Soft Magnetic Materials and Ordinary Differential Equations: From Linear Circuits to Neural Network Models

Thomas Guillod

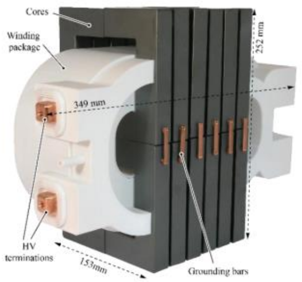
Dartmouth College, USA

IEEE MagNet Challenge Webinar
May 23, 2025

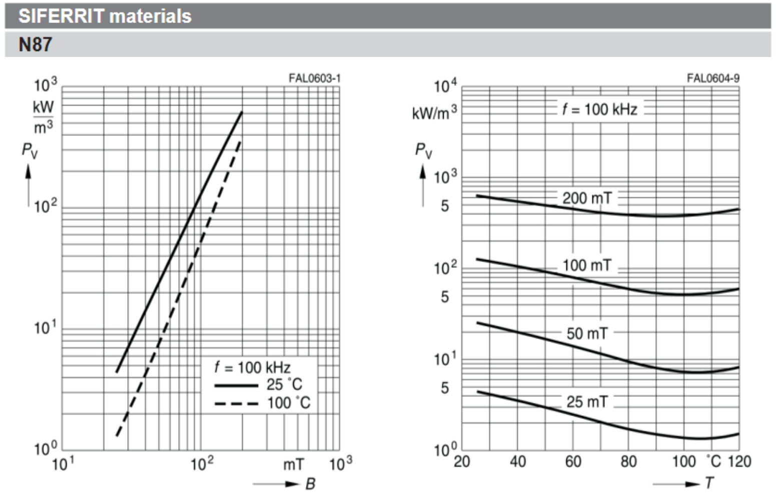
Magnetic Material Models

- **Magnetics** are a **bottleneck**
  - Bulky, expensive, lossy
  - Challenging design process
- **Soft magnetic** core **material**
  - Inductors, transformers, sensors, etc.
  - Datasheet: only sinusoidal and incomplete
  - Models: inaccurate (up to 100% deviation)
  - No accurate first principles model
- **Better models are required**

[Dartmouth]



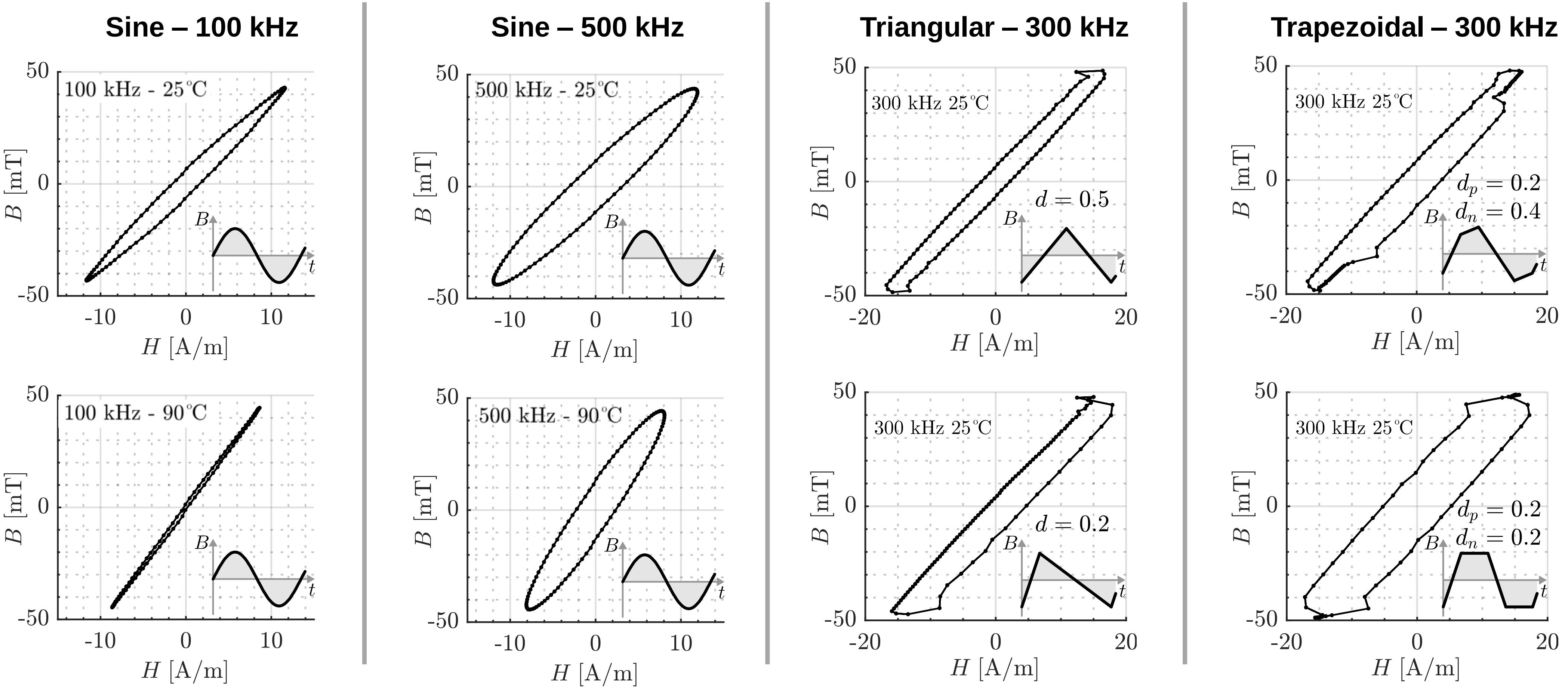
[ETHZ]



[TDK-EPCOS]

Magnet Challenge 1

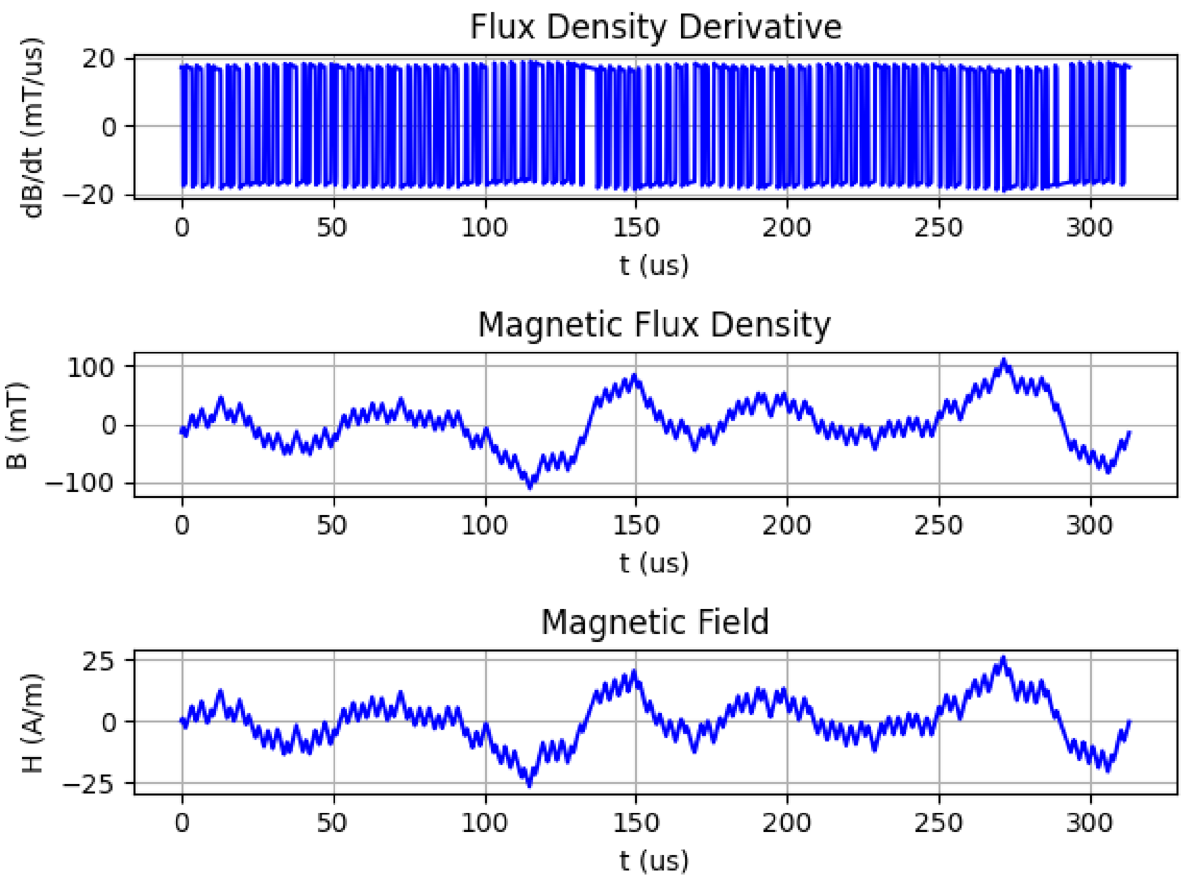
- **Nonlinear →** Amplitude, waveshape, frequency, temperature

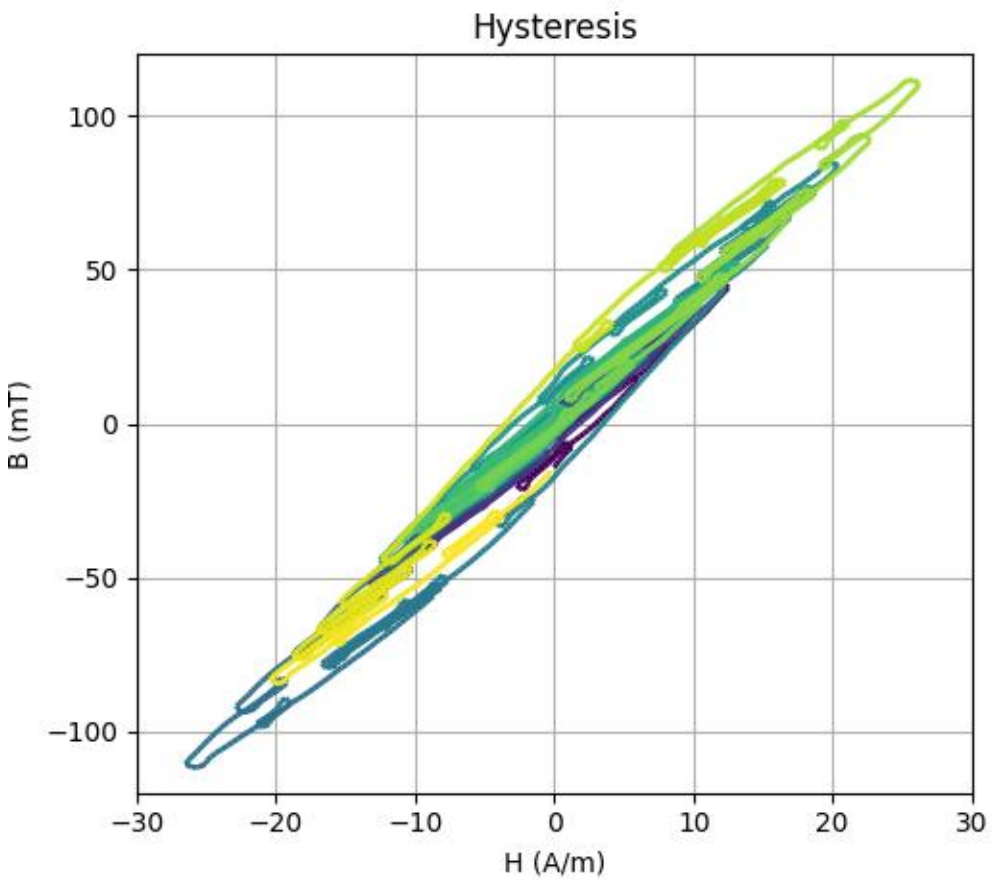


[example EPCOS/TDK N87]

Magnet Challenge 2

- **Arbitrary PWM →** longer waveforms with minor loops

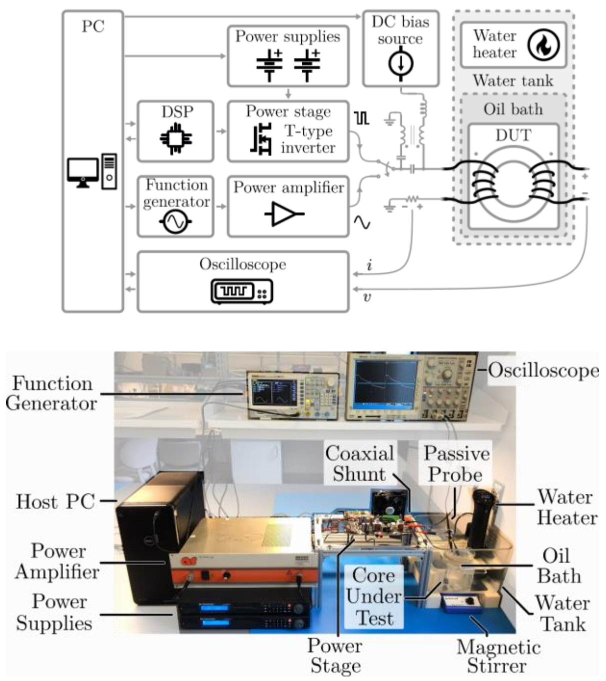




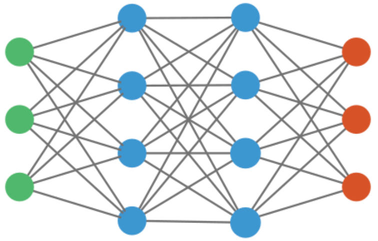
[example EPCOS/TDK N87 at 25C]

MagNetX Dataset

- **MagNetX Dataset**
  - 10 different materials
  - Over 10GB of measurements



- **MagNet Challenge 2**
  - Innovative models
  - Accurate & versatile
  - Usable for PE engineers



Machine Learning

$$P = \frac{1}{T}\int_0^T k\left|\frac{\mathrm{d}B}{\mathrm{d}t}\right|^{\alpha}(B_{\mathrm{pkpk}})^{\beta-\alpha}\,\mathrm{d}t$$

$$P = W_{\mathrm{hyst}} f_{\mathrm{eff}},$$
$$W_{\mathrm{hyst}} = a_1 B_{\mathrm{pkpk}} + a_2 B_{\mathrm{pkpk}}^2 + a_3 B_{\mathrm{pkpk}}^3$$
$$f_{\mathrm{eff}} = f\left(1 + c\left(\frac{1}{B_{\mathrm{pkpk}}}\int_0^T \left|\frac{\mathrm{d}^2 B}{\mathrm{d}t^2}\right|\mathrm{d}t\right)^{\gamma}\right)$$

Analytical Model

Part I:	**Modeling Workflow**
Part II:	**Circuit-Based Differential Equation Models**
Part III:	**Neural Differential Equation Models**
Part IV:	**Conclusion and Outlooks**

Part I:
Modeling Workflow

Workflow Step 1: Dataset

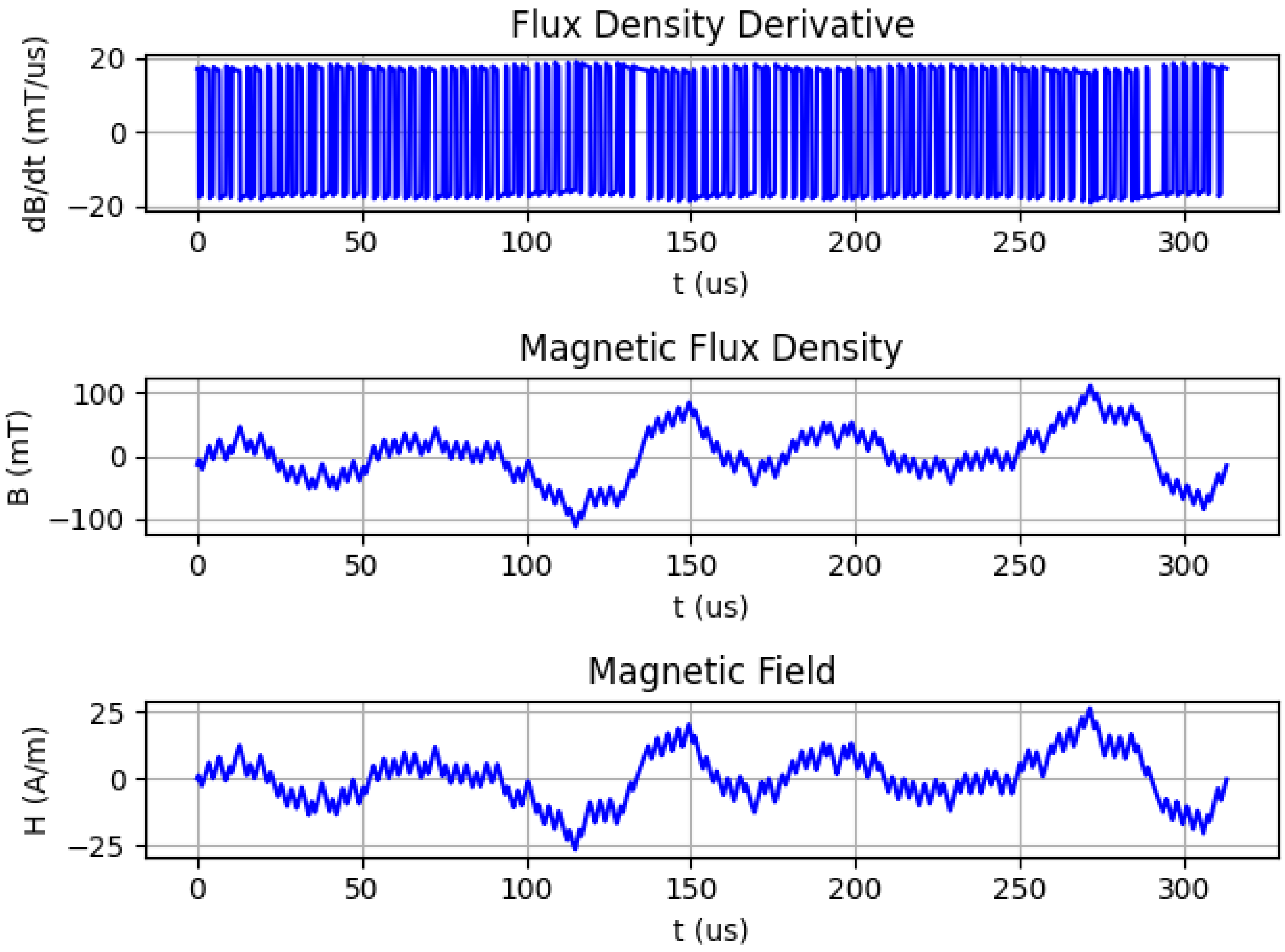
- **Load** the measurements
- **Select** the samples:
  - Select the training samples
  - Select the test samples
- **Prepare** the samples:
  - Compute the gradient of the flux density
  - Remove any offset and align the phase
  - Filter noise if required
- **Save** the resulting dataset

Data used in this presentation

MagNetX: “N_87_5”

- **EPCOS/TDK N87**
- **25°C operating temp.**
- **3.2 kHz rep. frequency**
- **48 training samples**
- **24 test samples**

Workflow Step 2: Training



Applied excitation

differentiate

Applied excitation

Used to fit the model

DARTMOUTH ENGINEERING

Workflow Step 2: Training

MagNet

1. Load the **dataset**
2. **Resample** the **dataset**
3. **Scale** the **model parameters**
4. **Split training and validation sets**
5. **Evaluate** the **model**
6. **Update param. with train. set.**
7. **Check convergence with valid. set.**
8. **Unscale** the **model parameters**
9. **Save** the model parameters

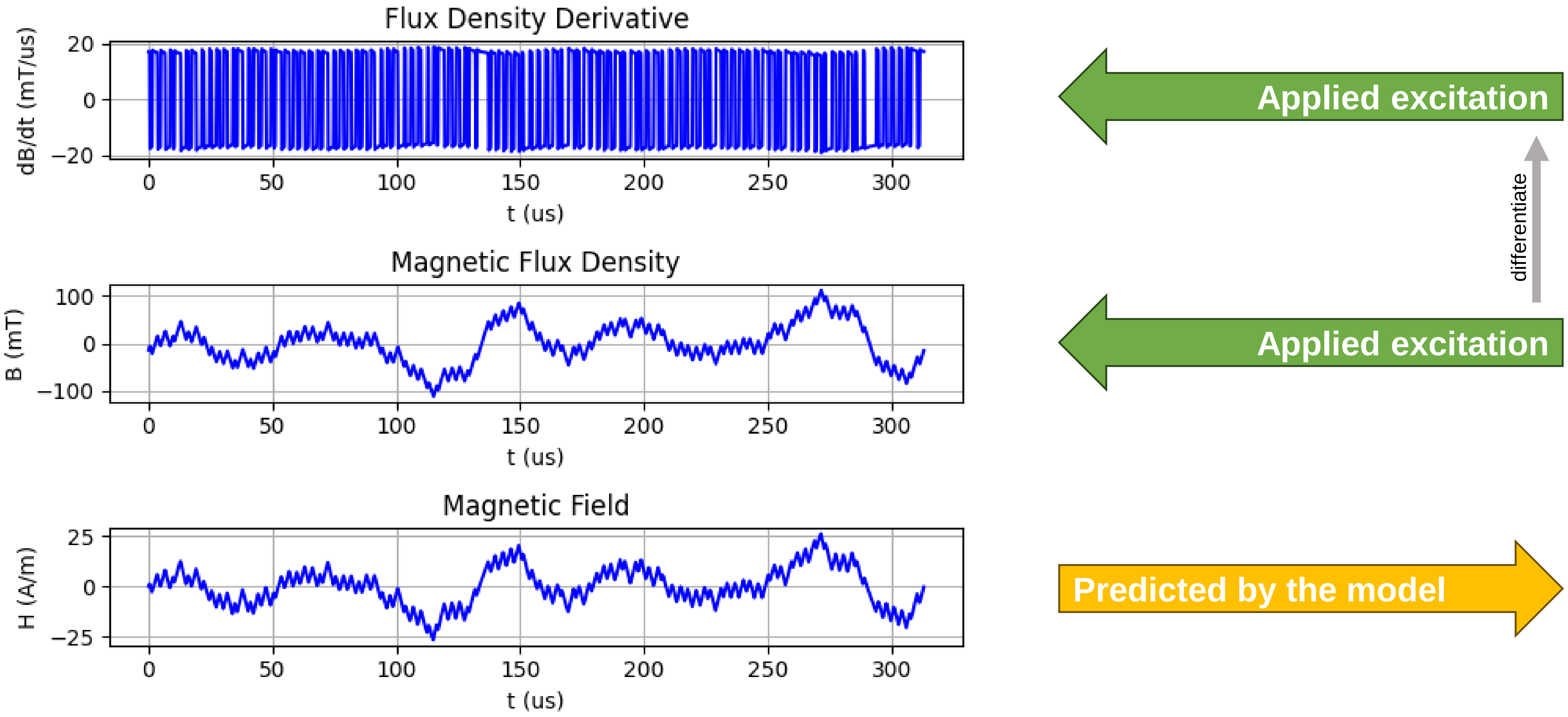
For each sample in the dataset.

- Unscale the model parameters
- Compute a penalty for bound violations
- Solve the model with the meas. dB/dt excitation
- Extract the predicted value for the magnetic field
- Compute the error metrics

$$\mathbf{err}_H = \frac{\mathbf{RMS}(H_{\mathbf{model}} - H_{\mathbf{ref}})}{\mathbf{RMS}(H_{\mathbf{ref}})}$$

$$\mathbf{err}_P = \frac{P_{\mathbf{model}} - P_{\mathbf{ref}}}{P_{\mathbf{ref}}}$$

Workflow Step 3: Inference



DARTMOUTH ENGINEERING

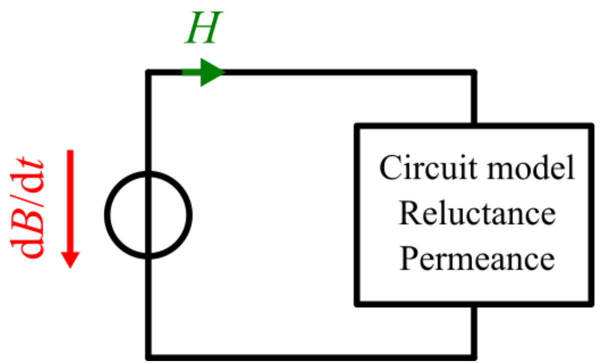
MagNet

Part II:
Circuit-Based Differential Equation Models

Circuit Model for Magnetics

- **Basic equations**
  - $\boldsymbol{V = N \cdot \int \frac{\mathrm{d}B}{\mathrm{d}t} \mathrm{d}A \ \Rightarrow \ V \propto \frac{\mathrm{d}B}{\mathrm{d}t}}$
  - $\boldsymbol{I = \frac{1}{\mathrm{N}} \cdot \int H \, \mathrm{d}l \ \Rightarrow \ I \propto H}$

- **Circuit model with ODEs**
  - Common for dynamic systems
  - Causality is enforced
  - Non-periodic waveforms
  - Compatible with SPICE
  - Compatible with FEM

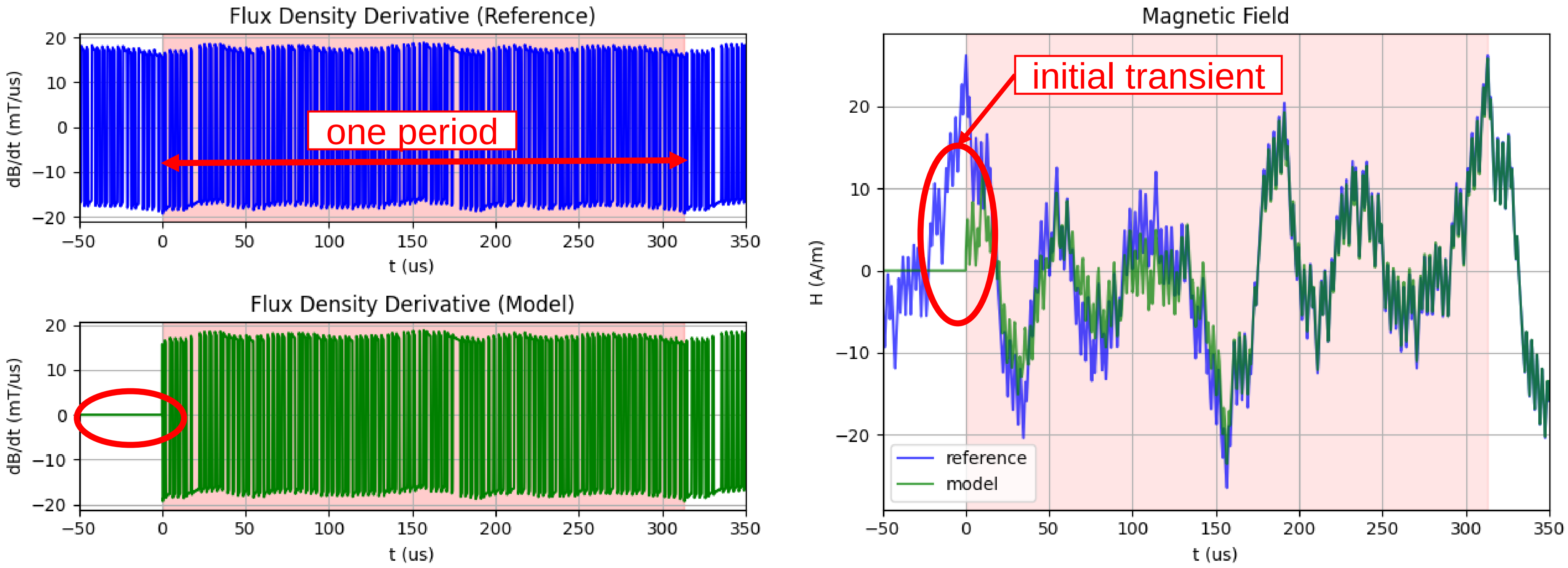


$$\frac{\mathrm{d}\vec{y}\,(t)}{\mathrm{d}t} = f\left(\vec{y}(t), \frac{\mathrm{d}B(t)}{\mathrm{d}t}\right)$$

$$H(t) = g\left(\vec{y}(t), \frac{\mathrm{d}B(t)}{\mathrm{d}t}\right)$$

Problem: Steady-State

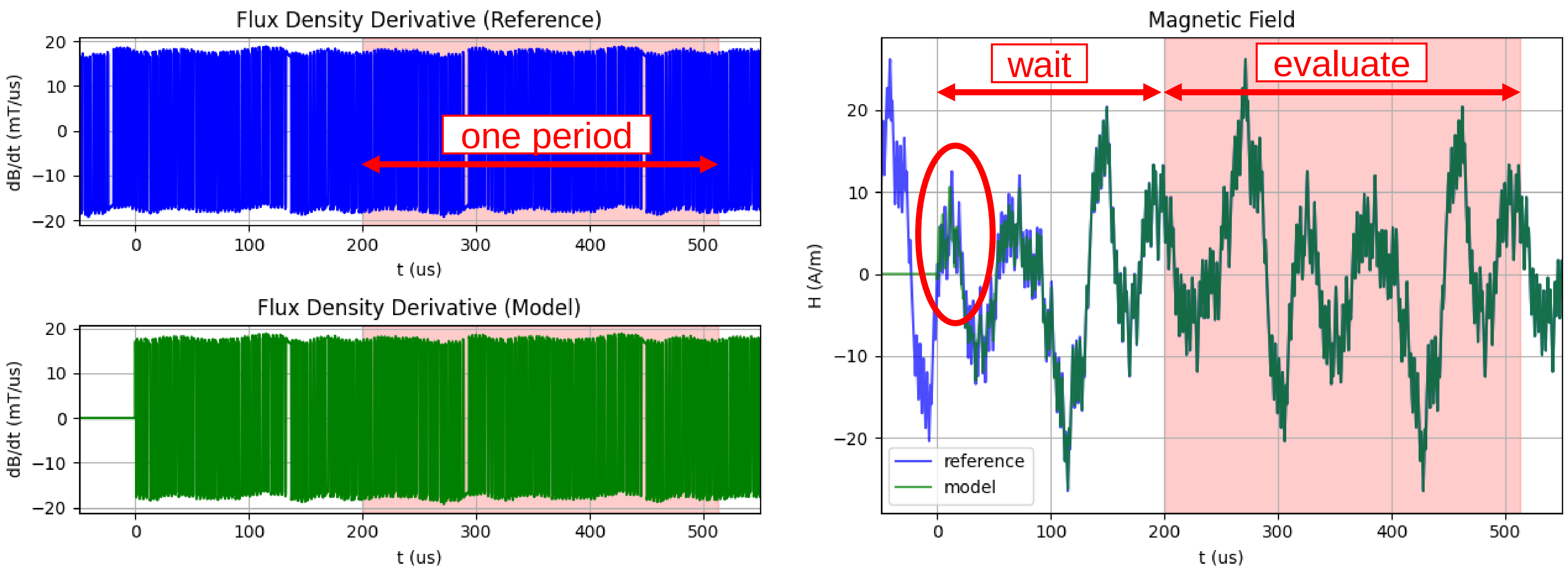
- The MagNetX dataset contains **periodic steady-state data**



- The **initial state** of the material is **not known**!

Problem: Steady-State

- The MagNetX dataset contains **periodic steady-state data**



- **Align the phase** with zero crossings and **wait** for steady-state!

Problem: Computational Cost

- Integrate **ODEs** for **thousand of time steps** with a measured excitation
- Repeat for the **hundreds of signals** composing the dataset
- Repeat **hundred of times** until the **optimal parameters** are found

[Oak Ridge, CC BY]

[Personal Laptop, CC BY]

JAX-Powered Computations

= **Python matrix / vector / array computation library**
Similar to NumPy/SciPy or MATLAB

\+ Just-In-Time **compiler, auto-vectorization, and parallelism**

\+ Transparent usage with **CPUs, GPUs, and TPUs**

\+ Extremely **powerful automatic differentiation** engine

\- **“The Sharp Bits”** - pure functions, immutable arrays, etc.

DARTMOUTH ENGINEERING

Simple JAX Example

MagNet

```
import jax
import jax.numpy as jnp

@jax.value_and_grad
def my_fun(val):
    return 1 / jnp.sqrt(val + 1.0)

# evaluate the function with a scalar
x = jnp.array(1.0)
(y, dydx) = my_fun(x)

# evaluate the function with vectorization
x = jnp.array([1.0, 2.0])
(y, dydx) = jax.vmap(my_fun)(x)
```

```
y = 0.7071067690849304
dydx = -0.1767767071723938

y = [0.70710677 0.57735026]
dydx = [-0.1767767  -0.09622505]
```

- Define a simple **mathematical function**.
- Compute the **value and gradient.**
- **Vectorize the function call for arrays.**
- **Possible with very complex operations.**

JAX as an Ecosystem

- **Neural networks:**
  - **Flax**: flexible neural networks
  - **Equinox**: lightweight neural networks
  - **Optax:** first-order gradient optimizers
  - **Leventer:** scalable foundation models

- **Other tools:**
  - **Optimistix**: root finding, minimization, etc.
  - **Diffrax**: differential equation solvers
  - **Lineax**: linear equation solvers
  - **BlackJAX**: probabilistic sampling

- In this work: JAX, Diffrax, Equinox, Optax, and Optimistix

ODE with JAX and Diffrax

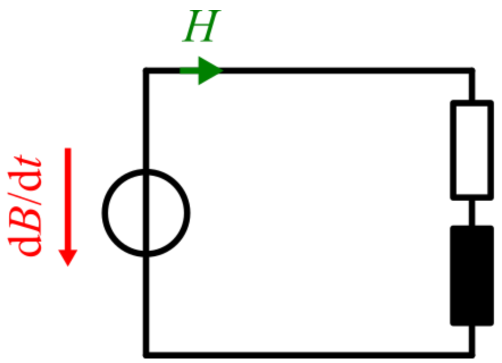
```python
import jax
import jax.numpy as jnp
import diffrax as dfx

@jax.jit
def eval_ode(R, L):
    # definition of the ODE: voltage step applied to a RL network
    term = dfx.ODETerm (lambda t, i, args: (1.0 - R * i) / L)

    # solve the ODE with the specified initial value
    sol = dfx.diffeqsolve(term, dfx.Dopri5(), t0=0e-6, t1=5e-6, dt0=50e-9, y0=0.0)

    # return the final value of the current
    return sol.ys[0]

# compute the ODE solution
R = jnp.array(5.0)
L = jnp.array(20e-6)
i = eval_ode(R, L)

# compute the derivative of the ODE solution
didR_auto = jax.grad(eval_ode, argnums=0)(R, L)

# compute the derivation with an approximation (slower, less accurate)
didR_diff = (eval_ode(R + 0.01, L) - eval_ode(R - 0.01, L)) / 0.02
```

i = 0.14269903302192688
didR_auto = -0.014214569702744484
didR_diff = -0.014217197895050049

- Define a **simple ODE**:
  - RL step response with 1V.
  - The ODE state is the current.
  - Return the current (last step).
- Evaluate the **solution of the ODE**.
- Evaluate the **derivative of the solution.**

Simple Linear R-L Circuit

- **Simplest possible circuit**



$$V(t) = R \cdot i(t) + L\frac{\mathrm{d}i(t)}{\mathrm{d}t}$$

$$\frac{\mathrm{d}B(t)}{\mathrm{d}t} = r \cdot H(t) + \mu_0\mu_r\frac{\mathrm{d}H(t)}{\mathrm{d}t}$$

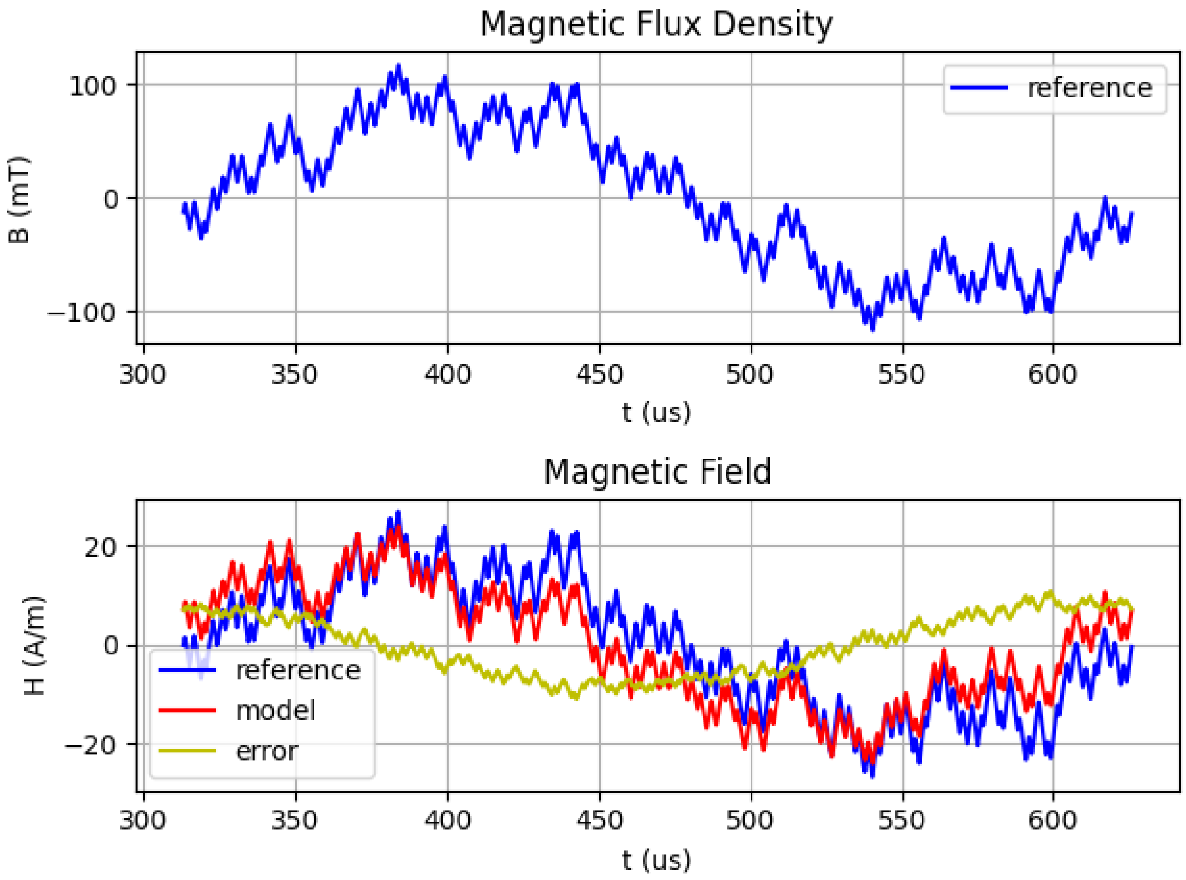
- **Single (linear) ordinary differential equation**
  - Two free parameters (resistance and permeability)
  - Cannot model saturation behavior
  - Cannot model nonlinear losses

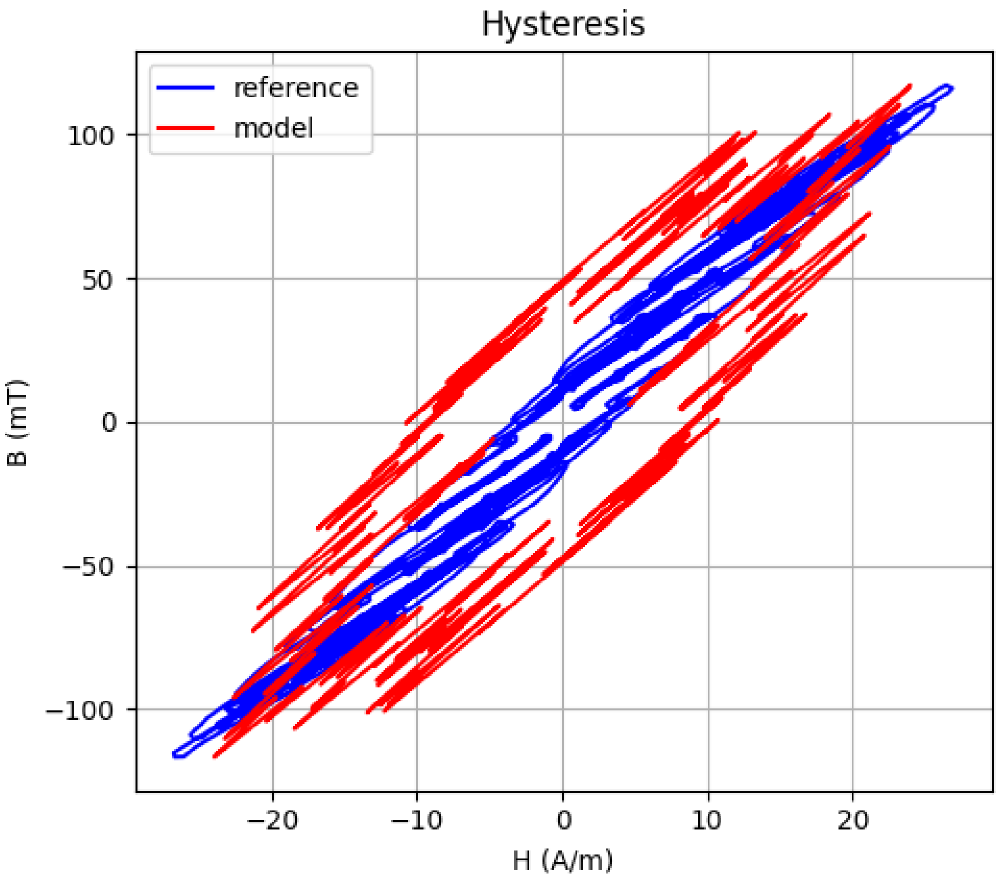
Simple Linear R-L Circuit

```python
@eqx.filter_jit
def get_ode(t, H, param, interp):
    # extract the variables
    r_lin = param["r_lin"]
    mu_lin = param["mu_lin"]

    # obtain the applied excitation
    dBdt = interp(t)

    # define the permeability
    mu0 = 4 * jnp.pi * 1e-7

    # compute the dynamic term
    dHdt = (dBdt - r_lin * H) / (mu0 * mu_lin)

    return dHdt
```

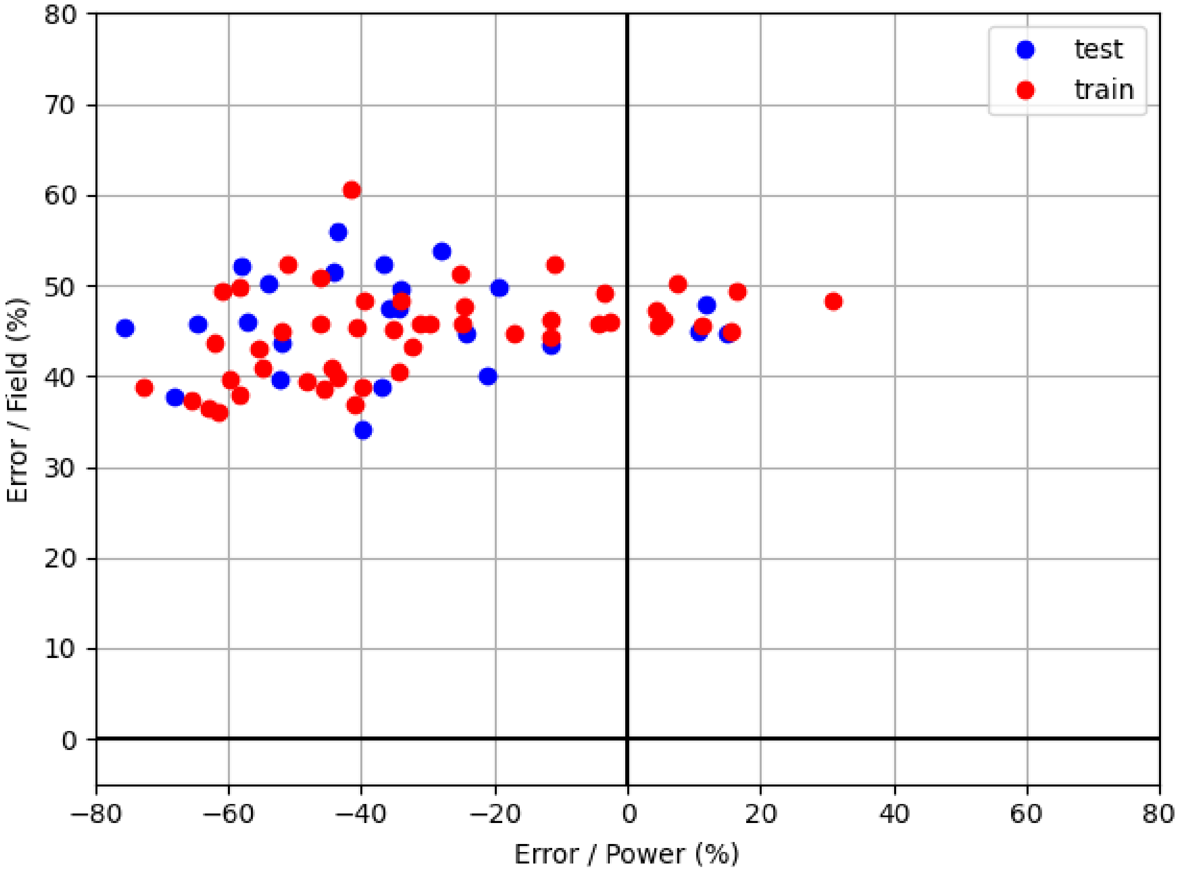
- $\frac{\mathrm{d}B(t)}{\mathrm{d}t} = r \cdot H(t) + \mu_0 \mu_r \frac{\mathrm{d}H(t)}{\mathrm{d}t}$
- Define a **circuit ODE.**
- **RL series** connection.
- The ODE **state** is the **mag. field**.
- The **excitation** is the **flux derivative**.
- The excitation is interpolated.

Linear R-L Circuit Results





Linear R-L Circuit Results

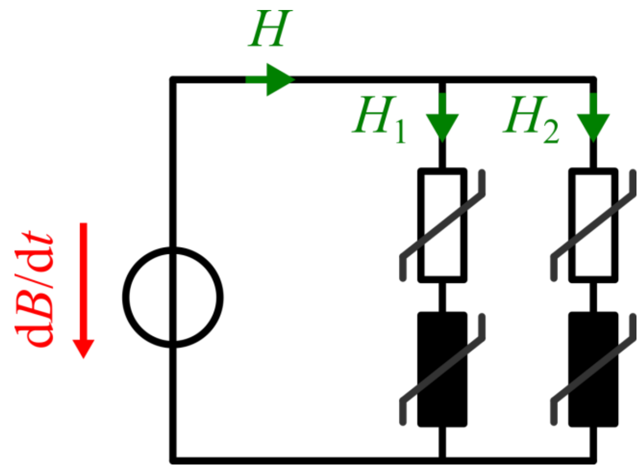


- **RMS error on the mag. field: 46%**
- **RMS error on the losses: 41%**

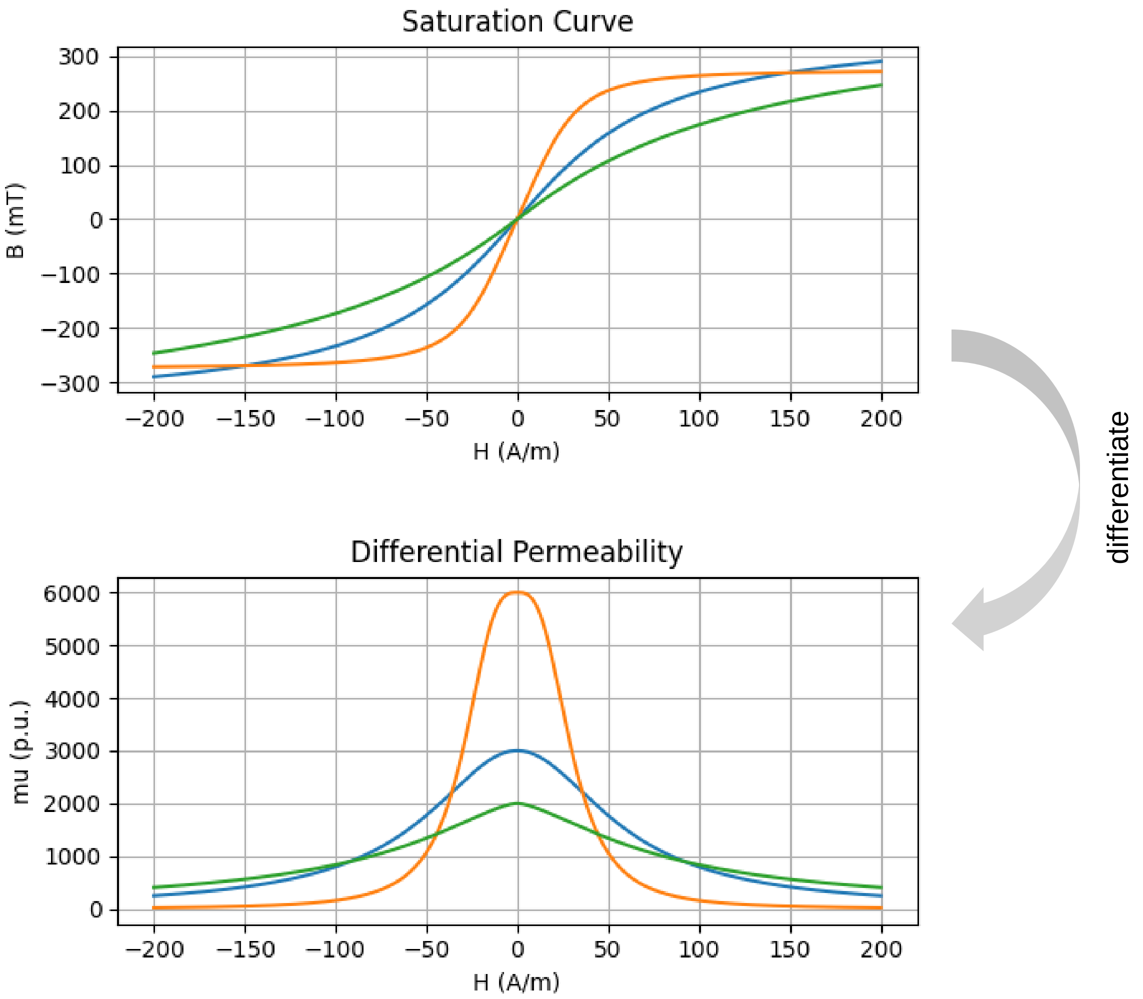
- **Only two parameters…**
- **Magnetics are non-linear…**

Nonlinear Circuit Definition

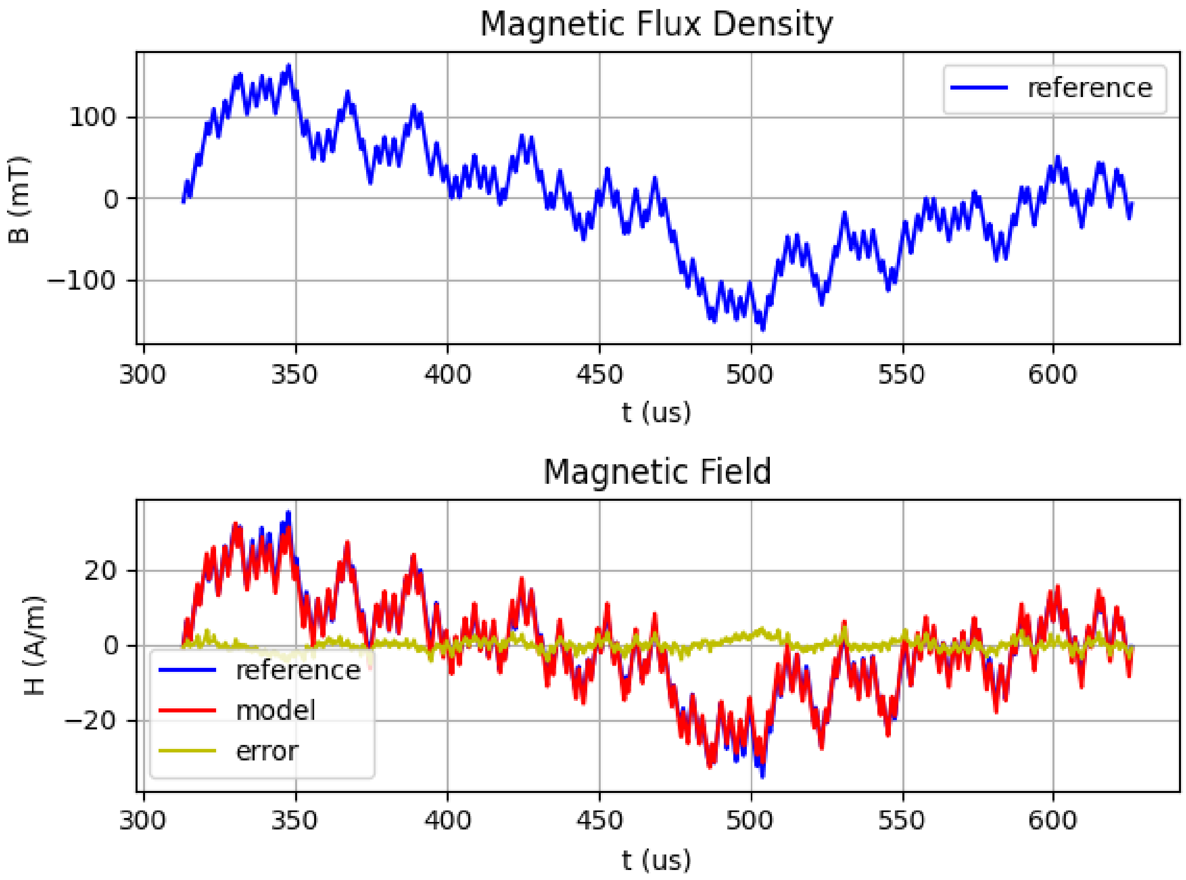
- **Transition to a nonlinear circuit**

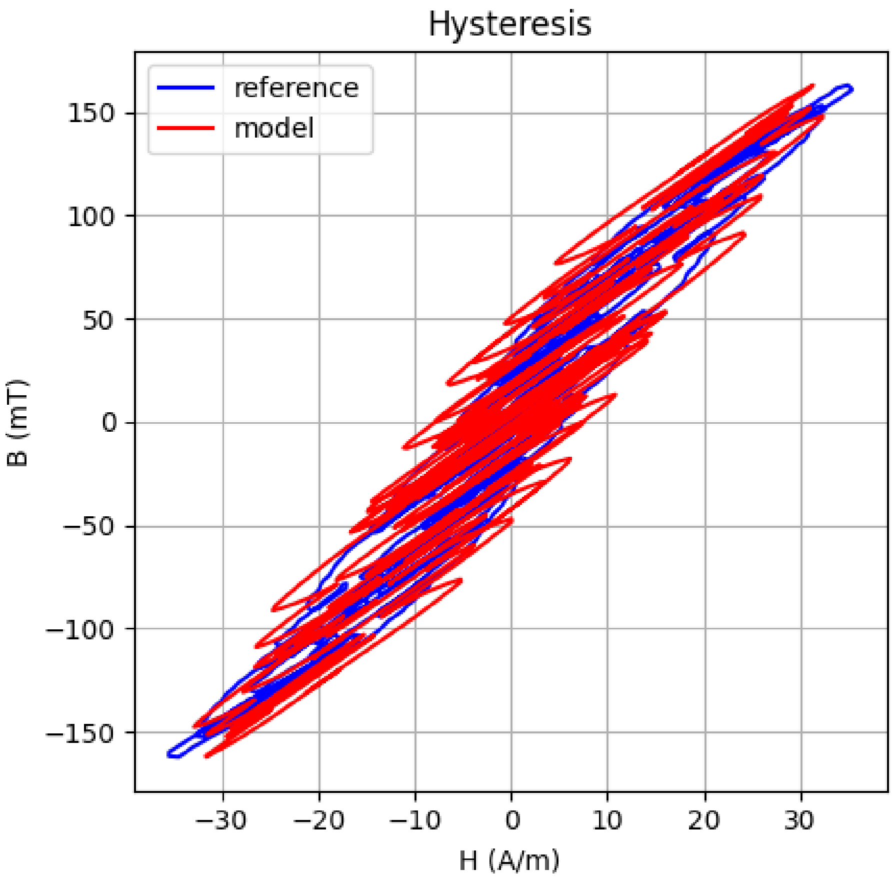


- **Two decoupled ODEs**
  - Two resistances ($R_i = R_i(H_i)$)
  - Two inductances ($L_i = L_i(H_i)$)
  - Non-linear equations
  - 10 free parameters

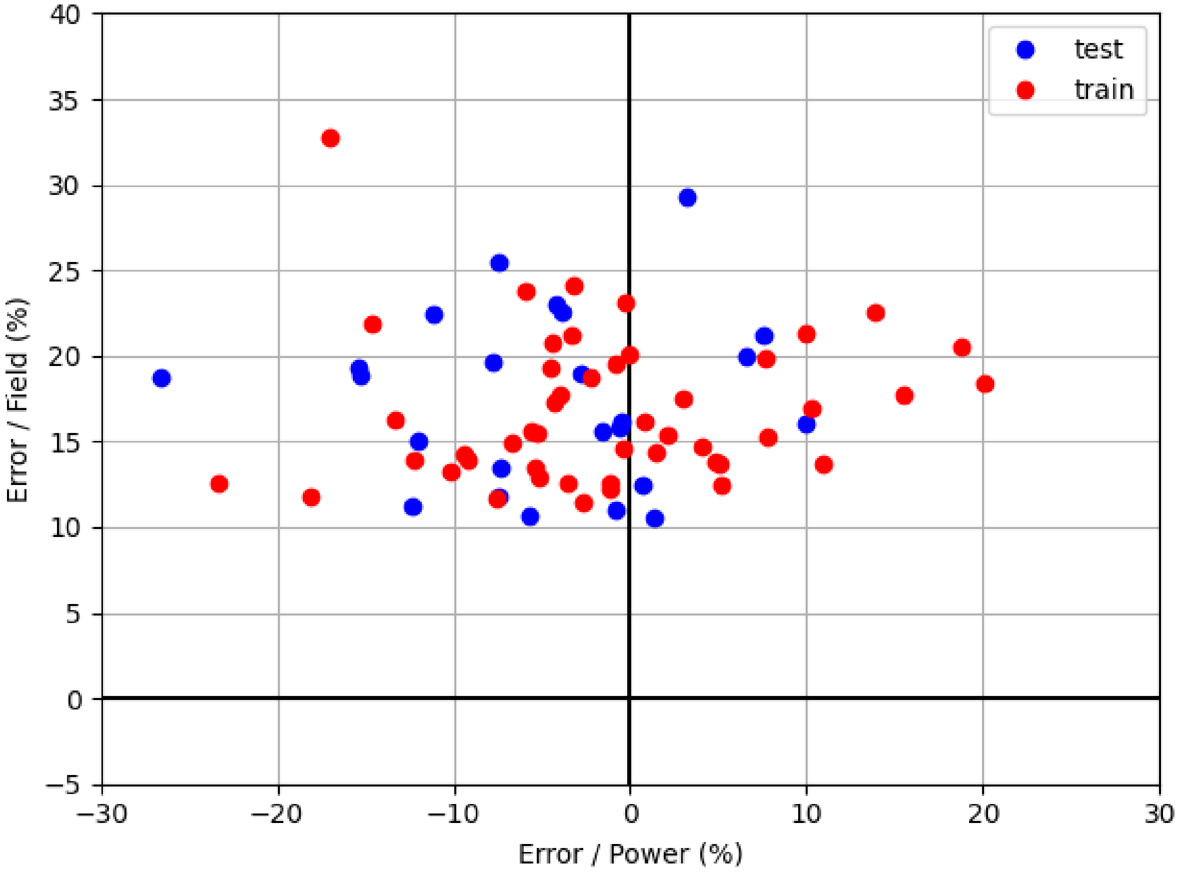


Nonlinear Circuit Results





Nonlinear Circuit Results



- RMS error on the mag. field: 18%
- RMS error on the losses: 9%
- No overfitting (test set)

- Number of parameters: 2 x 5
- Parameters bounds are enforced
- Initial values: latin hypercube
- Optimizer: AdaBelief grad. descent

Limitations and Opportunities

- **Equation-based analytical models have potential**
- **Physics-based** (or inspired) **models**
  - Minor loop splitting / decomposition
  - Steinmetz-based model (e.g. iGSE)
  - Using reluctance or permeance models
  - Landau–Lifshitz–Gilbert equation
  - Preisach Hysteresis model
  - Jiles–Atherton model
- **A last warning about such models:**

Physics-based material model $\not\Rightarrow$ **Fitted parameters have a physical meaning**

Some Interesting References

- **Loss Models**
  - K. Venkatachalam et. al., "Accurate Prediction of Ferrite Core Loss with Nonsinusoidal Waveforms using only Steinmetz Parameters", 2002, https://doi.org/10.1109/CIPE.2002.1196712
  - J. Mühlethaler et al., "Improved Core-Loss Calculation for Magnetic Components Employed in Power Electronic Systems", 2012, https://doi.org/10.1109/TPEL.2011.2162252
  - E. Stenglein et al., "Core Loss Model for Arbitrary Excitations With DC Bias Covering a Wide Frequency Range", 2021, https://doi.org/10.1109/TMAG.2021.3068188
  - T. Dimier et al, "Non-Linear Material Model of Ferrite to Calculate Core Losses With Full Frequency and Excitation Scaling," 2023, https://doi.org/10.1109/TMAG.2023.3277492

- **Hysteresis models**
  - D. C. Jiles et al., "Theory of Ferromagnetic Hysteresis". 1984, https://doi.org/10.1016/0304-8853(86)90066-1
  - H., K. Tanaka, Nakamura et al. "Calculation of Iron Loss in Soft Ferromagnetic Materials using Magnetic Circuit Model Taking Magnetic Hysteresis into Consideration“, 2015, https://doi.org/10.3379/msjmag.1501R001
  - M. Luo et al., "Modeling Frequency Independent Hysteresis Effects of Ferrite Core Materials Using Permeance–Capacitance Analogy for System-Level Circuit Simulations," 2018, https://doi.org/10.1109/TPEL.2018.2809704
  - G. Mörée et al., "Review of Play and Preisach Models for Hysteresis in Magnetic Materials", 2023, https://doi.org/10.3390/ma16062422

Part III:
Neural Differential Equation Models

Neural Differential Equations

- **Neural ODEs**

$$\frac{\mathrm{d}\vec{y}(t)}{\mathrm{d}t} = f\left(\vec{y}(t), \frac{\mathrm{d}B(t)}{\mathrm{d}t}\right)$$

$$H(t) = g\left(\vec{y}(t), \frac{\mathrm{d}B(t)}{\mathrm{d}t}\right)$$

$$\Rightarrow$$

$$\frac{\mathrm{d}\vec{y}(t)}{\mathrm{d}t} = \text{Neural Network}\left(\vec{y}(t), \frac{\mathrm{d}B(t)}{\mathrm{d}t}\right)$$

$$H(t) = \text{Neural Network}\left(\vec{y}(t), \frac{\mathrm{d}B(t)}{\mathrm{d}t}\right)$$

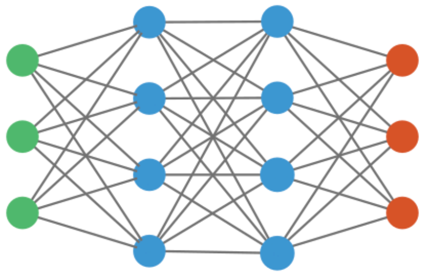
- **Neural networks are defining the ODE functions**
  - Coupled system of nonlinear differential equations
  - Very similar problem to circuit-based ODEs
  - Most of the code / libraries are the same

DARTMOUTH ENGINEERING

Neural Differential Equations

MagNet

- Standard **multilayer perceptron** (MLP) network
- Neural networks are **small and shallow**
- **Scaling** of the variables is **required**



State dynamics: single network:

$$\begin{bmatrix} \mathrm{d}H_1/\mathrm{d}t \\ \mathrm{d}H_2/\mathrm{d}t \\ \vdots \\ \mathrm{d}H_n/\mathrm{d}t \end{bmatrix} = \mathrm{MLP}\left(\begin{bmatrix} H_1(t) \\ H_2(t) \\ \vdots \\ H_n(t) \\ \mathrm{d}B(t)/\mathrm{d}t \end{bmatrix}\right)$$

State dynamics: two networks:

$$\begin{bmatrix} \mathrm{d}H_1/\mathrm{d}t \\ \mathrm{d}H_2/\mathrm{d}t \\ \vdots \\ \mathrm{d}H_n/\mathrm{d}t \end{bmatrix} = \mathrm{MLP}_1\left(\begin{bmatrix} |H_1(t)| \\ |H_2(t)| \\ \vdots \\ |H_n(t)| \end{bmatrix}\right)\begin{bmatrix} H_1(t) \\ H_2(t) \\ \vdots \\ H_n(t) \end{bmatrix} + \mathrm{MLP}_2\left(\begin{bmatrix} |H_1(t)| \\ |H_2(t)| \\ \vdots \\ |H_n(t)| \end{bmatrix}\right)\mathrm{d}B(t)/\mathrm{d}t$$

similar form

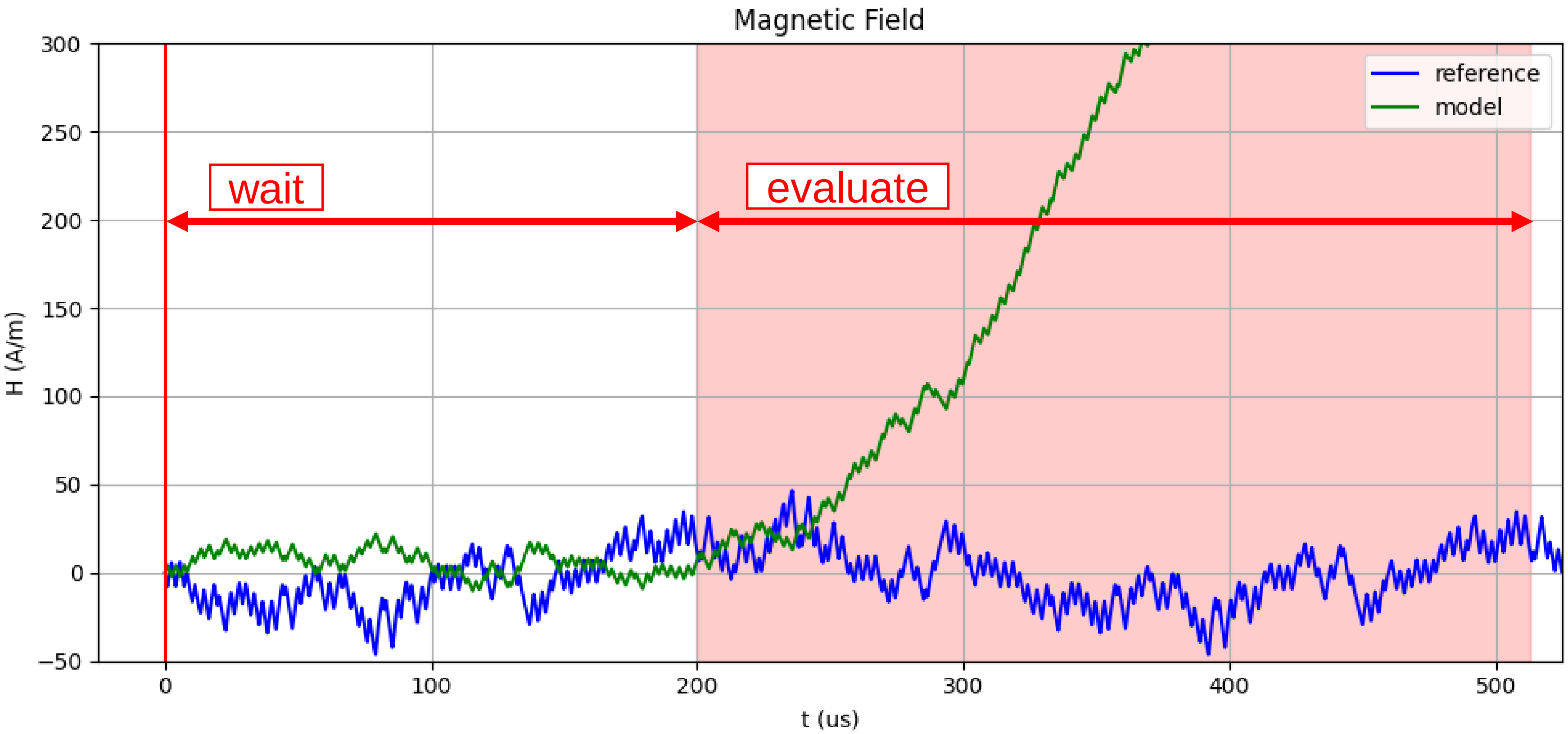
$$\begin{cases} \dfrac{\mathrm{d}B(t)}{\mathrm{d}t} = r(H)\cdot H(t) + \mu_0\mu_r(H)\dfrac{\mathrm{d}H(t)}{\mathrm{d}t} \\ \dfrac{\mathrm{d}H(t)}{\mathrm{d}t} = \dfrac{1}{\mu_0\mu_r(H)}\dfrac{\mathrm{d}B(t)}{\mathrm{d}t} - \dfrac{r(H)}{\mu_0\mu_r(H)}H(t) \end{cases}$$

Output functions: parallel connections:

$$H(t) = \sum_i H_i$$

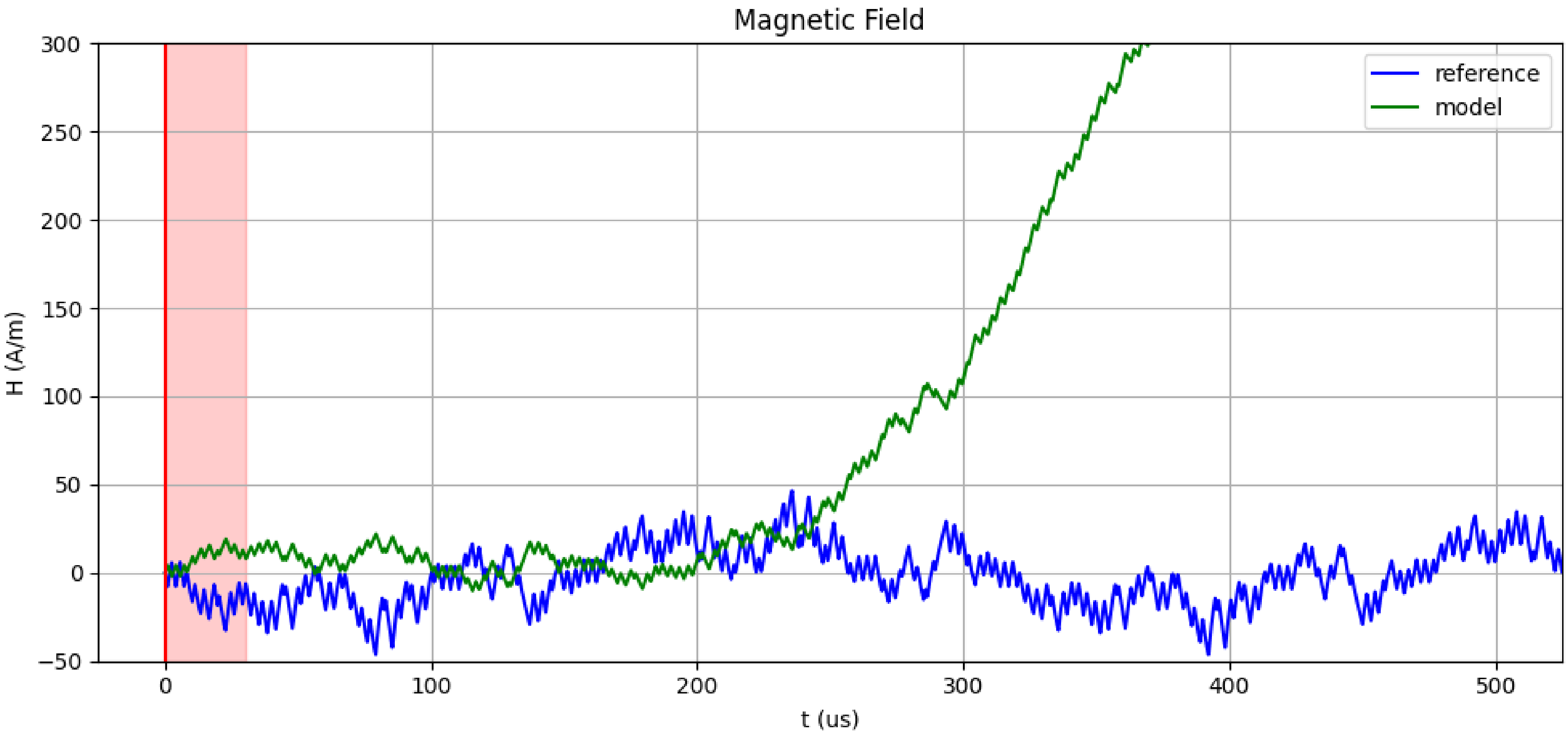
Problem: Initial Parameters

- The **initial** random networks **parameters** are a **bad model**
- This would imply a **very slow** and/or **unstable** training



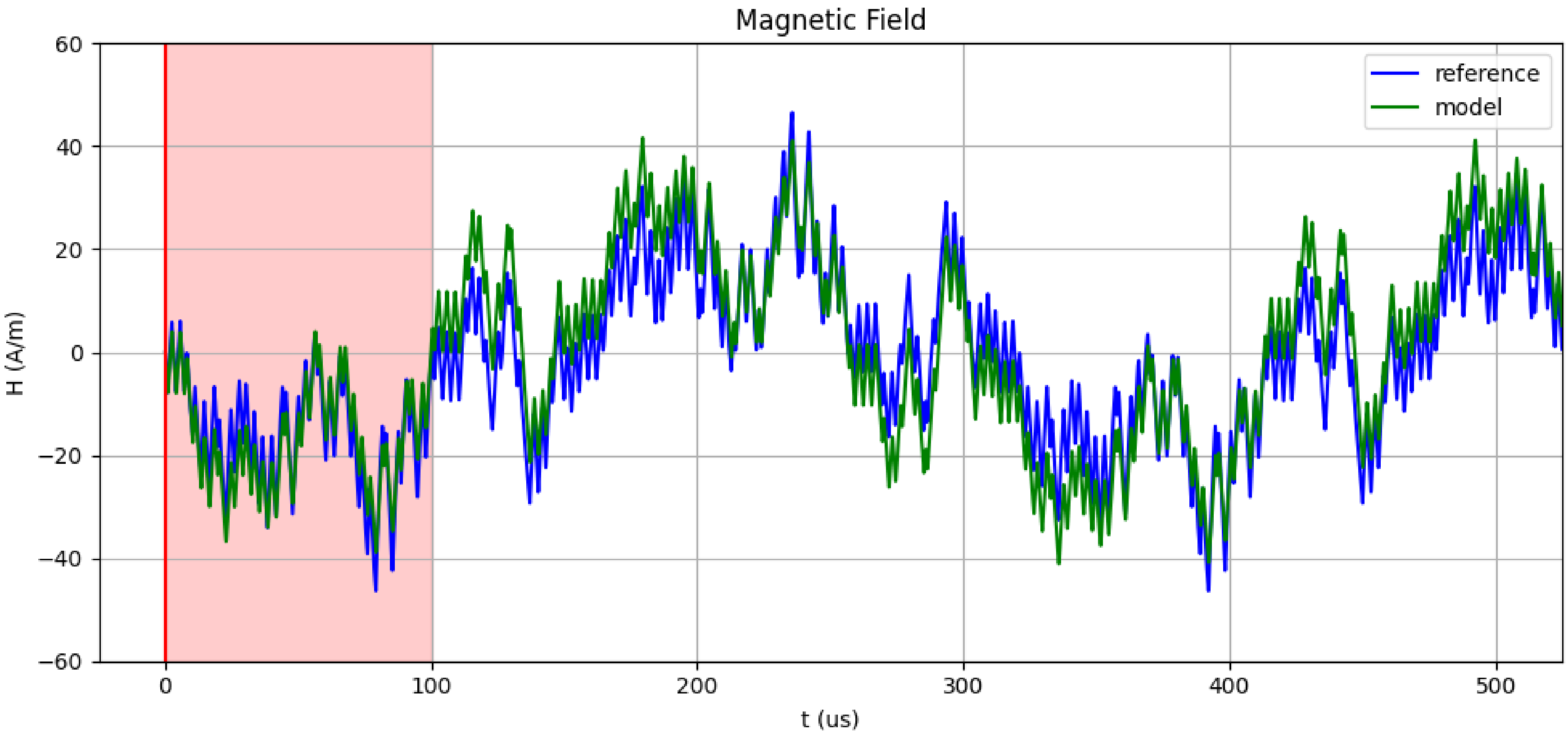
Problem: Initial Parameters

- **Solution**: start the training with a very **short window**
- **Drawback**: initial transient and hysteresis loop is not closed



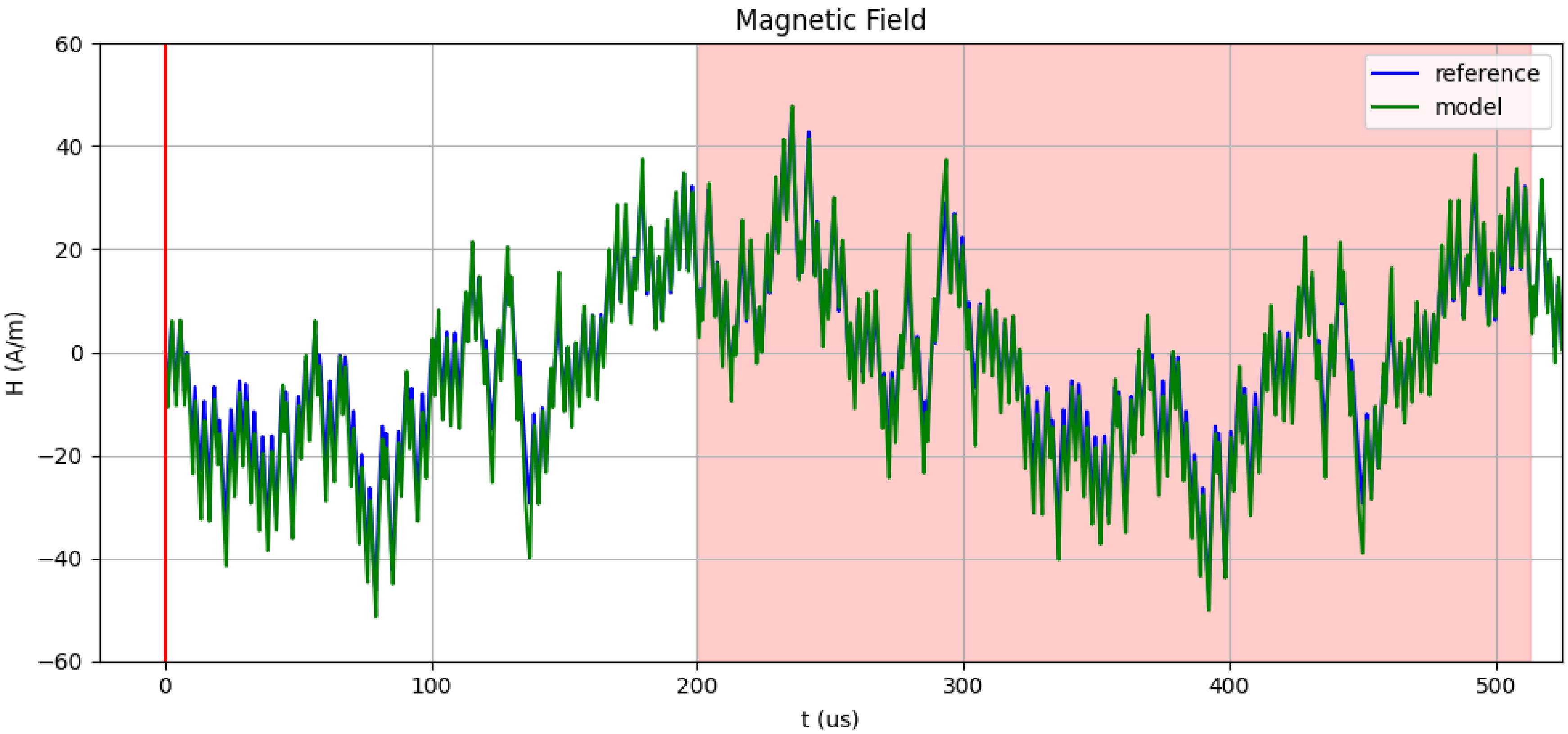
Problem: Initial Parameters

- **Solution**: start the training with a very **short window**
- **Goal**: slowly increase the size of the window

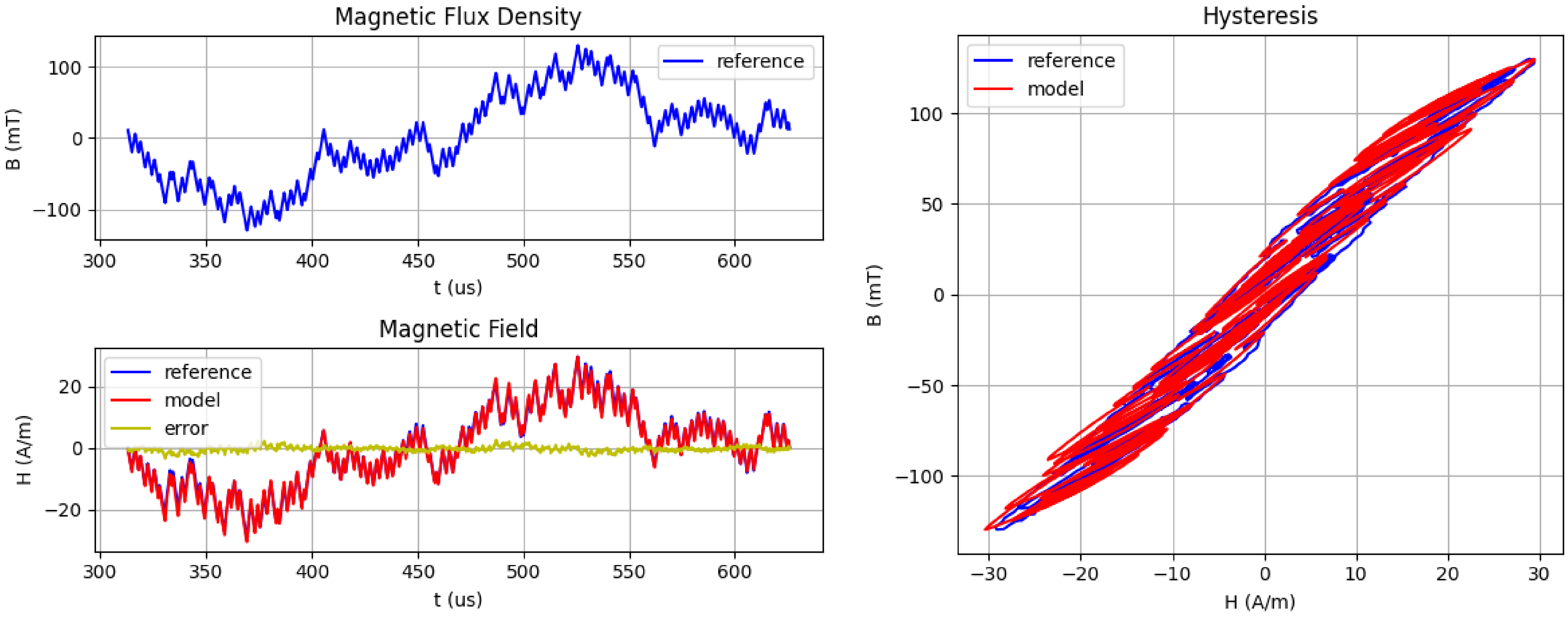


Problem: Initial Parameters

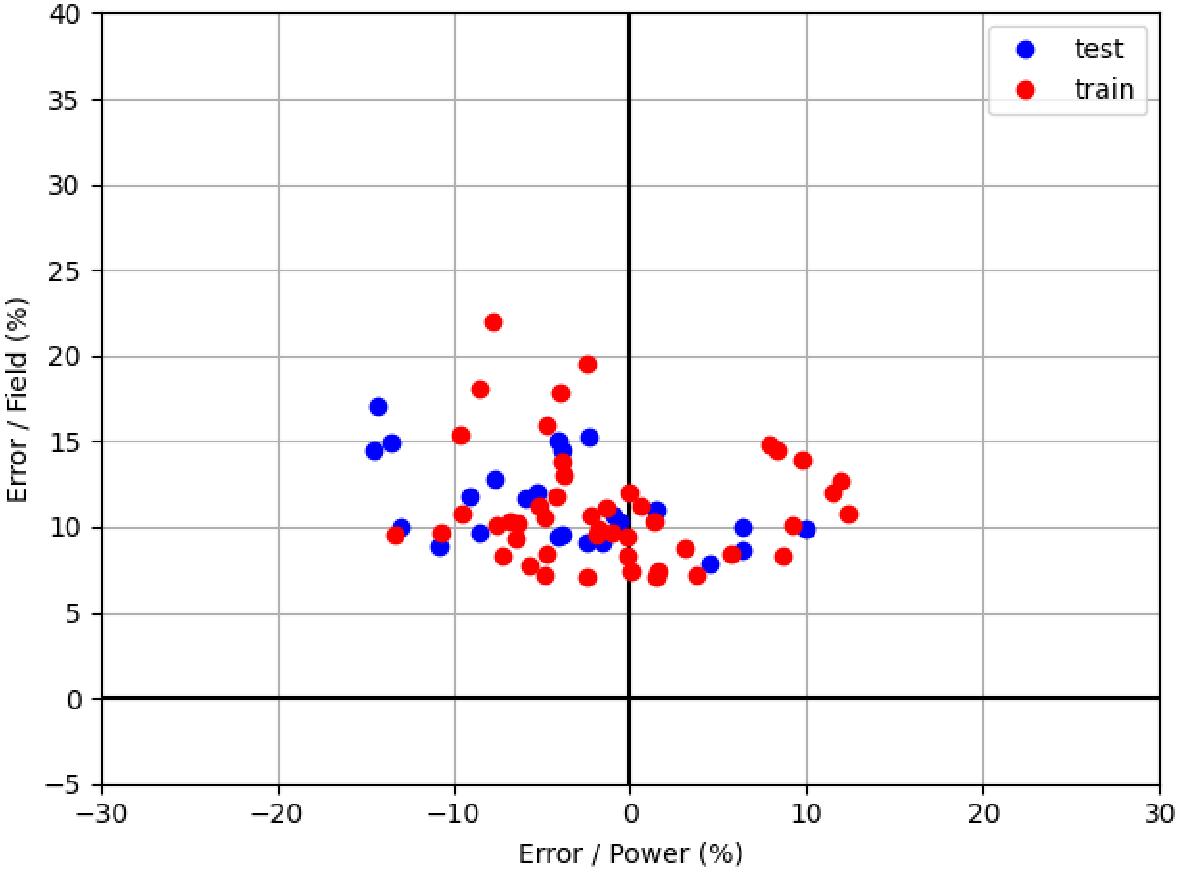
- **Solution**: start the training with a very **short window**
- **Goal**: training a full period with the model in steady-state



Neural ODE Results



Neural ODE Results



- **RMS error on the mag. field: 13%**
- **RMS error on the losses: 9%**
- **No overfitting (test set)**

- **Network: MLP with tanh activation**
- **Network size: 4 inputs and 3 outputs**
- **Hidden layers: 2 with 16 neurons**
- **Optimizer: AdaBelief grad. descent**

DARTMOUTH ENGINEERING

Limitations and Opportunities

MagNet

- **Neural ODEs**
  - No clear physical meaning
  - Opportunity for **compact models**
  - Compatible with **existing tooling**

- **Endless possibilities**
  - Neural SDE, neural CDE, latent ODE, etc.
  - System with discrete states (e.g. Preisach-based)
  - Possibility to implement a physics-informed network

- **Some references**

  - P. Kidger, “On Neural Differential Equations”, 2021, https://doi.org/10.48550/arXiv.2202.02435
  - P. Kidger, “Neural Differential Equations in Machine Learning”, 2021, https://kidger.site/links/KltsP45n2UP5/NDE_presentation.pdf

Part IV:
Conclusion and Outlooks

What is the ideal model?

- **Parameters and dataset**
  - Small number of parameters
  - Small dataset for the parametrization
  - Robust parametrization of the model
- **Model performance**
  - Accurate prediction for the field and the losses
  - Extrapolation outside the training/fitting range
  - Predicting waveshapes that are not in the training/fitting data
  - Transition to special cases (small signal, saturation, static curves, etc.)
  - Low computational cost (training and inference)
- **Other qualities**
  - Link with physical phenomena
  - Model debuggability and interpretability
  - Transfer of knowledge between materials
  - Possibility to extend the model (core geometry, DC bias, etc.)
  - Compatible with existing tooling (SPICE, FEM, etc.)

Advantage for analytical models?
Advantage for machine learning?
Unclear advantage?

Python Implementation

- **Used** to generate all the **presented results**
- Can be used as a **starting point** for the challenge

- **Implementation features**
  - **Training** and **inference** of **ODE models**
  - **Management** of the **dataset** with dataframes
  - Using **Python**, **JAX**, Diffrax, Pandas, etc.
  - Does not require a GPU
  - Open source (MPL 2.0)

- **Disclaimers**
  - The goal of this code is to demonstrate basic ODE models
  - The implementation is neither comprehensive nor optimized

- **https://github.com/otvam/magnetic_ode_models**

General Tips for the Challenge

- **The dataset is quite large**
  - Don’t hesitate to create a “simplified fake dataset”
  - Don’t hesitate to start with a subset of the dataset
  - Don’t hesitate to downsample / truncate the signals
- **Several factors are critical**
  - The model (structure, parameters, etc.)
  - The training/fitting process (optimizer, scaling, etc.)
  - The error metrics used as an objective
- **Programming tips**
  - Use clear interfaces between the training and inference
  - Use vectorized instructions (avoid loops and branches)
  - Nice data structures for storing the results
  - Make tools for displaying the results

Thank you! Questions?

Slides and Python source code:

https://github.com/otvam/magnetic_ode_models

Magnet Challenge GitHub:

https://github.com/minjiechen/magnetchallenge-2