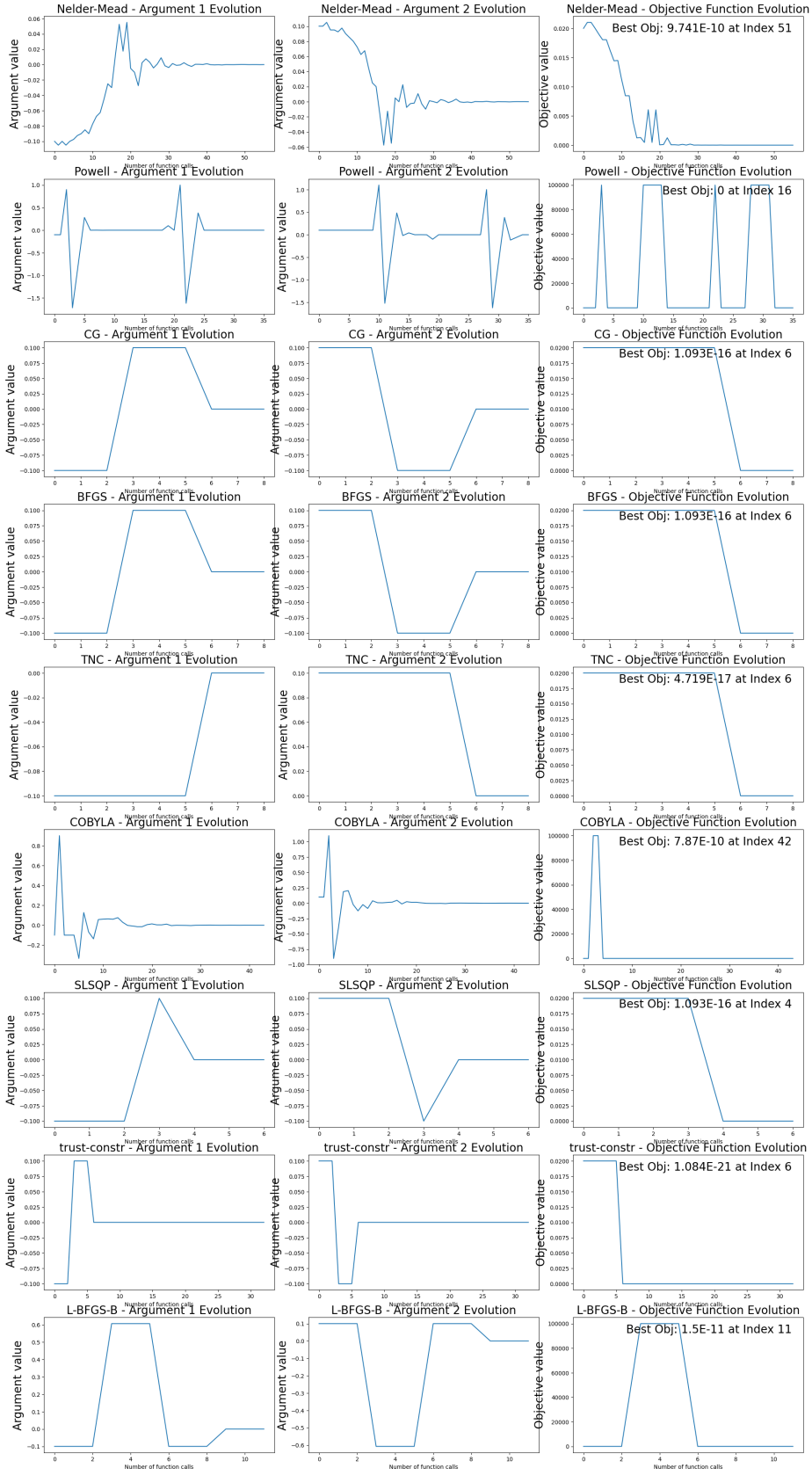


Figure 16: calls and objective function values with inf values if not admissible