Compensating the YIG filter's hysteresis : an informal manual

"F*ing magnets, how do they work ?" - not Lord Kelvin, incredibly**

Lou Bernabeu

August 2022

1 Introduction

1.1 Why the YIG filter?

In the Shapiro project, a clean RF source is crucial to get well-defined Shapiro peaks and very stable locking of the Josephson junction. However, the `Anapico APMS40G-2` source used at Φ_0 has enough noise that the Josephson junction sometimes loses lock, rendering the voltage unstable. Thus the need for a frequency-adjustable, reasonably sharp band-pass filter that can cover as much of the $0 - 40$ GHz frequency range of the source as possible, so as to follow the emission line of the source and get a cleaner output.

1.2 Principle and hysteresis

Yttrium Iron Garnet is a ferrimagnetic material ; Larmor-like resonant precession of its magnetic moments can allow it to selectively transfer some frequencies from one RF contact near it to another. Orienting the magnetization of the material changes the frequency of the passing band. This is the principle of the `Micro Lambda Wireless MLFP-1738PA` YIG filter we have at Φ_0 .

Whenever magnetization of a ferrimagnetic material comes into play, hysteresis is to be expected. This filter is no exception. The filter is controlled by a voltage V between 0 and 10 V. Its band frequency can be written as :

$$f(V) = f_0 + \gamma V + \Delta f$$

where f_0 and γ are known. Δf, the deviation from linearity, includes the anomaly due to the hysteretic behaviour. In the case of our filter, the behavior is just hysteretic enough that the passing band's width is not sufficient to make it negligible, but still very small (on the order of a few tens of MHz) when compared to the span of the filter frequency (38 GHz). So to make these hysteretic affects visible to the eye, it is more convenient to plot Δf, as in fig. 1.1.

2 The Preisach Model

All of the theoretical information on the model is taken from [Mayergoyz, 2003]. I highly recommend reading Chapter 1 and working through the different demonstrations at least once to see how the model works. Unless explicitly given in the caption, all graphs in this section are from [Mayergoyz, 2003].

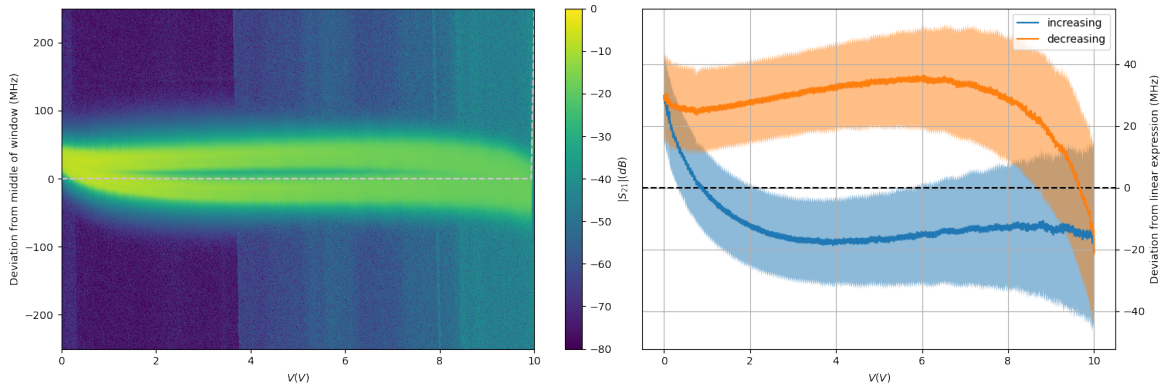


Figure 1.1: Measurement of Φ_0 's filter : Δf on $[0\text{V}, 10\text{V}]$. Blue curve is doing the sweep in the increasing V direction, orange curve is the reverse. Shaded regions denote the 3 decibel passing band of the peak.

2.1 Preisach's integral model of hysteresis

The idea of the Preisach model of hysteresis is inspired from the behavior of magnetic materials. Their macroscopic magnetic state corresponds to the sum of the magnetizations of many small magnetic domains within which the material has a constant, homogeneous magnetization. The magnetization of the whole is thus the sum of the magnetization of the parts. Each domain has its own hysteresis cycle, which hopefully is simpler than the whole, and contributes partially. These simple, elementary cycles are called **hysterons**. The idea is thus that the sum of the ouputs of these hysterons recreates the behavior of the whole system.

Since hysterons have an internal state that depends on the past history of the input, they are represented by operators rather than functions. For an input $u(t)$, the hysteron $\hat{\gamma}$ will give an output $f_{\hat{\gamma}} \underset{\text{def}}{=} \hat{\gamma}u(t)$. The output of the system, $f(t)$, is the sum of the outputs of all the hysterons in a set Γ, scaled by a positive weight function $\mu(\hat{\gamma})$:

$$f(t) = \int_{\hat{\gamma}\in\Gamma} \mu(\hat{\gamma}) f_{\hat{\gamma}}(t) = \int_{\hat{\gamma}\in\Gamma} \mu(\hat{\gamma})\hat{\gamma}u(t) \tag{1}$$

This leaves quite a lot of room for the design of the hysterons. The Preisach model choses them to be as simple as possible :

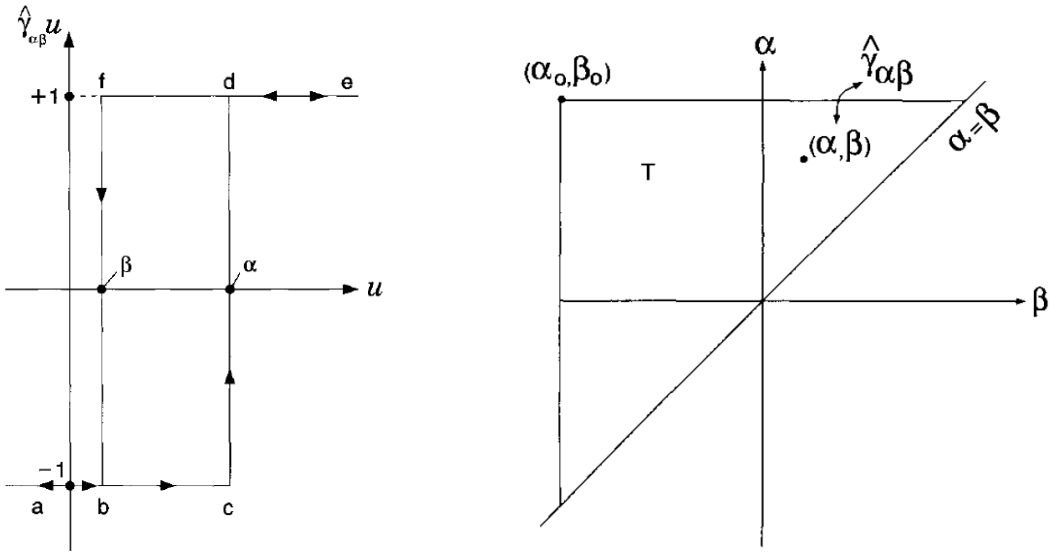
> In the Preisach model, hysterons have simple, rectangular hysteresis cycles that flip between $f_{\hat{\gamma}} = 1$ and $f_{\hat{\gamma}} = -1$ according to two parameters α and $\beta < \alpha$ as depicted in fig. 2.1a. The set of all hysterons Γ is thus indexed by $\mathcal{T} = \{(\alpha, \beta)\,|\alpha > \beta, |\alpha| < u_{max}, |\beta| < u_{max}\}$, forming a triangle on the (α, β) plane (limiting $|\alpha|$ and $|\beta|$ is necessary for the model to be well-behaved). Thus,
>
> $$f(t) = \int_{(\alpha,\beta)\in\mathcal{T}} \mu(\alpha, \beta)\hat{\gamma}_{\alpha,\beta}u(t)\mathrm{d}\alpha\mathrm{d}\beta$$

Note : The only hysterons whose cycles are symmetrical with respect to $u \mapsto -u$ are the ones where $\alpha = -\beta$. The asymmetrical ones do not translate directly to physical, single particles, but could represent domains where interactions between magnetic moments are strong.

2.2 The Preisach triangle, flipping fronts and memory effects

Representing data on the Preisach triangle $\mathcal{T}$ (fig. 2.1b), allows for some graphical intuition on the state of the model at a given time.

A hysteron is always in one of two possible states : "up" ($f_{\hat{\gamma}} = 1$) or "down" ($f_{\hat{\gamma}} = -1$). Thus, the Preisach triangle $\mathcal{T}$ is always partitioned into two regions S_+ and S_- which are the sets of up



(a) A hysteron's hysteresis cycle. α is the up-switching value, and β is the down-switching value. Due to energetic constraints, $\alpha > \beta$. Hysterons have no instrinsic dynamics, apart from the switching which happens instantly.

(b) The Preisach triangle as a way to index hysterons in an orderly fashion. Each hysteron is represented by a single point on the triangle.

Figure 2.1: Hysterons and their placement in the Preisach triangle.

and down hysterons respectively. When $|u| \geq u_{max}$, the system is said to be *saturated*, meaning all hysterons are in the same state (respectively "down" if $u = -u_{max}$ and "up" if $u = u_{max}$) and one of the regions ends up being of measure 0.

Let's suppose that the system is in its "down" saturated state. Starting from $u = u_{max}$, we increase u to a first value u_1. Then every hysteron for which $\alpha \leq u_1$ will have flipped up in the process. This creates a horizontal front in the Preisach triangle, of equation $\alpha = u_1$, which separates S_+ (at the bottom) from S_- (at the top). **In general, increasing u means creating a horizontal front at $\alpha = u$ below which all hysterons are flipped up. The state of the hysterons over this horizontal line remains unchanged.**

Now, let's decrease u from u_1 to $u_2 \leq u_1$. Then all hysterons whose β value gets crossed by u will flip down. That, is, when u reaches u_2, all hysterons to the right of $\beta = u_2$ will be flipped down, creating a vertical flipping front. Once again, variations of u create new conditions on which hysterons flip : **Decreasing u creates a vertical flipping front, moving towards low β, leaving its left unchanged and flipping all hysterons to its right in the "down" state**.

These two principles give insight into the Preisach model's hysteresis. What matters to understand this is the succession and extent of the periods of increasing and decreasing input, which will determine the (generally staircase-shaped) flipping front in the Preisach triangle (see fig. 2.2). This both embodies the memory and the possibility to "forget" that the Preisach model has : so long as the maxima and minima happen with decreasing amplitude (maxima are lower and lower while minima are higher and higher), the flipping front progresses less and less each time, so the total front is staircase-shaped and retains the whole information of the input. However, if, for example, a maximum u_{m+k} is higher than a preceding maximum u_m, then the horizontal front it creates will go further than $\alpha = u_m$, flipping all hysterons under it and erasing the front that u_m created. The model thus "forgot" part of it history.

The simple, rectangular hysteresis cycle of Preisach hysterons makes organizing them and tracking their states possible through the use of the Preisach triangle. The presence of maxima and minima in the input signal determines the shape of the domains S_+ and S_- that split the Preisach

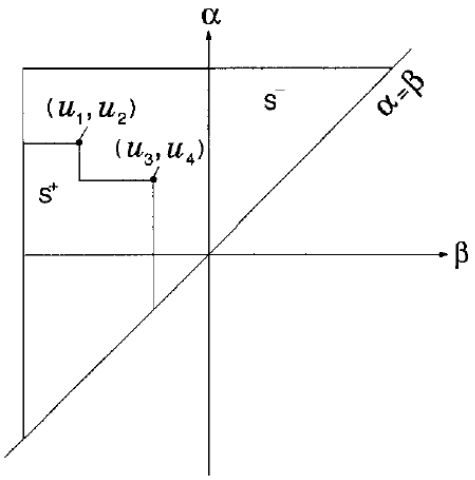


Figure 2.2: After a few variation changes, the flipping front in the Preisach triangle looks like a staircase.

triangle based on the state of its hysterons. The front separating the domains is generally staircase-shaped ; maxima of u determine its horizontal parts, and minima in u determine its vertical parts. Overpassing previous extrema of u overwrites their parts of the front, creating a "forgetting" effect. Taken to the extreme, bringing u to or beyond either $-u_{max}$ (resp. u_{max}) flips all hysterons and puts the system in a saturated "down" (resp. "up") state.

2.3 Identifying $\mu(\alpha, \beta)$ from experimental measurements

The Preisach model's way of shaping hysteresis cycles is through the weight function $\mu(\alpha, \beta)$. This gives more or less importance to some hysterons, shaping the resulting hysteresis loops. Thus, to fit the Preisach model to existing data, one may want to try and identify the corresponding weight function, which is not an easy task. The derivation of the following method is explained in detail in [Mayergoyz, 2003]. Here, only the result is presented.

Consider the system in a saturated down state. Then, the simplest way to sample its behaviour as a function of α and β is to subject it to one maximum and one minimum. That is, the input goes up to α, then down to β. The output curve corresponding to this input is called a *f*irst-order transition curve and its final value, when the input reaches and stops at β, is written $f_{\alpha\beta}$. If $\beta = \alpha$, that is, if the input signal does not decrease once it has reached α, then the resulting value is just abbreviated as f_α. A first-order transition curve is sketched out in fig. 2.3.

[Mayergoyz, 2003] derives that :

$$\mu(\alpha, \beta) = \frac{1}{2} \frac{\partial^2 f_{\alpha\beta}}{\partial\alpha\partial\beta} \tag{2}$$

While this gives an expression for μ, it is far from a perfect one, as taking derivatives of noisy, real-world data amplifies the noise. Moreover, it does not capture any offsets of the cycle, which will always be centered at $f = 0$. Finally, numerically getting the output from μ and u forces us to make a surface integral over the whole Preisach triangle, which is slow to numerically implement.

2.4 The method of sums to avoid derivatives

Another method proposed in [Mayergoyz, 2003] is to completely bypass the weight function. Looking at a staircase front in the Preisach triangle, [Mayergoyz, 2003] divides up the triangle in trapezoidal

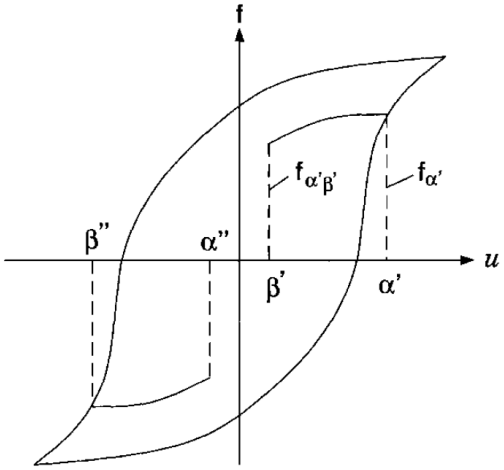


Figure 2.3: A first-order transition curve starts at the very bottom of the cycle, goes up to $u = \alpha'$ then down to $u = \beta'$.

regions, which allows, using only the knowledge of $f_{\alpha\beta}$ without taking derivatives, to implement the Preisach model. his negates the need for differentiation to identify the model and for integration to use it. One again, the derivation is long and available in [Mayergoyz, 2003]. Its result is :

Starting from negative saturation, and marking all successive maxima M_k and minima m_k of u that matter (ie, that are present as stair steps in the flipping front of the Preisach triangle), the system's output is :

$$\begin{cases} f(t) &= -f^+ + \sum_{k=1}^{n-1} \left(f_{M_k m_k} - f_{M_k m_{k-1}}\right) + f_{M_n u(t)} - f_{M_n m_{n-1}} \text{if u(t) is decreasing at current time} \\ f(t) &= -f^+ + \sum_{k=1}^{n-1} \left(f_{M_k m_k} - f_{M_k m_{k-1}}\right) + f_{-m_{n-1}} - f_{-m_{n-1},-u(t)} \text{if u(t) is increasing} \end{cases} \tag{3}$$

This expression us much more pleasant to work with : so long as we sample $f_{\alpha,\beta}$ regularly over the whole Preisach triangle, interpolation will allow us to estimate $f_{\alpha,\beta}$ to acceptable precision. By not using μ directly, we don't need to take derivatives of compute expensive integrals. This method also supports offsets in f naturally. However, we will need to keep track of the history of the maxima and minima of u and implement the dynamics of "forgetfulness" that corresponds with the evolution of the flipping front in the Preisach triangle. This is a small price to pay in comparison to the better quality of results and the performance increase. This is the preferred method for the numerical implementation of the model.

3 Inverting the Preisach Model

Now that the Preisach model has been introduced and a viable method to numerically compute its output has been derived, the question of inverting the model is raised. That is, with the system in a certain state, what input should one choose to bring the system to another, specific state ? Since the system is neither linear nor time-invariant, a nice, closed-form inversion of the model seems out of reach. Instead, a simple stepped method is described.

3.1 Going step by step : A simple feed-forward control setup

One approach to solving this problem is to imagine that the nonlinearities of the model are akin to errors and try and implement a sort of feed-forward control setup where we use this known non-linearity to take a guess and iterate taking small steps to reach our goal.

We first begin by writing, once again, $f(u) = f_0 + \gamma u + \Delta f$ where γ is some kind of linear part (for example, we take $\gamma = \frac{f_{max} - f_{min}}{u_{max} - u_{min}}$ in the Python implementation. A linear fit of the maximal hysteresis cycle would probably be alright too.) We also choose a number $0 \leq \alpha \leq 1$ that will dictate how the actual step taken is scaled with respect to the linear prediction. To avoid backtracking and the issues it'd cause with keeping track of the history of the model, it is preferable to approach the desired value by always undershooting the steps we take, hence $\alpha \leq 1$. Denoting the current output of the system by f_t and the setpoint to reach by f, the step δu to take is then calculated like this :

$$\delta u = \alpha \frac{f - f_t}{\gamma} \tag{4}$$

The algorithm to follow is relatively simple :

1. Compute δu from the current state of the system using (4).
2. Update $u_{new} = u + \delta u$.
3. Compute the state & output of the system at this new value of u.
4. Repeat until the output is within a desired range of the set point.

This is, in the Python code, implemented using a `while` loop in the `aim` method of the `Transducer` class. So long as the desired precision hasn't been reached, and the number of steps taken in under some limit, the algorithm keeps stepping.

If the adjustment to be made in f is small enough and provided there will be no overshoots, the algorithm described corresponds to simulating a first-order non-linear discrete control system. Choosing a small enough value of α should prevent overshoots (cf. first-order linear discrete control theory), and the algorithm will converge towards the right value of f. However, if α and γ are not chosen carefully enough, there may be overshoots, which adds unnecessary stair steps in the Preisach triangle and does not guarantee convergence as the system will exhibit hysteretic nonlinearities.

3.2 Optional recursion for overshoot prevention

Choosing α is difficult without studying the complete dynamics of the system around f, so adjusting it by hand could possibly be very tedious. To circumvent this issue, we detect overshoots, roll the system's history back to where it was before we started stepping, and retry with a smaller value of α until no overshoot is detected. This is done by recursion, finding overshoots by looking for changes in sign of δu (or equivalently, in $f - f_t$). If an overshoot is detected in the current `while` loop, the current value of α is saved, the history is rolled back to where it was at the beginning of the `aim` function call, then `aim` is called again with $\alpha \leftarrow \frac{\alpha}{2}$.

3.3 Preventing excursions away from the model's input limits

Near the edges of the input domain, the hysteresis curve flattens. This, along with noise in the data used to interpolate $f_{\alpha\beta}$, can cause the algorithm to predict an input that is outside the authorized range, raising an error in the interpolation function. In this case, the value is clipped to the nearest allowed input. If this passes the precision requirements of the algorithm, then function execution resumes as is. Else, since overshoot prevention measures are already in place, this means that we can't match the precision requirements. The value is returned anyway along with a warning.

4 Implementation and measurements

4.1 Managing the memory behavior

Equation (3) states that the only relevant data to compute the output of a Preisach system is its past minima and maxima in u or, equivalently, the positions of the flipping front's steps in the Preisach triangle. As discussed in part 2.2, subsequent maxima and minima in u will negate the effects of lower-amplitude extrema by making the corresponding steps in the Preisach triangle disappear. Thus, to completely know the state of a Preisach system, in addition to $f_{\alpha\beta}$ which can be measured, we must keep track of this list of extrema and make sure it always corresponds to its representation in the Preisach triangle by enforcing rules on the insertion of new values. The data is stored in a 2 by n `NumPy` array of variable length depending on the number of steps. If need be, new values are inserted at the end of the array. The array is to be thought of as a list of n vectors having 2 components. The n-th vector is thus $\left[\begin{smallmatrix} M_n \\ m_n \end{smallmatrix}\right]$, respectively the n-th maximum and the minimum which follows it.

The array is initialized when the system receives its first input V by taking the value $[[V, V]]$. By default, the system was considered to begin at negative saturation, so the first point in the Preisach triangle is that of a maximum. The next minimum can be seen as degenerate and having the same value. The history array is now set up in a valid state and can be worked on.

Next comes the question of how to insert values. Let u_{n+1} be the new input value to insert into the history. There are three cases, depending on the relative position of u_{n+1}, M_n and m_n :

- $m_n < u_{n+1} < M_n$: u_{n+1} is a new max that doesn't cause any forgetting. In the same vein as when initializing the history, we concatenate $[[u_{n+1}, u_{n+1}]]$ at the end of the history array.
- $u_{n+1} \geq M_n$: u_{n+1} is a new max but has caused forgetting. We need to delete every $[M_k, m_k]$ pair where $M_k \leq u_{n+1}$, and then concatenate $[[u_{n+1}, u_{n+1}]]$ at the end of the resulting array.
- $u_{n+1} \leq m_n$: u_{n+1} is a new minimum that causes some forgetting. We delete every $[M_k, m_k]$ pair where $m_k \geq u_{n+1}$, and replace the minimum of the last remaining pair by u_{n+1}, making it the last minimum.

These three cases are summed up in fig 4.1.

4.2 Python object-oriented implementation for the model

The simulations are coded in `Python`, using classes to have instances of the model be as self-contained as possible.

In order to make sure that the sum-based implementation of the Preisach model does simulate the model accurately, it needs to be tested on sample measurements of Preisach hysteresis loops. The samples are generated using the (slower, but most importantly, differently-structured) integral method of calculating Preisach behavior. This way, once the integral model is considered to be sufficiently bug-free, its output can be used as an input to the sum-based model for validation of agreement between the two implementations.

The development of both objects was done in a sequential way, creating parent classes that inherit from one another.

4.2.1 Common to both models : `History` and `History_primitive`

Both models need to manage their history as described in 4.1. This is grouped in a class called `History`, that implements the following methods:

- `insert` uses the algorithm described in 4.1 to insert values into an existing history array.
- `update` wraps `insert` and adds auto-creation of the array if it is not yet initialized.
- `reset` resets the history to it uninitialized value : `None`.

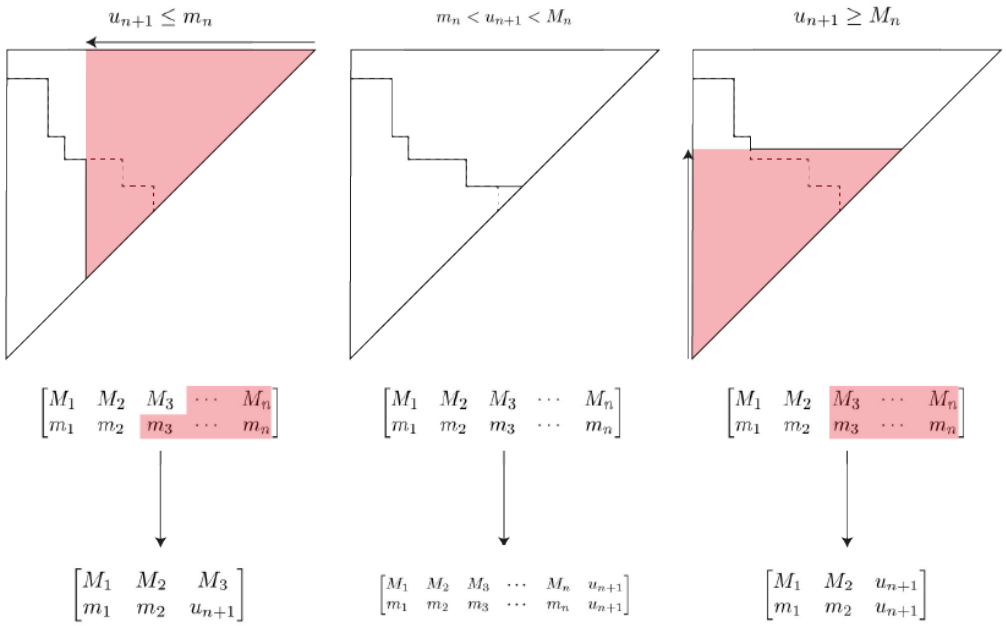


Figure 4.1: A summary of the three cases to consider when adding a new input value to the history. Arrows and the corresponding red zones denote the effect of u_{n+1}.

`History_primitive` is the true base class in the inheritance tree. It packages an instance of `History` along with some IO (by making relevant sets/gets on `History` objects got through `@property` decorators). It also adds methods[1] to make debugging and plotting states easier :

- `generate_hysteron_state_function` creates a function `delta(alpha, beta)`, defined over the Preisach triangle, that returns 1 or -1 depending on the current state of the hysteron at `(alpha, beta)`.
- `make_flipping_front` returns the coordinates of the front separating S^+ and S^- in the Preisach triangle, for plotting purposes.

4.2.2 `Integral_Preisach` and `Preisach`

These two classes work together to implement the Preisach model in two different ways, to provide mutual validation.

`Integral_Preisach` uses the integral definition to compute the model's output. It performs two integrals, one over S^+ and one over S^- which are defined as bounds[2] for `scipy`'s `dblquad` integration function, and substracts their results. Of course, it needs a weight function μ, which is given at instantiation. It is then used through the `to_value` method, which inserts its input in the history, and then computes the corresponding output. `get_value` gives the current output without inserting any new input.

`Preisach` implements the Preisach model using the sum-based expression. At instantiation, it takes as an input a measurement of the first-order transition curves of the system of interest. Just like it intergal-based counterpart, it has two methods `to_value` and `get_value`, that behave in the same way. Under the hood, `generate_model` interpolates the measurements, `plot_mesh` visualizes the measured first-order transition curves in the Preisach triangle, and `clip_input` provides a way for the daughter class `Transducer` to limit its inputs.

The validation process goes as follows :

[1] The method `make_integrand_bound_functions` is not described here as it was moved to the `Integral_Preisach` class. This should not break anything as it is the only class that uses it, but it has not been tested yet.

[2] Done by the `make_integrand_bound_functions` method.

1. Instantiate `Integral_Preisach` with a known μ function.
2. Generate the set of its first order transition curves.
3. Instantiate `Preisach` using this set.
4. Reset the integral model's history.
5. Generate an array $u(t)$ to take as an input, feed it to both models using their respective `to_value` methods, and get the corresponding outputs.
6. Compare the resulting output curves.

This validation process is used to debug the implementations. The code has now been validated ; all this is done in a Jupyter notebook.

4.2.3 `Transducer`

This class inherits from `Preisach` and implements the aiming algorithm described in section 3.
It has the following new methods :

- `check_validity` is used under the hood to make sure that the setpoint used in `aim` is valid.
- `aim` returns the next input value to take in order reach the setpoint passed to it as an argument. The algorithm described in section 3 is implemented in this function.

`Transducer` is the class to use in practical settings.

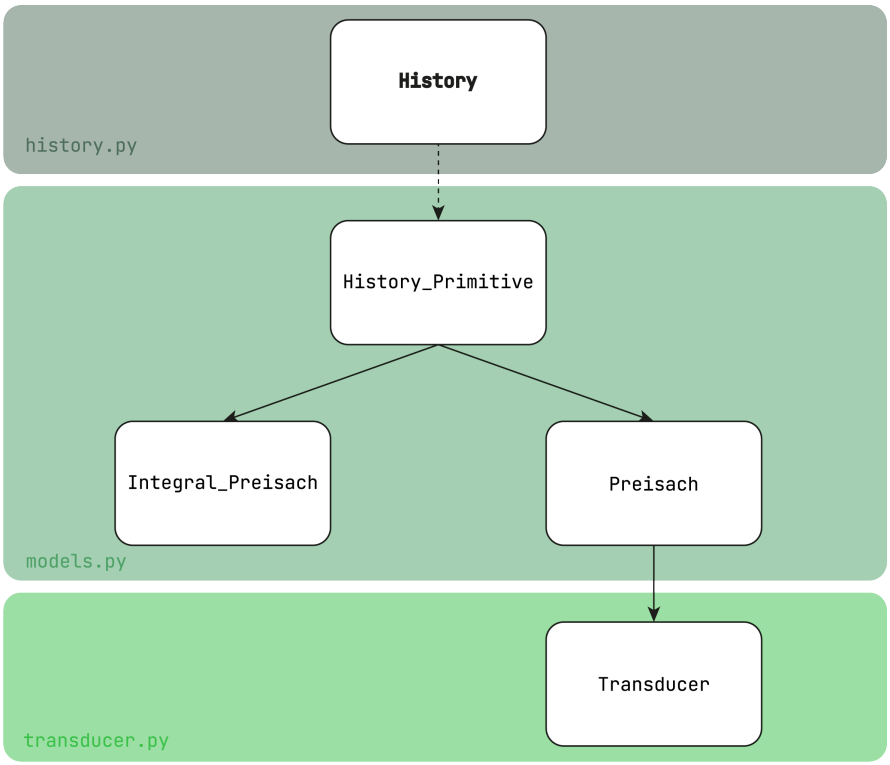


Figure 4.2: Inheritance diagram for the Python project. Full arrows convey inheritance, dashed arrow indicates wrapping.

4.3 Measurements

4.3.1 Setup

The filter is controlled by a Yokogawa 7651 DC Source and its RF characteristics are measured by Collège de France's Rohde&Schwarz ZVK VNA. The filter also has to be powered and cooled, which is managed by a breakout box Jean-Loup made. The Yokogawa source is programmed on the fly to provide relatively smooth, linear transition between voltages, rather than steps.

The idea of the following tests is to make sure that the algorithm and model described during this report give acceptable results, with low enough drift. As the passing band of the filter is at least 20 MHz wide, this is chosen as the max allowed error on the output of the Preisach prediction.

The model is calibrated by measuring the first-order transition curves as described earlier and using them when creating the `Transducer` objects.

4.3.2 Dampened sine setpoint : a simple test

The frequency setpoint is chosen to follow a damped sine curve so as to visit most of the filter's range, and then sampled. We then call `aim` on the successive values of the setpoint, creating a series of values of V. They are successively used as inputs to the filter, taking traces with the VNA, and the position of the band center is compared with the predicted position by the Preisach aiming algorithm. This error is then plotted, as well as a comparison of the general shapes of f_{setpoint} and V, and the evolution of the VNA trace. A measurement is shown in fig.4.3 and show that once the model is calibrated to the filter's first-order transition curves, the error is up to spec.

The damped sine is chosen because its many maxima and minima of decreasing amplitude force the model to keep track of a long history, making this a stress test of sorts.

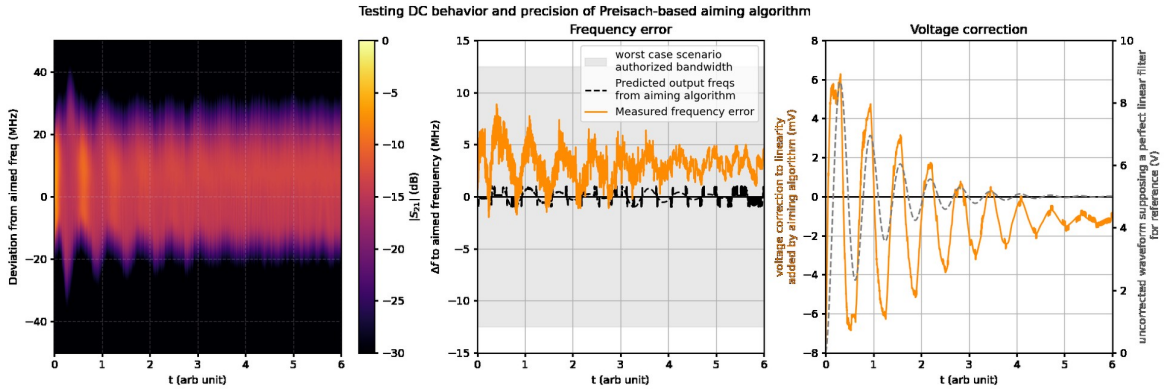


Figure 4.3: First measurement of filter accuracy.

4.3.3 Amplitude-modulated sine : verifying that no drift or error accumulates

Instead of exponentially damping the sine amplitude, it is now periodically modulated, to make the filter follow the same path multiple times and check that errors don't accumulate and that no drift happens.

The results are shown in fig. 4.4. Even after many periods of the envelope, the error does not seem to drift and is still up to spec.

4.3.4 An interactive widget for setting the filter to arbitrary frequencies on the fly

A simple Jupyter Widget was coded that displays the filter's current history, the VNA trace of the current state of the filter, as well as comparisons of aimed, predicted, and real positions of the filter band center. The frequency setpoint can be adjusted on the fly, as depicted in fig.4.5.

Playing around with this tool, it seems that the filter does show some dynamics effects which the Preisach model doesn't capture : reaching a frequency that is more than around 10 GHz away by

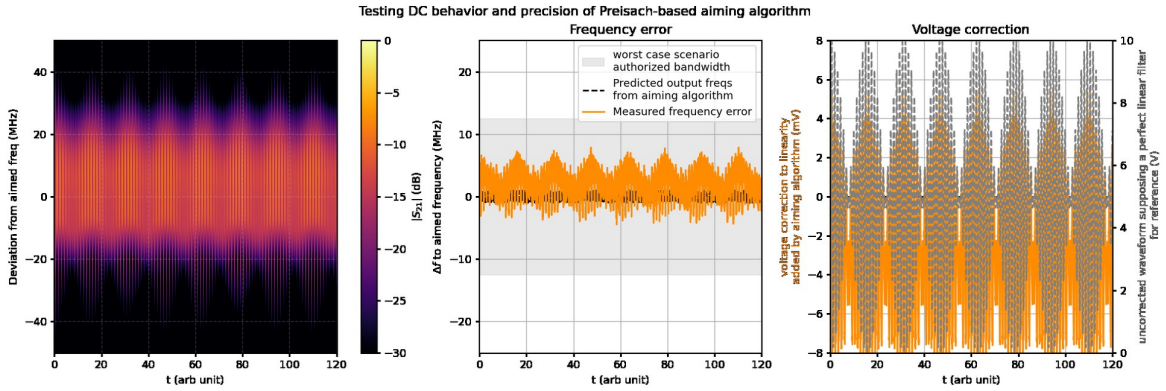


Figure 4.4: Measurement of model drift.

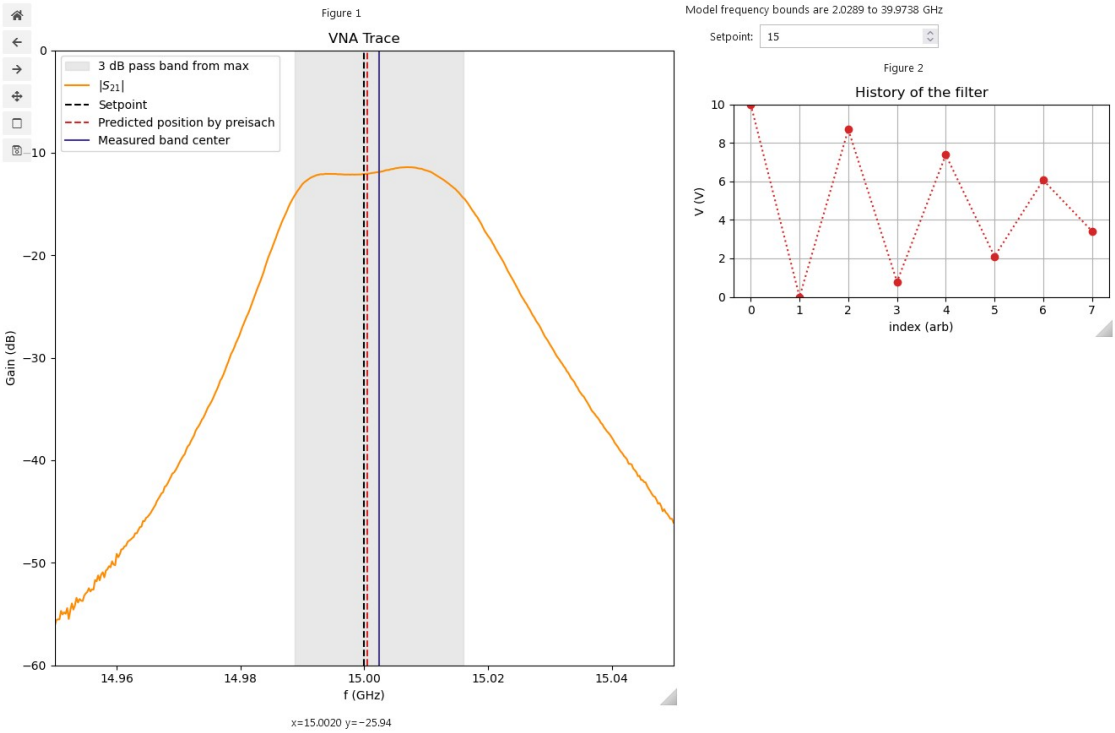


Figure 4.5: A simple interface for setting frequency on-the-fly.

doing successive setpoints that are closely spaced, the filter behaves as predicted, while doing one big jump introduces some kind of irreversible error that is only accounted for by recalibrating. Since the Yokogawa source is set to sweep voltage steps during a constant time regardless of the step's amplitude, the only difference is the speed at which the applied voltage varies. It seems, from rough experimentation, that jumps up to 1 GHz pose no issue at all.

Based on the relatively low frequency at which this happens (in comparison to the magnetic behavior of the YIG crystal, which happens at the GHz scale), my hypothesis is that the filter's driver is at fault here. I think voltage steps that are too fast may cause an overshoot of some type in the driver, introducing small oscillations that cause the behavior of the filter to diverge from the Preisach model. The phenomenon I describe here could also be an issue of the Yokogawa voltage source.

4.3.5 A very rough estimation of a maximum voltage slope

From very quick experimentation at the very end of my internship, it seems to me that the maximum voltage slope allowed before the Preisach model breaks down is somewhere between 25 $V.s^{-1}$ (a very conservative lower bound, it seems that 50 may be more likely) and 100 $V.s^{-1}$ (a very loose upper bound). However, more experimentation is needed.

4.4 Using the code

Most of the code is hosted at `https://github.com/MechanicalPenguin225/preisach`, in the form of a Python package for the transducer code, and an annex for the practical measurement code. A `.md` file at the repo's root describes the structure of the data I used during the internship, which is located on the Win7 PC.

4.4.1 Measuring the first-order transition curves

Code has already been written to measure first-order transition curves. The file `measure_preisach_triangle.py` measures and saves the curves with the structure used by the `Transducer` class. 51-point measurements are usually alright for shorter calibrations, but to get good precision, 101-point to 201-point measurements are strongly recommended. Be warned, however, that they take a long time (a few hours at worst).

4.4.2 General usage

`measure_preisach_triangle.py` outputs a `.npz` file, which is loaded and whose entries are used at construction of an instance of `Transducer`. The transducer object is initialized at negative saturation, that is, 0V, so make sure that the filter also begins at 0V. Then, calling `TRansducer.aim(frequency)` will return the voltage that you need to set the filter to to get to the required frequency. You may thus do the computation inside a measurement loop, or, if the aimed frequencies are known, do the whole computation of the voltages in advance.

5 Conclusion and perspectives

5.1 The Preisach model for modelling the YIG filter

So long as the control voltage is varied slowly enough in time, **the Preisach model is sufficient** to estimate the behavior of the YIG filter and implement a simple feed-forward control setup. Its time-independence (in the sense that only input voltages, and not their time derivatives, are relevant to the model) means that it is cheaper to compute and easily computable in advance if the desired setpoint evolution is known.

However, a lot of work is still needed to turn this proof of concept into a usable component that can be worked as a black box. A dynamical approach could be taken, though there seems to be very little literature on which model is useful and how to fit different parameters.

5.2 Sourcing better-suited equipment

The Yokogawa source is well-suited to high-precision, DC applications. An AWG with a precision at least on the order of 1 mV is needed to ensure that the filter responds well (for example, the GWinstek sources caused issues with skips in their voltage when passing certain thresholds). To me, this is the most urgent change as it will allow us to see if the voltage slope requirements change with a different voltage source (which would point towards the Yokogawa source being the culprit for the voltage slope issues).

5.2.1 Alternative : a Pound-Drever-Hall setup

A Pound-Drever-Hall setup may be used in place of simulating the behavior of the filter, as it wil only be used to follow a particular frequency. However, due to the very wide frequency range to cover, finding suitable components (notably a phase-shifter) may prove impossible or very expensive.

References

[Mayergoyz, 2003] Mayergoyz, I. (2003). *Mathematical Models of Hysteresis and Their Applications*. Elsevier.