



2023 MagNet Challenge Webinar: Equation-Based Baseline Models

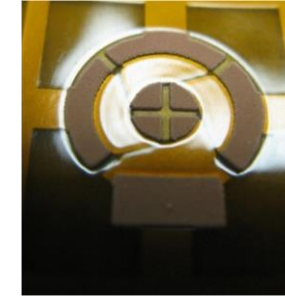
Thomas Guillod

Dartmouth College, USA

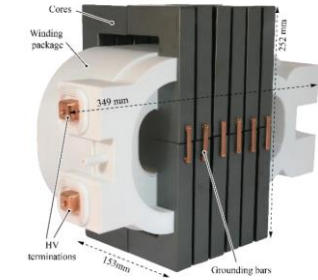
IEEE MagNet Challenge Webinar
May 12, 2023



- **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]

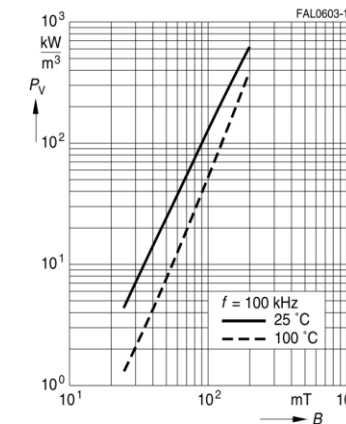


[ETHZ]

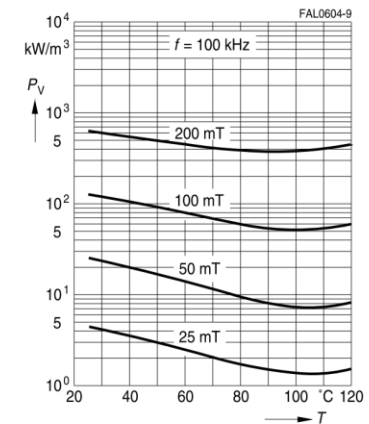
SIFERRIT materials

N87

Relative core losses
versus AC field flux density
(measured on R34 toroids)

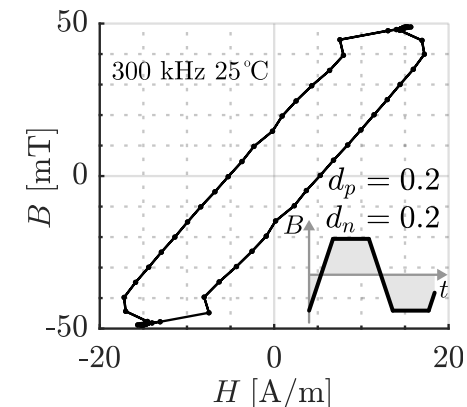
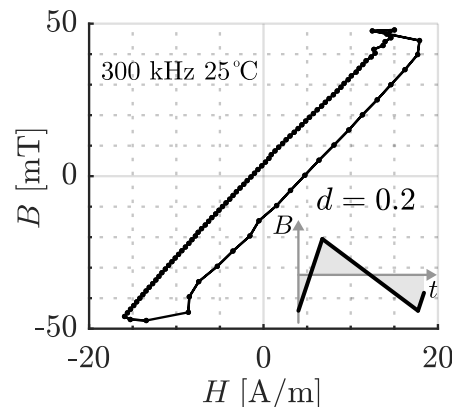
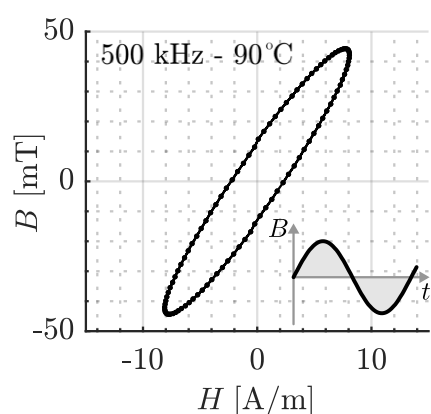
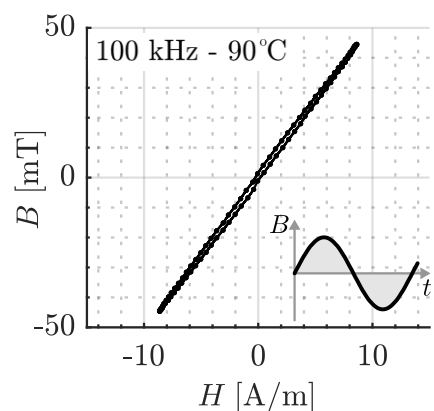
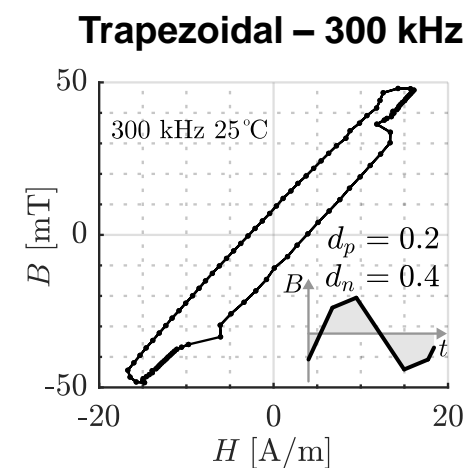
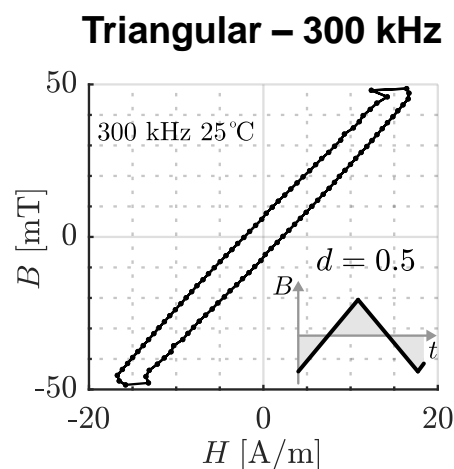
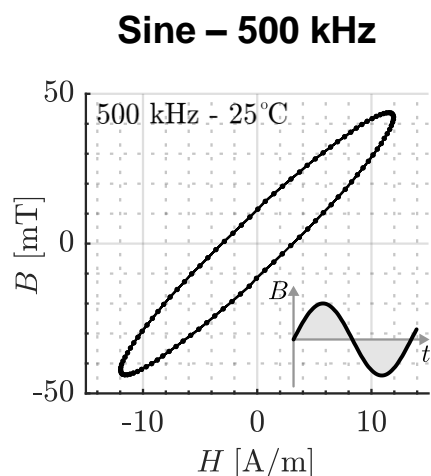
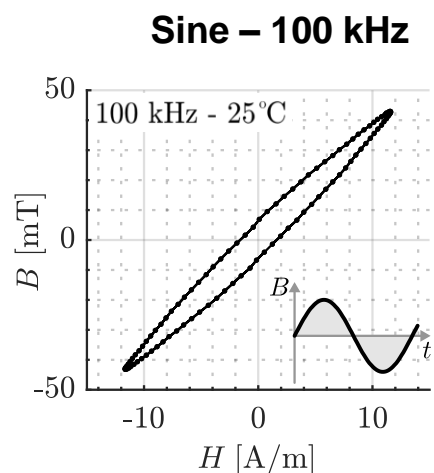


Relative core losses
versus temperature
(measured on R34 toroids)



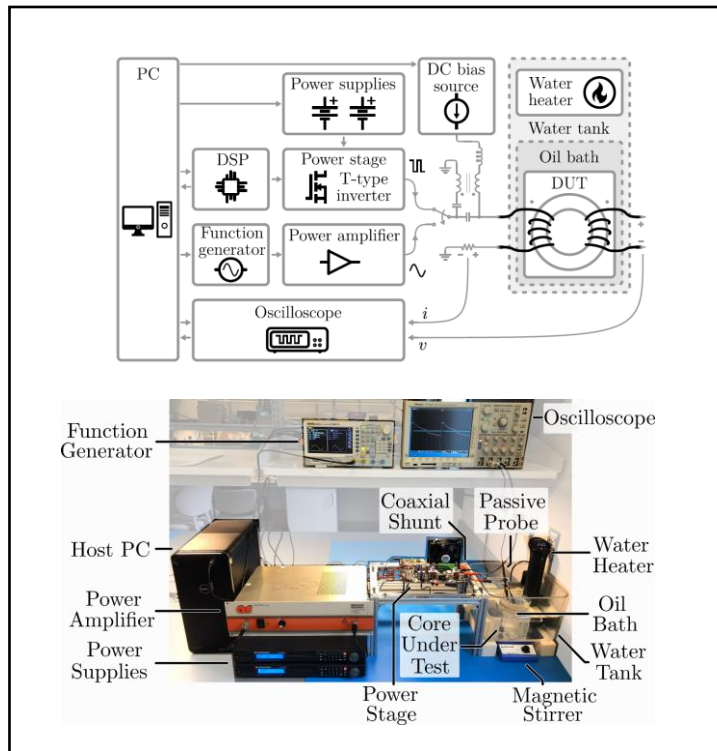
[TDK-EPCOS]

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

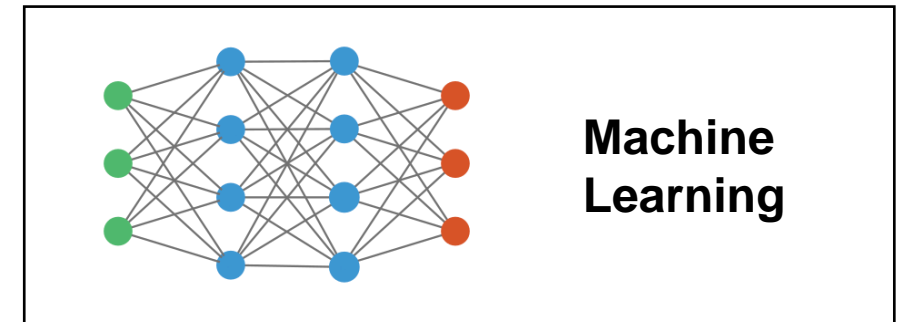


[example for R 22.1×13.7×7.9 N87 core, 7 turns]

- **MagNet Dataset**
 - 10 different materials
 - Over 500,000 measurements



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



$$P = \frac{1}{T} \int_0^T k \left| \frac{dB}{dt} \right|^\alpha (B_{pkpk})^{\beta-\alpha} dt$$

$$W_{hyst} = a_1 B_{pkpk} + a_2 B_{pkpk}^2 + a_3 B_{pkpk}^3$$

$$f_{eff} = f \left(1 + c \left(\frac{1}{B_{pkpk}} \int_0^T \left| \frac{d^2 B}{dt^2} \right| dt \right)^\gamma \right)$$

Equation Based

Part I: Overview of Equation-Based Models

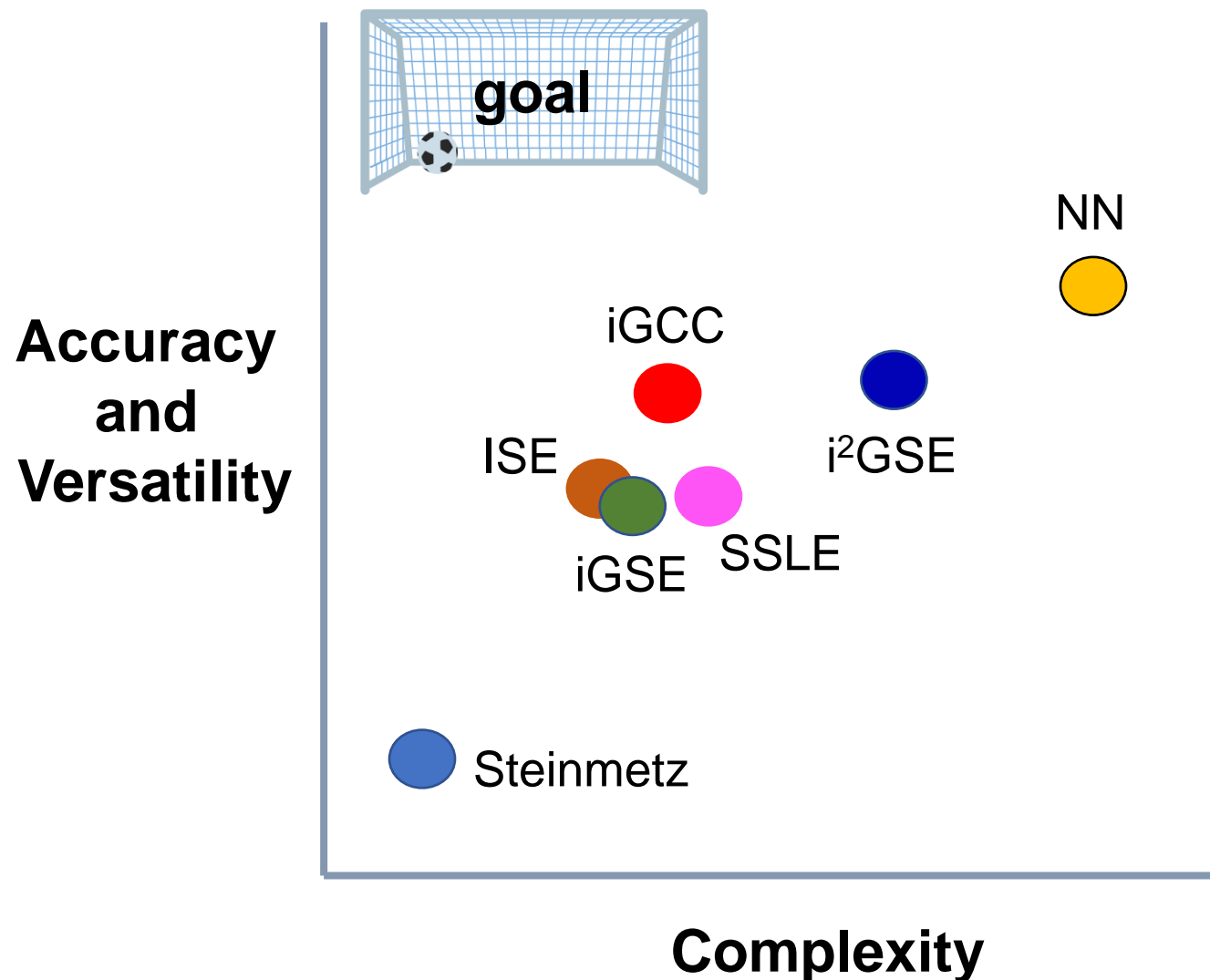
Part II: Equation-Based vs. Machine Learning

Part III: Implementation of the iGSE



Part I:

Overview of Equation-Based Models



- Trade-off
 - Accuracy and versatility
 - Complexity
- iGSE: only 3 parameters
- NN: up to 50'000 parameters
- Apple to orange comparison !

- **Equation-based models**

- Analytical formulation
- Fully empirical or physics-inspired
- Empirical parameters extracted from measurements

- **Steinmetz equation [Steinmetz, 1890]**

- Original form without frequency-dependency
- Modified in order to include frequency-dependency
- $P = k f^{\alpha} B_{pk}^{\beta}$
- Based on the Steinmetz parameters (k , α , and β)
- Parameters are typically fitted with sinusoidal waveforms
- No dependencies on the waveshape (sine, triangular, trapezoidal, etc.)

- **Improved generalized Steinmetz equation (iGSE) [Venkatachalam, 2002]**

- Loss computation for arbitrary waveforms
- Based on the Steinmetz parameters (k , α , and β)

- $$P = \frac{1}{T} \int_0^T k \left| \frac{dB}{dt} \right|^\alpha (B_{pkpk})^{\beta-\alpha} dt$$

- **Second derivative based models [Stenglein, 2021]**

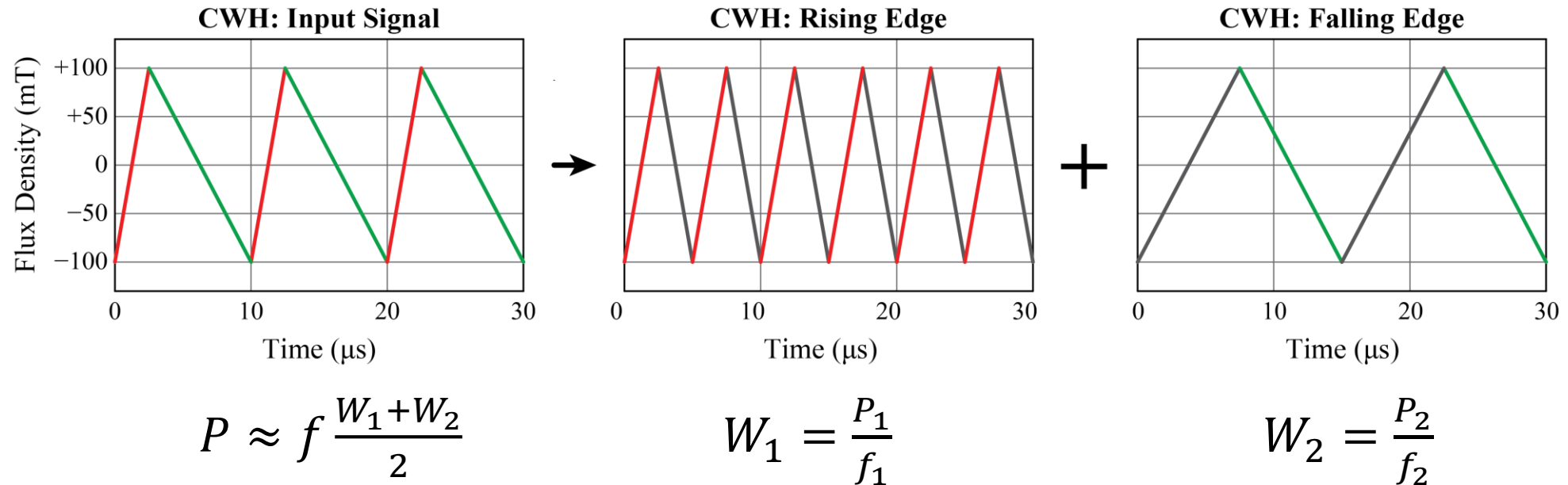
- **SSLE (3 parameters)**

$$P = k f (f_{eq})^{\alpha-1} B_{pkpk}^\beta$$
$$f_{eq} = \frac{1}{4\pi B_{pkpk}} \int_0^T \left| \frac{d^2 B}{dt^2} \right| dt$$

- **SEFLE (5 parameters)**

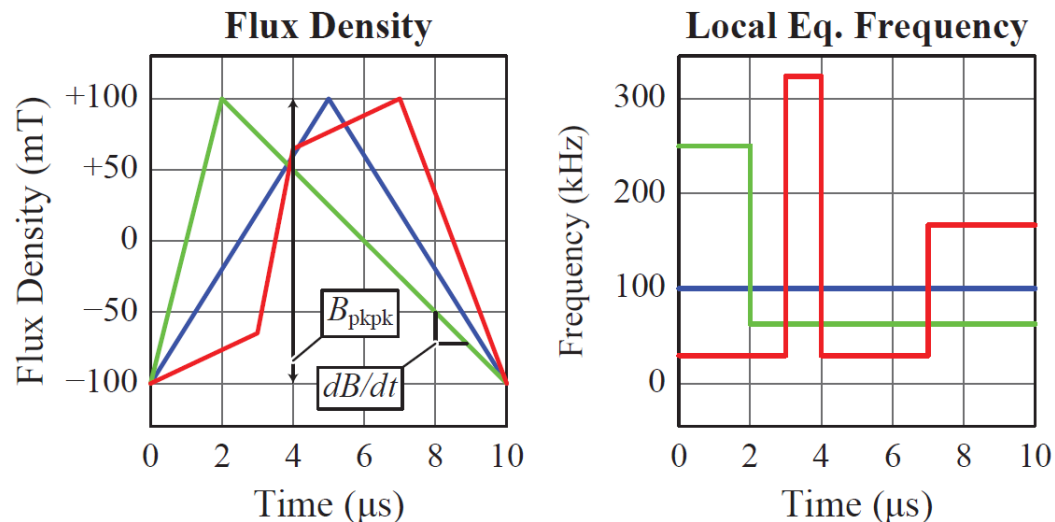
$$P = W_{hyst} f_{eff},$$
$$W_{hyst} = a_1 B_{pkpk} + a_2 B_{pkpk}^2 + a_3 B_{pkpk}^3$$
$$f_{eff} = f \left(1 + c \left(\frac{1}{B_{pkpk}} \int_0^T \left| \frac{d^2 B}{dt^2} \right| dt \right)^\gamma \right)$$

- **Composite waveform hypothesis (CWH) [Sullivan, 2010]**
 - A waveform can be decomposed in segments
 - The losses associated with the segments can be computed separately
 - Many analytical method relies (explicitly or implicitly) on the CWH
- **Triangular waveforms**



- How to decompose an arbitrary waveform?

- Local equivalent frequency: $\tilde{f}(t) = \frac{1}{2} \frac{\left| \frac{dB}{dt} \right|}{B_{pkpk}}$
- Property (after loop splitting): $\int_0^T \tilde{f}(t) dt = 1$



- **Improved Generalized Composite Calculation (iGCC)**

- Local equivalent frequency :
$$\tilde{f}(t) = \frac{1}{2} \frac{\left| \frac{dB}{dt} \right|}{B_{pkpk}}$$

- Losses of 50% triangular waveforms:
$$P_{\text{sym}}(f, B_{pkpk})$$

- iGCC integral form:
$$P = \frac{1}{T} \int_0^T P_{\text{sym}}(\tilde{f}(t), B_{pkpk}) dt$$

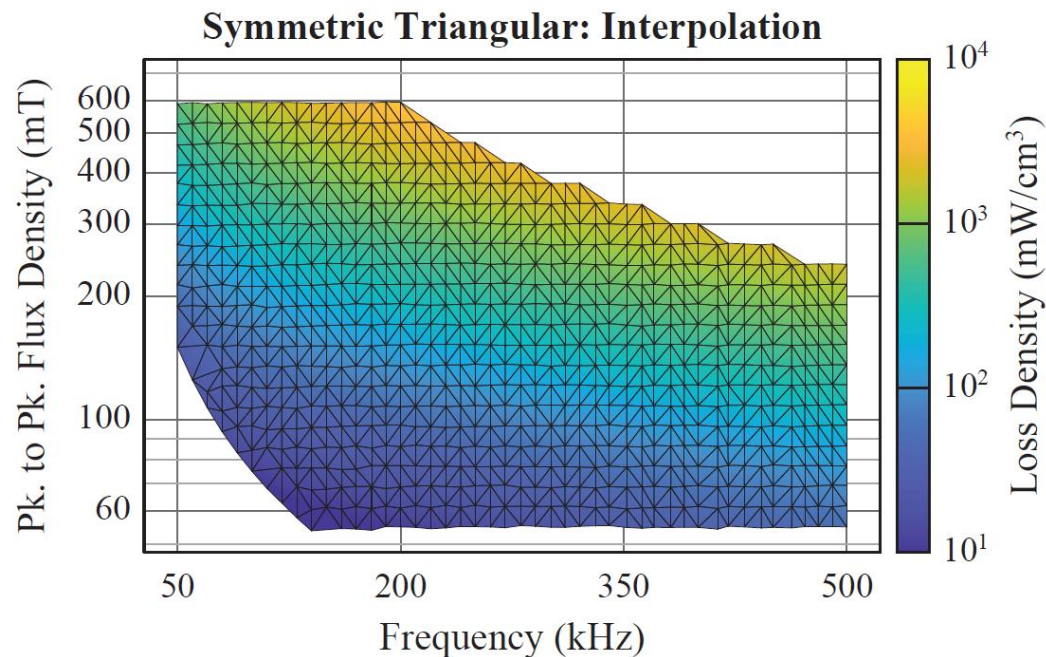
- iGCC piecewise linear form:
$$P = f \sum_{i=1}^n P_{\text{sym}} \left(\frac{1}{2} \frac{\left| \frac{\Delta B_i}{\Delta t_i} \right|}{B_{pkpk}}, B_{pkpk} \right) \Delta t_i$$

- **How to obtain $P_{\text{sym}}(f, B_{pkpk})$?**

- iGCC_{int.} : loss map with interpolation
- iGCC_{fit.} : curve fitting of Steinmetz parameters

- **Loss map with interpolation**

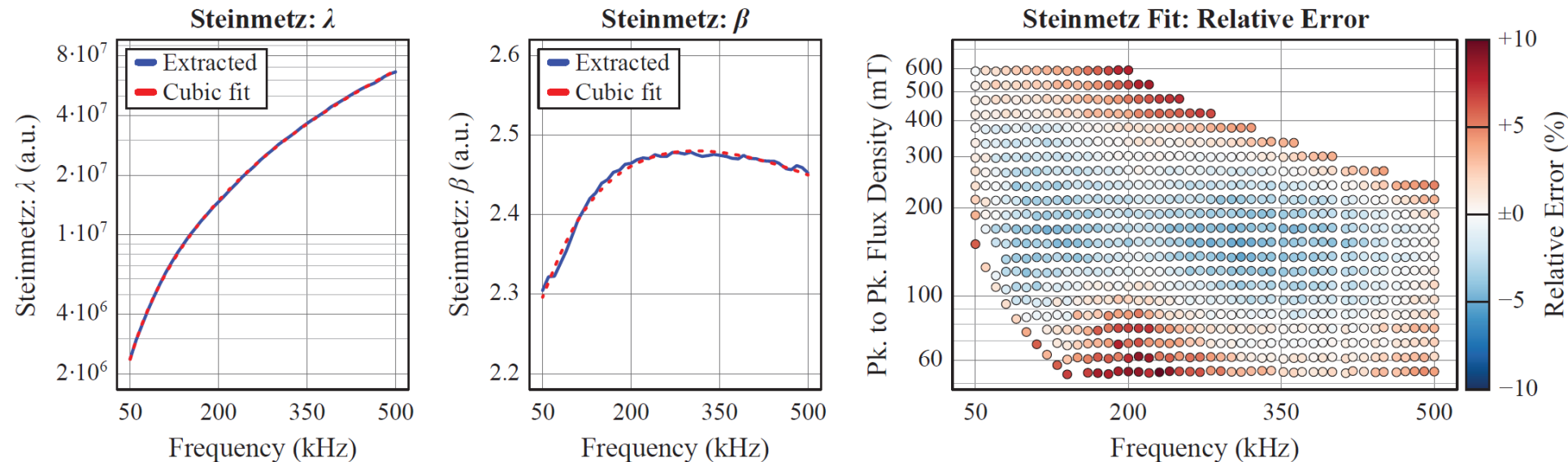
- 50% triangular waveforms
- Different frequencies and flux densities
- Advantage: simple and accurate
- Drawback: requires a large dataset



- Linear interpolation (in log scale)
- Meas. points are not on a regular grid
- Delaunay triangulation of the points

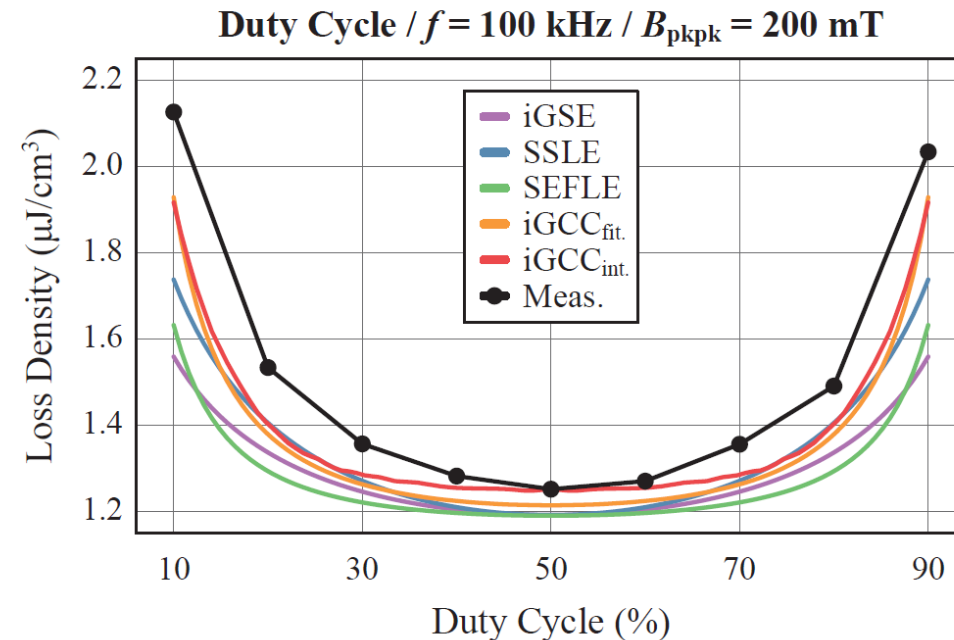
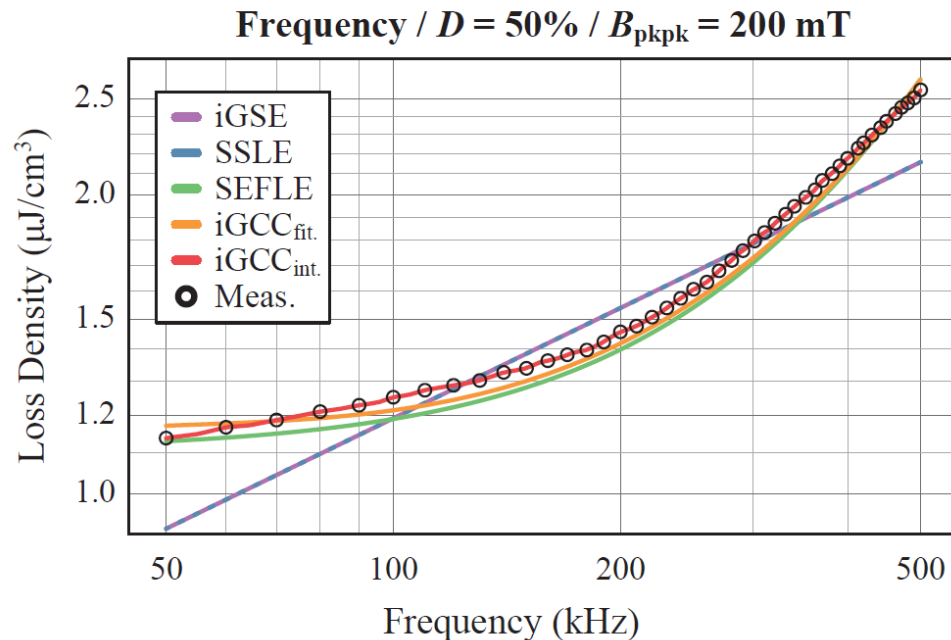
• Frequency-dependent Steinmetz parameters

- Expression: $P_{\text{sym}}(f, B_{\text{pkpk}}) = \lambda(f) B_{\text{pkpk}}^{\beta(f)}$
- Fitting of λ and β for different frequencies
- Cubic curve fitting of the obtained values
- Advantage: no extraction of α is required



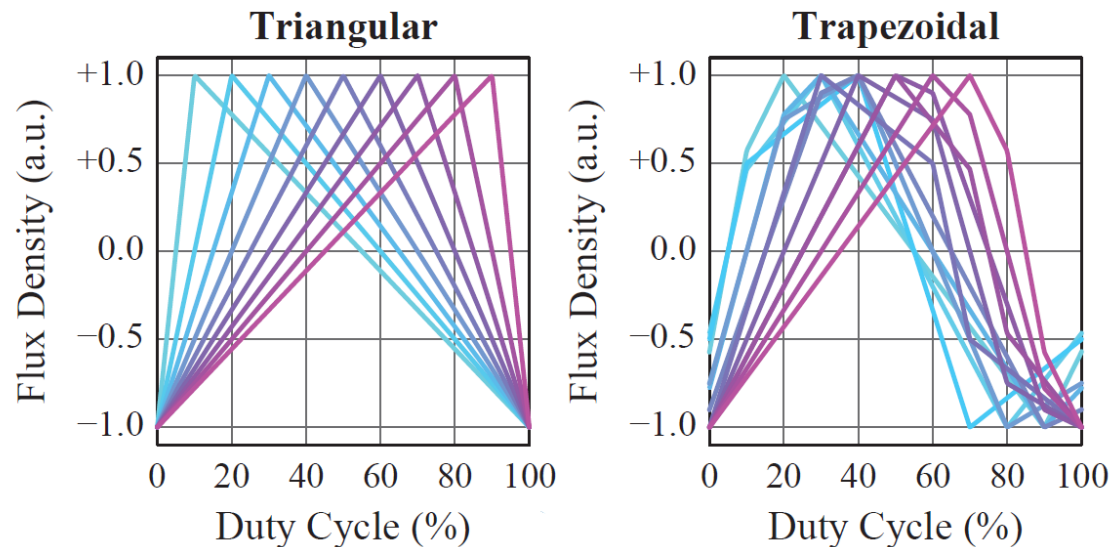
- **Triangular signals**

- N87 material at 25°C
- iGCC is better at extreme duty cycles
- iGCC is better in a wide frequency range



- **Large test dataset**

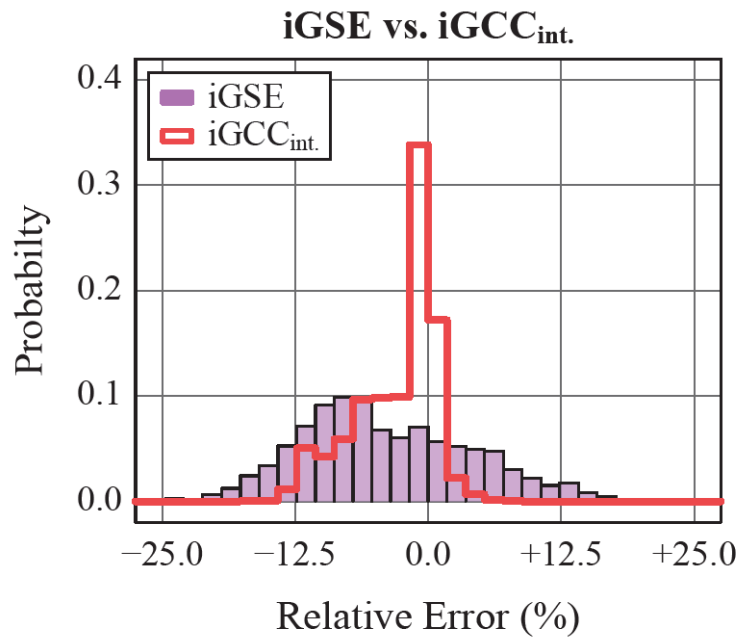
- Extracted from the **MagNet** dataset
- **N87 material**, different frequencies, amplitudes, waveshapes, temperatures
- 4720 triangular and trapezoidal signals



- N87 Material
- $f \in [50, 500]$ kHz
- $B_{pkpk} \in [50, 600]$ mT
- $T \in [25, 90]$ °C
- $P > 5$ mW/cm³

• Measurements at 25°C

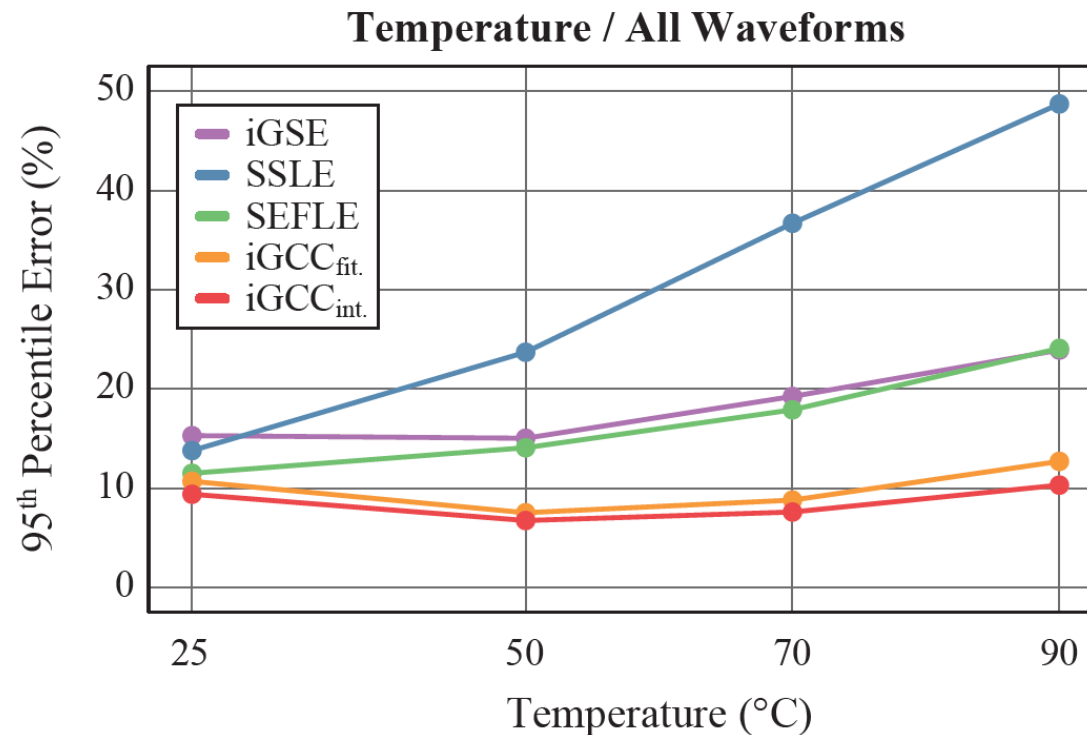
- iGCC clearly outperform the iGSE, SSLE, and SEFLE
- 95th percentile error below 12%



Model	Avg.	RMS	95 th pct.	Max.
iGSE	7.5 %	9.0 %	16.2 %	27.7 %
SSLE	6.0 %	7.4 %	14.3 %	20.8 %
SEFLE	5.4 %	6.6 %	12.4 %	26.1 %
iGCC_{fit.}	4.7 %	5.9 %	11.9 %	16.9 %
iGCC_{int.}	3.3 %	4.8 %	11.1 %	16.9 %

- **Impact of the core temperature**

- iGCC performs well across the complete range
- 95th percentile error below 13%



- **Limitations of the iGSE, iGCC, SSLE, and SEFLE**

- **Relaxation losses** are not considered
- **Temperature dependencies** are not part of the model
- **DC biases** are not considered (not relevant for the MagNet Challenge)
- **Core shape** are not considered (not relevant for the MagNet Challenge)

- **Equation-based model references**

- K. Venkatachalam et. al., “Accurate Prediction of Ferrite Core Loss with Nonsinusoidal Waveforms using only Steinmetz Parameters,” 2002
- J. Mühlethaler et al., “Improved Core-Loss Calculation for Magnetic Components Employed in Power Electronic Systems, 2012
- E. Stenglein et al., “Core Loss Model for Arbitrary Excitations With DC Bias Covering a Wide Frequency Range,” 2021
- T. Guillod et al., “Calculation of Ferrite Core Losses with Arbitrary Waveforms using the Composite Waveform Hypothesis”, 2023



Part II:

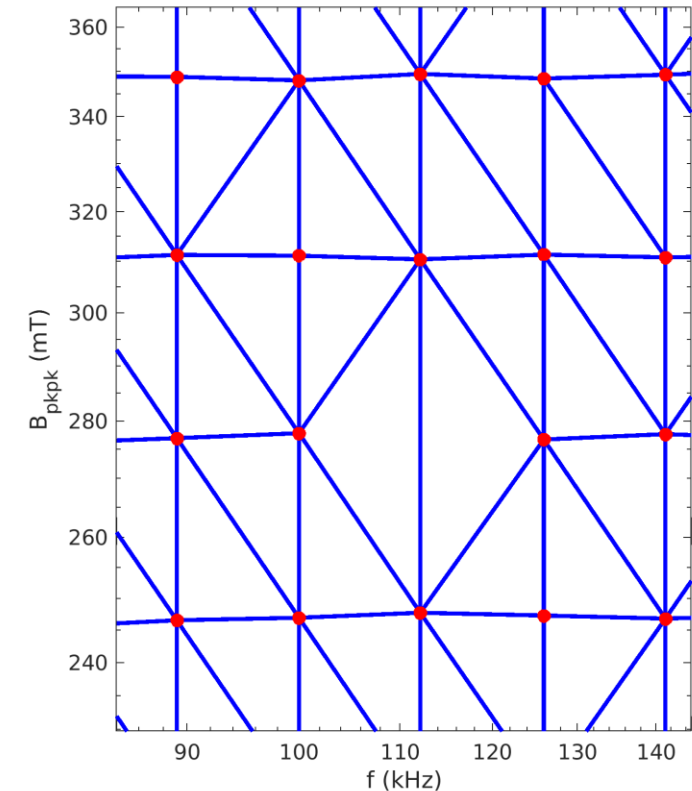
Equation-Based vs. Machine Learning

- **Number of parameters**
 - Equation-based: 3 – 30 parameters
 - Machine learning: 500 – 50000 parameters
- **Required dataset**
 - Equation-based: small datasets (3 – 500 points)
 - Machine learning: large datasets (over 1000 points)
- **Link with physical phenomena**
 - Equation-based: relatively easy to achieve
 - Machine learning: possible but much more difficult
- **Model debuggability and interpretability**
 - Equation-based: not easy but achievable
 - Machine learning: extremely difficult

- **Predicting waveshapes that are not in the training/fitting data**
 - Equation-based: standard for state-of-the-art models (iGSE, iGCC, etc.)
 - Machine learning: possible but more difficult and unpredictable
- **Extrapolation outside the training/fitting range**
 - Equation-based: possible but risky
 - Machine learning: extremely risky
- **Detection of poor dataset quality**
 - Equation-based: possible but not guaranteed
 - Machine learning: difficult (garbage in, garbage out)

- **Model versatility (operating conditions, materials, etc.)**
 - Equation-based: limited to the used equations
 - Machine learning: models can “self-adapt” to various conditions
- **Possibly to extend the model (DC bias, temperature, etc.)**
 - Equation-based: require an update of the equations (can be difficult)
 - Machine learning: easy if the model paradigm allows it
- **Achieved accuracy**
 - Equation-based: good but difficult to achieve over wide ranges
 - Machine learning: extremely good (same range as the dataset accuracy)
- **Dataset pre-processing**
 - Equation-based: required, dataset should be pre-processed and sorted
 - Machine learning: possible to directly use the raw dataset

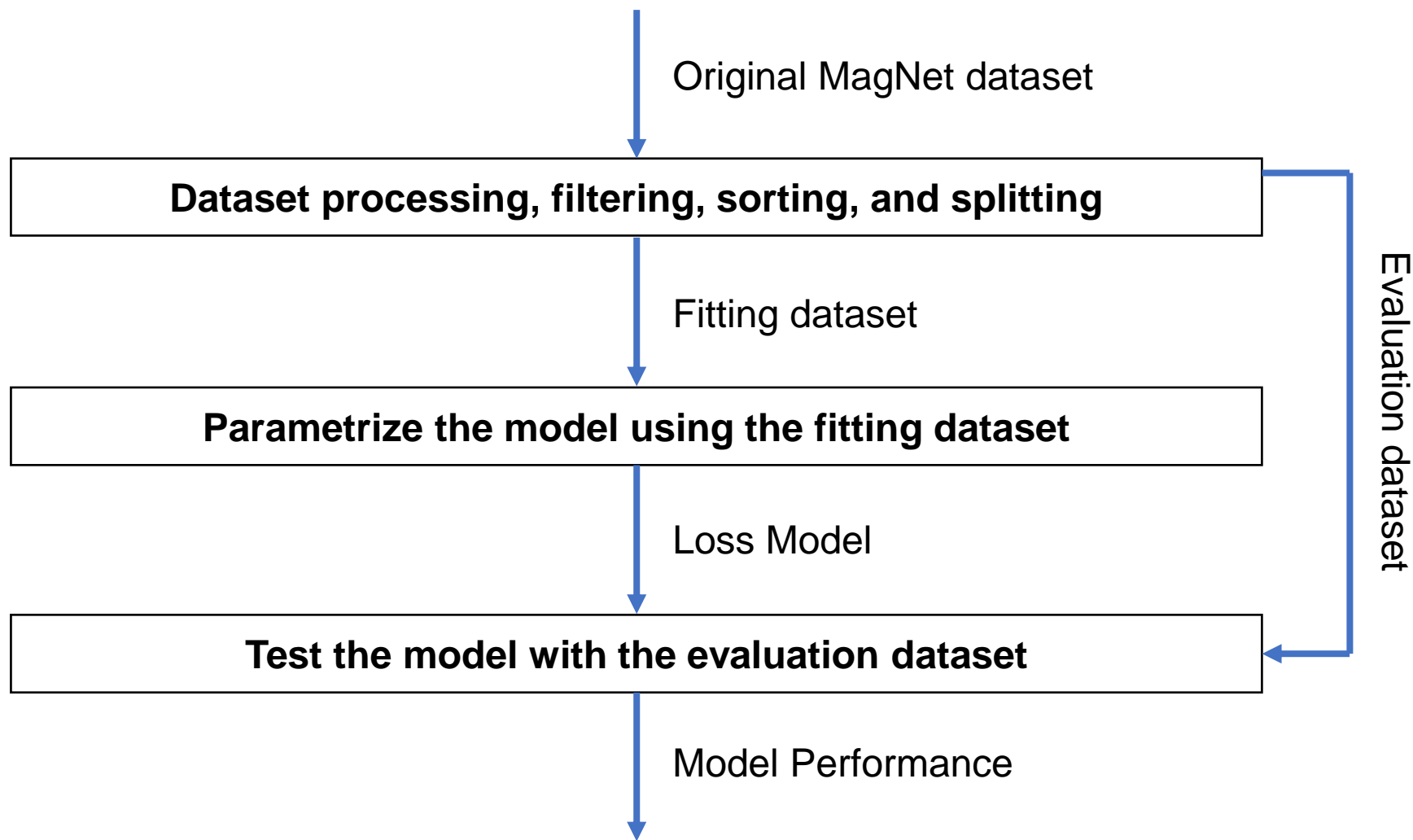
- For equation-based models, several pitfalls should be avoided
- Dataset organization
 - Pre-processing, filtering, and sorting
 - The points are **not** on a **regular grid**
 - Some **points** might be **missing**
- Dataset range
 - The **dataset range** might not be what you want/need
 - All the points are between **50 kHz** and **500 kHz**
 - N27 material: optimal between 10 kHz and 100 kHz
 - 3F4 material: optimal between 750 kHz and 2000 kHz
 - This can be critical for physics-based models





Part III:

Implementation of the iGSE



- **MATLAB implementation**

- Code **snippets** in the **slides** for the iGSE
- **More complete code** for the iGSE and iGCC on **GitHub**
- https://github.com/otvam/magnet_webinar_eqn_models

- **Disclaimers**

- The goal of this code is to **highlight** the **typical workflow** of equation-based models
- The implementation is **not** meant to be **comprehensive and/or accurate**

- **Assumptions**

- Single material measured at ambient temperature
- Only triangular signals are considered
- Simple model parametrization

```
function run_igse()
% Parametrize and evaluate the iGSE loss model.

% load the fitting and evaluation sets
map_fit = load('data/N87_25C_fit.mat');
map_eval = load('data/N87_25C_eval.mat');

% parametrize the loss model with the loss map
fct_model = get_model(map_fit);

% evaluate a loss model and compare the results
map_eval = get_eval(map_eval, fct_model);

% save the results
save('data/N87_25C_res.mat', '-struct', 'map_eval');

end
```

Step 1: load the datasets

Step 2: fit the model

Step 3: eval. the model

Step 4: save the data

```
function run_igse()
% Parametrize and evaluate the iGSE loss model.

% load the fitting and evaluation sets
map_fit = load('data/N87_25C_fit.mat');
map_eval = load('data/N87_25C_eval.mat');

% parametrize the loss model with the loss map
fct_model = get_model(map_fit);

% evaluate a loss model and compare the results
map_eval = get_eval(map_eval, fct_model);

% save the results
save('data/N87_25C_res.mat', '-struct', 'map_eval');

end
```

Step 1: load the datasets

Step 1: Load the Datasets

- **Selected material: N87 at 25°C**
- **Fitting set (346 points)**
 - Should only contain **symmetric triangular signals**
 - f_vec signal frequencies
 - B_pkpk_vec peak-to-peak flux densities
 - p_meas_vec measured loss densities (used for fitting)

Field	Value
B_pkpk_vec	1x346 double
f_vec	1x346 double
p_meas_vec	1x346 double

- **Evaluation set (2446 points)**
 - Could contain any type of **piecewise linear waveforms**
 - f_vec signal frequencies
 - d_mat duty cycles defining the piecewise linear waveforms
 - B_mat flux densities defining the piecewise linear waveforms
 - p_meas_vec measured loss densities (used for comparison)

Field	Value
B_mat	3x2446 double
d_mat	3x2446 double
f_vec	1x2446 double
p_meas_vec	1x2446 double

```
function run_igse()  
% Parametrize and evaluate the iGSE loss model.  
  
% load the fitting and evaluation sets  
map_fit = load('data/N87_25C_fit.mat');  
map_eval = load('data/N87_25C_eval.mat');  
  
% parametrize the loss model with the loss map  
fct_model = get_model(map_fit);  
  
% evaluate a loss model and compare the results  
map_eval = get_eval(map_eval, fct_model);  
  
% save the results  
save('data/N87_25C_res.mat', '-struct', 'map_eval');  
  
end
```

Step 2: fit the model

Step 2: Fit the Model

```
function fct_model = get_model(map_fit)
% Parametrize a loss model (iGSE or iGCC) with a measured loss map.

f_vec = map_fit.f_vec;
B_pkpk_vec = map_fit.B_pkpk_vec;
p_meas_vec = map_fit.p_meas_vec;

% extract the range (frequency and flux density) of the loss map
fct_range = get_range(f_vec, B_pkpk_vec);

% fit the parametrized fitting function with the provided data
param_fit = get_igse_fit(f_vec, B_pkpk_vec, p_meas_vec);

% get a function handle describing the fitted loss model
fct_model = @(f_vec, d_mat, B_mat) get_igse_model(f_vec, d_mat, B_mat, fct_range, param_fit);

end
```

Get the dataset

Find the fitting range

Find the optimal fit

Get the model

Step 2: Fit the Model



```
function fct_model = get_model(map_fit)
% Parametrize a loss model (iGSE or iGCC) with a measured loss map.

f_vec = map_fit.f_vec;
B_pkpk_vec = map_fit.B_pkpk_vec;
p_meas_vec = map_fit.p_meas_vec;

% extract the range (frequency and flux density) of the loss map
fct_range = get_range(f_vec, B_pkpk_vec);

% fit the parametrized fitting function with the provided data
param_fit = get_igse_fit(f_vec, B_pkpk_vec, p_meas_vec);

% get a function handle describing the fitted loss model
fct_model = @(f_vec, d_mat, B_mat) get_igse_model(f_vec, d_mat, B_mat, fct_range, param_fit);

end
```

Find the fitting range

Step 2: Find the Range

```
function fct_range = get_range(f_vec, B_pkpk_vec)
% Extract the range (frequency and flux density) of a loss map.

% alpha radius (see alphaShape, 'Inf' for full triangulation)
alpha = 0.2;

% shape object describing the loss map range
shp_obj = alphaShape(log10(f_vec).', log10(B_pkpk_vec).', alpha);

% function testing if query points are within the loss map range
fct_range = @(f, B_pkpk) shp_obj.inShape(log10(f), log10(B_pkpk));

end
```

```
function fct_model = get_model(map_fit)
% Parametrize a loss model (IGSE or IGCC) with a measured loss map.

f_vec = map_fit.f_vec;
B_pkpk_vec = map_fit.B_pkpk_vec;
p_meas_vec = map_fit.p_meas_vec;

% extract the range (frequency and flux density) of the loss map
fct_range = get_range(f_vec, B_pkpk_vec);

% fit the parametrized fitting function with the provided data
param_fit = get_igse_fit(f_vec, B_pkpk_vec, p_meas_vec);

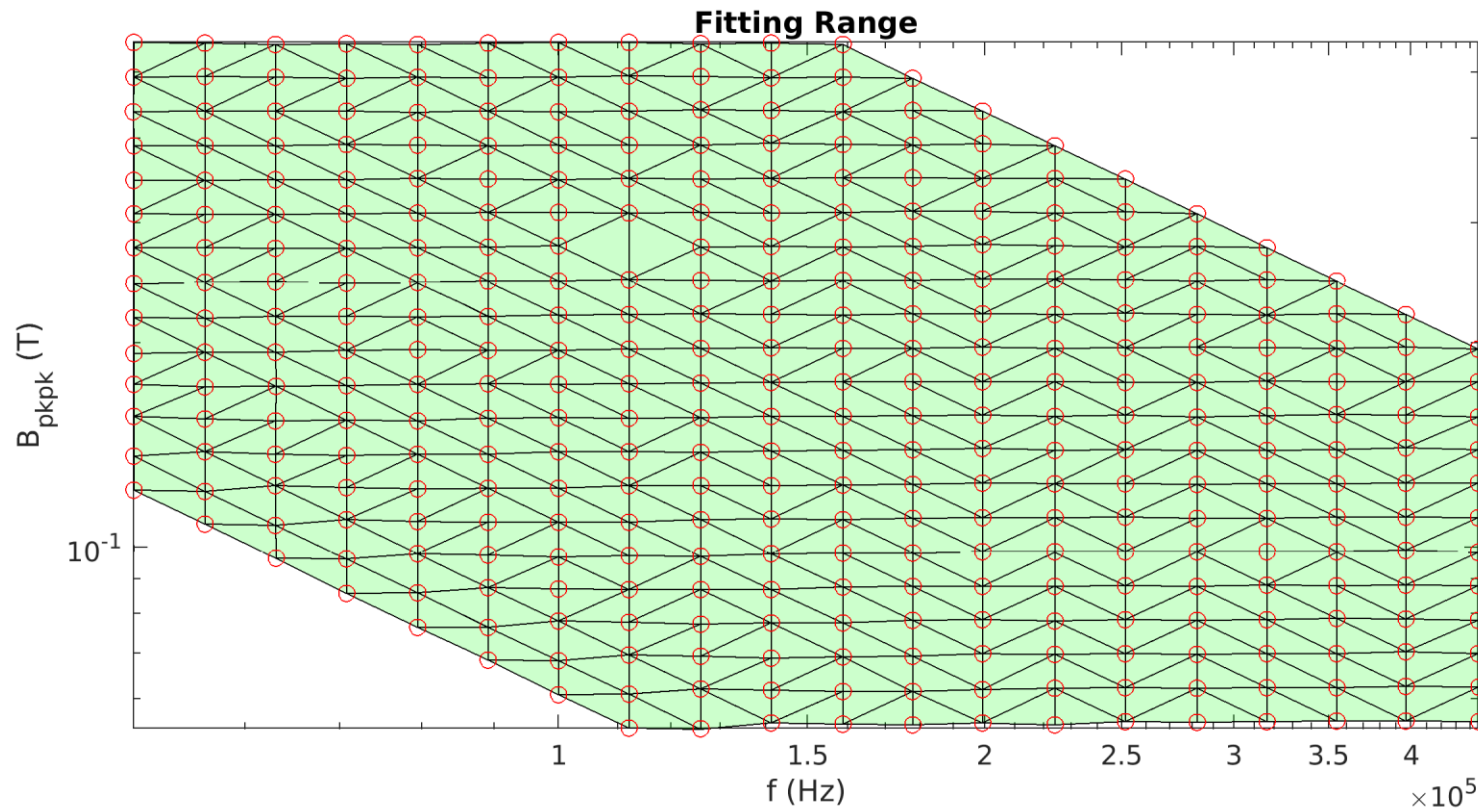
% get a function handle describing the fitted loss model
fct_model = @(f_vec, d_mat, B_mat) get_igse_model(f_vec, d_mat, B_mat, fct_range, param_fit);

end
```

- Create a shape representing the fitting range
- Return a function detecting evaluation outside the range

Step 2: Find the Range

- Find the fitting dataset range
- Detect extrapolation during model evaluation



Step 2: Fit the Model

```
function fct_model = get_model(map_fit)
% Parametrize a loss model (iGSE or iGCC) with a measured loss map.

f_vec = map_fit.f_vec;
B_pkpk_vec = map_fit.B_pkpk_vec;
p_meas_vec = map_fit.p_meas_vec;

% extract the range (frequency and flux density) of the loss map
fct_range = get_range(f_vec, B_pkpk_vec);

% fit the parametrized fitting function with the provided data
param_fit = get_igse_fit(f_vec, B_pkpk_vec, p_meas_vec);

% get a function handle describing the fitted loss model
fct_model = @(f_vec, d_mat, B_mat) get_igse_model(f_vec, d_mat, B_mat, fct_range, param_fit);

end
```

Find the optimal fit

Step 2: Find the Optimal Fit

```
function param_fit = get_igse_fit(f_vec, B_pkpk_vec, p_meas_vec)
% Extraction of a least-square fit of a function with respect to a loss map.

% get the initial value vector
x0 = [0.0, 0.0, 0.0];

% function evaluating the fit function for given parameters
fct_eval = @(x) x(1).*(f_vec.^x(2)).*(B_pkpk_vec.^x(3));

% function describing the relative error between the fits and the measurements
fct_fun = @(x) (fct_eval(x)-p_meas_vec)./p_meas_vec;

% get the options for the least-square fitting algorithm
fit_options = struct('FunctionTolerance', 1e-6, 'Display', 'off');

% solve the fitting problem with a least-square fitting algorithm
x = lsqnonlin(fct_fun, x0, [], [], fit_options);

% extract the fitted parameters
param_fit = cell2struct(num2cell(x.'), {'k', 'alpha', 'beta'});

end
```

```
function fct_model = get_model(map_fit)
% Parametrize a loss model (iGSE or iGCC) with a measured loss map.

f_vec = map_fit.f_vec;
B_pkpk_vec = map_fit.B_pkpk_vec;
p_meas_vec = map_fit.p_meas_vec;

% extract the range (frequency and flux density) of the loss map
fct_range = get_range(f_vec, B_pkpk_vec);

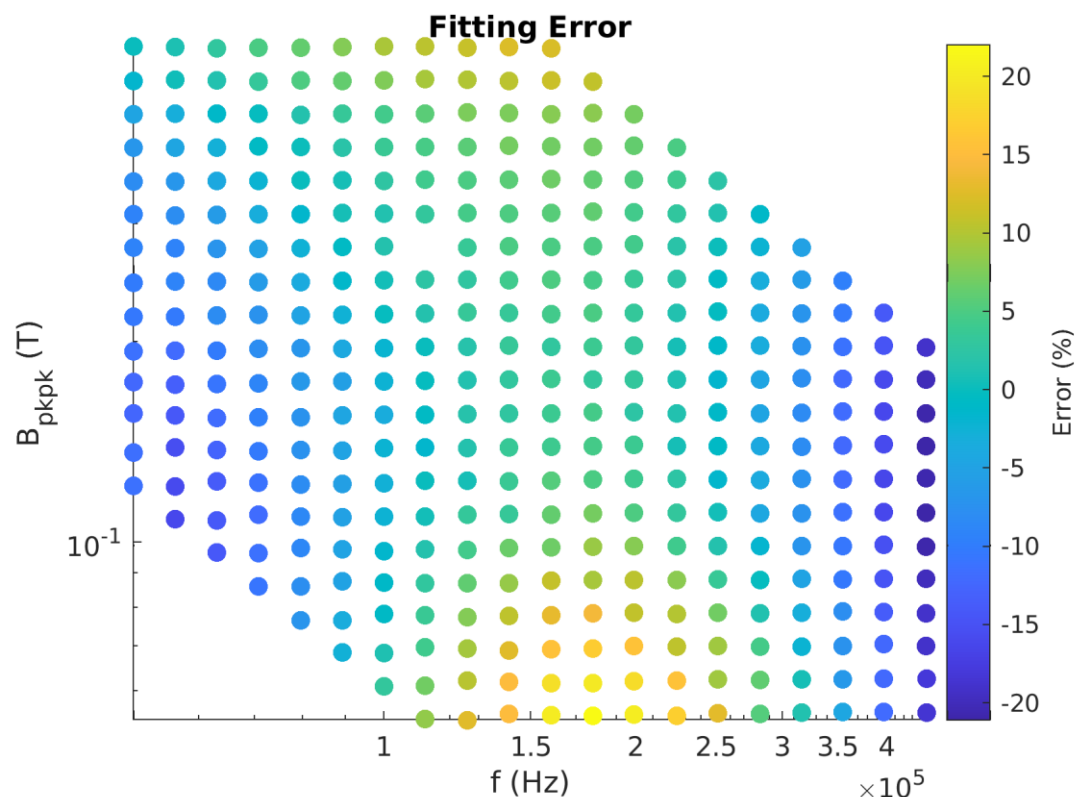
% fit the parametrized fitting function with the provided data
param_fit = get_igse_fit(f_vec, B_pkpk_vec, p_meas_vec);

% get a function handle describing the fitted loss model
fct_model = @(f_vec, d_mat, B_mat) get_igse_model(f_vec, d_mat, B_mat, fct_range, param_fit);

end
```

- **Get a function returning the relative errors for given fitting parameters**
- **Find the optimal fitting Steinmetz parameters with a least-square algorithm**

- Evaluate the performance of the fit



```
fit
errors
  n_points = 346
  err_mean = 6.920 %
  err_rms = 8.646 %
  err_95th = 18.161 %
  err_max = 22.032 %
parameters
  k = 1.397e+00
  alpha = 1.332e+00
  beta = 2.423e+00
```

Step 2: Fit the Model

```
function fct_model = get_model(map_fit)
% Parametrize a loss model (iGSE or iGCC) with a measured loss map.

f_vec = map_fit.f_vec;
B_pkpk_vec = map_fit.B_pkpk_vec;
p_meas_vec = map_fit.p_meas_vec;

% extract the range (frequency and flux density) of the loss map
fct_range = get_range(f_vec, B_pkpk_vec);

% fit the parametrized fitting function with the provided data
param_fit = get_igse_fit(f_vec, B_pkpk_vec, p_meas_vec);

% get a function handle describing the fitted loss model
fct_model = @(f_vec, d_mat, B_mat) get_igse_model(f_vec, d_mat, B_mat, fct_range, param_fit);

end
```

Get the model

Step 2: Get the Model

```
function [valid_vec, p_model_vec] = get_igse_model(f_vec, d_mat, B_mat, fct_range, param_fit)
% Definition of the iGSE model.

k = param_fit.k;
alpha = param_fit.alpha;
beta = param_fit.beta;

% compute the duration and gradient of the segments
dd_mat = diff(d_mat, 1, 1);
dB_mat = diff(B_mat, 1, 1);
dB_dt_mat = f_vec.*(dB_mat./dd_mat);

% extract the peak-to-peak flux densities
B_pkpk_vec = max(B_mat, [], 1)-min(B_mat, [], 1);

% check which points are within the fitting range
valid_vec = fct_range(f_vec, B_pkpk_vec);

% compute the iGSE integral (for piecewise linear waveforms)
w_mat = (k./(2.^alpha)).*(B_pkpk_vec.^(beta-alpha)).*(abs(dB_dt_mat).^alpha);
p_model_vec = sum(dd_mat.*w_mat, 1);

end
```

```
function fct_model = get_model(map_fit)
% Parametrize a loss model (iGSE or iGCC) with a measured loss map.

f_vec = map_fit.f_vec;
B_pkpk_vec = map_fit.B_pkpk_vec;
p_meas_vec = map_fit.p_meas_vec;

% extract the range (frequency and flux density) of the loss map
fct_range = get_range(f_vec, B_pkpk_vec);

% fit the parametrized fitting function with the provided data
param_fit = get_igse_fit(f_vec, B_pkpk_vec, p_meas_vec);

% get a function handle describing the fitted loss model
fct_model = @(f_vec, d_mat, B_mat) get_igse_model(f_vec, d_mat, B_mat, fct_range, param_fit);

end
```

- **Compute the gradient of the piecewise linear segments**
- **Get the pk-to-pk flux**
- **Detect extrapolation**
- **Compute the iGSE summation for piecewise linear signals**


```
function run_igse()
% Parametrize and evaluate the iGSE loss model.

% load the fitting and evaluation sets
map_fit = load('data/N87_25C_fit.mat');
map_eval = load('data/N87_25C_eval.mat');

% parametrize the loss model with the loss map
fct_model = get_model(map_fit);

% evaluate a loss model and compare the results
map_eval = get_eval(map_eval, fct_model);

% save the results
save('data/N87_25C_res.mat', '-struct', 'map_eval');

end
```

Step 3: eval. the model

Step 3: Evaluate the Model

```
function map_eval = get_eval(map_eval, fct_model)
% Evaluate a loss model and compare the results with the measurements.

f_vec = map_eval.f_vec;
d_mat = map_eval.d_mat;
B_mat = map_eval.B_mat;
p_meas_vec = map_eval.p_meas_vec;

% evaluate the loss model
[valid_vec, p_model_vec] = fct_model(f_vec, d_mat, B_mat);

% compute the relative error between the loss model and the measurements
err_model_vec = (p_model_vec-p_meas_vec)./p_meas_vec;

% add the predicted losses to the loss map
map_eval.valid_vec = valid_vec;
map_eval.p_model_vec = p_model_vec;
map_eval.err_model_vec = err_model_vec;

end
```

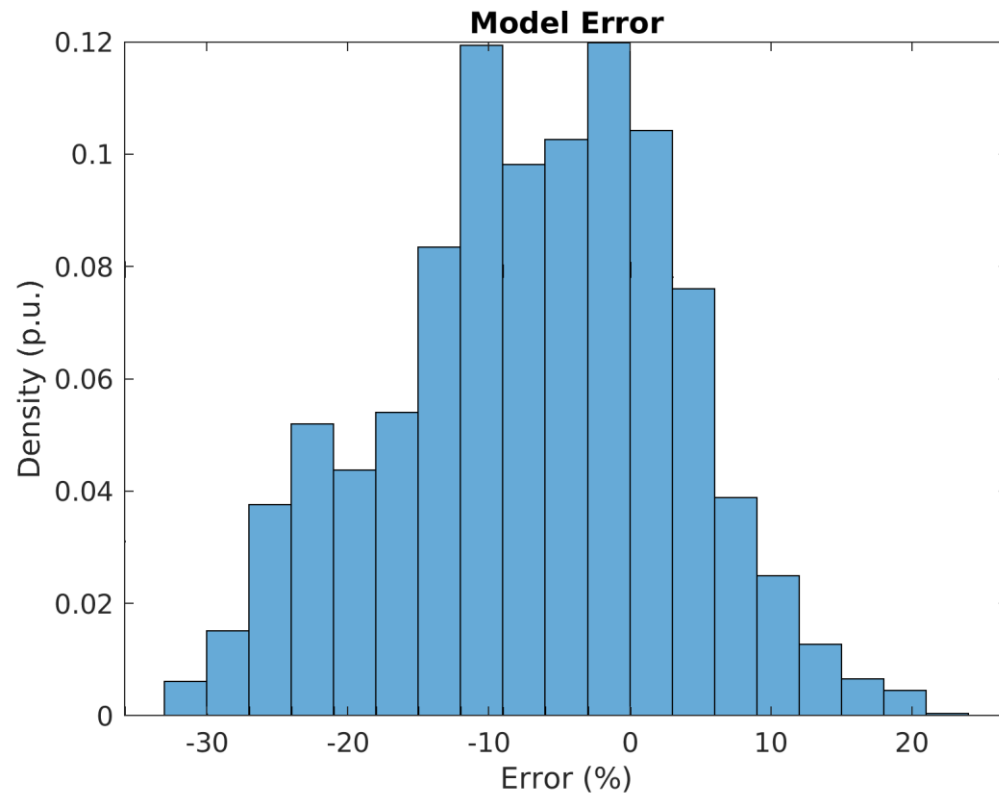
Get the dataset

Evaluate the model

Compute the deviation

Assign the results

- Evaluate the model performance



```
eval
  all points
    n_points = 2446
    err_mean = 9.642 %
    err_rms = 12.195 %
    err_95th = 24.498 %
    err_max = 32.038 %
  valid points
    n_points = 2279
    err_mean = 9.510 %
    err_rms = 12.139 %
    err_95th = 24.632 %
    err_max = 32.038 %
  invalid points
    n_points = 167
    err_mean = 11.439 %
    err_rms = 12.934 %
    err_95th = 20.696 %
    err_max = 22.796 %
```

```
function run_igse()
% Parametrize and evaluate the iGSE loss model.

% load the fitting and evaluation sets
map_fit = load('data/N87_25C_fit.mat');
map_eval = load('data/N87_25C_eval.mat');

% parametrize the loss model with the loss map
fct_model = get_model(map_fit);

% evaluate a loss model and compare the results
map_eval = get_eval(map_eval, fct_model);

% save the results
save('data/N87_25C_res.mat', '-struct', 'map_eval');

end
```

Step 4: save the data

- **Tools for development and debugging**
 - Display the results and metrics
 - Plot the results for the complete dataset
 - Plot the results for a single datapoint
- **Between the model fitting and the model evaluation**
 - Minimize the coupling
 - Use clear interfaces
- **Code performance**
 - Use vectorized instructions (no loops)
 - Downsampling / simplifying of the waveshapes
 - Do not overoptimize the code !
- **Make the code easy to test and deploy !**

Thank you! Questions?



<https://mag-net.princeton.edu>

<https://github.com/PrincetonUniversity/magnet>

<https://github.com/minjiechen/magnetchallenge>

https://github.com/otvam/magnet_webinar_eqn_models