PHyMOS

Accelerating the simulation of equation-based models by replacing non-linear algebraic loops with error-controlled machine learning surrogates

Andreas Heuermann and Bernhard Bachmann

Faculty of Engineering and Mathematics

Bielefeld University of Applied Sciences, Germany

FH Bielefeld
University of
Applied Sciences

Why non-linear Loops?

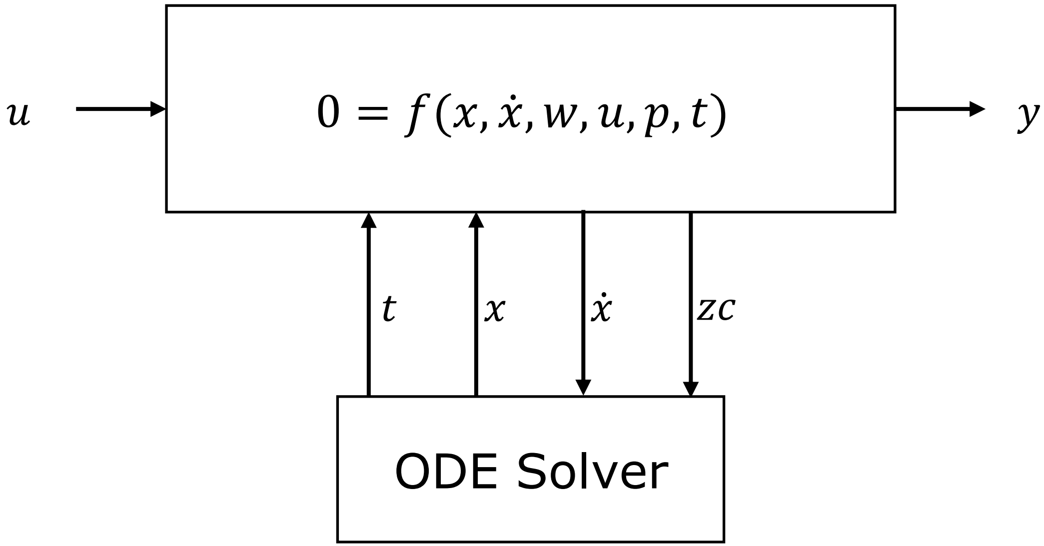
A.k.a. non-linear equation systems / strong components

Replacing Strong Components

Why replace non-linear algebraic loops?

- Expensive to solve
- Error control possible
- Improve ODE solver step size
- Learning task easier than learning whole ODE
- Improved start values for classic NLS solver
- Keep most of the physics

Replacing Strong Components



t	time
p	parameters
$u(t)$	inputs
$x(t)$	continuous states
$w(t)$	local variables
$y(t)$	outputs
$zc(t)$	event indicators

Replacing Strong Components

$$z(t) := \begin{pmatrix} \dot{x}(t) \\ w(t) \end{pmatrix} \text{ system unknowns}$$

$$\begin{aligned} f_1(z_3, z_4) &= 0 \\ f_2(z_2) &= 0 \\ f_3(z_2, z_3, z_5) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \end{aligned}$$

$$\begin{aligned} f_2(z_2) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_3(z_2, z_3, z_5) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \\ f_1(z_3, z_4) &= 0 \end{aligned}$$

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{array} \begin{array}{c} \begin{array}{ccccc} z_1 & z_2 & z_3 & z_4 & z_5 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

$$\begin{array}{c} \\ f_2 \\ f_4 \\ f_3 \\ f_5 \\ f_1 \end{array} \begin{array}{c} \begin{array}{ccccc} z_2 & z_1 & z_3 & z_5 & z_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & \boxed{1 \quad 1} & & 0 \\ 0 & 1 & \boxed{1 \quad 1} & & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Replacing Strong Components

$$z(t) \coloneqq \begin{pmatrix} \dot{x}(t) \\ w(t) \end{pmatrix} \text{ system unknowns}$$

$$\begin{aligned} f_1(z_3, z_4) &= 0 \\ f_2(z_2) &= 0 \\ f_3(z_2, z_3, z_5) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \end{aligned}$$

$$\begin{aligned} f_2(z_2) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_{NLS}(z_1, z_2, z_3, z_5) &= 0 \\ f_1(z_3, z_4) &= 0 \end{aligned}$$

Machine learning surrogate f_S →

$$\begin{aligned} f_2(z_2) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_S(z_1, z_2) &= z_3, z_5 \\ f_1(z_3, z_4) &= 0 \end{aligned}$$

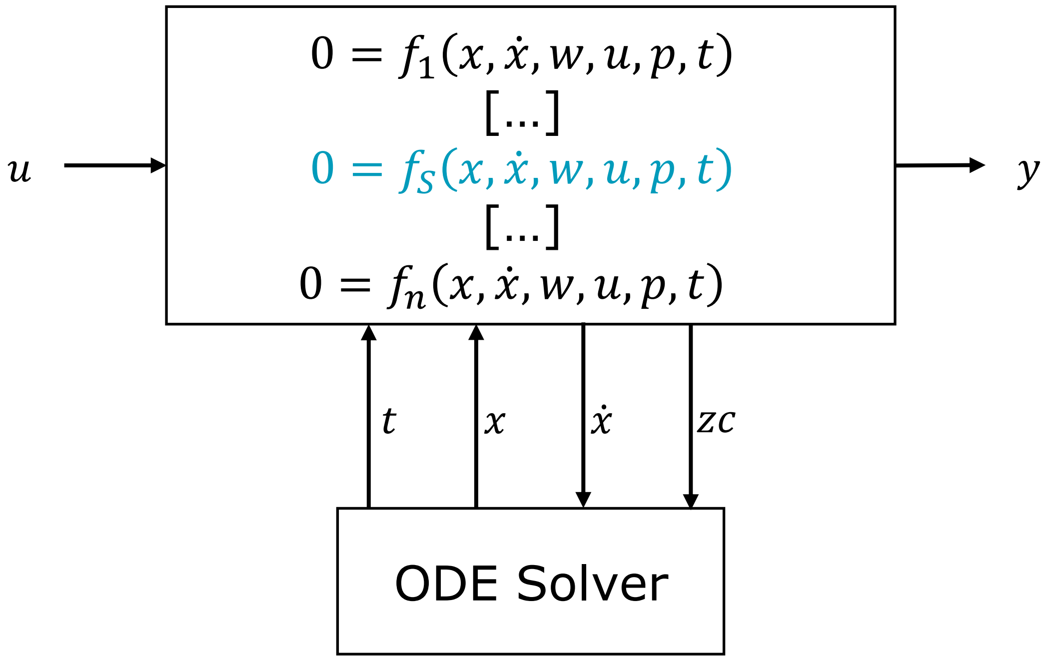
$$\begin{array}{c|ccccc} & z_1 & z_2 & z_3 & z_4 & z_5 \\ \hline f_1 & 0 & 0 & 1 & 1 & 0 \\ f_2 & 0 & 1 & 0 & 0 & 0 \\ f_3 & 0 & 1 & 1 & 0 & 1 \\ f_4 & 1 & 1 & 0 & 0 & 0 \\ f_5 & 1 & 0 & 1 & 0 & 1 \end{array}$$

$$\begin{array}{c|ccccc} & z_2 & z_1 & z_3 & z_5 & z_4 \\ \hline f_2 & 1 & 0 & 0 & 0 & 0 \\ f_4 & 1 & 1 & 0 & 0 & 0 \\ f_3 & 1 & 0 & 1 & 1 & 0 \\ f_5 & 0 & 1 & 1 & 1 & 0 \\ f_1 & 0 & 0 & 1 & 0 & 1 \end{array}$$

$$\begin{array}{c|cccc} & z_2 & z_1 & z_3, z_5 & z_4 \\ \hline f_2 & 1 & 0 & 0 & 0 \\ f_4 & 1 & 1 & 0 & 0 \\ f_S & 1 & 1 & 1 & 0 \\ f_1 & 0 & 0 & 1 & 1 \end{array}$$

FH Bielefeld University of Applied Sciences

Replacing Strong Components



t	time
p	parameters
$u(t)$	inputs
$x(t)$	continuous states
$w(t)$	local variables
$y(t)$	outputs
$zc(t)$	event indicators

Example

Simple Loop with two unknowns

Example: Simple Loop
Intersection between Circle and Line

Non-linear system (solve for x, y):

$$r^2 = x^2 + y^2$$
$$rs + b = x + y$$

Tearing

Transformed into:

Inner equation { $x = rs + b - y$

Residual equation { $0 = y^2 + x^2 - r^2$

2 Unknowns: x, y

1 Iteration variable: y

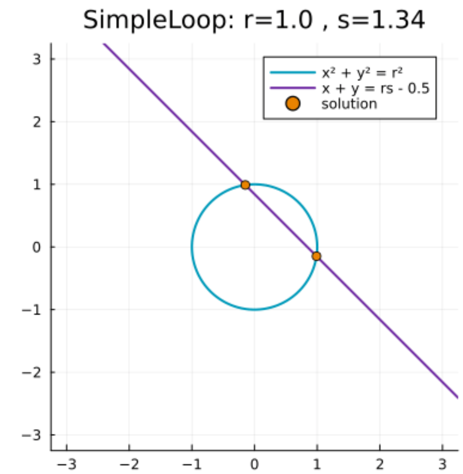
Parameters: b

Knowns: r, s

```
model simpleLoop
  Real r(min = 0);
  Real s(min = -sqrt(2), max = sqrt(2));
  Real x(start=1.0), y(start=0.5);
  parameter Real b = -0.5;
equation
  r = 1+time;
  s = sqrt((2-time)*0.9);

  r^2 = x^2 + y^2;
  r*s + b = x + y;
end simpleLoop;
```

Example: Simple Loop
Intersection between Circle and Line



Non-linear system (solve for x, y):

$$r^2 = x^2 + y^2$$
$$rs + b = x + y$$

Transformed into:

Tearing

Inner equation $\{ \quad x = rs + b - y$

Residual equation $\{ \quad 0 = y^2 + x^2 - r^2$

2 Unknowns:	x, y
1 Iteration variable:	y
Parameters:	b
Knowns:	r, s

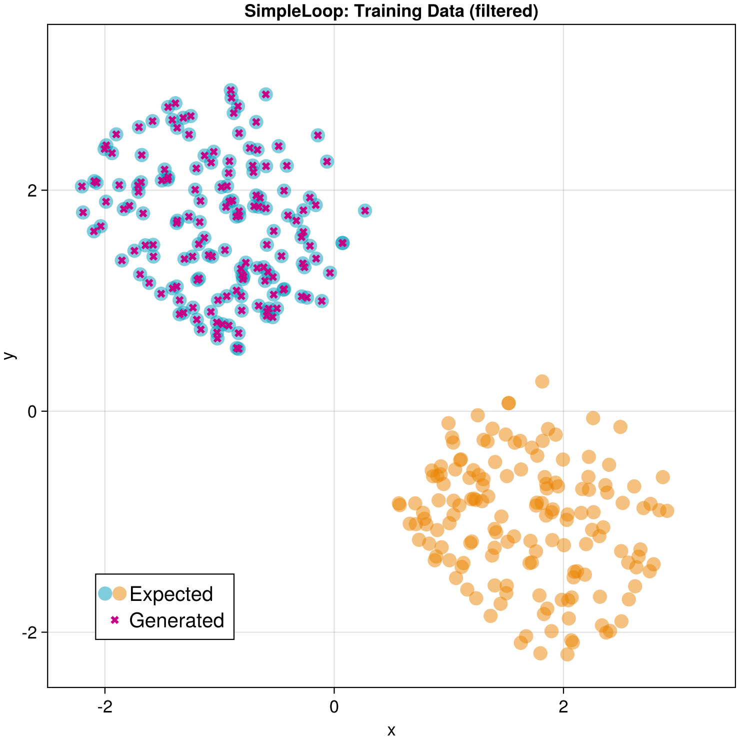
ML Surrogate:

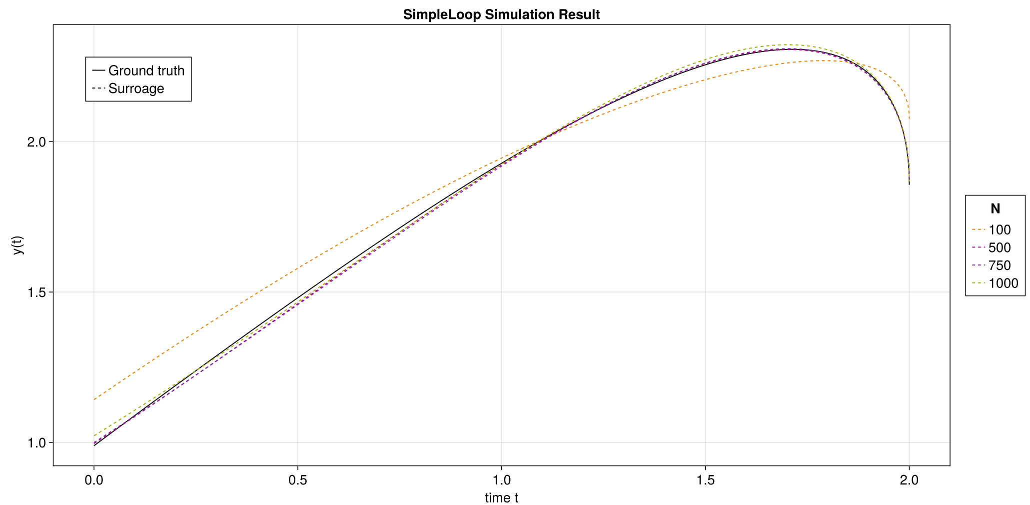
$$y = f_S(r, s, b)$$
$$x = rs + b - y$$

Inputs:	r, s, b
Outputs:	y
Evaluate Inner Equations:	x

FH Bielefeld
University of
Applied Sciences

Example: Simple Loop
Intersection between Circle and Line





Requirements on the Surrogate

Classic Newton–Raphson method

- Quadratic convergence
  - Each step doubles number of correct decimals
  - Start values extrapolated from previous solution
- Complexity: $\mathcal{O}(N^3)$

Surrogate f_S needs to be:

- Sufficiently faster than NLS solver
- Accurate
- Independent of solver step size
- Trained on relevant input space

Method Overview

Method Overview

1. Identify slow equation sets
2. Generate training data
3. Train ANN surrogate
4. Replace equation set with ANN surrogate

FH Bielefeld
University of
Applied Sciences

Automated Profiling

1. Simulate with Profiling

- `-d=infoXmlOperations` and `-clock=CPU -cpu`
- Profiling information and reference data

2. Process profiling JSON file

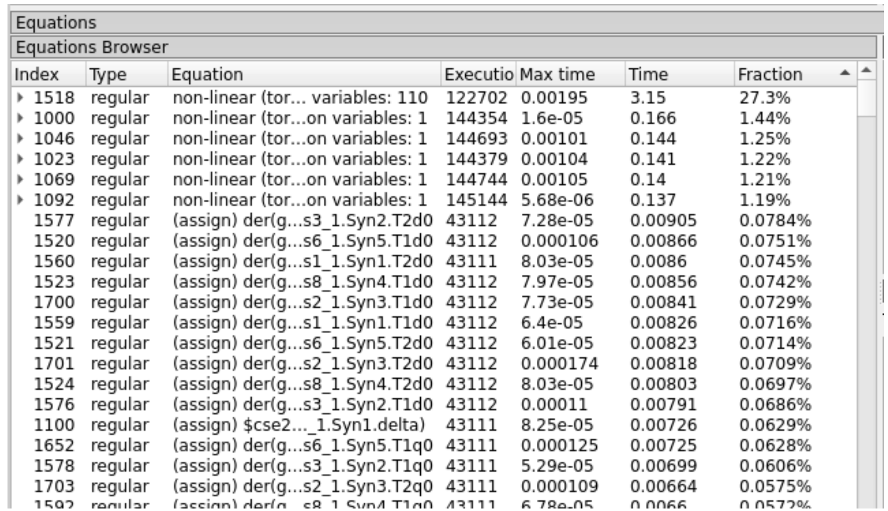
- Sort for total time
- Return equation systems over threshold

3. Process info JSON file

- Get dependent variables of equation

4. Process reference results

- Get min/max values of input variables



Equations
Equations Browser

Index	Type	Equation	Executio	Max time	Time	Fraction
1518	regular	non-linear (tor... variables: 110	122702	0.00195	3.15	27.3%
1000	regular	non-linear (tor...on variables: 1	144354	1.6e-05	0.166	1.44%
1046	regular	non-linear (tor...on variables: 1	144693	0.00101	0.144	1.25%
1023	regular	non-linear (tor...on variables: 1	144379	0.00104	0.141	1.22%
1069	regular	non-linear (tor...on variables: 1	144744	0.00105	0.14	1.21%
1092	regular	non-linear (tor...on variables: 1	145144	5.68e-06	0.137	1.19%
1577	regular	(assign) der(g...s3_1.Syn2.T2d0	43112	7.28e-05	0.00905	0.0784%
1520	regular	(assign) der(g...s6_1.Syn5.T1d0	43112	0.000106	0.00866	0.0751%
1560	regular	(assign) der(g...s1_1.Syn1.T2d0	43111	8.03e-05	0.0086	0.0745%
1523	regular	(assign) der(g...s8_1.Syn4.T1d0	43112	7.97e-05	0.00856	0.0742%
1700	regular	(assign) der(g...s2_1.Syn3.T1d0	43112	7.73e-05	0.00841	0.0729%
1559	regular	(assign) der(g...s1_1.Syn1.T1d0	43112	6.4e-05	0.00826	0.0716%
1521	regular	(assign) der(g...s6_1.Syn5.T2d0	43112	6.01e-05	0.00823	0.0714%
1701	regular	(assign) der(g...s2_1.Syn3.T2d0	43112	0.000174	0.00818	0.0709%
1524	regular	(assign) der(g...s8_1.Syn4.T2d0	43112	8.03e-05	0.00803	0.0697%
1576	regular	(assign) der(g...s3_1.Syn2.T1d0	43112	0.00011	0.00791	0.0686%
1100	regular	(assign) $cse2..._1.Syn1.delta)	43111	8.25e-05	0.00726	0.0629%
1652	regular	(assign) der(g...s6_1.Syn5.T1q0	43111	0.000125	0.00725	0.0628%
1578	regular	(assign) der(g...s3_1.Syn2.T1q0	43111	5.29e-05	0.00699	0.0606%
1703	regular	(assign) der(g...s2_1.Syn3.T2q0	43111	0.000109	0.00664	0.0575%
1592	regular	(assign) der(g...s8_1.Syn4.T1q0	43111	6.78e-05	0.0066	0.0572%

Automated Profiling

Model:
OpenIPSL.Examples.IEEE14.IEEE_14_Buses
OpenIPSL version 3.0.1

- Non-linear system 1403:
  - Unknowns: 204
    - Iteration variables: 110
    - Inner equations: 94
  - Knowns: 16 (time and states)

Index	Time [s]	Fraction [%]
1403	1.31	13.70
1594	0.06	0.61
1686	0.05	0.57
1640	0.05	0.56
1617	0.05	0.56
1663	0.05	0.55

Automated Profiling

1. Identify slow equation sets ☑
2. Generate training data ☐
3. Train ANN surrogate ☐
4. Replace equation set with ANN surrogate ☐

Generation of Training Data

1. Generate 2.0 ME C Source-Code FMU

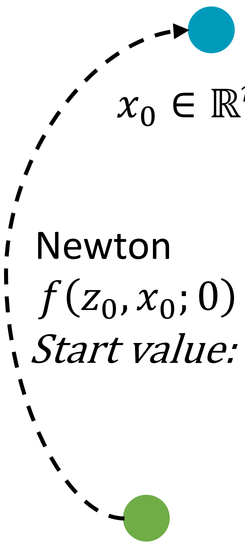
2. Add C extension

- Make it possible to evaluate single equations
- Re-compile FMU with changed sources

3. Generate training data

- Instantiate, setup experiment & initialize system
- Evaluate loop for used Variables (input) and initial values for iteration variables
- Save training data to CSV

Generation of Training Data



$x_0 \in \mathbb{R}^{n_{itter}}$

Newton

$f(z_0, x_0; 0) = 0$

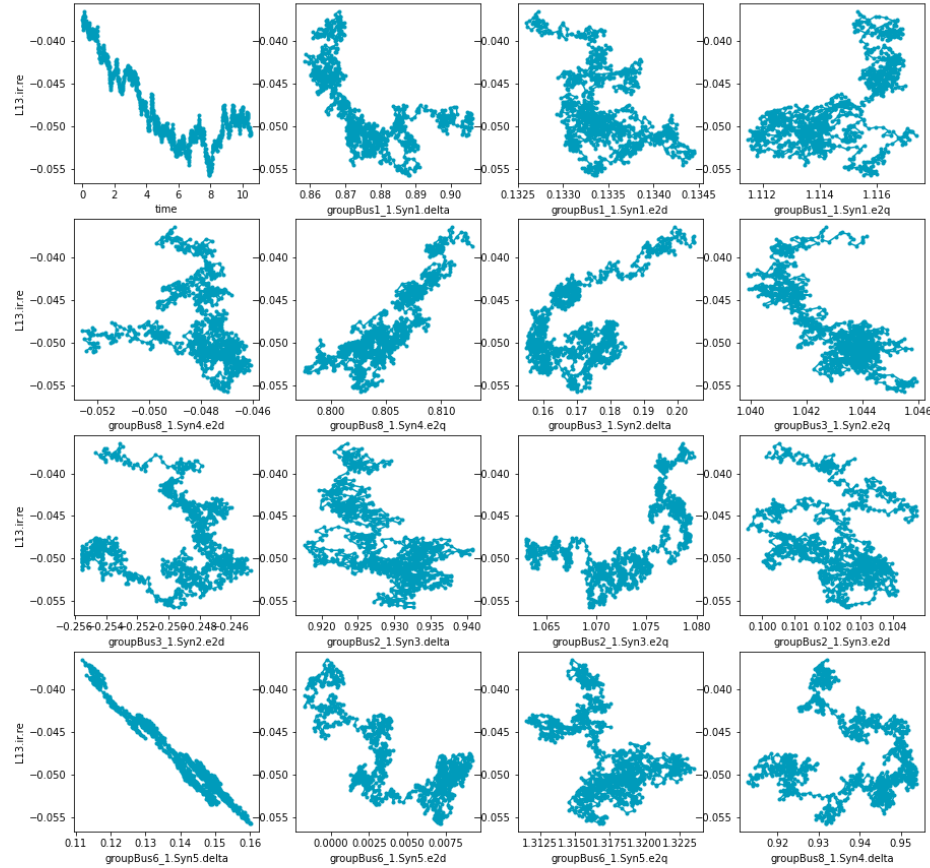
Start value: $0 \in R^{n_{itter}}$

$z_0 \in \mathbb{R}^{n_{input}}$ random

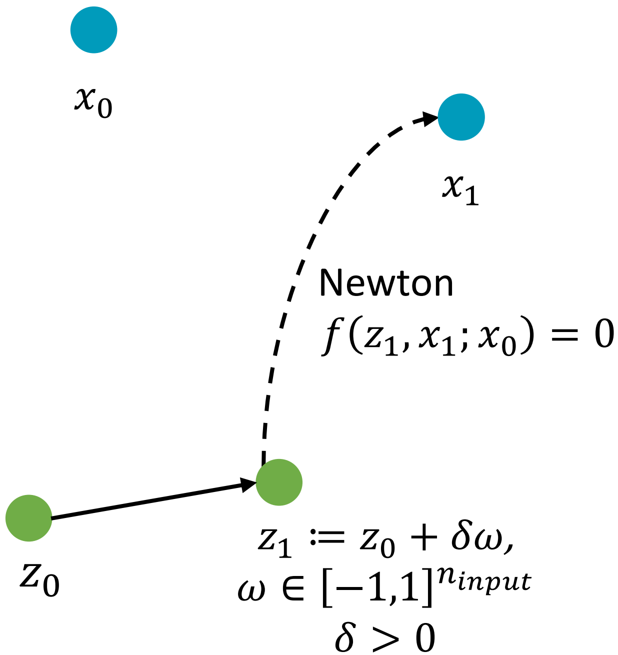
Save (z_0, x_0)

IEEE_14_Buses

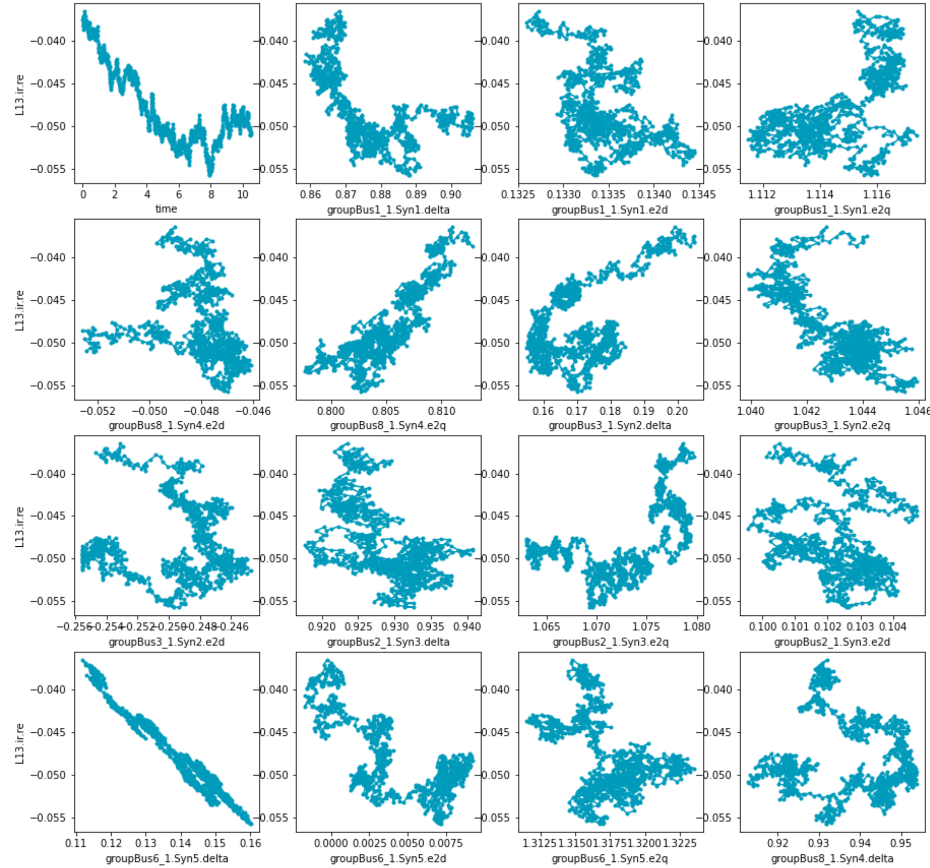
Inputs z_i vs Output $x_{i,1}$



Generation of Training Data



IEEE_14_Buses
Inputs z_i vs Output $x_{i,1}$



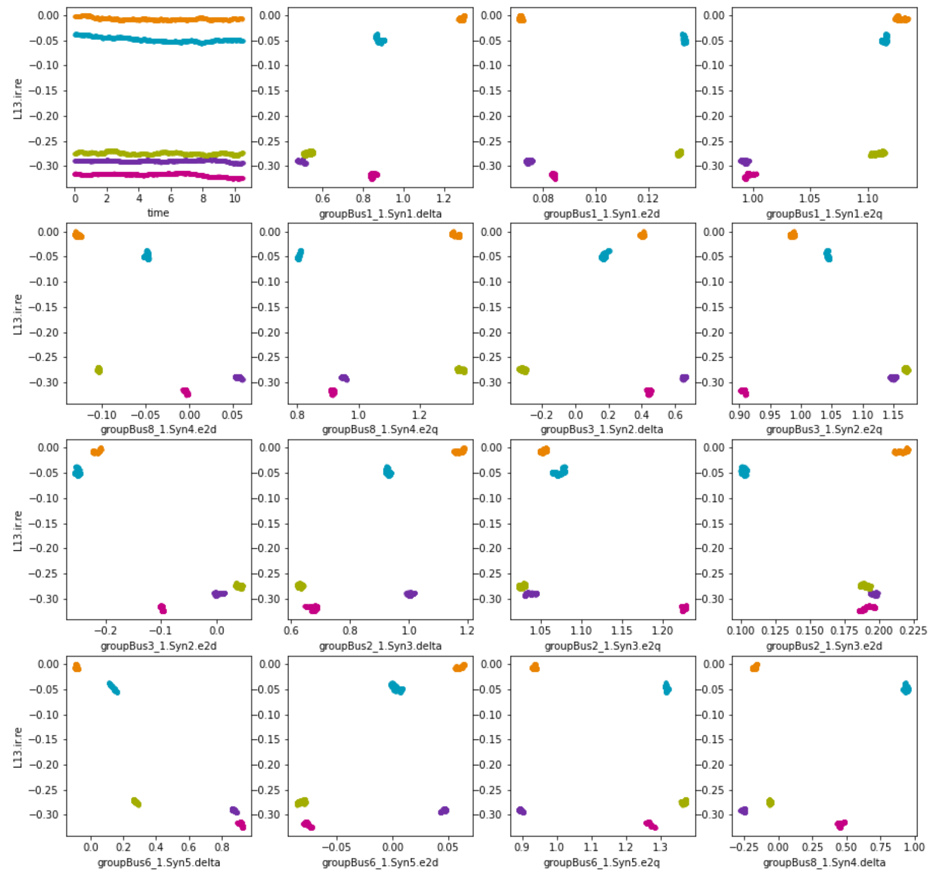
Generation of Training Data

Data generation is fast:

- Parallelized
- Only evaluating non-linear system equations.
- Start values for Newton iteration close to solution.

IEEE_14_Buses

Inputs z_i vs Output $x_{i,1}$

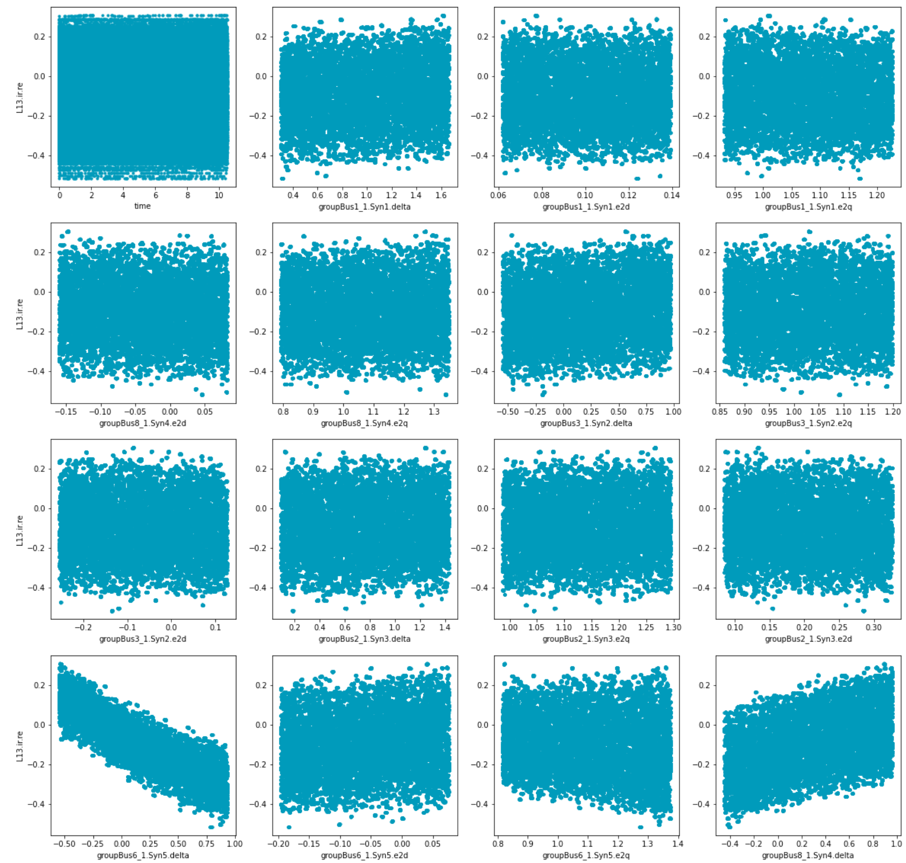


FH Bielefeld University of Applied Sciences

Generation of Training Data

IEEE_14_Buses

Inputs z_i vs Output $x_{i,1}$

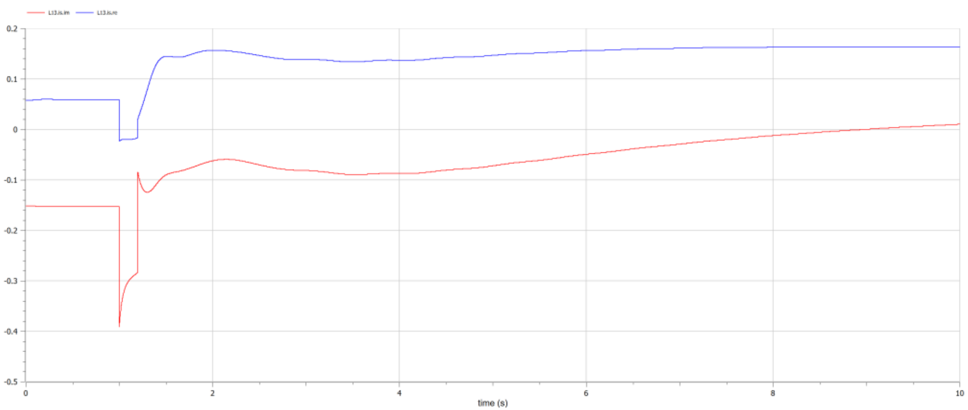


Data generation is fast:

- Parallelized
- Only evaluating non-linear system equations.
- Start values for Newton iteration close to solution.

Generation of Training Data

- Possible to have complicated equations inside residuum equations
  - Events at 1s and 1.124s

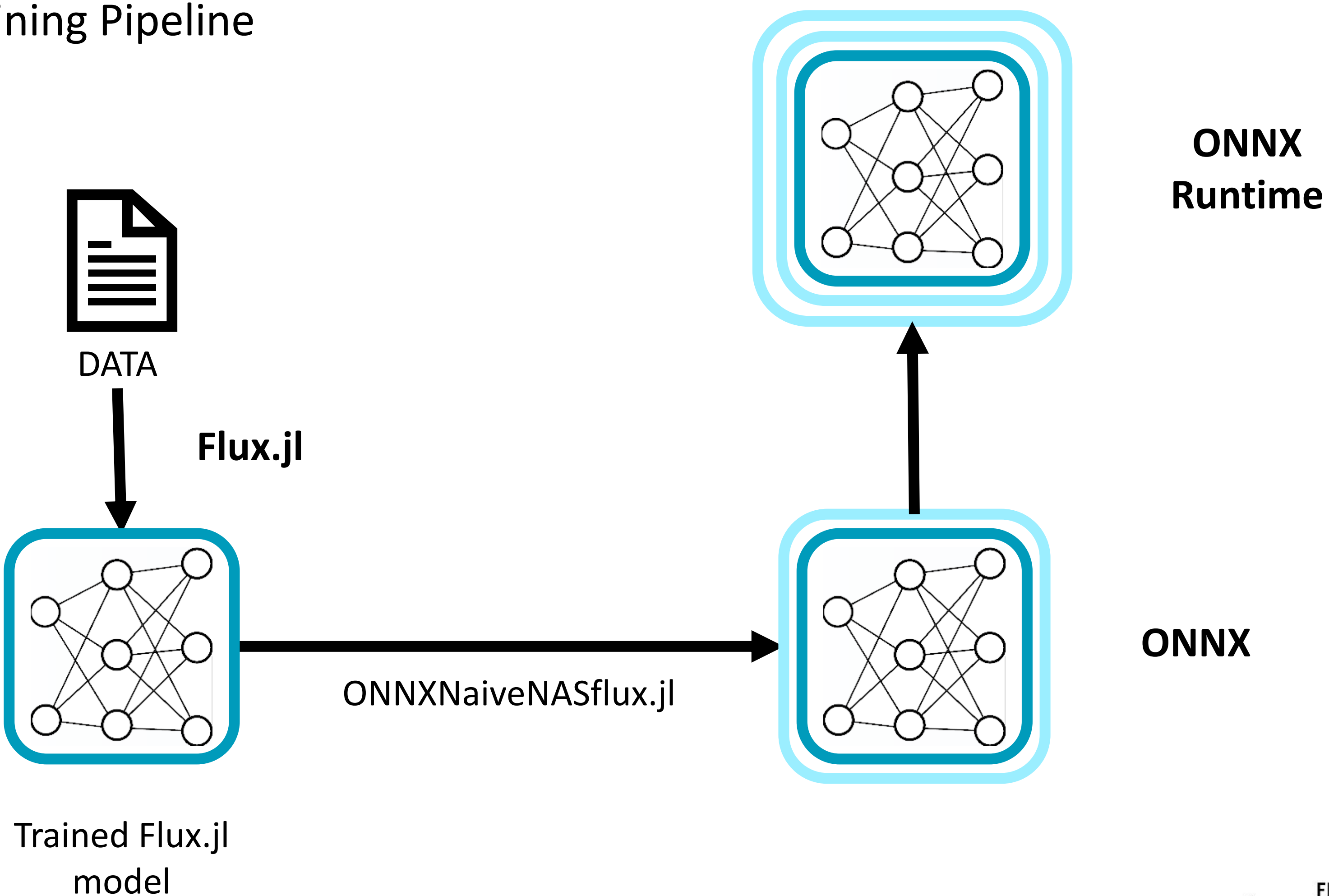



```
model OpenIPSL.Electrical.Branches.PwLine
  [...]
  if time >= t1 and time < t2 then
    if opening == 1 then
      is = Complex(0);
      ir = Complex(0);
    elseif opening == 2 then
      is = Complex(0);
      ir = (vr - ir*Z)*Y;
    else
      ir = Complex(0);
      is = (vs - is*Z)*Y;
    end if;
  else
    vs - vr = Z*(is - vs*Y);
    vr - vs = Z*(ir - vr*Y);
  end if;
end OpenIPSL.Electrical.Branches.PwLine;
```

Generation of Training Data

1. Identify slow equation sets ☑
2. Generate training data ☑
3. Train ANN surrogate ☐
4. Replace equation set with ANN surrogate ☐

Training Pipeline



Surrogate strategy

If $z_{t_i} \in TrainedArea \subset \mathbb{R}^{n\_input}$ then:

$\tilde{x} := f_S(z_{t_i})$

If $|f_{NLS}(z_{t_i}, \tilde{x})| > \epsilon$ then:

Newton: Solve $0 = f_{NLS}(z_{t_i}, x; \tilde{x})$ for x

$x_{t_i} := x$

Else:

$x_{t_i} := \tilde{x}$

Else:

$\tilde{x} := extrapolate(x_{t_{i-1}}, x_{t_{i-2}})$

Newton: Solve $0 = f_{NLS}(z_{t_i}, \mathrm{x}; \tilde{x})$ for x

$x_{t_i} := x$

ML Surrogate (first block)

Default NLS process (second block)

Replace Strong Component

Pros:

- Ensure correct solution up to given precision.
- Ensure performance outside of trained are (use default solver).
- Evaluation (mostly) independent of step-size / distance to previous solution.

Cons:

- Too slow to justify training effort:
  - Additional overhead for loading ONNX.
  - Evaluating default solver anyway.
- Solution not smooth enough rendering ODE solver step-size control useless.

Replacing Strong Components

1. Identify slow equation sets ☑
2. Generate training data ☑
3. Train ANN surrogate ☑
4. Replace equation set with ANN surrogate ☑

Ongoing Work

Problems and what‘s next

Problems with Prototype
Training and use-case

- Simple models
  - Precise, but too slow
- Large models
  - Training difficult, maybe fast
  - Speed-up potential not as high as expected
- Finding use-cases:
  - Large non-linear systems not best-practice
  - Large non-linear loops hard to train

Student project: Sensitivity analysis used vars <-> iteration variables

Problems with Prototype
FMI Standard

- Using FMU’s to evaluate only a part of the equations
  - Not always allowed to call fmi2SetXXX in FMU state
  - Variable time is special, can’t use arbitrary values

    → Don’t rely on FMI / Create simpler runtime for strong components

Next Steps

- Improve net topology
- Improve training process
  - Linear interpolation as additional input to ANN
- Improve data generation
  - Less post-processing
- Publish results
  - Julia package NonLinearSystemNeuralNetworkFMU.jl

FH Bielefeld
University of
Applied Sciences

Proper **H**ybrid **M**odels for **S**marter **V**ehicles

The presented work is part of the PHyMoS project, supported by the German Federal Ministry for Economic Affairs and Climate Action.

Homepage: https://phymos.de/

Supported by:

Federal Ministry
for Economic Affairs
and Climate Action

on the basis of a decision
by the German Bundestag

Project number: 19I20022G

Questions Remarks Comments

FH Bielefeld
University of
Applied Sciences