Development of a Simulation Environment for Space Magnetic Sensors

Maxime RONCERAY

IUT d'Orléans - Computer Science Departement

Université d'Orléans

26/02/2024 - 14/06/2024

Internship Supervisor: Malik MANSOUR

Co-Supervisor: Claire REVILLET

Academic Advisor: Julien ARSOUZE

LABORATOIRE de PHYSIQUE des PLASMAS
Centre National de la Recherche Scientifique - École Polytechnique
Route de Saclay F-91128 PALAISEAU CEDEX,
https://www.lpp.polytechnique.fr/index.php

Preamble

In the context of my internship report, we raised a question regarding the language in which it should be written. Although both the IUT and LPP are French-speaking institutions, a significant portion of the documents produced in the laboratory are in English. This practice enhances the longevity of the documents and facilitates communication with various foreign individuals present in the laboratory.

Given that many of the findings and analyses in my report may later, it is practical to write the report in English. This approach avoids the need for subsequent translation and ensures that the document is readily accessible to a broader audience.

For these reasons, I have chosen to write my internship report in English.

Acknowledgements

I would like to express my gratitude to several individuals who played a role during my internship:

Firstly, I extend my deepest thanks to Malik Mansour, my internship supervisor. I would like to thank him for his dedication, patience, and the amount of time he invested.

I am also profoundly grateful to Claire Revillet, my co-supervisor, who provided me with essential guidance on computing aspects. Her willingness to dedicate time and her kindness have greatly enhanced my experience.

I must also acknowledge Dominique Alison, a colleague in the office, who was very helpful with his electronic knowledge and with whom I shared many enjoyable coffee breaks.

Lastly, I thank the entire team at my office and the laboratory at large for their cheerful disposition, which created a welcoming working environment.

CONTENTS

CHAPTER 1

INTRODUCTION

As a third-year student in the Computer Science department at IUT of Orléans, I had the opportunity to complete my end-of-year internship at Laboratoire de Physique des Plasmas (LPP), a research laboratory. My role during this internship was to contribute to the development of tools to simulate magnetic sensor behaviours. These simulations will be used by scientists and engineers when designign a magnetic sensor for a project.

The primary goal of this internship was to immerse myself in a professional environment while applying the knowledge and skills I have acquired during my studies.

My goal was to develop tools for magnetic sensor simulation by re-implementing a model and creating both a modular graphical interface and a flexible calculation engine, ensuring the application's and the model's future evolution.

In this report, I will first provide an overview of the project, then describe the sensor and the software I worked on. I will then detail the working methods I employed and the challenges I faced. After that, I will elaborate on the work I accomplished and the details of the project. To conclude, I will provide a human and technical assessment of my internship experience at the LPP.

CHAPTER 2
INTERNSHIP FRAMEWORK

2.1 Context

2.1.1 CNRS Presentation

The Centre National de la Recherche Scientifique (CNRS) is the largest public research organization in France. Its mission is to produce and diffuse scientific knowledge. It was established on October 19, 1939, to encourage and coordinate scientific research in all fields. The founders of the CNRS are Jean Perrin, a Nobel Prize-winning physicist, and Jean Zay, who was the Minister of National Education at the time. Today, the CNRS is one of the largest research organizations in the world.

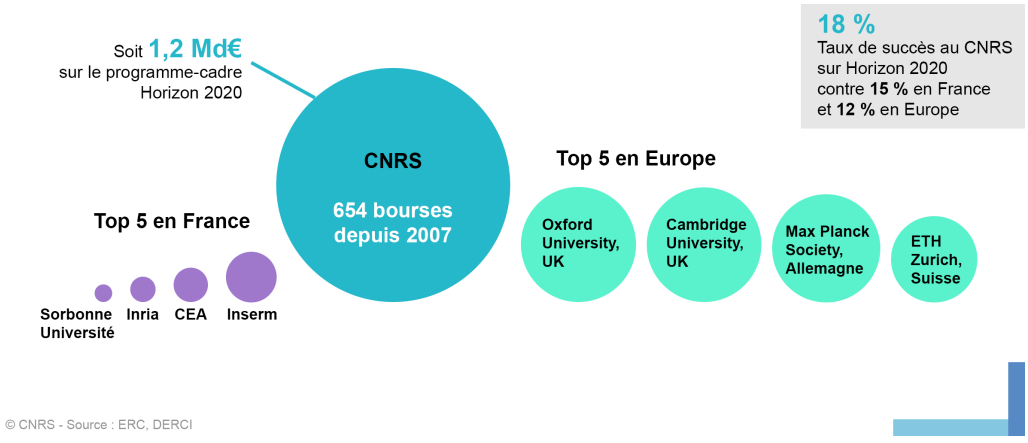


Figure 2.1: The importance of CNRS in Europe, Extracted from: [1]

The history of the CNRS began during World War II, and despite the challenges created by the German Occupation, the organization continued to develop after the Liberation. Over the years, the CNRS has established close relationships with other research actors, including universities, and has diversified its areas of expertise.

The CNRS focuses on six main objectives:

- Supporting high-level fundamental research in all fields.

- Encouraging interdisciplinarity, especially for major societal issues.
- Collaborating with industry and the economy on significant innovations.
- Strengthening the presence of French research internationally, particularly in major European projects.
- Working with autonomous universities to strengthen partnerships.
- Providing scientific expertise to policymakers and society at large.

The CNRS is organized into institutes that cover a wide range of fields of study, including mathematics (INSMI), physics (INP, IN2P3), biology (INSB), chemistry (INC), engineering sciences (INSIS), Earth and universe sciences (INSU), humanities and social sciences (INSHS), and many others. The LPP, as a laboratory under the supervision of the CNRS, focuses on plasma science. Its research covers topics such as Cold Plasmas, Fusions Plasmas and Space Plasmas.

2.1.2 Presentation of the LPP

After setting the scene and the central role of CNRS in scientific research, we can now focus more closely on the Laboratoire de Physique des Plasmas (LPP).

The LPP is a Mixed Research Unit (UMR) that focuses on research in the fields of plasma physics.

The LPP was established in January 2009 through the merge of the Laboratoire de Physique et Technologie des Plasmas (LPTP) and the Centre d'étude des Environnements Terrestre et Planétaires (CETP). The LPP conducts research across all fields of plasma physics, from hot to cold plasmas and from laboratory to space plasmas, using theoretical, simulation, and experimental approaches. The LPP is now located in Palaiseau, next to the Ecole Polytechnique (l'χ) and Paris (Jussieu campus).

The LPP's objectives are to make significant contributions to major contemporary international projects in plasma physics, including space plasma research into the sun-earth and other planetary systems, and the ITER project, where magnetically confined hot plasmas will be harnessed to achieve controlled nuclear fusion. The LPP is also heavily involved with plasma technologies such as plasmas for nanotechnology, pollution abatement, and plasma sources.

Whether in astrophysics or laboratory settings, the focus at the lab is the study of plasmas. In all the LPP research teams, the primary research goal is understanding the physics of plasmas. This ionized gas, either fully or partially ionized, is the fourth state of matter and the least known. On Earth, plasma is rare (lightning is an example), but a few hundred kilometers above, the ionosphere is made of plasma. In the universe, 99% of observed matter is in the plasma state. Plasmas are also found in everyday items like fluorescent lamps and plasma TVs.

Plasma physics is essential for many technologies. Over half of the processes to make microchips are done in plasma reactors, and plasma thrusters are seen as the best option for a manned mission to Mars. Plasma technologies also help reduce pollution and generate high power microwaves and X-rays. Lastly, understanding plasma physics is crucial for studying

systems used in thermonuclear fusion research.
[4]

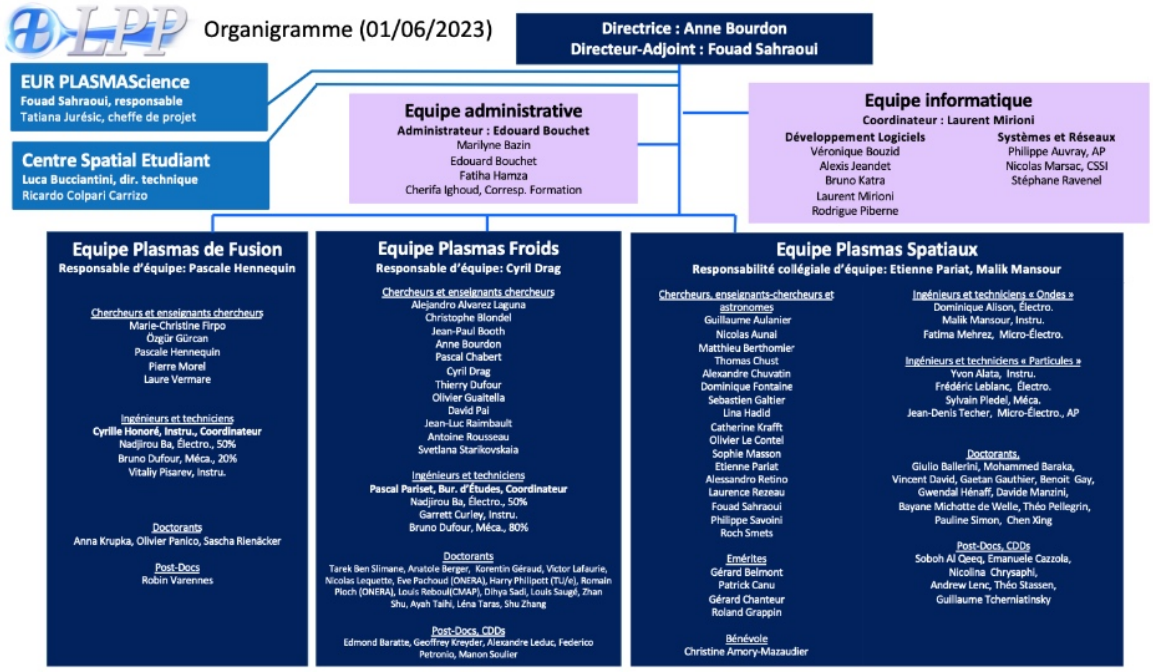


Figure 2.2: Organisational Chart of the LPP[4]

The laboratory is organised around three main scientific teams: Fusion Plasmas, Cold Plasmas, Space Plasmas, and also an Administrative and IT team. Each team is supported by a range of engineers and technicians.

I had the opportunity to complete my internship within the Space Plasmas team, specifically under the supervision of Malik Mansour. My work involved close interaction with the Applied Computing team, particularly in collaboration with Claire Revillet from another laboratory, the LPC2E. At the LPP, each member of the lab can be involved in many different projects. Specifically, my internship advisor was involved in the JUICE space mission and is currently engaged in the HelioSwarm and Plasma Observatory (PO) projects. I will delve into more details about my role and the work I accomplished later in the report. However, before diving into these details, it is essential to understand the general context of the PO project in which my internship was embedded, in order to apprehend its scientific interest.

2.2 Presentation of the Plasma Observatory Project

The Plasma Observatory (PO) is a mission proposed by the plasma physics community to the European Space Agency (ESA). This mission comprises a constellation of seven spacecraft designed to explore the environment of electrically charged particles, or plasma, surrounding Earth. This mission is centered on two fundamental inquiries: how are particles energised within space plasmas, and what processes govern the transport of energy and mediate interactions among various regions of Earth's magnetospheric system? The Plasma Observatory aims to augment ESA's ongoing and forthcoming missions that examine the interactions between the Sun and Earth. This initiative will enhance our comprehension of how the solar wind impacts our planet, with the ultimate goal of safeguarding life and technology from its adverse effects.
[5]

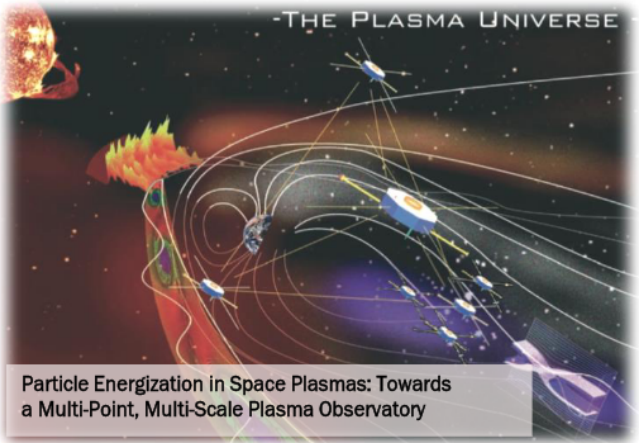


Figure 2.3: Plasma observatory presentation picture [6]

To date, various missions have attempted to study space plasma, but most relied on limited measurements from single locations. A breakthrough was achieved with multi-point missions using several satellites, yet even these were unable to simultaneously measure at different spatial scales comprehensively.

The PO project aims to deploy a constellation of seven satellites to form a network that can carry out multi-scale measurements. This network includes a central hub satellite and six smaller satellite nodes that vary in distance from each other and the central hub. Each satellite will carry many different sensors :

- A Fluxgate Magnetometer (FGM) measures the intensity and direction of the ambient magnetic field, enabling measurement of the magnetic field and its low-frequency frequency fluctuations,
- A Faraday Cup (FC) is used to measure the flow of charged particles, or plasma, in space, which helps us to understand the density, temperature and speed of these particles.
- A Search Coil Magnetometer (SCM) that is used to measure very weak magnetic field variations.

2.3 Introduction to the SCM

Speaking about sensors, the LPP is specialised in designing specific sensors called Search-Coil Magnetometer (SCM). This is a passive induction sensor. To provide context, let's begin with an overview of what a sensor is.

2.3.1 What is a Sensor?

A sensor is a device that detects and responds to some type of input from the physical environment. The specific input could be light, heat, motion, moisture, pressure, or any one of a laege number of other environmental phenomena. The output is generally a signal that is converted to human-readable display at the sensor location or transmitted electronically over a network for reading or further processing. Globally, a sensor is a device that converts a physical value to another physical value.

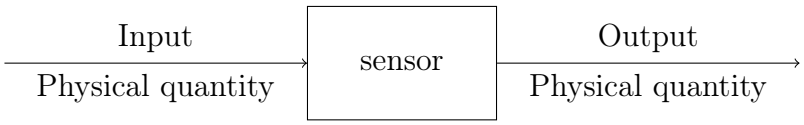


Figure 2.4: Sensor framework

2.3.2 The SCM Sensor

The SCM sensor specifically converts the flux variation of the magnetic field $\overrightarrow{H}$ into an electric voltage. Its behaviour is based on the physical phenomenon of induction.

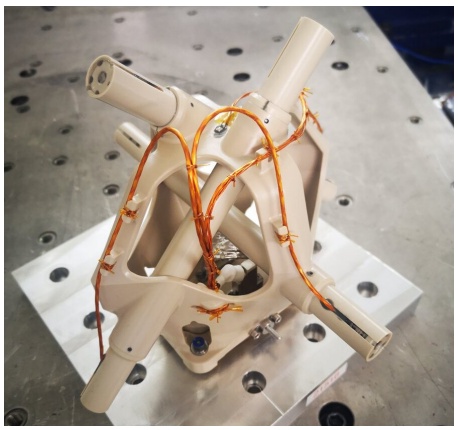


Figure 2.5: Structure of the SCM Sensor from JUICE mission [2]

2.3.3 Physical Principle: Induction

The induction is a characteristic of electrical conductors, like coils of wire, that allows them to store energy in a magnetic field. Essentially, when the a magnetic field current flowing through a conductor changes, it generates an electromotive force (voltage) across the conductor as well.

Lenz's Law

Lenz's Law is a fundamental principle that describes how induced electromotive forces oppose the change in magnetic flux that causes them. Mathematically, Lenz's Law is expressed as:

$$\mathcal{E} = -\frac{d\Phi_B}{dt}$$

where $\mathcal{E}$ is the induced electromotive force, and Φ_B is the magnetic flux. The negative sign indicates that the induced electromotive force generates a current whose magnetic field opposes the change in the original magnetic field. [3]

2.3.4 Implementation

The SCM consists of two coils and a core:

Coil 1: Pickup Coil

The pickup coil generates a voltage resulting from the variation of the magnetic flux through it, according to Lenz's Law. Lenz's Law states that the direction of the induced electromotive force will be such that it opposes the change in magnetic flux that produced it. The pickup coil

behaves as an intrinsic electrical circuit, functioning as a resonant circuit (RLC circuit). This resonant behaviour limits the sensor's application over a wide frequency range. Therefore, the signal needs to be conditioned.

Coil 2: Feedback Coil

To condition the signal, we use the feedback coil, which provides counter-reaction flux to cancel all or part of the magnetic field. This helps to extend the usable frequency range of the sensor.

Magnetic Core

The magnetic core serves a dual purpose: it amplifies the magnetic field for the pickup coil and provides selective feedback around the resonance frequency to ensure stable operation.

2.3.5 Characteristics of the SCM Sensor

To fully understand this sensor and its performance, we need to consider its NEMI (Noise Equivalent Magnetic Induction), impedance, electrical noise and transfer functions. These characteristics will be discussed in detail later in this report.

2.4 Presentation of the Need

After exploring the characteristics of the SCM sensor, it is essential to understand the complexities involved in designing a high-quality sensor.

2.4.1 The Modelling process

Modelling is an important step in the design of sensors. It involves creating a mathematical representation, to predict and analyse the behaviour of the sensor under various conditions. This process is essential for refining the sensor design and ensuring it fit to scientific requirements.

2.4.2 How to Model

The choice between using analytical equations or numerical simulations depends on the specific requirements of the project. Analytical methods provide a direct understanding of the relationships and dependencies within the sensor, while numerical simulations offer a more comprehensive analysis by considering a wider range of variables and complex interactions that may not be easily captured analytically, generating a more accurate result. Sometimes even a mix of analytical and numerical simulations can be an option.

2.4.3 Using the Model

Understanding the sensor's characteristics, such as its resonant circuit properties is fundamental. It is very practical to have tools that allow for the easy manipulation and visualisation of how changes in the sensor's physical and geometrical parameters affect its performance. This is then used for optimising sensor design.

Now that we have a good understanding of the sensor modelling process and its importance, we can move on to discussing the specifics of my internship tasks and objectives.

CHAPTER 3
INTERNSHIP MISSION DESCRIPTION

3.1 Internship Topic

The subject of my internship is :
Development of a Simulation Environment for Space Magnetic Sensors

During my placement, I was tasked with developing a magnetic sensor simulation software : PLASMAG. More specifically, the aim was to design and produce a user interface and a modular engine with a view to initiating the development of a long-lasting software.

3.2 Work Environment

3.2.1 Human

My office was located on the ground floor of the building and was shared with three other colleagues, including Dominique Alison, an electronics engineer who was very helpful. My office was not far from my supervisor's office, making communication and interaction easier.

3.2.2 Technical

My work environment consisted of a simple setup: a portable computer connected to the internet and a desk. Given that my internship focused on simulations, this setup met all my needs.

The computer operated on Fedora 29, which, although different from the systems I used in class, was familiar to me from my previous internship at LPC2E. With assistance from the IT team, I was able to freely install my entire working environment, ranging from the email system to the Integrated Development Environment (IDE).

As I was working with python, for programming, I chose to install PyCharm, which I was accustomed to using.

To manage and version-control my code, I initially used "Forge OSUC," a Git repository hosted by the LPC2E. However, as the project expanded and needed to be accessible to a broader community towards the end of my internship, we migrated all releases to a public GitHub repository under the LPP's organisation (https://github.com/LaboratoryOfPlasmaPhysics/PLASM

Also, to continue the software development with good version control and high code quality, I employed a static code analyser, *Pylint*. This tool provided reports on the overall quality of

my code, including function names, spacing, and line lengths. Throughout the development process, I aimed to adhere to certain coding guidelines intrinsic to the project, such as organising code into modules and standardising the naming of variables, methods, and classes.
To document these rules and the software's functionality, I was responsible for creating various documents:

- A User Manual, explaining how to use the software
- A general presentation of the application.
- A Contributing Guide, for developers or scientists who may wish to contribute to the project.
- A summary of the model equations in LaTeX
- A comprehensive technical documentation, using the python module Sphinx, continuously updated and available online at https://plasmag.readthedocs.io/en/master/

All these documents were written in English using Markdown, reStructuredText (.rst), or LaTeX, formats that allow an easy compilation and deployment of the documentation.

3.3 Related Constraints

This section delves deeper into the constraints I faced during my internship, which are categorised into three key areas: Time, Quantity, and Quality.

3.3.1 Time Constraints:

I was faced with a significant time constraint, as I only had four months to achieve the project objectives.

3.3.2 Quantity Constraints:

The main quantitative constraint involved the requirement to produce tangible products, a functional application or usable modules.

Requirements Specifications

I worked with a technical specifications document that outlined 25 key points the project needed to meet. An overview of these specifications is available in the annex.

The primary goals for the software included developing a simulation engine, a Python wrapper for Simulation Program with Integrated Circuit Emphasis (SPICE) and FreeCAD[1], and a graphical user interface (GUI).

As we will discuss later in the report, most of these targets were met, except for the FreeCAD integration. My preliminary studies indicated that integrating FreeCAD into the app might be too ambitious. Thus, following discussions with my advisor, we decided to put this idea. Instead we discussed adding an optimisation module to the application. This module would reverse the usual process by not just allowing the user to tune parameters for a good design but instead automatically finding a good design that meets certain performance criteria, before

[1]simple CAD software widely used in the industry

tuning by hand. This was an area of interest for me and we agreed to allocate time at the end of the internship to explore this option. The optimisation process will be explained more in details in its own section.

We also organised the objectives and clarified my work plan for the 16 weeks, all around and updated specification file provided in appendix **??**.

3.3.3 Quality Constraints:

The software developed during the internship needed to meet professional or near-professional standards, suitable for use by other scientists or developers.

This required well-documented and commented code with a robust module architecture. More than just coding, it was mandatory to develop a modular software. In the next section, we'll focus onto the modularity aspect. However, the essential requirement was to create a tool and engine that would permit evolving models for the sensor. It was crucial that my development work did not obstruct future software evolutions, regardless of the technical choices made.

This leads us into the next section of the report, which involves the actual system design and development.

CHAPTER 4

SYSTEM DESIGN AND DEVELOPMENT

4.1 Work Done

After 16 weeks of development, we have made substantial progress with our application. The screenshot below provides an overview of almost all the important features. The application is organised into sections: parameters/strategy selection/optimisation, SPICE, and display. Each section was a part of the development process and will be explored in detail later in this chapter.

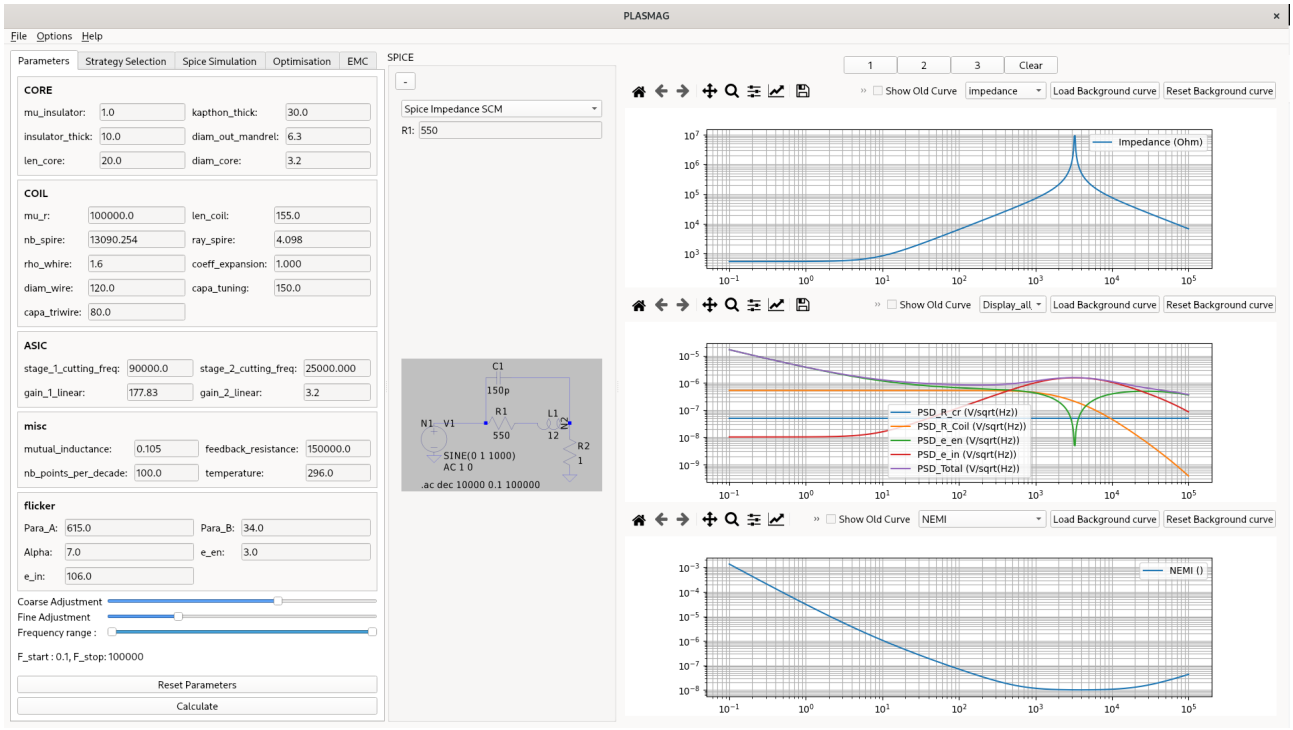


Figure 4.1: Application overview

4.1.1 Preliminary Studies

Before diving directly into the development of the application, it was important to define an approach strategy. The preliminary studies played an important role in structuring my work plan. During the first week, I dedicated most of my time to research and explore the feasibility of the different features.

Understanding SPICE

Initially, my focus was on understanding Simulation Program with Integrated Circuit Emphasis (SPICE) : what it is, how it functions, why it is used, and its advantages. Understanding how SPICE work can be complex and I'll be addressing this topic in the dedicated chapter. After a few experiments using online documentation and the knowledge of my electronics colleagues, we concluded that implementing SPICE was quite feasible.

Interfacing with FreeCAD

My next task was to determine the possibility of interfacing with the FreeCAD software. The general idea was to utilise geometric parameters for simulations within the application and in FreeCAD to verify the mechanical feasibility of the sensor's design. However, after several tests, even though FreeCAD can be interfaced and wrapped with Python, we decided to set this idea aside. It would have required a disproportionate investment for a *non-essential* feature, and it was more valuable to use that time to develop other features.

Expanding Knowledge

Throughout my internship, I tried to deepen my knowledge of electronics and sensor physics. Although my academic background included many relevant courses, I had never had the opportunity to use this knowledge for real-world applications. While not strictly a *Preliminary Study*, this ongoing study was crucial. Understanding the broader context and behaviour of the sensor and its simulations was necessary to develop an application that would meet the client's needs and deepen my understanding of the field.

Naming PLASMAG

Another part of the preliminary studies *(often during coffee breaks)* was brainstorming a name for the project. After many discussions and numerous propositions, the name "PLASMAG" was chosen, standing for Python Library for Accurate Space Magnetometers Adjustments with Genetics. This name was initially used for the first release and was later updated at the end of my internship to be more generic and omit "genetics," as I implemented various optimisation techniques, not solely genetic ones. During my free time, I used my Photoshop skills to create a logo with the updated name.

Figure 4.2: PLASMAG logo

The logo represents a fusion between the well-known Python logo and an antenna of a

search-coil (a copper wire wound around a core). For the remainder of this document, I'll refer to the application as PLASMAG.

4.1.2 Technology Study

Now that the problem has been clearly defined, and I had a clear understanding of my tasks, the second part of the first weeks involved addressing any technical challenges that might arise. It was essential to have a clear idea of the libraries I would use, the methods I would employ to find solutions, and the architecture I would use for the application.

Library for SPICE wrapping

We identified the *PySpice* library, which wraps the Simulation Program with Integrated Circuit Emphasis (SPICE) engine, enabling its easy implementation in Python within PLASMAG. This library was perfectly suited to our needs due to its user-friendliness and extensive online documentation.

GUI Development and Library Selection

After prototyping the graphical user interface (GUI) with various libraries. Discussions with my technical advisor, Claire Revillet, led to the selection of PyQt6. Despite its apparent complexity, PyQt6 offers extensive customisation options and is well-suited for medium to large projects. My experience with PyQt from a previous internship was also beneficial.

Choice of Scientific Libraries

Initially, I considered using specialised libraries like XArray, which supports parallelised dynamic labelled arrays. However, due to its hight learning curve and minimal advantages for our needs, we decided to use more generic packages: NumPy and pandas for data structure and scientific computations, and Matplotlib for visualisation.

Application Deployment

An important requirement was the ease of deploying the application on other computers. I chose Anaconda, a robust virtual environment manager, for this purpose. Despite its size, Anaconda simplifies installation via an *Environment.yaml* file and is compatible across all platforms (Linux, Windows, and macOS), making it the optimal choice given my existing knowledge about packaging.

Proof of Concept Development

With the foundational decisions in place, I was able to construct an initial Proof of Concept (PoC) for the application. This involved separately building prototype versions of the following modules:

- An interactive UI prototype that dynamically re-plots on parameter updates.
- An initial version of the Engine with a specific architecture (detailed in the upcoming engine section).
- A PoC demonstrating the use of AI to fit real data, as opposed to using predefined equations, detailed further in the Fit Module section.

- Later in the internship, an Optimisation module featuring a preliminary implementation of a genetic algorithm, validating the feasibility of this requirement.

Agile Development Methodology

Despite not being a *Technology* in the literal sense, before beginning the work, we needed to decide on a work methodology. Due to the short time frame and the proximity of our offices, we opted for an approach that is very similar to *Agile*. The approach I used was as follows:

- Conceptualise a module or feature.
- Develop a PoC or prototype.
- Present the PoC or prototype at weekly meetings or to my supervisor.
- Integrate the concept into PLASMAG.
- Validate the implementation during weekly meetings.

Existing Solutions

An essential part of the technology studies was understanding the existing situation, which comprised two parts:

- Reviewing the existing code for the model that my supervisor had developed during the COVID-19 pandemic.
- Examining existing sensor simulation software. In this phase, I had a close look to *Pyxel 1.0: an open-source Python framework for detector and end-to-end instrument simulation*. Although their sensors differ significantly from ours, their data treatment pipeline, engine, and software architecture provided valuable guidelines. From this analysis, I designed the initial version of PLASMAG, starting with the architecture design and some preliminary sketches.

4.2 Software Architecture

4.2.1 Architecture Overview

Given the requirement for modularity, it was imperative to establish a clear separation between computation and the Graphical User Interface (GUI). To achieve this, I designed the engine to operate headlessly and the UI to function with any engine that fit to the defined data interfaces.

Software Design

To address this requirement, we adopted a simple Model-View-Controller (MVC) architecture. In this framework:

- The **Model** is the engine that handles data processing.
- The **View** is the GUI that is used to interact with the user.
- The **Controller** acts as an interface, between the model and the view.

The diagram below illustrates how the classes are organised within PLASMAG:

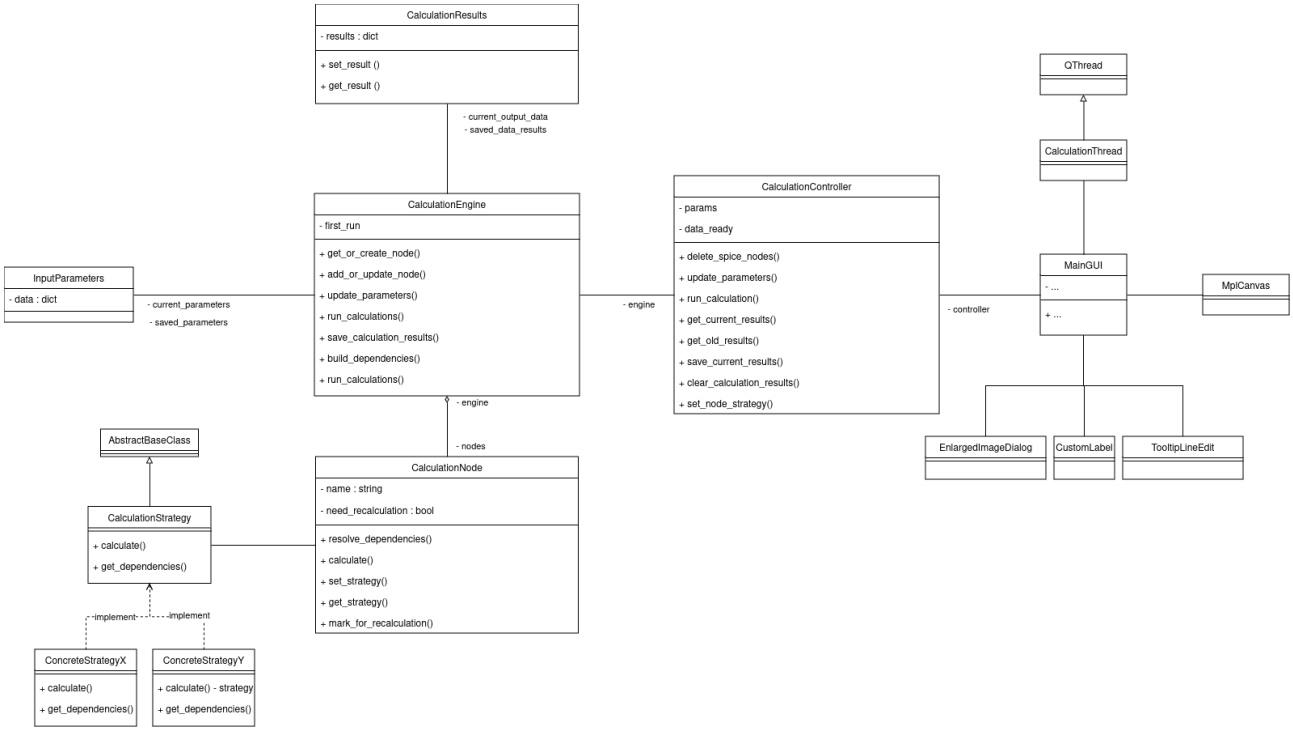


Figure 4.3: PLASMAG Class Diagram realized with draw.io - *Detailed version: A.2*

The three main classes are *CalculationEngine*, *CalculationController*, and *MainGUI*.

4.2.2 GUI Overview

This subsection provides an overview of the GUI, from its creation process to its organisation layout. While this section does not cover every detail, it focuses on the primary features.

The GUI is divided into three main parts:

- **Plot Section** (right side): This section allows the display of various physical values like resistance or impedance over specific x-axes. It is modular, supporting up to five plots, with live options for users to select the displayed outputdata. Additional functionality includes the ability to save and compare up to three different curves with the current simulation results.

- **Parameter Section** (left side): Dedicated to inputting geometrical and physical parameters, this section dynamically generates input fields from a *.json* configuration file. This design makes it easy to modify the model without changing the GUI code. Parameter adjustments are facilitated by sliders at the bottom, as shown in the figures below.

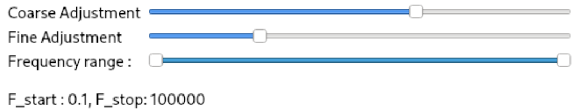


Figure 4.4: View of sliders

```
"diam_wire": {
    "default": 90.0,
    "min": 10,
    "max": 300,
    "description": "Diameter of the wire ...",
    "input_unit": "micrometer",
    "target_unit": "meter"
},
```

Figure 4.5: .json configuration for wire diameter

Units of measurement are handled through the *Pint[8]* library to convert input values into SI units, accommodating various scales without user complexity. For example, entering a value like $3.510^{-12}T$ (Tesla) can be hard, but with *Pint*, users can input values in more manageable units, which are then automatically converted to the standard international units, simplifying the process.

- **SPICE Section** (center): This interactive panel becomes available when electrokinetic simulations using SPICE are activated. Like the parameter section, its inputs fields are dynamically generated and validated to ensure correct data entry.

User Interface Documentation

The PyQt GUI is often a complex intrication of widgets and layouts. Widgets are tangible components that can be customised and interacted with by the user. Layouts describe how these widgets are organised within the GUI—such as in tables, rows, or columns.

Due to the rapid complexity that can arise with PyQt's layouts, maintaining a clear documentation of how these components are structured became essential. To address this, I progressively documented all new UI elements in the layout diagrams. This approach not only helped my development process but also ensures that future developers can easily understand and modify the GUI structure. As the layout is not fixed and may evolve, these diagrams are saved in an *SVG* file format, allowing them to be easily opened, edited, and updated using an appropriate application.

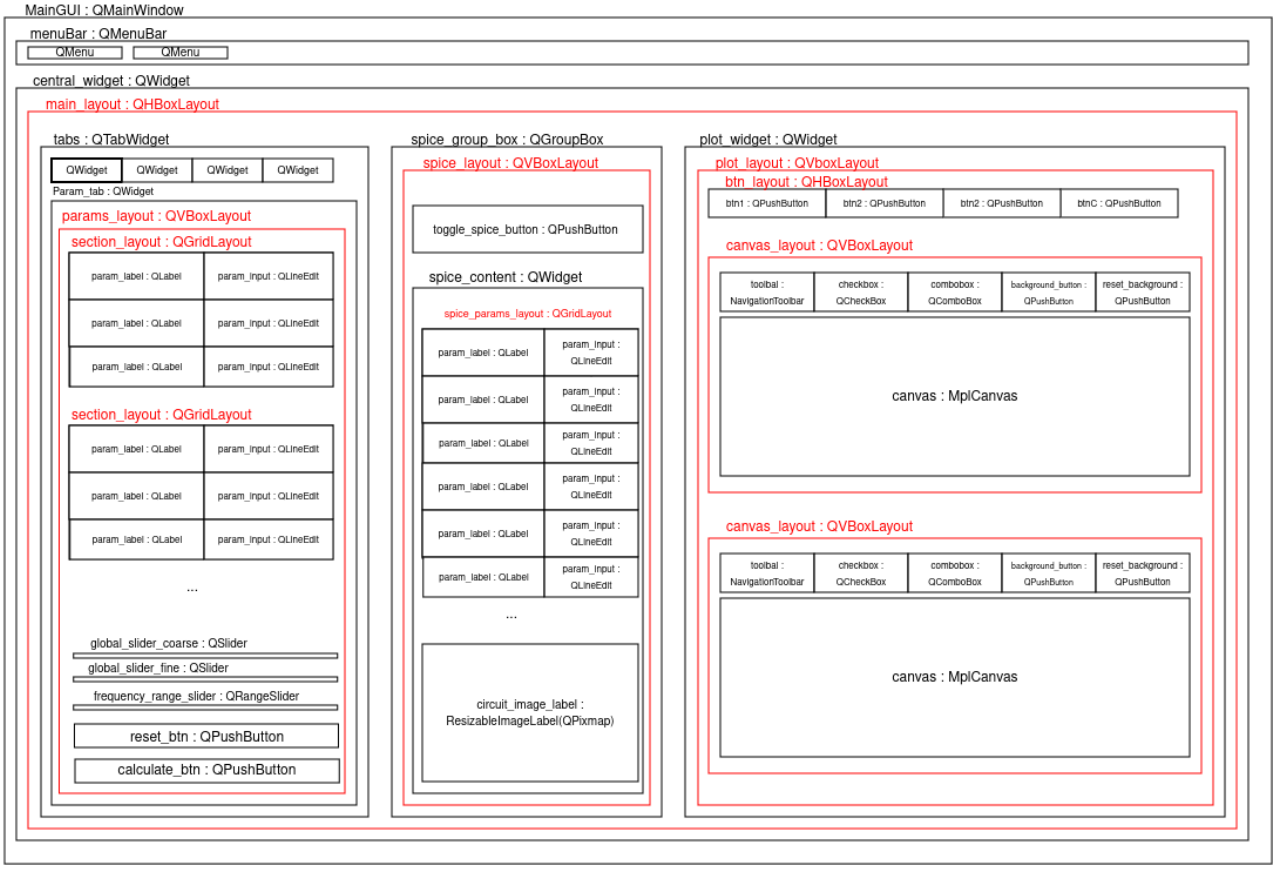


Figure 4.6: PLASMAG Layout Diagram realized with draw.io - *Detailed version: A.1*

Now we have described in detail the structure of the application and we have illustrated how users can interact with the UI. Let us now proceed to the next section to explore in detail how the simulation engine is built.

CHAPTER 5

SIMULATION ENGINE DESIGN

This section discusses the architecture of the simulation engine, explaining the reasons for choosing this specific design and its benefits. The guiding principle during its development was **Modularity**. The goal was to allow the evolution of the computational model by maintaining modularity while maintaining near-real-time processing capabilities.

5.1 Problem

The simulation involves numerous parameters and products that are highly interconnected. To illustrate, consider four main equations: Resistance, Inductance, Capacitance, and Impedance:

Equations

$$\textbf{Resistance} \quad R = N \cdot (2\pi r_s) \cdot \rho$$

$$\textbf{Inductance} \quad L = (4\pi \times 10^{-7}) \cdot \mu_{\text{app}} \cdot N^2 \cdot (\pi \cdot (R_s)^2) \cdot \frac{\lambda_{\text{param}}}{len_{coil}}$$

$$\textbf{Capacitance} \quad C = \frac{\pi \cdot (\mu_0 \cdot 4\pi \cdot 10^{-7}) \cdot \mu_{\text{insulator}} \cdot len_{coil}}{(Kt + 2 \cdot It) \cdot N_{\text{layer}}^2} + \left((N_{\text{layer}} - 1) \cdot D_{mandrel} + N_{\text{layer}} \cdot (N_{\text{layer}} - 1) \cdot (D_{wire} \cdot Kt)\right) \cdot \frac{1}{2} + capa\_tuning + capa\_triwire$$

$$\textbf{Impedance} \quad Z(f) = \sqrt{\frac{R^2 + (L \cdot 2\pi f)^2}{(1 - L \cdot C \cdot (2\pi f)^2)^2 + (R \cdot C \cdot 2\pi f)^2}}$$

Parameter Descriptions

- R: Resistance of the coil in ohms (Ω).
- L: Inductance of the coil in henrys (H).
- C: Capacitance in farads (F).
- $Z(f)$: Impedance at frequency f in ohms (Ω).
- N: Number of spires in the coil (unitless).
- R_s: Radius of the spire in meters (m).
- μ_{app}: Apparent permeability of the coil's core material (unitless).
- λ_{param}: Computed as $\left(\frac{len_{coil}}{len_{core}}\right)^{\frac{-2}{5}}$ (unitless).
- len_{coil}: Length of the coil in meters (m).
- $\mu_{\text{insulator}}$: Permeability of the insulator material (unitless).
- Kt: Thickness of the Kapton tape in meters (m).
- It: Thickness of the insulator in meters (m).
- $D_{mandrel}$: Outer diameter of the mandrel in meters (m).
- D_{wire}: Diameter of the wire in meters (m).

- capa_tuning: Tuning capacitance in farads (F).
- capa_triwire: Triwire capacitance in farads (F).
- N_{layer}: Number of layers, calculated as $\frac{N}{N_{\text{spire\_per\_layer}}} + 1$.
- $N_{\text{spire\_per\_layer}}$: Number of spires per layer, calculated from coil length and wire diameter considering the coefficient of expansion K_{exp} as $\frac{len_{coil}}{D_{wire} \cdot K_{exp}}$.
- f: Frequency in hertz (Hz).

See PLASMAG's full model given in TODO add annexe

An analysis of these equations reveals frequent reuse of certain geometric and physical parameters. The computation of impedance (Z) depends on the values of R, L, and C, necessitating their prior calculation. This dependency pattern appears throughout the model, demanding precise sequential computation to ensure accurate results. Modifying the model would create a problem; precise interventions are required at specific lines of code to implement changes or additions, demanding a thorough understanding of the model's code structure.

To visualise the interdependencies among equations, outputs, and parameters, we constructed the following graph:

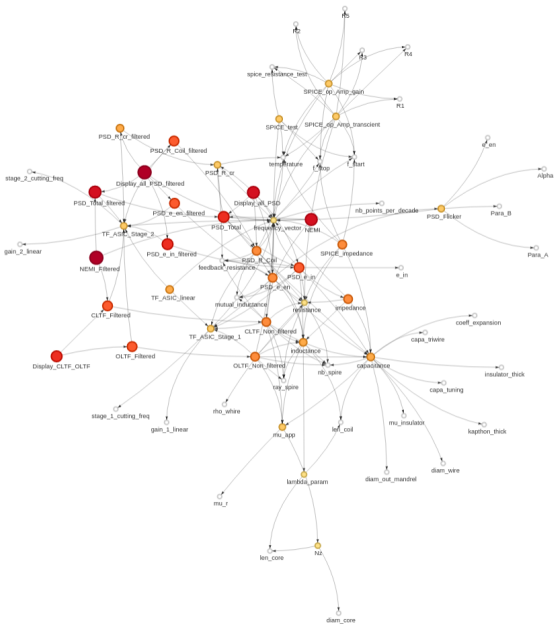


Figure 5.1: PLASMAG model visualisation

In this graph, the lighter nodes represent lesser interdependencies. White nodes are input parameters, while dark red nodes indicate parameters that require multiple computational steps. This graphical representation mimics a tree structure, encapsulating the core idea behind the PLASMAG engine: to represent computations as a graph rather than a linear sequence of steps.

5.2 Design

5.2.1 Overview

From now on, the calculation engine can be called 'graph-based'. The graph-based calculation engine presents a flexible and efficient framework for managing complex computational tasks. By organising calculations as a graph of dependencies, the engine enables dynamic calculation flows, easy updates to calculation logic, and the ability to handle a wide range of computational scenarios. The engine also includes computation optimisation techniques that minimise

recalculations by only updating necessary nodes, as well as supporting parallelisable operations for improved performance.

5.2.2 How It Works

The engine is based on calculation nodes, each embodying a specific computational task (methods to compute the output). These nodes are intricately linked with a calculation strategy, a blueprint detailing both the nature of the calculation and its reliance on the results of other nodes. It orchestrates the sequence of calculations, guaranteeing that all necessary computations are performed in the correct order, based on their inter-dependencies.

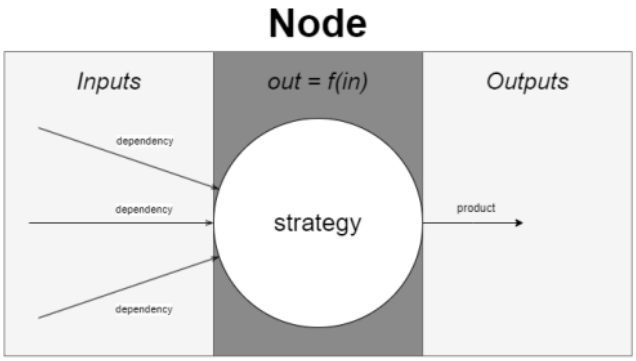


Figure 5.2: Node Framework

Here is a simple representation of a Node. It simply takes parameters as inputs, and act like a black box that outputs results.

Now if we assemble nodes, we can form a graph. Let's take the same example for the computation of R,L,C and Z.

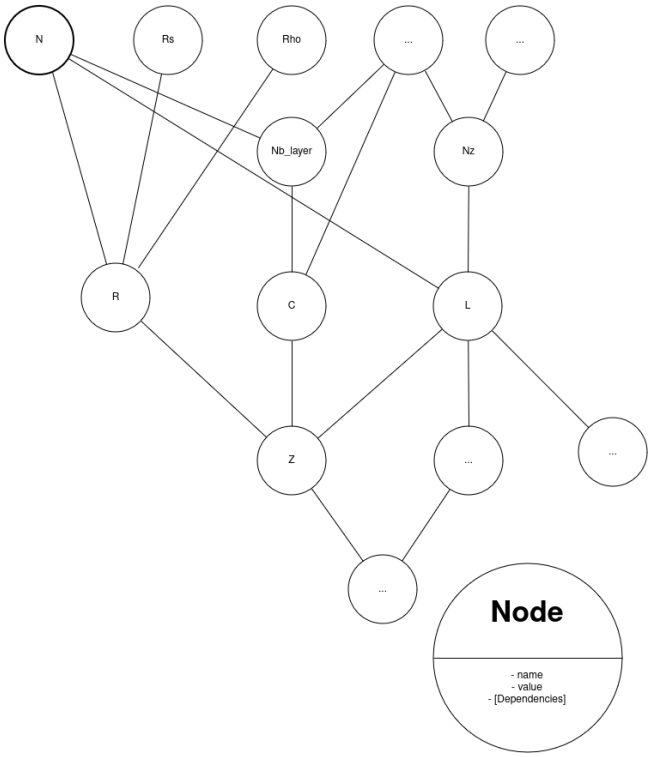


Figure 5.3: Calculation graph example

Figure 5.3 depicts an example of a calculation graph, where each node represents a computational task or a data input. Nodes such as **R**, **C**, and **L** correspond to specific calculations or inputs, while the connections between them illustrate dependencies. For instance, as I mentioned before, node **Z** depend on the values from **R**, **L** and **C**, signifying that **Z**s calculation cannot proceed until **R**, **L** and **C** have been computed.

5.3 Implementation

The engine is implemented in Python, using object programming features to create a versatile and easily extensible framework. Each component is represented by a class, with abstract base classes used to define interfaces for extensible behaviour.

The whole graph framework is open design, it welcome the integration of new calculation strategies with ease. Developers can introduce new strategies that uses already defined parameters. From a class speaking point of view, here is how we can describe the engine :

5.3.1 Key Classes

- **InputParameters** : A container for all input data required by the calculations. It allows for structured access to parameters across the engine.
- **CalculationResults** : Acts as a repository for storing and retrieving the outcomes of calculations.
- **CalculationStrategy (Abstract Base Class - ABC[1])** : Defines the interface for calculation strategies. Each concrete strategy implements the logic for a specific calculation and its dependencies.
- **CalculationNode** : Represents a node in the calculation graph. It's capable of performing calculations using a specified strategy and dynamically resolving dependencies.
- **CalculationEngine** : Orchestrates the entire calculation process, managing nodes, strategies, and data flow.

[1]Library used to implement abstract classes and interfaces in Python

5.3.2 Class Diagram

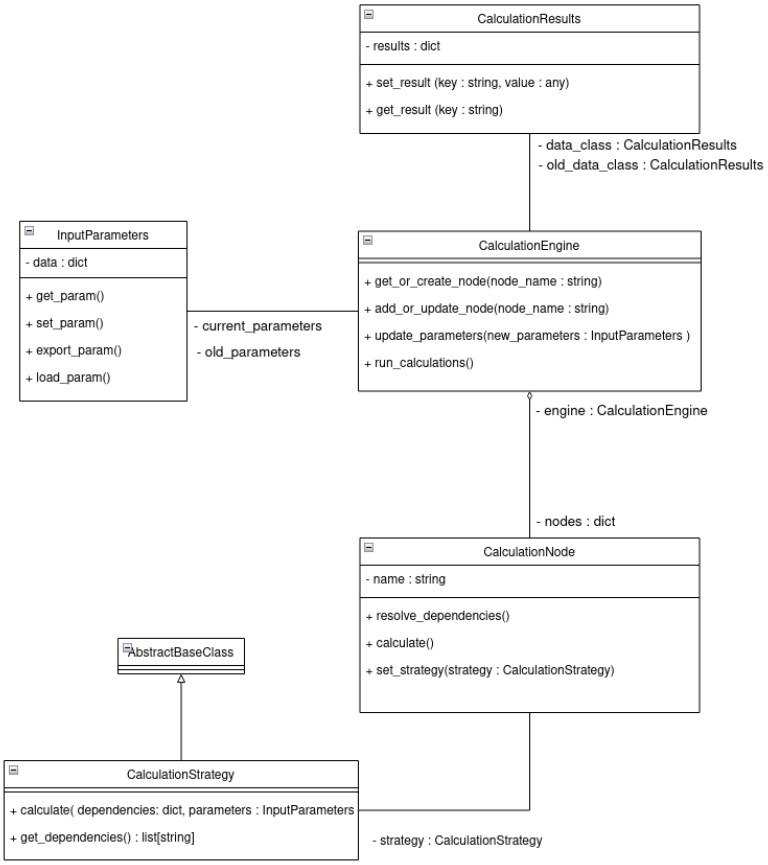


Figure 5.4: Calculation engine class diagram

The class diagram represent an implementation of the strategy pattern within the calculation engine, demonstrating how the *CalculationEngine* manages *CalculationNode* objects. Each node employs a specific *CalculationStrategy*, an abstract class that requires concrete subclasses to define their dependencies and calculation logic.

This approach aligns with the factory pattern, allowing for dynamic instantiating of strategies and nodes.

The *InputParameters* class provides the nodes with the necessary inputs, serving as a data source, while the *CalculationResults* class captures and stores the outputs.

5.3.3 Execution Diagram

The engine instantiation and execution flow are depicted as a four-step process in the following diagram.

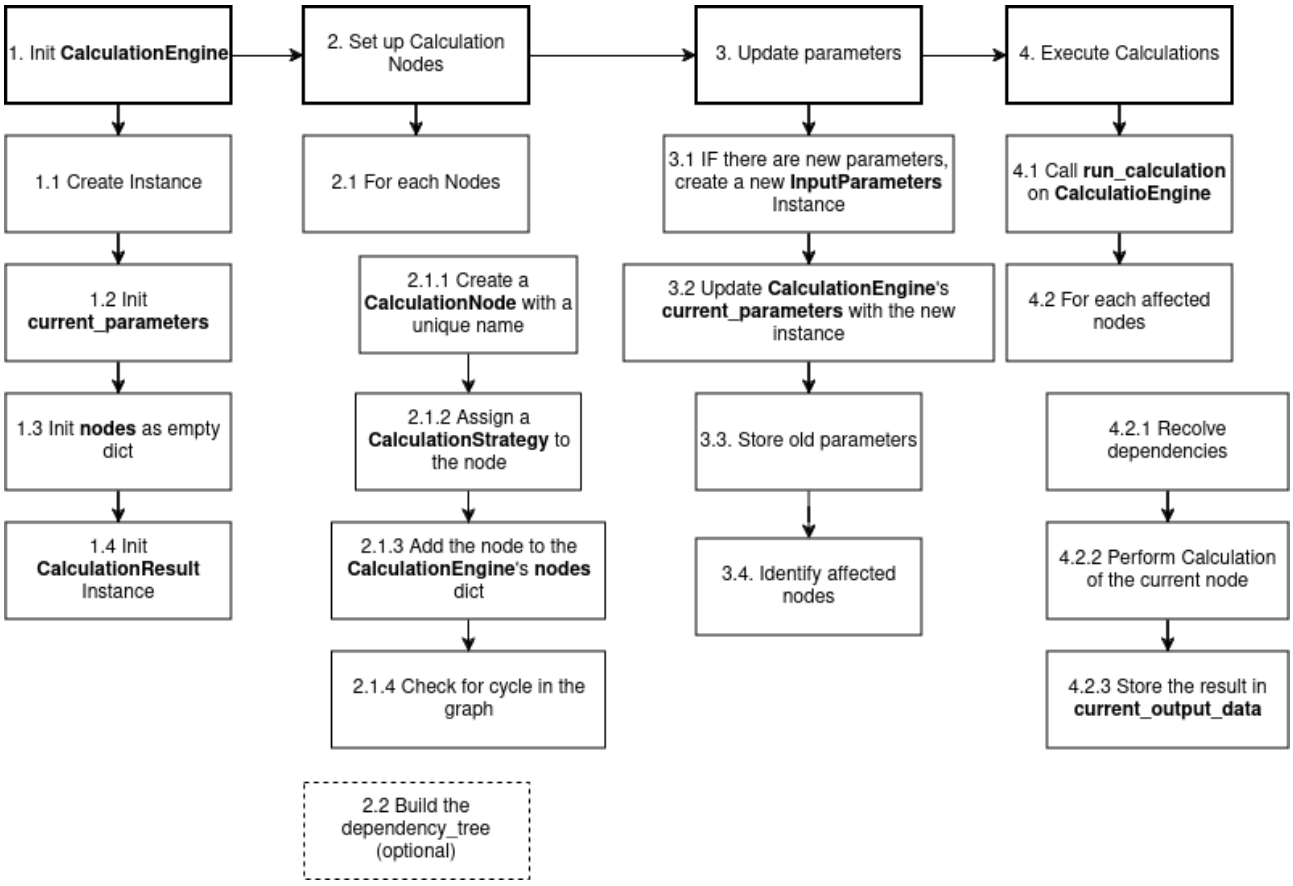


Figure 5.5: Execution diagram

Initially, the CalculationEngine is initiated by creating an instance, initialising the current parameters, setting up an empty dictionary for nodes, and creating a CalculationResults instance to store calculation outcomes.

Next, calculation nodes are set up, where each node is created with a unique name, assigned a calculation strategy, added to the engine's node dictionary, and checked for cycles to prevent infinite loops.

Following this, parameters are updated/defined; if new parameters are introduced, a new *InputParameters* instance is created, and the engine's current parameters are refreshed with this instance while the old parameters are stored.

Lastly, calculations are executed by calling the *run_ calculation* method on the *CalculationEngine*, which resolves dependencies, performs calculations for each node, and stores results in the *current_ output_ data*. This structured approach ensures a systematic progression from engine initialisation to calculation execution, with checks for cyclic dependencies and the ability to handle parameter updates efficiently.

5.3.4 Cycle Check

Implementing new calculation strategies with varying dependencies carries the risk of inadvertently creating cycles, which can lead to infinite recursion. To avoid this issue, the engine incorporates a cycle check mechanism.

The cycle check mechanism utilises a depth-first search (DFS) [9] algorithm to traverse the calculation graph. It detects cycles by marking nodes as visited and keeping track of the current traversal path using a stack. If a node is encountered that is already in the stack, a cyclic dependency is identified, and an exception is raised.

5.4 Engine upgrade

During the development of the engine, I implemented unit tests to ensure that all features function as intended. Alongside these unit tests, I conducted benchmarks to evaluate the engine's performance.

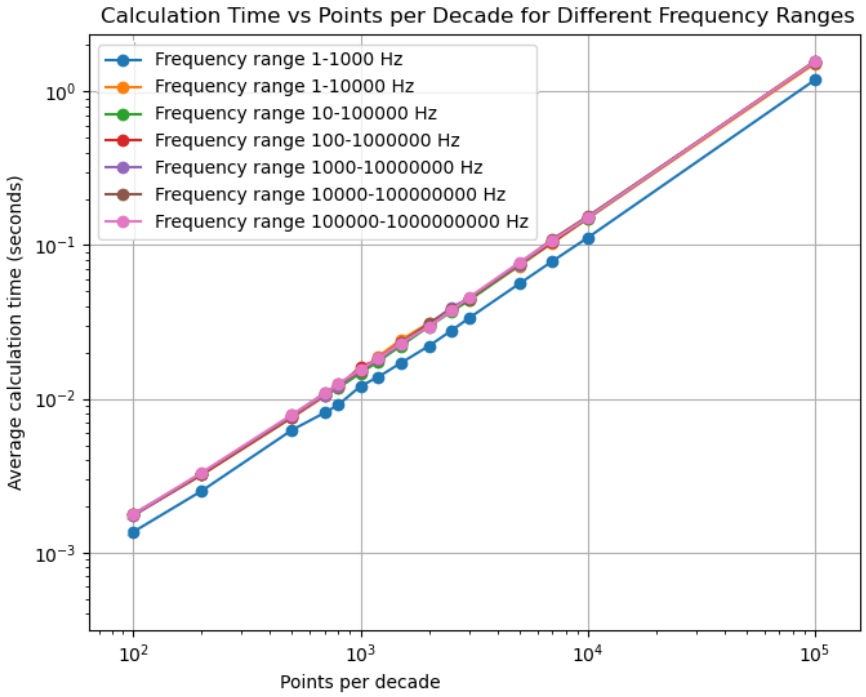


Figure 5.6: Engine First benchmark

This figure represent the computation times for the simulation across various frequency ranges, assessed at different numbers of points per decade. I had the intuition from these initial results that there was potential for significant performance enhancements.

The following graph illustrates the comparative engine performance, evaluated using the same benchmark parameters, before and after applying computation optimisation techniques we will discuss in the following section.

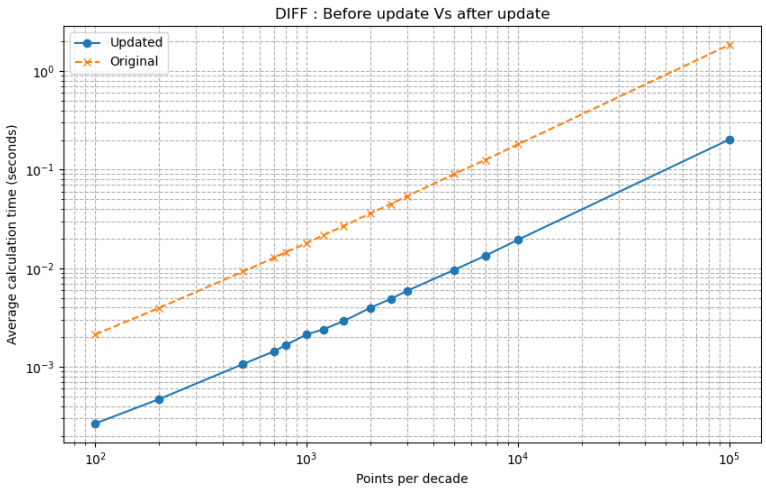


Figure 5.7: Engine Performance Improvement

We can observe that in the worst-case scenario, the computation time remained the same, while in the best scenario, it improved by a factor of 10.

Now, let's explore what contributed to this performance improvement.

5.4.1 How to upgrade the engine?

Instead of recalculating every nodes of the graph when a parameter is updated. I had the feeling that it would be possible to detect and only recalculated nodes that depends on this parameters.

This would addresses the inefficiencies that arise when a single parameter change triggers widespread recalculations throughout the system.

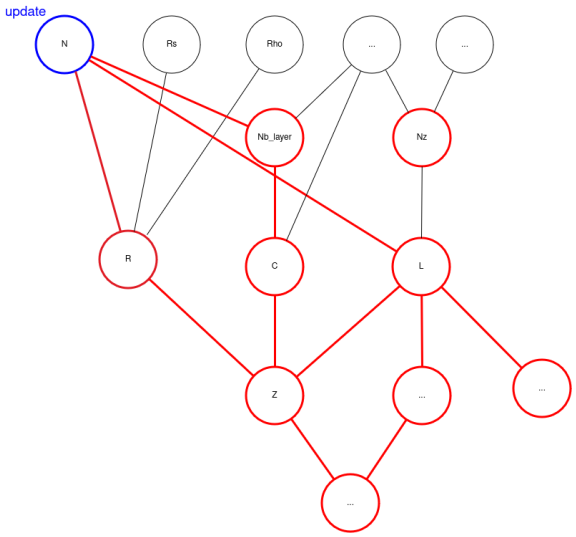


Figure 5.8: All nodes recalculated due to a single change - Worst scenario

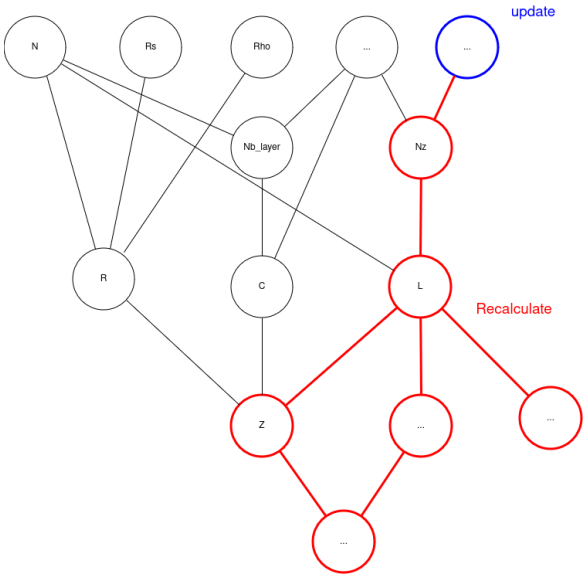


Figure 5.9: Recalculating only directly affected nodes - Best scenario

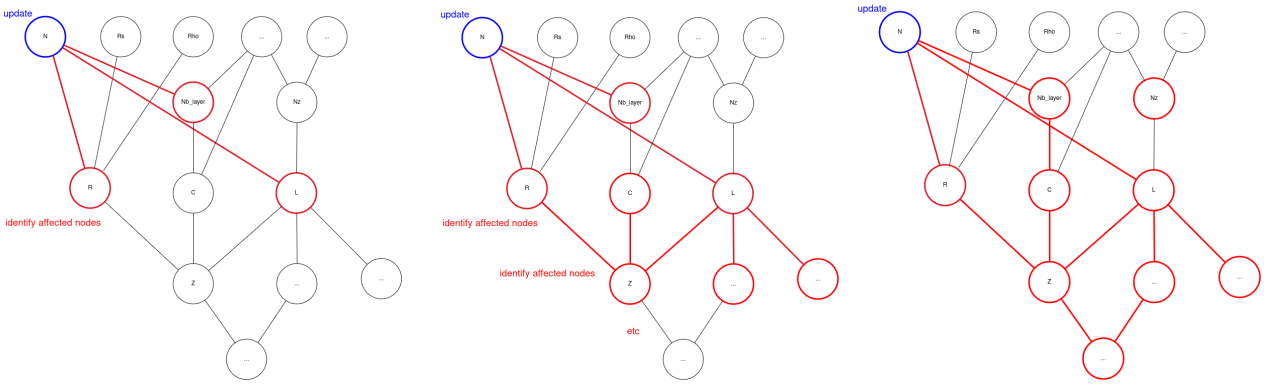
Imagine adjusting a single parameter, and that parameter update to trigger a recalculation across every node in the system (Figure 5.8). This represents our worst-case scenario. But in the best case, if an updated parameter is not highly interconnected, and only used for top level nodes, the difference between recomputing everything and only necessary nodes is huge, because we can just keep the previously calculated values for almost all nodes. This is the best case.

5.4.2 Identify impacted nodes

To improve the performance of our graph-based calculation, I integrated to the engine integrates a dependency resolution mechanism. Upon updating a node, the engine recalculates only those nodes directly dependent on the changed node

Identifying which nodes require recalculation may seem straightforward, as illustrated step by step in Figures 5.10a, 5.10b, and 5.10c. However, this task can become time-consuming.

This process of identifying affected nodes cannot be done in real-time without creating an useless computation load, and it's unnecessary to do so, as the strategies used for each nodes shouldn't change at every computation.



(a) Impact of change step 1. (b) Impact of change step 2. (c) Impact of change step 3.

Figure 5.10: Sequential impacts of a change on the graph.

5.4.3 Introduction to Inverse Dependency Trees

To efficiently navigate this complexity, we employ an inverse dependency tree, similar to a map, describing which node should be re computed when any node is updated. This structure maps out dependencies in reverse, from nodes back to their dependencies.

The inverse dependency tree is essential for tracing back which nodes influence a given node. Here's an illustrative example of what such a tree might look like in JSON format:

```
{  "Z": {
        "R": {
            "N": {},
            "Rs": {},
            "rho": {}
        },
        "C": {
            "R": {
                "N": {},
                "Rs": {},
                "rho": {}
            }  }  }  }
```

Figure 5.11: Dependencies tree

```
"N": ["R", "C", "Z"],
"Rs": ["R", "C", "Z"],
"rho": ["R", "C", "Z"],
"R": ["C", "Z"],
"C": ["Z"],
"Z": [],
```

Figure 5.12: Inverse Dependencies Tree Example

In real use, the trees are way more complex, but this only serve as an example to visualise how such a tree could exist and be used.

It is important to note that in the inverse dependency tree, not only leaf nodes are used as keys. This structure also includes intermediate and higher-level nodes as keys because changes can occur in the computational method of any node, not just the inputs at the leaves. When the calculation method of a node is modified, all nodes that depend on it must be recalculated to reflect these changes. This inclusion ensures that such updates do not necessitate recalculations of nodes closer to the leaves.

5.4.4 Algorithmic Efficiency and Complexity

When it comes to algorithmic efficiency and complexity, we encounter a trade-off between computation and storage costs.

For the Depth-First Search (DFS) algorithm utilised in constructing the inverse dependency tree, its time complexity is $O(V + E)$, where V represents the number of nodes and E the number of edges in the graph. This complexity arises from traversing all nodes and edges to identify dependencies.

Regarding storage costs, the generated inverse dependency tree file incurs minimal overhead compared to the computation cost. The size of the file is proportional to the number of nodes and edges in the graph. Despite this, the trade-off between storage and computation favours precomputing the inverse dependency tree due to the reduced computational overhead during runtime.

As our engine upgrade adjusts the recalculations needed after a change. Instead of recalculating all nodes, we now only recalculate $N - L + 1$ nodes in the worse scenario, where N represents the total number of nodes and L represents the number of leaf nodes directly impacted by the change.

5.4.5 Parallelizable Nodes

Last advantage of working with an tree-based calculation engine : the parallelisation. Within our dependency tree, nodes can be categorized into different layers based on their proximity to the leaf nodes. The concept of layering facilitates efficient parallelization of computations. While we described layers as levels (e.g., first-level nodes, second-level nodes). Nodes within the same layer can execute their computations at the same time, as they operate independently and do not share dependencies within that layer.

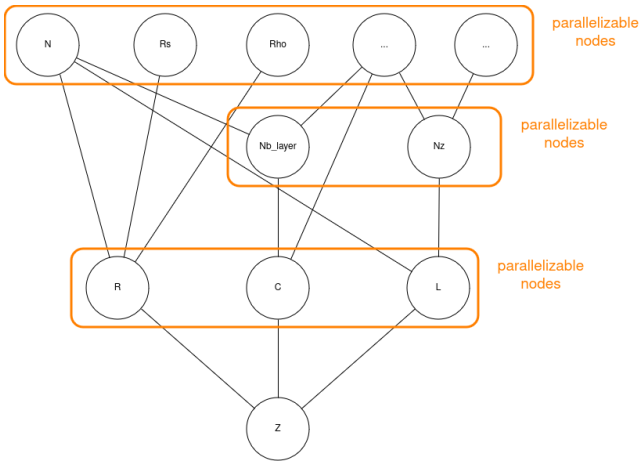


Figure 5.13: Parallelizable nodes categorised by layer proximity to leaf nodes.

5.5 Tree Visualisation

Finally, as PLASMAG tends to evolve in the future, the model tends to be more and more complex. It's important to be able to visualise the whole tree for many reasons. Like knowing which parameters can be used for implementing a new strategies, visualise the impact of the modification of a Node, and so on. Thus I developed two main features to do so.

Firstly, a simple dump to JSON option. In fact with a simple function we can generates a nested dictionary representing the dependency tree, which can optionally be saved to a JSON file. This feature provides a snapshot of the entire calculation graph, useful for debugging and

documentation purposes.

Secondly, a more detailed method using graph representation using *NetworkX* and *Pyvis* [10] [11]:

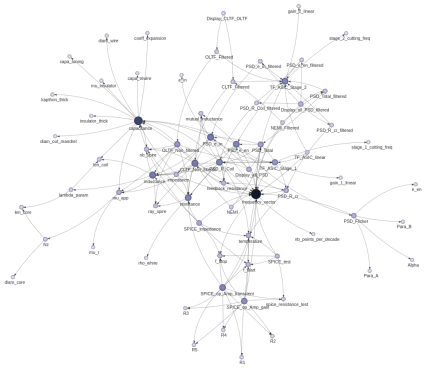


Figure 5.14: Nodes Degree representation in the graph

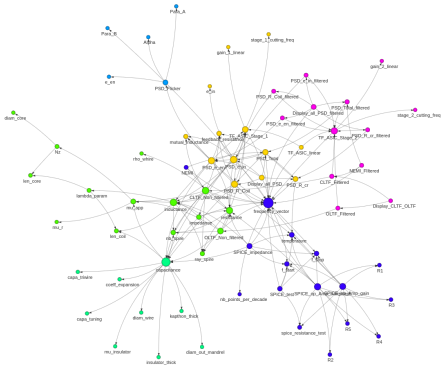


Figure 5.15: Community representation in the graph

Combined with this visualisation method, it's very easy to use some clustering techniques to highlights structural features and community within the graph.

The image on the left, represent the centrality and influence of nodes: lighter colors indicate fewer connections (peripheral nodes or leaves), while darker colors denote nodes with many connections (central nodes). This color-coding is not merely aesthetic but serves a functional purpose. Darker-colored nodes, which represent nodes with a higher number of connections, are critical within the graph as they influence many other nodes. Modifying the computation strategy of these nodes, may impact the results of the whole simulation. Therefore, any recalculations required for these nodes due to parameter changes or updates are likely to necessitate widespread recalculations across the graph.

The image on the right represent the different community in the graph. Nodes from different community are rarely interdependent, this visualisation helps to understand the different parts of the simulation. For example in yellow, all the nodes corresponding to a computation of a Noise can be found, while the dark blue part represent the electrokinetic part of the simulation model, simulated with SPICE.

CHAPTER 6

SPICE

So far, we have discussed only about analytical simulations, which involve systems that can be described by known equations. We will now explore the integration of numerical simulations within PLASMAG. This section aims to detail what SPICE is, why we have incorporated it into PLASMAG, and the process behind its integration.

6.1 Why SPICE

SPICE, an acronym for Simulation Program with Integrated Circuit Emphasis, was developed by the University of Berkeley. It is a programme "used in integrated circuit and board-level design to check the integrity of circuit designs and to predict circuit behavior" [12]. In simpler terms, SPICE is a high-level simulation tool that operates at the transistor level. Instead of describing the sensor with equations, SPICE allows us to simulate its behaviour by modelling it as an electrical circuit.

6.2 SPICE for Sensor Simulation

For our specific application, SPICE provides several advantages. By modelling the sensor as an electrical circuit, we can fully simulate its behaviour using just three basic components: Resistance, Inductance, and Capacitance. These components can be adjusted based on the sensor's physical parameters, such as the number of coil turns or length, providing a simple yet effective model for simulation. This basic model can be expanded to incorporate more complex parameters as needed.

SPICE is a method rather than a standalone tool, necessitating the use of SPICE-compatible software for our initial sensor simulation tests. The subsequent two figures illustrate a circuit designed using LTSpice , a GUI-based software for electrical circuit design. The first figure shows the graphical view, which is native to LTSpice, while the second figure presents the *Netlist*—a file that describes the circuit and its simulation parameters, which is then processed by the SPICE engine.

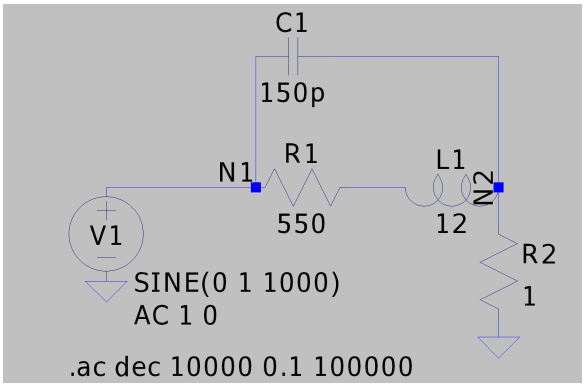


Figure 6.1: RLC Circuit

```
V1 N1 0 SINE(0 1 1000) AC 1 0
R1 N001 N1 550
C1 N2 N1 150p
L1 N2 N001 12
R2 N2 0 1
.ac dec 10000 0.1 100000
.backanno
.end
```

Figure 6.2: RLC Netlist

Having explored how we can model a sensor using an electrical circuit, let us now proceed to discuss the process of integrating this model into our system.

6.3 How to integrate it

As previously mentioned, SPICE is only a simulation methodology and requires a computational engine to perform the simulations. In our project, we have opted to use Ngspice [14], a popular SPICE implementation. Directly writing and parsing our own Netlists could have been an option. However, upon further investigation, we found that SPICE Netlists are not standardised and can be challenging to interpret and modify due to their complex syntax.

To address this, we utilise PySPICE [13], a Python library that facilitates the writing of SPICE circuits in Python code and automatically translates them into SPICE Netlist format. This approach streamlines the integration process by eliminating the need to manually handle Netlist syntax, allowing us to focus on the simulation aspects witho getting restrained by Netlist management.

```
circuit = Circuit(' Imp JUICE')

circuit.SinusoidalVoltageSource('V1', 'N1', circuit.gnd, amplitude=1 @ u_V, frequency=1 @u_kHz)

circuit.R('1', 'N001', 'N1', resistance @u_ohm)
circuit.C('1', 'N2', 'N1', capacitance @u_Farrad)
circuit.L('1', 'N2', 'N001', inductance @u_Henry)
circuit.R('2', 'N2', circuit.gnd, 1 @u_kohm)
```

Figure 6.3: Representation of the RLC 6.1 circuit using PySpice

This integration approach simplifies the inclusion of SPICE into PLASMAG significantly. By first computing the resistance, capacitance, and inductance values analytically, we can easily incorporate these components into our SPICE simulations. Users wishing to simulate with SPICE simply add a node to our calculation tree. That specific node has to use PySpice in its strategy and with dependencies on the resistance (R), inductance (L), and capacitance (C) values.

It can be seen in the full view of the GUI that SPICE has been integrated as an activatable option that can be folded and unfolded. Numerical simulations with SPICE are somewhat more resource-intensive than analytical ones. To avoid slowing down the results too much, we only allow one circuit to be simulated live at a time.

6.4 First results

The first results are very promising. Spice allows for the simulation of many different types of circuits, including simple noise level measurements, transfer functions and Bode diagrams.

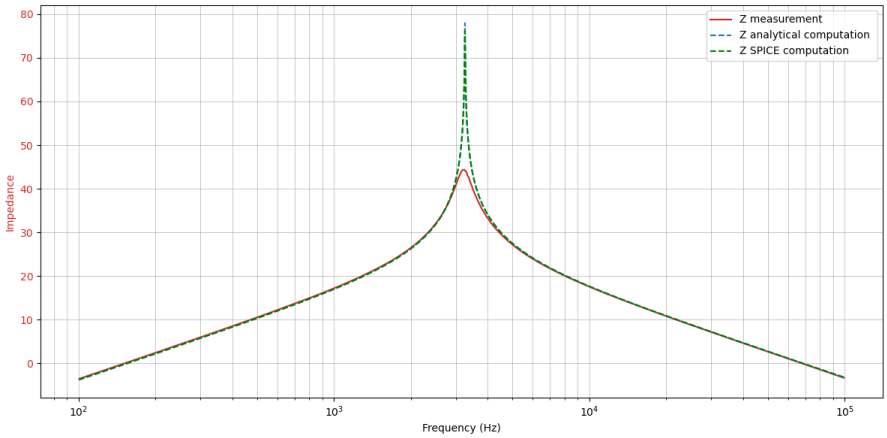


Figure 6.4: Impedance of JUICE SCM instrument plot

This figure shows the first result obtained. There are three curves on the same graph: in red, the real data for the impedance of a sensor measured during a previous space mission (JUICE); in blue, the result of an analytical model; and in green, the results are computed using SPICE. The curves are almost identical, showing similar resonance frequencies. The only notable difference is the amplitude of the peak at the resonance frequency, which can be explained by the measurement sensitivity of the tool used to measure the impedance of the real sensor.

CHAPTER 7

DATA FIT MODULE

The penultimate module of PLASMAG is the last component of the simulation engine that I developed during my internship. A small data fitting module embedded in the application. To provide more context, although we can analytically or numerically simulate a sensor, there are instances where certain parameters may not be modelable. This includes cases where the simulation requires parameters that have unmodeled behavior.

One of the best example is the *Flicker Noise*, which is present in every electronic component and appears at low frequencies. Currently, our model lacks equations to describe this noise, and in numerical simulations, it is included in the total noise, making it indistinguishable from other noise contributions. However, we can measure it from a real sensor and obtain a curve. Thus, our goal was to be able to have a computation node in engine that could utilise real data. Let's keep the example of the *Flicker Noise* to explain how this module work.

7.0.1 Module Principle

The main problem is that measured data are often extremely noisy. To achieve accurate simulation results, we need to denoise this data. This involves "fitting" a curve to the data points, which means constructing a curve that best matches the data, often within certain constraints. We use a simple three-layer neural network for this task, but I will not focus too much deeply into the neural network's architecture here.

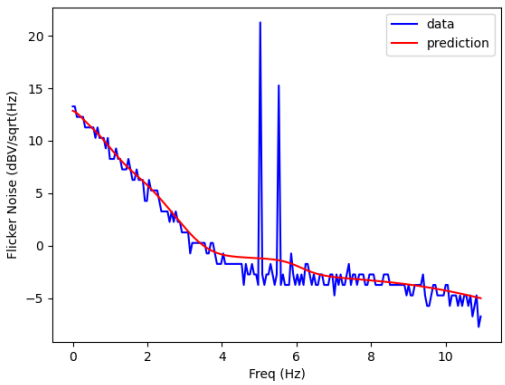


Figure 7.1: First fitting attempt

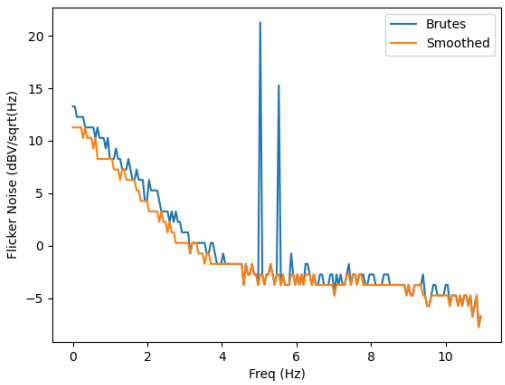


Figure 7.2: Denoised data curve

These two figures illustrate the initial attempts at fitting real Flicker Noise data. In the

first figure 7.1, the fitted curve, shown in red, matches the data almost perfectly across most of the frequencies. However, the data in blue includes two aberrant spikes, which excessively influence the curve's shape.

To address this, we applied a floating median-based outlier smoothing algorithm to the data, as shown in the second figure (7.2). This algorithm effectively smooths the data, although it slightly reduces the amplitude at lower frequencies. The algorithm works by "eliminating" values that deviate significantly from the median within a given window, then sliding this window across the entire frequency range.

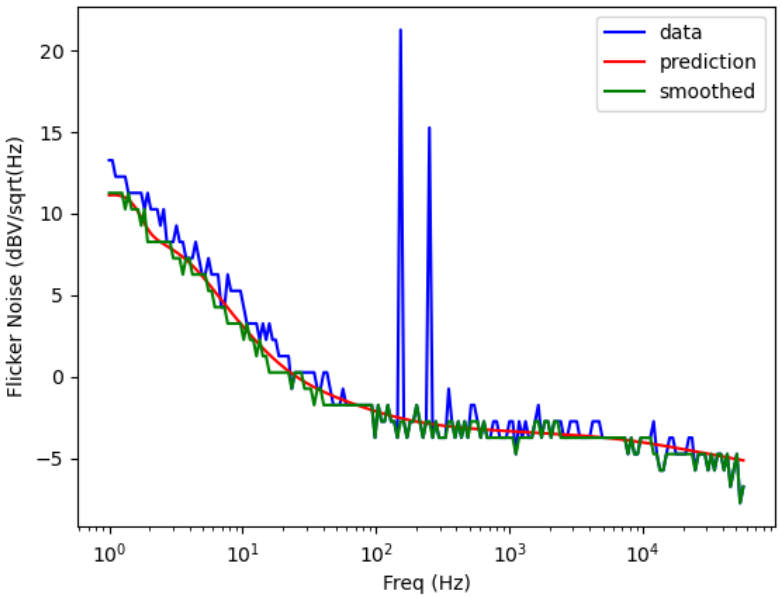


Figure 7.3: Final results of fitted curves on noisy data

The final image represents our results. The data, shown in blue, are fitted almost perfectly by the curve (in red), minimising deviations from the original curve shape and are now ready for use in a computation node.

Having thoroughly explained how we can simulate the behaviour of a sensor using the PLASMAG simulation engine, let's move on to the next chapter to see how this engine can be used to find a sensor that meets specific requirements automatically.

CHAPTER 8
OPTIMISATION

8.1 Why Optimize

Last but not lease, let's focus on the optimisation module. Finding the right design that meets specific performance criteria can be complex. Therefore, we introduced an optimisation module that performs the inverse operation. The concept involves entering performance goals (e.g., a value of NEMI at a certain level, and a different value at another level, etc., through a table) or a specific resonance frequency, or any other relevant metric. We then employ algorithms to identify one or more designs that best meet these performance objectives. The purpose is to prepare process of manually tuning parameters.

8.2 What is Optimisation?

Optimisation in this context refers to the systematic process of adjusting design parameters to find the most effective solution that meets specific criteria. This is typically represented mathematically by a cost function. A cost function quantifies how far a particular design is from the ideal, providing a scalar value that optimisation algorithms aim to minimise or maximise.

Optimisation techniques can be categorised into two main types: maximising or minimising the cost function. Maximising might be used when improving factors like efficiency or performance, whereas minimising is used to reduce costs or errors.

8.3 How to Optimise

Optimisation requires an iterative approach to gradually refine the design parameters. The process begins with the definition of a cost function, which evaluates the performance of each potential solution based on the design objectives. Through a series of iterations, the optimisation algorithm adjusts the parameters and continuously evaluates the resulting outcomes through the cost function, guiding the search towards the optimal solution.

In PLASMAG, we perform optimisation using different algorithms

8.4 Algorithms:

To identify the optimal design, we utilise evolutionary algorithms such as the Genetic Algorithm, Particle Swarm Optimization, and Simulated Annealing.

I selected these specific optimisation algorithms for several reasons. First, our cost function may not be differentiable, making classical gradient methods unsuitable. Second, we often face a vast parameter space in some scenarios. Lastly, as discussed in subsequent sections, some situations require optimisation across multiple targets and parameters. These factors, combined with the limitations of classical deterministic algorithms, make heuristic algorithms preferable due to their ability to explore larger solution spaces effectively due to their non-deterministic nature.

We will now briefly see how these three algorithms function and their respective advantages.

8.4.1 Genetic

The Genetic Algorithm was the initial choice for the optimisation module of PLASMAG. It is implemented using the Python library DEAP [15], with its principle base on Darwinian Evolution.

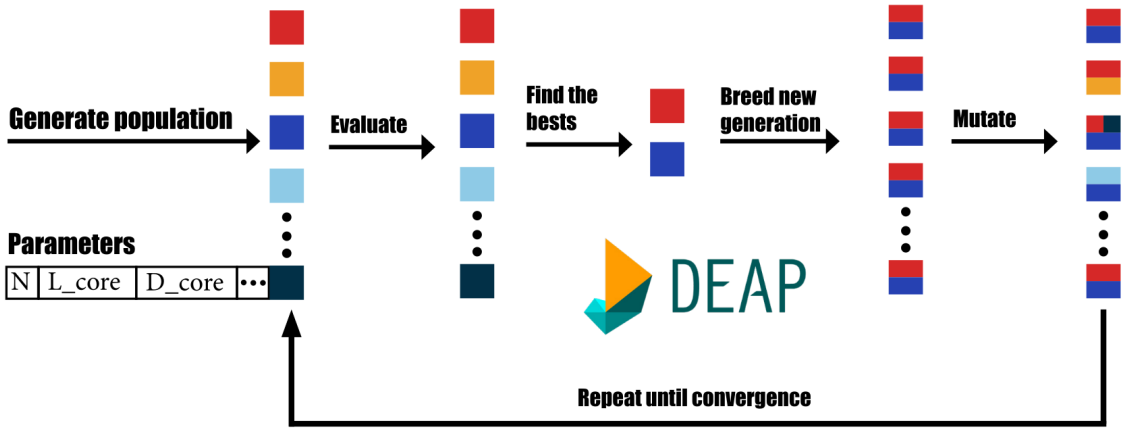


Figure 8.1: Genetic optimisation algorithm principle

The diagram illustrates the main principles of the Genetic Algorithm. We can conceptualise a sensor as an individual in a population, with parameters such as the number of coils and the length of the core and so on, are representing the genome of the individual.

- Initially, we randomly generate a population of N individuals.
- Each individual is then evaluated, we determine their performance with a cost function.
- Based on this evaluation, we identify the best performers in their generation (those the closest to meet the objective).
- These select individuals have their genomes mixed to breed a new generation.
- To prevent convergence on local minima, we introduce mutations, i.e., small variations in parameters.

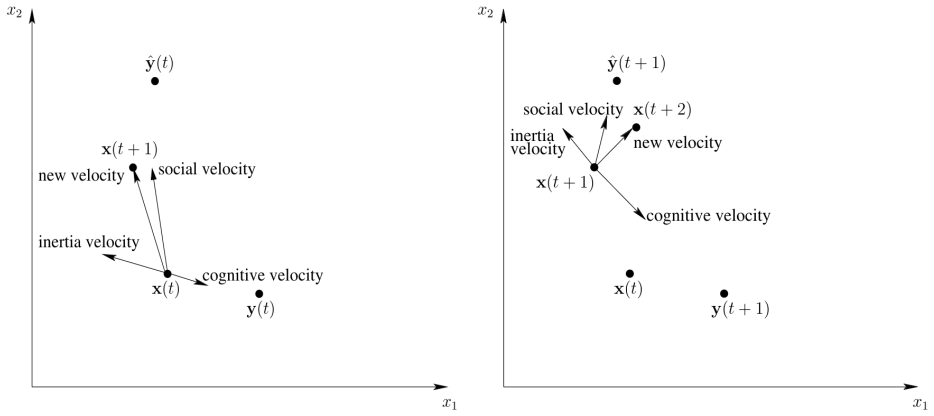
These steps are repeated until convergence is achieved, defined as reaching the target within a specific error range or after a maximum number of iterations.

This algorithm, when finely tuned, generally avoids local minima and converges relatively quickly, often leading to effective solutions.

8.4.2 Particle Swarm Optimisation

Although the genetic algorithm can effectively explore large parameter spaces, it may be slow to converge. Consequently, I implemented the Particle Swarm Optimization (PSO) algorithm manually. This method is inspired by the natural dynamics of a swarm.

The core principle of PSO is to manage a population of candidate solutions, where each particle represents a sensor. These particles move within the search space based on simple mathematical formulas that update the particle's position and velocity.[7].



(a) Time Step t (b) Time Step $t+1$

Figure 8.2: Particle Swamr Optimisation overview [7]

The velocity and position equations are given by:

$$V_i(t+1) = \omega V_i(t) + c_1 r_1 (P_i - X_i) + c_2 r_2 (P_g - X_i) \tag{8.1}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{8.2}$$

where:

- $V_i(t)$ is the velocity of particle i at time t.
- $X_i(t)$ is the position of particle i at time t.
- ω is the inertia weight factor.
- c_1 and c_2 are cognitive and social scaling coefficients. It represents how much the particle is respectively attracted by the global best and personal best solution.
- r_1 and r_2 are random numbers uniformly distributed in [0,1], used to avoid being stuck in local optima.
- P_i is the best position encountered by the particle i.
- P_g is the best position encountered by any particle in the swarm.

To illustrate, consider on the next graphs the movement of a swarm in a 2D optimisation problem. Here, the optimisation process explores the space incrementally, step by step, to locate the optimal solution.

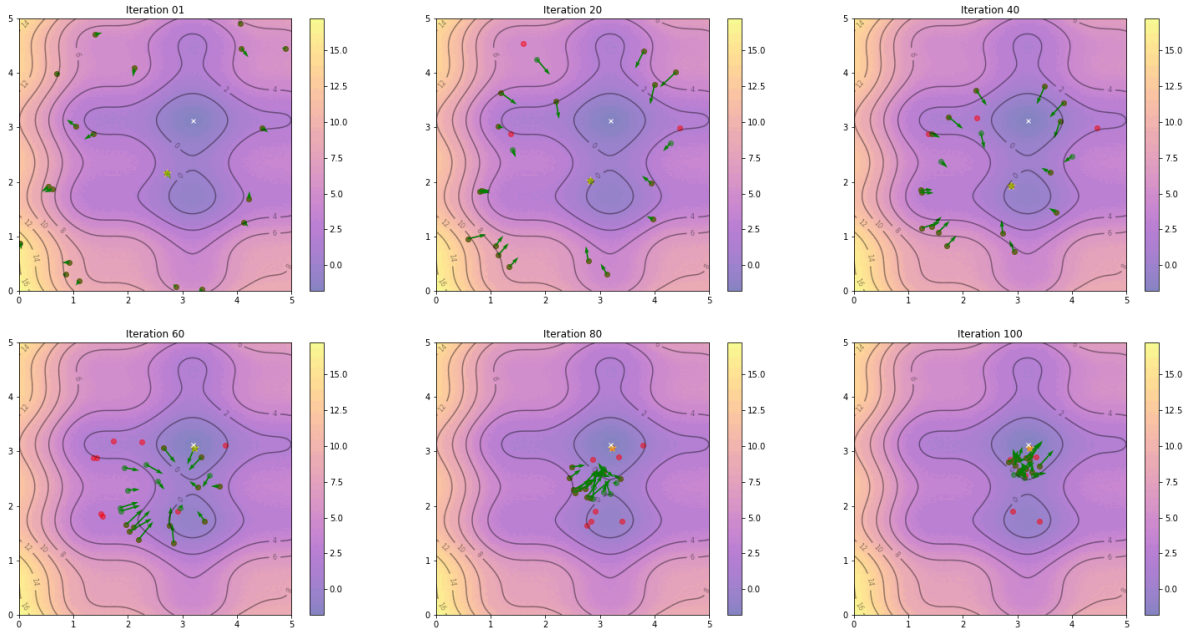


Figure 8.3: PSO iteration steps [7]

This algorithm has the advantage of rapid convergence, making it well-suited for initial explorations. As the computations for all particles at a given time step t can be parallelised, it quickly covers the entire search space. However, if not properly tuned (specifically the values of inertia, social, and cognitive coefficients) it may fall into local optima.

8.4.3 Simulated Annealing

Simulated Annealing is the last optimisation method we will talk about. While it may not be as rapid as the previous methods, it has the distinct advantage of reliably finding the optimal solution given sufficient time.

This method is metaphorically derived from the metallurgical process of annealing, where a material is heated to a high temperature and then allowed to cool slowly to ensure that it reaches a stable, low-energy state.

In the context of optimisation, we can think of the process similarly:

- We initialise the system at a high "Temperature" T, with a predefined cooling rate λ.
- At each iteration, a new candidate solution is generated near the current solution. A solution that parameters slightly variate from the actual state.
- This new solution is evaluated and accepted with a probability that depends on whether it improves the objective function and how significantly the temperature has decreased.
- The acceptance probability, expressed as $P(e, e', T) = \exp\left(\frac{e'-e}{T}\right)$ where e is the current energy (performance of the individual), e' is the energy of the new solution, and T is the current temperature.

Simulated Annealing works like a Markov chain, with the cooling schedule ensuring that the system eventually converges to a global optimum over time. This method is particularly effective because it avoids getting trapped in local minima by occasionally accepting worse solutions, thus exploring more of the solution space. Although it is the slowest method among those discussed, its ability to consistently reach the most accurate solution makes it invaluable for problems where precision is important.

8.5 Cost Functions:

In the previous chapter, I frequently mentioned "evaluating" when discussing sensors or individuals. This evaluation is done using a **cost function**, which quantifies how far a sensor's performance deviates from a desired target. This target could be a specific resonance frequency, performance level at a certain frequency, or other predefined criteria.

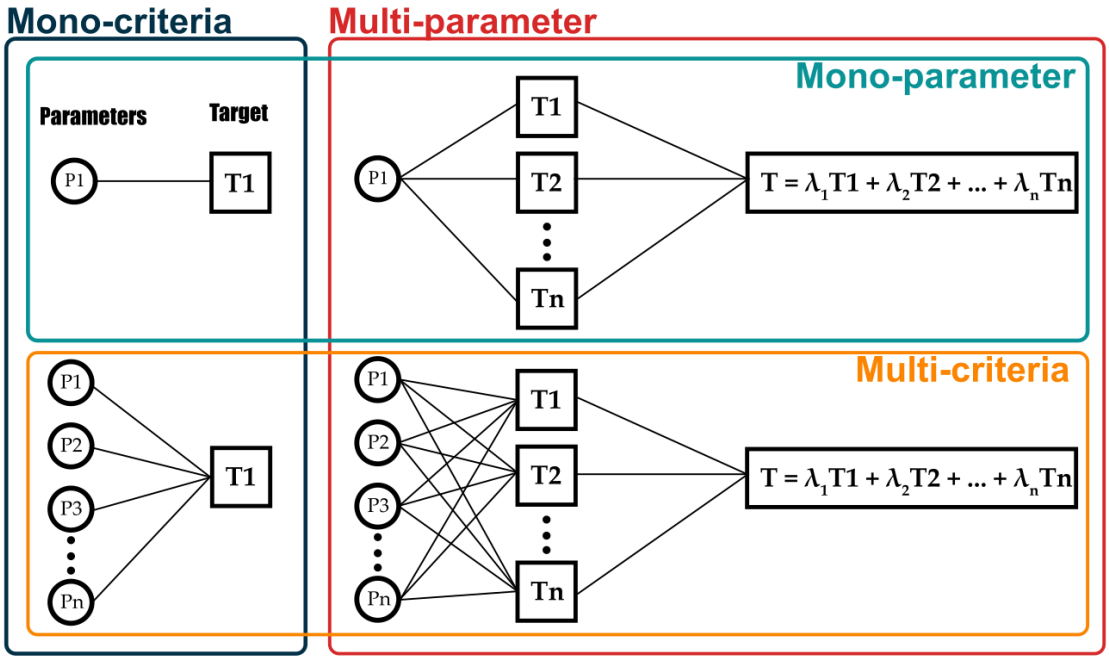


Figure 8.4: Optimisation cases overview

The diagram above illustrates various evaluation scenarios we encounter. In each case, we need to obtain a scalar value used for comparison. As the graph shows, we may encounter up to four different cases:

TODO : maybe add graph to describe how we calculate difference from the curve

8.5.1 mono-objective

mono-parameter

This case is straightforward: we vary one parameter at a time, aiming to achieve a single target value.

Multi-parameter

It should be a not so simple case, but since we're using our engine. Evaluating with a variation of different parameters is the same as variating a single one. For each solution, we just "send",

the parameters list to the Controler that operate the engine. And all the simulation data are returned. Thus both single and multi parameters are almost the same.

While evaluating multiple parameters simultaneously might be complex regarding the context. But for our case, the engine simplifies this process. Whether varying a single parameter or multiple parameters, the approach is essentially the same. We transmit the list of parameters to the controller that operates the engine, which then returns all simulation data. Thus, handling single and multiple parameters is nearly identical.

8.5.2 Multi-objective

However, there are situations where a single target is insufficient. For instance, when considering the Noise Equivalent Magnetic Induction (NEMI) of a sensor, one might prioritise specific sensitivity at low frequencies over lesser sensitivity at high frequencies. Also having a certain sensitivity while keeping the sensor's length below a specified value could be desired. In these cases, we are in multi-objective scenarios.

Currently, our cost function should only return scalar values. Therefore, we evaluate each target separately, assigning weights to indicate their relative importance. We then aggregate these weighted targets into a scalar value using a combination method. Common aggregation methods include weighted sums or weighted products, depending on the specific requirements of the optimisation task.

CHAPTER 9

CHALLENGES AND SOLUTIONS

9.1 Technical Problems and Solutions

During my internship I had to face some difficulties. Initially, as mentioned in the specifications section, my task was not merely to contribute to an ongoing project, but to help start a project from scratch. I had to produce something that would work at the end. Alongside my two advisors, we had to establish a clear and structured plan for what needed to be accomplished within the 16 weeks.

Another significant challenge was the limited availability of my supervisors, who were concurrently involved in several projects. Their time was often important, and they were not always available for long discussions at every scheduled meeting. This situation necessitated that I be efficient and proactive in my approach. Additionally, during their vacations, I was required to manage my tasks independently while keeping a certain productivity. Despite these periods of reduced direct supervision, I had the support of the IT team at the laboratory. It was important for me to adhere strictly to the planned tasks to avoid any deviations that could lead to unproductivity.

Toward the end of my internship, my advisor presented me with the opportunity to prepare and present a poster at the scientific council of the laboratory. The challenge here was the tight deadline; I was informed about this opportunity just five days before the event. This was my first experience creating a scientific poster, which proved to be an intense learning process. With guidance and support from both of my supervisors, we successfully produced a nice poster. This poster is now included in the Annex for reference.

CHAPTER 10

ASSESSMENT AND REFLECTIONS

Finally, I'm going to tell you about my work placement at LPP as a computer science intern. During this experience, as mentioned throughout this report, I worked on the PLASMAG simulation application. I am pleased to report that we not only met our initial objectives (reimplementing the model and developing a GUI with a SPICE interface) but also exceeded them, particularly with the development of optimization and fit modules.

10.1 Technical Assessment

My internship afforded me the opportunity to develop a range of technical and scientific skills. I discovered and mastered new tools, assimilated working methodologies and contributed to a project.

10.1.1 Acquired Skills

I gained a deeper understanding of Fedora and enhanced my overall skills in Linux. In terms of development environments, I learned how to construct a project from scratch, and also reinforced my proficiency with Git and PyCharm.

In the execution environment, I solidified my knowledge of Python and gained knowledge with several Python libraries: PyQt6 for graphics, NumPy for scientific computing, and Matplotlib for plotting. Additionally, I explored new libraries and modules, such as DEAP for genetic algorithms, and learned new computing techniques and optimisation algorithms.

I also contributed to make the project as modular as possible using Object-Oriented Programming (OOP) and a good architectural design. I hope this will facilitate future enhancements of the application.

Finally, I expanded my scientific knowledge through interactions with various researchers and engineers, collaborations with PhD students, and by attending multiple seminars and presentations. These experiences enriched my understanding and contributed significantly to my professional growth.

10.2 Human Assessment

The opportunity to work in a research laboratory was an enriching experience. Taking part in a number of meetings also helped me to develop my communication and project management skills.

10.2.1 Integration and Collaboration

The human aspect of this internship was incredibly positive. I successfully integrated into the team and engaged in constructive exchanges with my colleagues. This integration was a valuable experience.

I was fortunate to have advisors who dedicated time for weekly video-conference meetings to discuss the project's progress. Additionally, I had the opportunity to attend the scientific council, where I presented a poster and explained my work to many people. I participated in various presentations and meetings, including a PhD thesis presentation and a conference at the Ecole Polytechnique, where an ESA astronaut presented his work. These experiences were very rewarding, both personally and professionally, and were usefull in expanding my professional network.

I also made valuable contacts with engineer, researchers and PhD students. These interactions were very much appreciated and gave me a lot, thanks to their expertise and good humour.

10.3 Improvement Perspectives and Future Recommendations

As of writing these lines, 22 out of the 25 requirements for the project have been met, one was canceled, and 4 remain to be validated. I have implemented nearly all the features, released three versions of PLASMAG (with a fourth forthcoming), and we have confirmed the functionality of the engine and the model by comparing our results with measurements taken from the JUICE mission sensor (detailed in the Annexes).

Moving forward, the plan is to adhere to the project requirements as detailed in the documentation and address issues tracked on Git. If possible, I aim to continue contributing to the project as much as I can.

CHAPTER 11

CONCLUSION

This internship at LPP was a significant step in my professional and academic career. I am satisfied with the work done and the results obtained.

Having the opportunity to work within a scientific research environment—a field I am deeply passionate about—has been an invaluable experience. Not only did it give me an insight into the research environment, it also strengthened my conviction that I should continue my studies in this direction.

The warm welcome and support extended by the laboratory team were instrumental to the success of this internship. I would like to thank them for their guidance and for the opportunities for learning and exchange that they offered me.

All in all, this internship has been an enriching experience that has enabled me to progress and to look to the future with greater certainty. It is with a sense of achievement and renewed determination that I conclude this report.

GLOSSARY

Symbols | A | C | D | I | L | M | N | P | Q | R | S | X

Symbols

.json JavaScript Object Notation, a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. 20

A

ABC Abstract Base Class, a class in object-oriented programming that cannot be instantiated and is designed to be subclassed, enforcing the presence of certain methods in derived classes. 25

Agile A methodology for software development that emphasises incremental delivery, splitting the development in *Sprint*. 18

C

Capacitance The ability of a system to store an electric charge. 33

CNRS Centre National de la Recherche Scientifique. 6

Complexity In the context of an algorithms, it's a measure of the amount of resources, such as time and space, that an algorithm consumes relative to the input size. 3, 30

D

DFS Depth-First Search, an algorithm for traversing or searching tree or graph data structures, where one starts at the root and explores as far as possible along each branch before backtracking. 31

I

Impedance The total opposition that a circuit offers to the flow of alternating current or any other varying current at a particular frequency, combining the effects of resistance, inductance, and capacitance. 35

IN2P3 Institut National de Physique Nucléaire et de Physique des Particules. 7

INC Institut National de la Chimie. 7

Inductance Property of a circuit or coil that causes an electromotive force to be set up due to a rate of change of current in the circuit or coil. 33

INP Institut National de Physique. 7

INSB Institut National des Sciences Biologiques. 7

INSHS Institut National des Sciences Humaines et Sociales. 7

INSIS Institut National des Sciences de l'Ingénieur et des Systèmes. 7

INSMI Institut National des Sciences Mathématiques et de leurs Interactions. 7

INSU Institut National des Sciences de l'Univers. 7

L

LPC2E Laboratoire de Physique et Chimie de l'Environnement et de l'Espace. 12

LPP Laboratoire de Physique des Plasmas. 5, 7, 9

M

Matplotlib A Python library for creating scientific visualizations. 17

N

NEMI Noise Equivalent Magnetic Induction, a measure of the sensitivity of magnetic sensors in terms of the equivalent magnetic noise. 38

NetworkX A Python library for the creation, manipulation, and study of networks of nodes and edges. 32

NumPy A Python library for numerical computing, providing support for arrays, matrices, and mathematical functions to operate on them. 17

P

pandas A Python library providing data structures and data analysis tools for manipulating numerical tables and time series. 17

PoC Proof of Concept, a demonstration to verify that certain concepts or theories have the potential to work for real applications. 17

PyQt6 Python wraping for the Qt application framework, allowing the development of cross-platform applications with a native lookl. 17

PySpice A Python library for interacting with SPICE simulator, allowing the analysis and simulation of electrical circuits. 17

Pyvis A Python library for creating interactive network visualizations. 32

Q

Qt A free and open-source widget toolkit for creating graphical user interfaces as well as cross-platform applications. 49

R

Resistance A measure of the opposition to the flow of electric current in a conductor. 33

S

SPICE Simulation Program with Integrated Circuit Emphasis. 13, 16, 17, 33

X

XArray A Python library designed for working with labeled multi-dimensional arrays. 17

BIBLIOGRAPHY

[1] “Le cnrs affiche SES ambitions pour l’Europe,” CNRS, https://www.cnrs.fr/fr/cnrsinfo/le-cnrs-affiche-ses-ambitions-pour-leurope

[2] "Juice SCM", https://x.com/ESA_JUICE/status/1297774546477633540

[3] Libretexts, “13.3: Lenz’s law,” Physics LibreTexts, https://phys.libretexts.org/Bookshelves/University_Physics/University_Physics_(OpenStax)/Book%3A_University_Physics_II_-_Thermodynamics_Electricity_and_Magnetism_(OpenStax)/13%3A_Electromagnetic_Induction/13.03%3A_Lenz%27s_Law (accessed May 30, 2024).

[4] “LPP - Laboratoire de Physique des Plasmas - UMR 7648 - A propos du LPP,” Polytechnique.fr, 2019. https://www.lpp.polytechnique.fr/-A-propos-du-LPP- (accessed May 31, 2024).

[5] “Final three for ESA’s next medium science mission,” www.esa.int. https://www.esa.int/Science_Exploration/Space_Science/Final_three_for_ESA_s_next_medium_science_mission (accessed May 31, 2024).

[6] A. Retino, “Particle Energization in Space Plasmas: Towards a Multi-Point, Multi-Scale Plasma Observatory,” Oct. 29, 2019. https://www.cosmos.esa.int/documents/3273648/3495481/2_03_ARetino-esa-voyage-workshop-2019.pdf/6073323d-2e45-c8d6-1851-b07e7e498406?t=1573466575654

[7] A. P. Engelbrecht, Computational Intelligence. John Wiley & Sons, 2007.

[8] “Pint: makes units easy — pint 0.23rc3.dev1+g52ac9f5 documentation,” pint.readthedocs.io. https://pint.readthedocs.io/en/stable/

[9] K. Moore, K. Jennison, and J. Khim, “Depth-First Search (DFS) | Brilliant Math & Science Wiki,” brilliant.org. https://brilliant.org/wiki/depth-first-search-dfs/

[10] NetworkX, “NetworkX — NetworkX documentation,” networkx.org. https://networkx.org/

[11] "Interactive network visualizations — pyvis 0.1.3.1 documentation,” pyvis.readthedocs.io. https://pyvis.readthedocs.io.

[12] “SPICE (Simulation Program with Integrated Circuit Emphasis) | EECS at UC Berkeley,” www2.eecs.berkeley.edu. https://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html

[13] F. SALVAIRE, “PySpice,” PySpice. https://pyspice.fabrice-salvaire.fr/ (accessed Jun. 04, 2024).

[14] “Ngspice, the open source Spice circuit simulator - Intro,” ngspice.sourceforge.io. https://ngspice.sourceforge.io/.

[15] “DEAP documentation — DEAP 1.3.0 documentation,” Readthedocs.io, 2019. https://deap.readthedocs.io/en/master/

APPENDIX A

ANNEXES

A.1 Démonstration de la compétence 2

Consignes : *Annexe Démonstration de compétence : Vous inclurez dans le rapport une annexe intitulée « Démonstration de la compétence + n° de compétence » (dans la limite de 2 pages) dans laquelle vous ciblez une des 6 compétences du BUT dont vous expliquerez la mise en œuvre lors de votre travail de stage. Vous vous appuierez pour cela sur les apprentissages critiques du niveau 3 du BUT. Il s'agit en somme de démontrer, en vous appuyant sur des exemples concrets, que vous avez acquis le niveau 3 de la compétence choisie.* (Celene, consulté le 05/06/2024)

Dans ce portfolio je vais principalement cibler la compétence "Optimiser". Je vais maintenant tâcher de démontrer ainsi que citer les points acquis et mis en oeuvre durant mon stage, en essayant de tendre vers une exhaustivité. J'ometterais volontairement de citer chacune des compétences en 1er et 2e niveau, en admettant que ces dernières sont déja acquises.

Premièrement, **CE2.01 - Formaliser et modéliser des situations complexes** et **AC32.01 - Profiler, analyser et justifier le comportement d'un code existant**: Lors du développement de PLASMAG, il était nécessaire de m'imprégner du problème dans sa globalité, d'exprimer la problématique en code et de proposer une solution adaptée. J'ai repris le code existant, analysé son comportement et proposé une structure de moteur adaptée aux besoins de modularité tout en offrant un modèle performant pour ce besoin complexe.

Plus particulièrement, **AC32.01 - Anticiper les résultats de diverses métriques (temps d'exécution, occupation mémoire, etc.)**: Lors de la conception du moteur, j'ai porté une attention particulière à la performance. Pour chaque version, du prototype à la version finale, j'ai réalisé des benchmarks pour évaluer le temps d'exécution moyen de la simulation. Bien que les schémas ne soient pas présents dans ce rapport, j'ai également vérifié la charge mémoire lors des calculs.

Aussi, **CE2.03 - S'appuyant sur des schémas de raisonnement**: Lors de la première proposition d'amélioration du moteur de calcul, j'ai raisonné sur le nombre de recalculs nécessaires sur les nœuds de l'arbre de calcul pour proposer une solution améliorant les performances. Pour **CE2.02 - Recenser les algorithmes et les structures de données usuels**, lors de la conception et de l'amélioration des performances du moteur, j'ai travaillé et implémenté, pour mon cas particulier, des algorithmes de parcours d'arbre et de recherche dans des données tampon.

En continuant sur la compétence **CE2.02 - Recenser les algorithmes et les structures de données usuels**, lors de la partie optimisation, j'ai recherché des méthodes et algorithmes appropriés au problème. Après avoir formalisé l'objectif et réfléchi au processus d'optimisation à effectuer, il était nécessaire de choisir précisément quels algorithmes utiliser, que ce soit pour l'optimisation par l'algorithme génétique, l'optimisation par essaims particulaires ou le recuit simulé, ou même la méthode pour calculer des fonctions de coût lorsqu'il y avait plusieurs objectifs.

AC32.01 - Choisir et utiliser des bibliothèques et méthodes dédiées au domaine d'application: J'ai réussi à utiliser les bonnes bibliothèques adaptées à ce contexte scientifique. Que ce soit pour le module d'optimisation, où j'ai utilisé la bibliothèque spécialisée DEAP [15], l'affichage graphique via PyQt6, les diagrammes avec Matplotlib, ou pour les autres calculs effectués où les opérations sont vectorisées grâce à NumPy, ou encore avec des bibliothèques

très spécifiques comme PySPICE, que j'ai appris à fonctionner et utilisée pour intégrer les simulations SPICE dans l'application. Cela montre bien que j'ai su utiliser des librairies spécifiques et très pertinentes pour mon domaine de stage.

Finalement, **CE2.04 - Justifier les choix et valider les résultats**: Pour chaque choix d'architecture, j'ai su proposer des prototypes démontrant que mes idées fonctionnaient et appuyer mes propos lors des réunions hebdomadaires, où je justifiais mes choix techniques et d'architecture, et ce pour chaque module ou sous-partie réalisée dans l'application PLASMAG.

GUI Hierarchy

```
MainGUI (QMainWindow)
├── central_widget (QWidget)
│   └── main_layout (QHBoxLayout)
│       ├── tabs (QTabWidget)
│       │   ├── param_tab (QWidget)
│       │   │   └── params_layout (QVBoxLayout)
│       │   │       ├── section_scroll_area (QScrollArea)
│       │   │       │   └── section_widget (QWidget)
│       │   │       │       └── section_layout (QGridLayout)
│       │   │       │           ├── param_label (QLabel)
│       │   │       │           └── param_input (QLineEdit or QComboBox)
│       │   │       ├── global_slider_coarse (QSlider)
│       │   │       ├── global_slider_fine (QSlider)
│       │   │       ├── frequency_range_slider (QRangeSlider)
│       │   │       └── calculate_btn (QPushButton)
│       │   ├── strategy_tab (QWidget)
│       │   ├── spice_tab (QWidget)
│       │   ├── optimisation_tab (QWidget)
│       │   └── EMC_tab (QWidget)
│       ├── spice_group_box (QGroupBox)
│       │   └── spice_layout (QVBoxLayout)
│       │       ├── toggle_spice_button (QPushButton)
│       │       └── spice_contents (QWidget)
│       │           └── spice_contents_layout (QVBoxLayout)
│       │               ├── params_layout (QGridLayout)
│       │               │   ├── spice_param_label (QLabel)
│       │               │   └── spice_param_input (ToolTipLineEdit)
│       │               └── circuit_image_label (ResizableImageLabel)
│       └── plot_widget (QWidget)
│           └── plot_layout (QVBoxLayout)
│               ├── btn_layout (QHBoxLayout)
│               │   ├── btn1, btn2, ..., btnN (QPushButton)
│               │   └── btnC (QPushButton)
│               ├── canvas_layout_1 (QVBoxLayout)
│               │   ├── top_layout (QHBoxLayout)
│               │   │   ├── toolbar (NavigationToolbar)
│               │   │   ├── checkbox (QCheckBox)
│               │   │   ├── combobox (QComboBox)
│               │   │   ├── background_button (QPushButton)
│               │   │   └── reset_background_button (QPushButton)
│               │   └── canvas (MplCanvas)
└── menuBar (QMenuBar)
    ├── file_menu (QMenu)
    └── options_menu (QMenu)
```

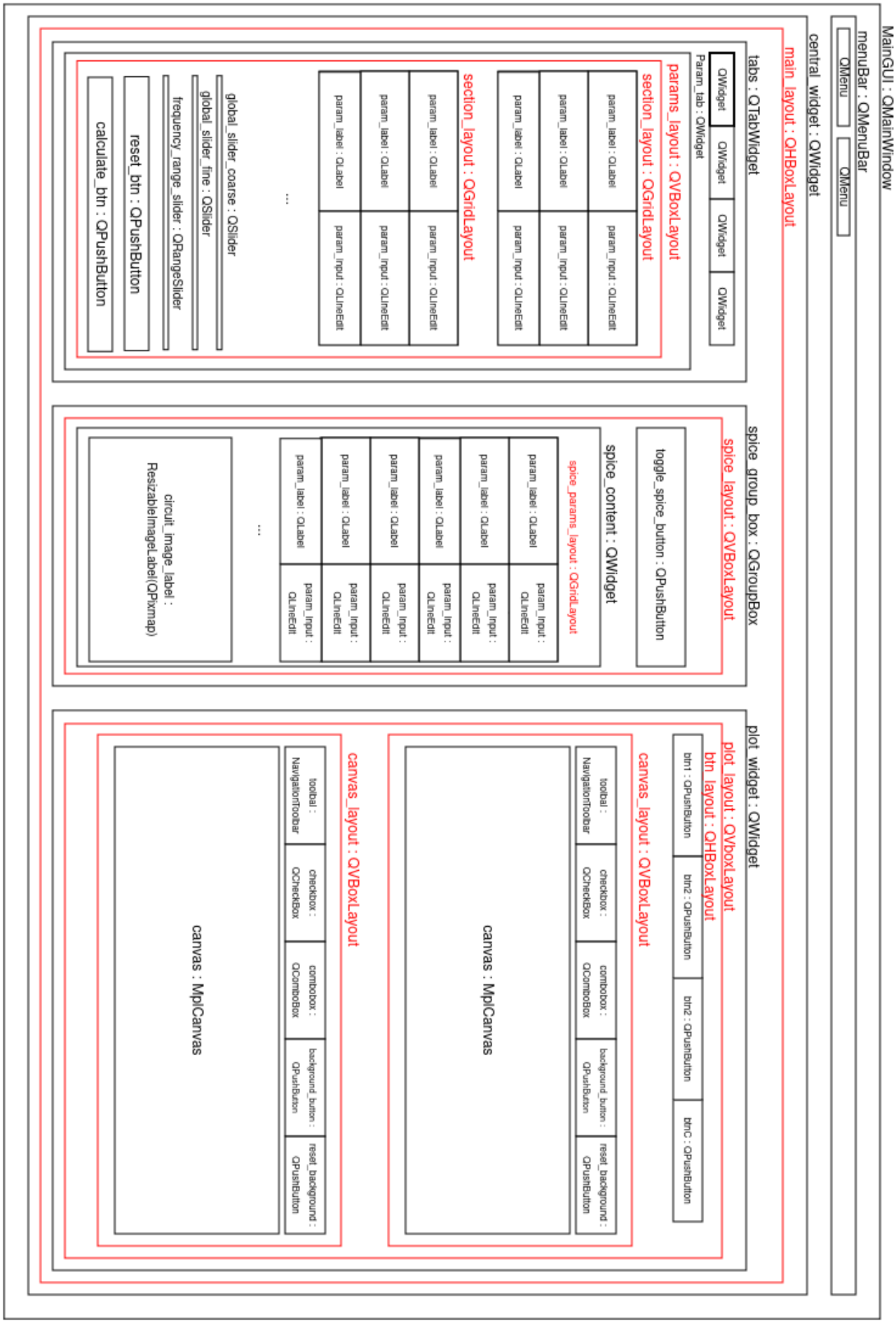


Figure A.1: Layout disposition Full version

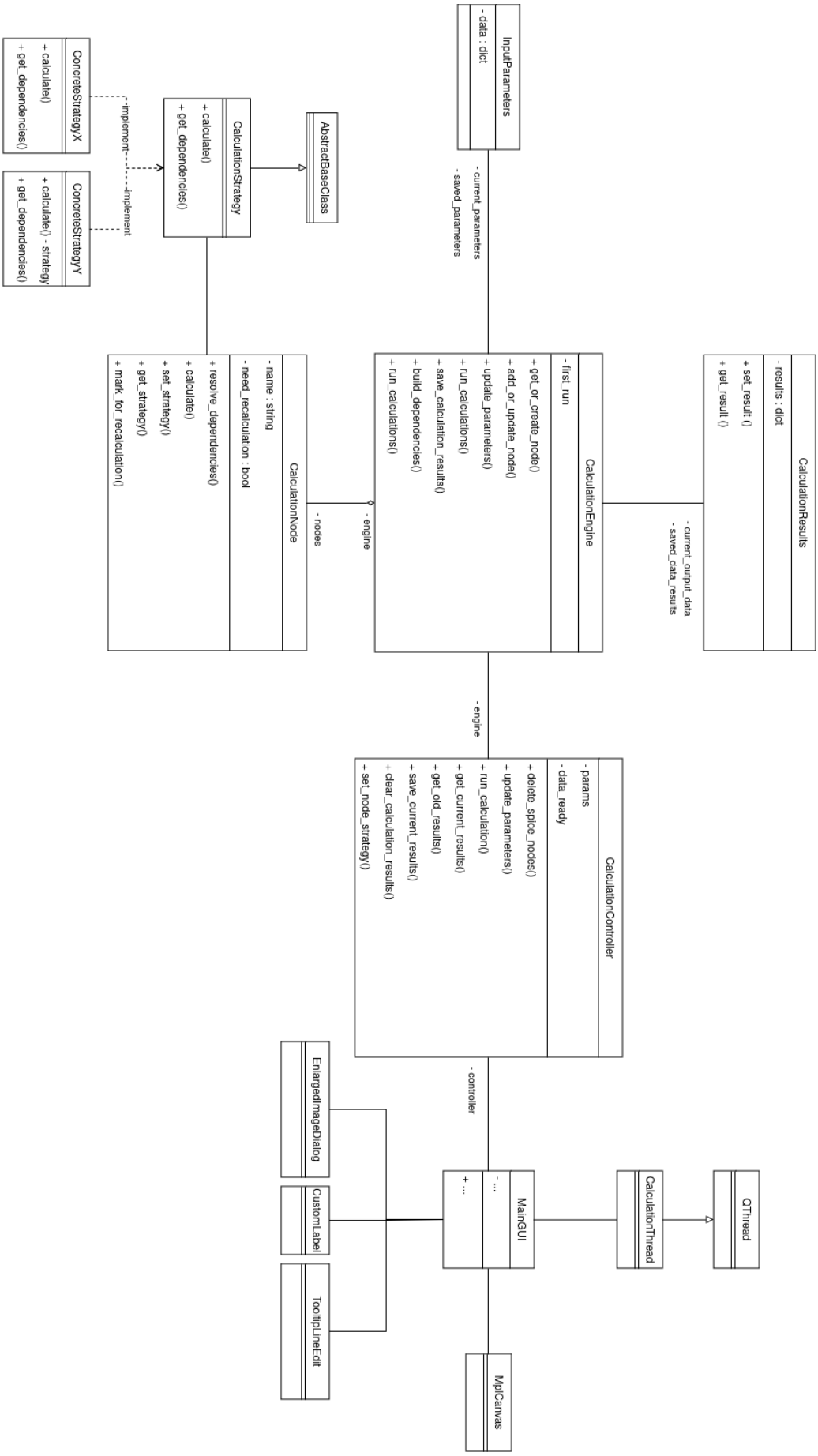
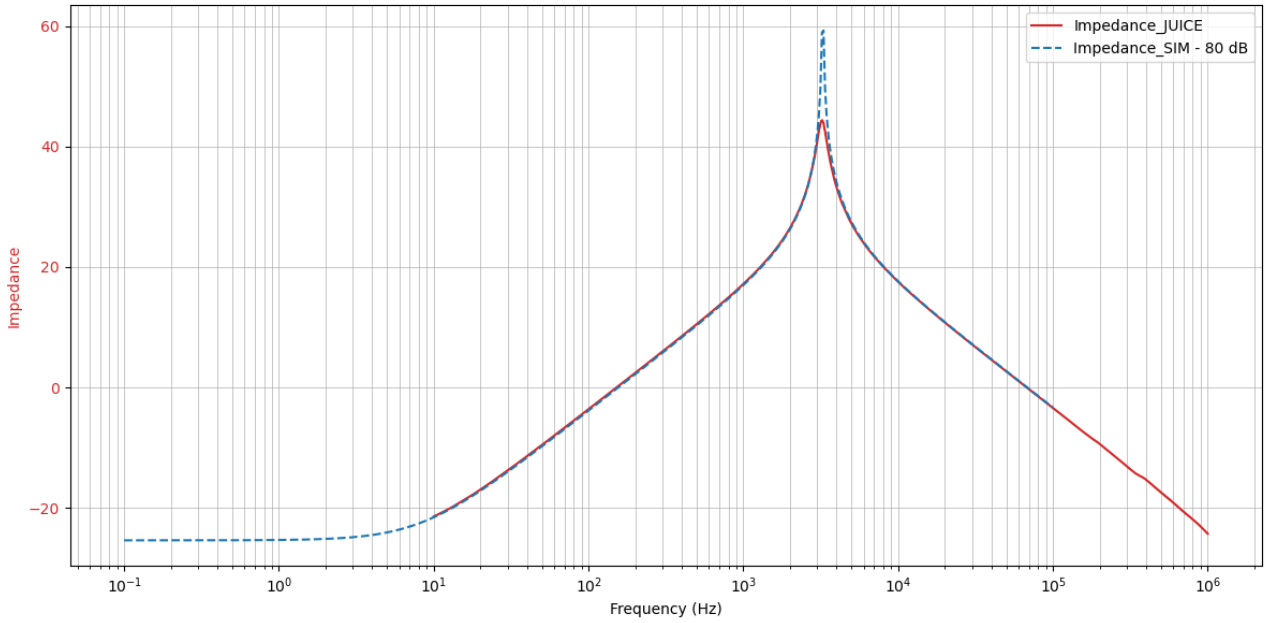
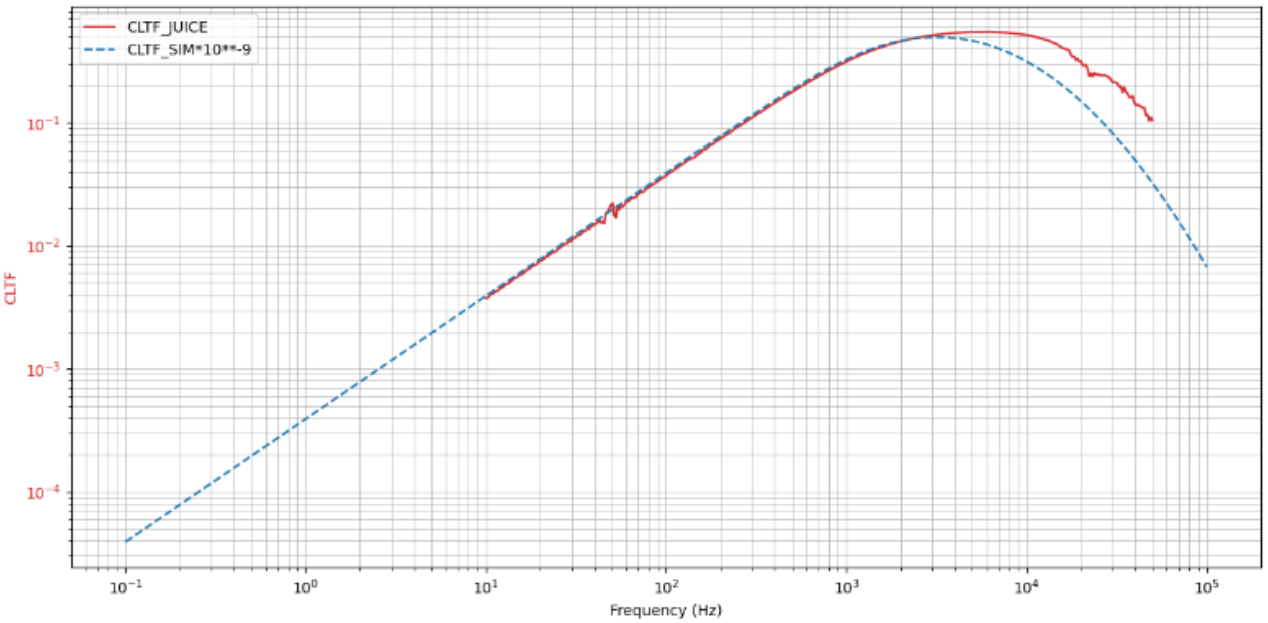
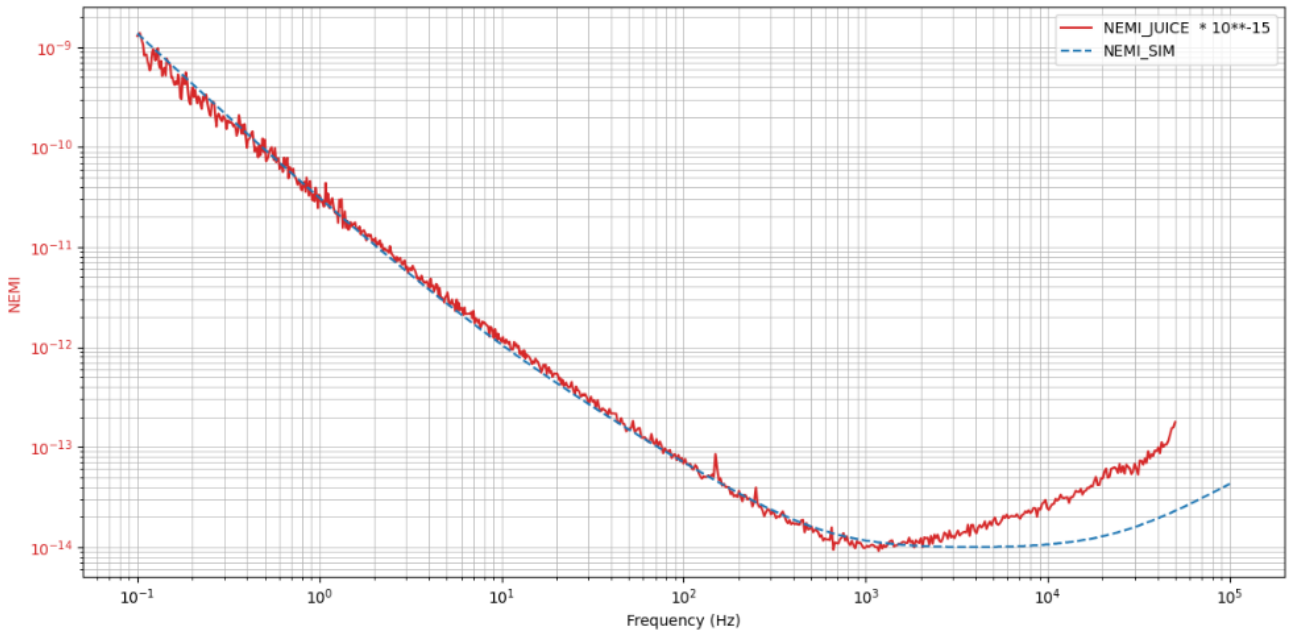


Figure A.2: Class Diagram Full version


Impedance_JUICE
Impedance_SIM - 80 dB
Impedance
Frequency (Hz)


CLTF_JUICE
CLTF_SIM*10**-9
CLTF
Frequency (Hz)



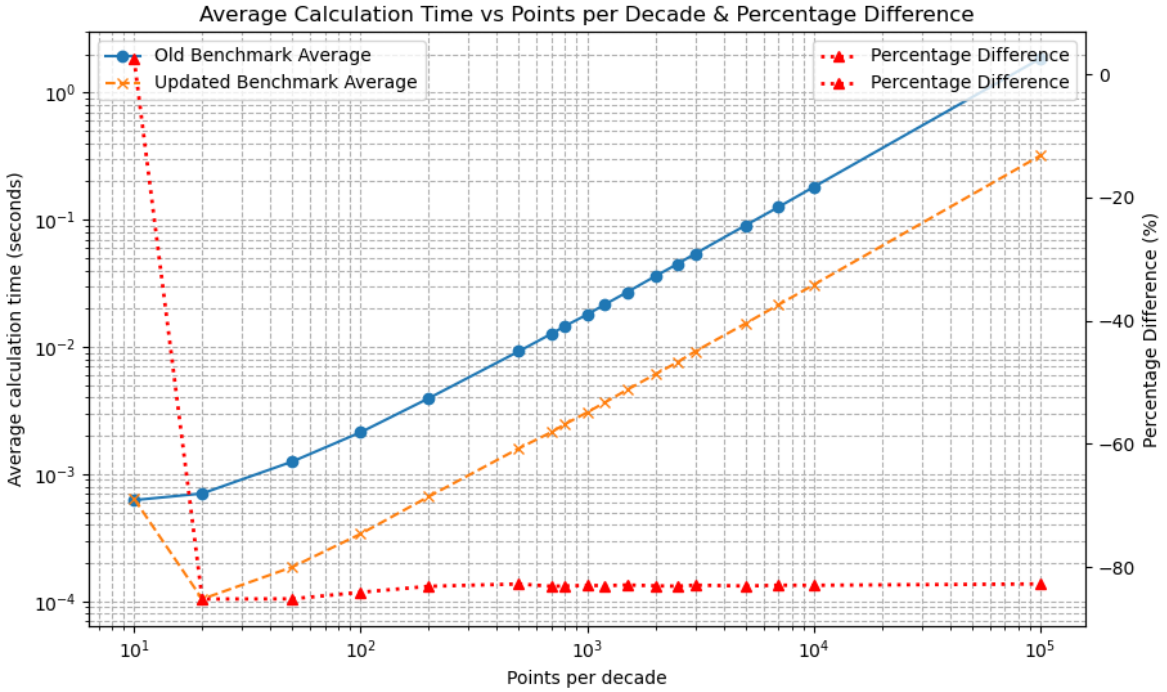
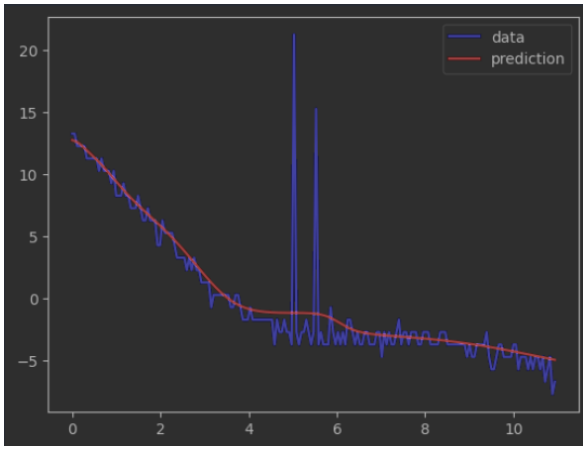
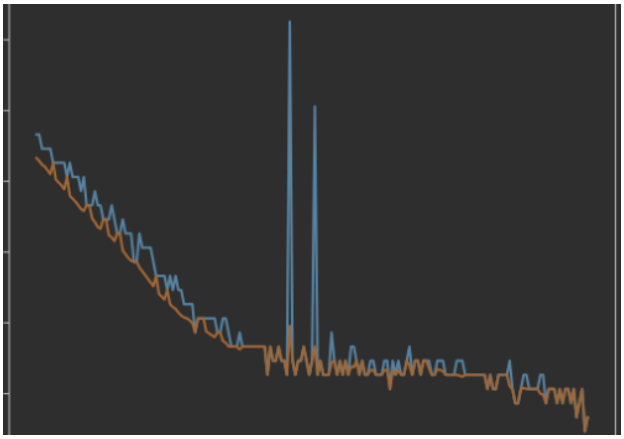


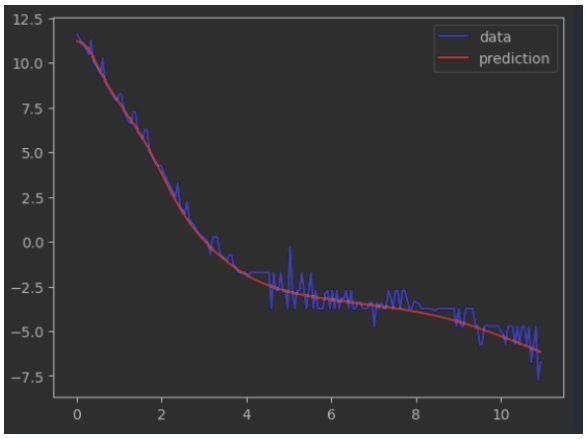
Figure A.3: Performances improvement



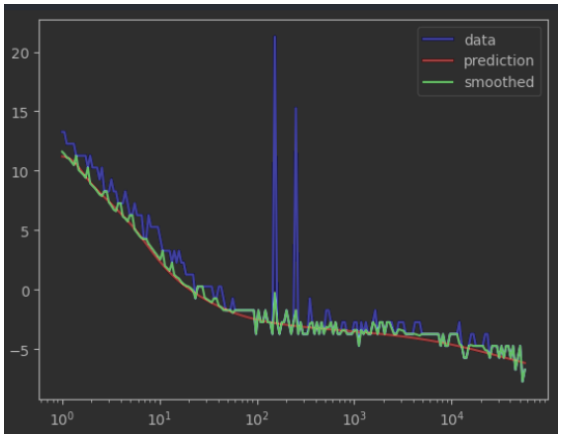
(a) Flicker curve fit - 1st attempt



(b) Flicker curve fit - Smooth the curve



(c) Flicker curve fit - 2nd attempt



(d) Flicker curve fit - Final results

Figure A.4: Flicker curve fitting

PLASMAG

Python library for Accurate Search-coil Magnetometer Adjustments with GUI

M.Ronceray[1] , M.Mansour[1], C.Revillet[2] , G.Jannet[2]

1 - Laboratoire de Physique des Plasmas (CNRS/X)

2 - Le Laboratoire de Physique et de Chimie de l'Environnement et de l'Espace (CNRS/CNES)

Context

The precise design of an instrument meeting scientific needs requires an iterative optimization process, made more efficient through the use of a **virtual tool** such as PLASMAG. Developed jointly by LPP and LPC2E, PLASMAG is specifically dedicated to design **induction magnetometers,** including air-core coils as well as single-band and bi-band search coils. Leveraging decades of expertise in instrumental development, PLASMAG offers plasma physicists and Earth physicists reliable and proven **simulation models**. PLASMAG is designed to be a **versatile tool** and Integrates into all phases of an instrumental project from proposal to validation catering to the needs of both scientists and engineers. You want to obtain a basic but realistic design in response to a **scientific proposal**? You want to **visualize** the impact of various design parameters on the instrument's properties or to reproduce faults for effective correction of detected non-conformities? PLASMAG helps you to do it! Additionally, PLASMAG is designed **collaboratively**, allowing community contributions to enhance it with new **simulation models** or **predefined designs** for specific applications.

Engine

To allow evolution of the computation models, PLASMAG is designed as a **modular** engine. The design difficulty comes from computing **highly interconnected** products while keeping the engine modular.

Our approach make it possible by representing the calculation process as a **tree** and an appropriate software design patern.

Ex : the NEMI (Noise Equivalent Magnetic Induction), is a product requiring several dependencies that are computed using differents approachs (analytical, numerical).

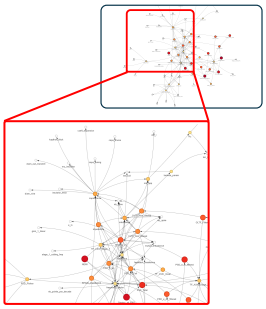


Figure 1.A - Dependency tree visualisation

Tree

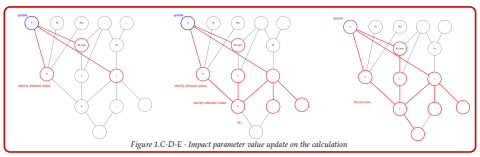


Figure 1.C-D-E - Impact parameter value update on the calculation

The tree is composed of nodes, each has dependencies and a "strategy" defining how to compute it based on its given parameters and dependencies.This structure make parallelization possible for nodes on the same level. Caraterized by improved computation efficiency and enhanced user experience.

Node

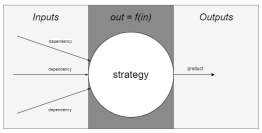


Figure 1.B - Node template

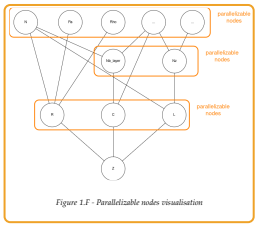


Figure 1.F - Parallelizable nodes visualisation

Advantages

- Only recalculate necessary nodes on parameter update.
- Easily parallelizable.
- Full graph visualisation tools.
- On the fly, computation method changes.

Strategy design pattern

A design pattern is a repeatable solution to a common programming problem. To solve our modularity problem, we use the **Strategy** pattern [1].

This approach allows each part of our system to change its calculation method as needed.

It helps us adapt easily to new requirements, making our system both **flexible** and **efficient**

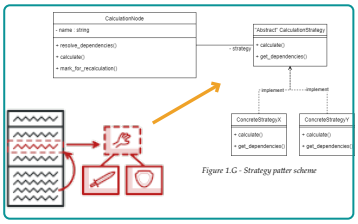


Figure 1.G - Strategy patter scheme

Optimisation

Our optimisation problem's complexity and the limitations of deterministic algorithms necessitate the use of heuristic algorithms, which are better suited due to their non-deterministic nature for exploring larger solution space.

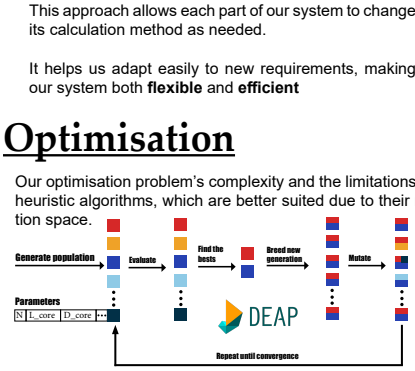


Figure 4.A - Genetic optimisation algorithm principle

Plasmag embed an optimisation module, allowing the user to find an optimal parameter set for a given target.

Currently implemented :
- Genetic optimisation (DEAP) [5].
- Particle Swarm Optimisation [2][3].
- Simulated Annealing

- Mono-parameter
- Multi-parameter
- Mono-criteria
- Multi-criteria - *streamlined into a single-criterion framework using a weighted polynomial.*

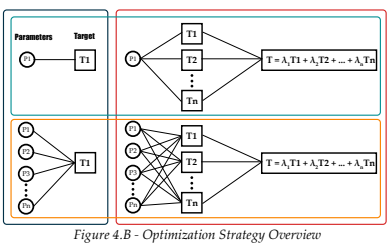


Figure 4.B - Optimization Strategy Overview

First results

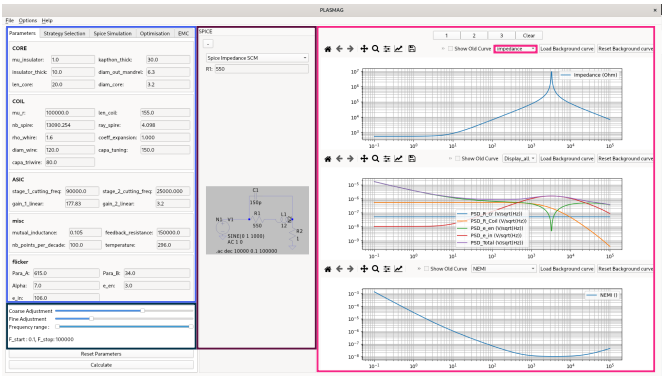


Figure 2.A - PLASMAG Gui with detailed sections [4].

- Full **electrokinetic** simulation using SPICE kernel - *using PySPICE.*
- **Selectable** calculation strategy for each products.
- **Adaptative** input section depending on a given magnetomener parameter set.
- **Sliders**, combined with real-time calculations, allow users to visualize parameters variation impact.
- Fully modular plot section allowing up to 5 **simultaneous** display of a variety of product.

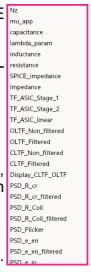


Figure 2.B - Plottable products combo box

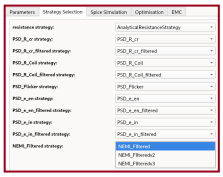


Figure 2.C - Strategy selection panel

Visualisation example of product obtained using different strategies :

- Analytical computation.
- SPICE simulation.
- Real data measurement.

PLASMAG allows to export any product or parameters to be plotted or re imported later.

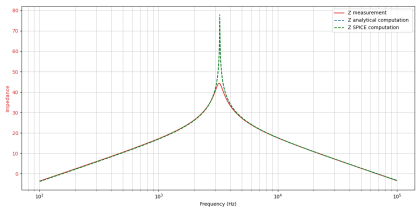


Figure 2.D - Impedance of JUICE SCM instrument plot

Real data fitting

Simulation may requires unmodelled parameter behaviors. To address this, we propose using fitted measured data as inputs.
PLASMAG's fit module is a two step data fitting tool using both a denoising process and a neural network to fit the data.

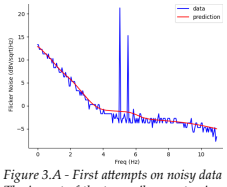


Figure 3.A - First attempts on noisy data The impact of the two spikes are too important on the fitted curve

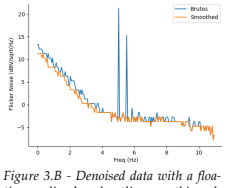


Figure 3.B - Denoised data with a floating median based outlier smoothing algorithm

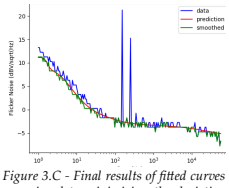


Figure 3.C - Final results of fitted curves on noisy data minimizing the deviation from the original curve

References

[1]. Refactoring Guru. "Strategy Design Pattern." https://refactoring.guru/design-patterns/strategy.

[2]. Engelbrecht, A.P. "Computational Intelligence: An Introduction." Chapter 16. Carnegie Mellon University in Qatar, https://web2.qatar.cmu.edu/~gdicaro/15382/additional/CompIntelligence-Engelbrecht-ch16.pdf

[3]. https://en.wikipedia.org/wiki/Particle_swarm_optimization

[4]. Riverbank Computing. "PyQt6 Documentation." https://www.riverbankcomputing.com/static/Docs/PyQt6/.

[5]. DEAP. "Distributed Evolutionary Algorithms in Python." https://deap.readthedocs.io/en/master/.

SPEC-#ID	The application must :	State
Spec-01	Be deployable on : Fedora/Win/Mac	DONE
Spec-02	Dimension : SCM or Air coil	DONE
Spec-03	Be modular and evolutive	DONE
Spec-04	Have an evolutive data structure	DONE
Spec-05	permit to calculate the following physical paramaters : ... (cf full spec)	DONE
Spec-06	be able to calculate the following geometric parameters: ... (cf full spec)	~ DONE
Spec-07	allow the value of all the following geometric dimensioning parameters to be entered graphically: ... (cf full spec)	~ DONE
Spec-08	be able to calculate Geometric, Electrical and Magnetic parameters using one or more analytical models.	TODO
Spec-09	allow graphical input of the type of Meometric, Electrical and Magnetic parameter calculation model to be used.	DONE
Spec-10	allow a later implementation of a new analytical model	DONE
Spec-11	be able to interface with a SPICE-type electrokinetic resolution engine in order to calculate the following quantities: ... (cf full spec)	~ DONE
Spec-12	Interface with the mechanical resolution engine of the FREECAD application.	CANCELED
Spec-13	Implement a graphic visualisation module	DONE
Spec-14	Implement an Human-computer interface	DONE
Spec-15	allow 2 curves of the same type to be displayed simultaneously on the same graph	DONE
Spec-16	be able to calculate and plot the difference between 2 curves of the same type in order to quantify the differences.	TODO
Spec-17	Allow the implementation of an automatic optimisation module of the genetic algorithm type.	DONE
Spec-18	enable the various parameters to be plotted on a linear or logarithmic scale.	DONE
Spec-19	Allow the following parameters to be plotted in V²/Hz and V/sqrt(Hz).	DONE
Spec-20	Allow export and import of a configuration or scenario file in .csv format, saving all the input parameters for a given simulation.	~ DONE
Spec-21	allow export of one or more simulation results files in .csv format containing all the Geometric Electrical and Magnetic parameters resulting from the calculation of a given dimensioning.	DONE
Spec-22	be able to import curves in .CSV format for the following parameters - NEMI - DSP_Vn_Preamp-Input	DONE
Spec-23	implement a module for generating a fit from experimental data for the following data types: - DSP_Vn_Preamp-Input_	DONE
Spec-24	Enable experimental and simulation data to be plotted on the same graph for the following parameters: - NEMI - CLTF	DONE
Spec-25	implement a dedicated graph enabling the following quantities to be viewed simultaneously and as required: ... (cf full spec)	DONE

Figure A.5: Specification table