Replacing Strong Components with ANN Surrogates in an Open-Source Modelica Compiler

Andreas Heuermann, Philip Hannebohm and Bernhard Bachmann

Faculty of Engineering and Mathematics

Bielefeld University of Applied Sciences, Germany

FH Bielefeld
University of
Applied Sciences

Strong Components

Strong Components

- A.k.a. algebraic loops, loops, blocks
- Equations that need to be solved simultaneously

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

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

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

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

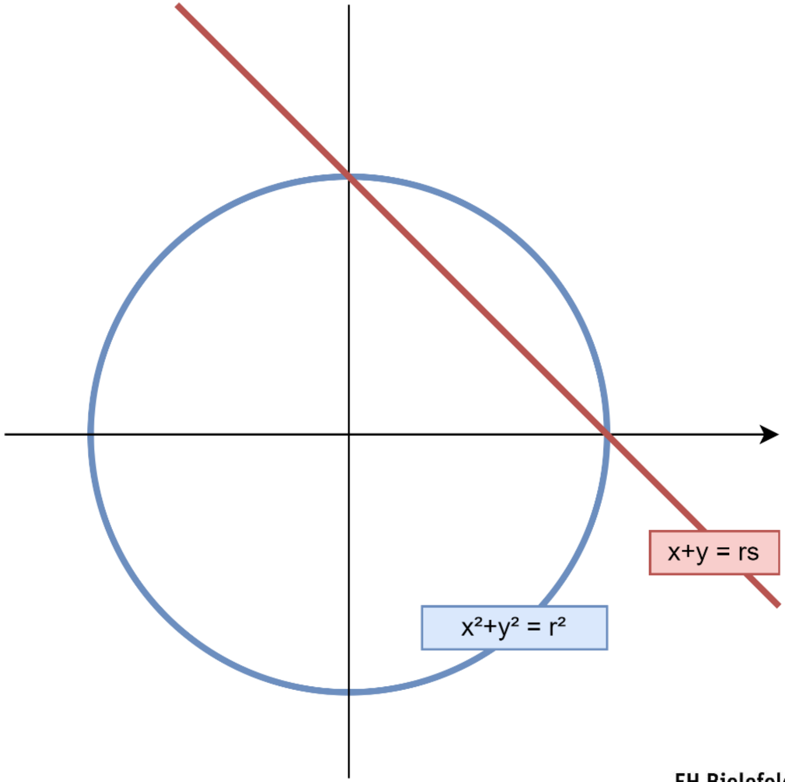
$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{array} \begin{array}{c} \begin{array}{ccccc} z_1 & z_2 & z_3 & z_4 & z_5 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & \color{red}{\mathbf{1}} & 0 \\ 0 & \color{red}{\mathbf{1}} & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & \color{red}{\mathbf{1}} \\ \color{red}{\mathbf{1}}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

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

Strong Components

```
model intersectionPoints
  Real r, s;
  Real x(start=1.0), y(start=0.5);
equation
  r = 1+time;
  s = sqrt((2-time)*0.9);

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



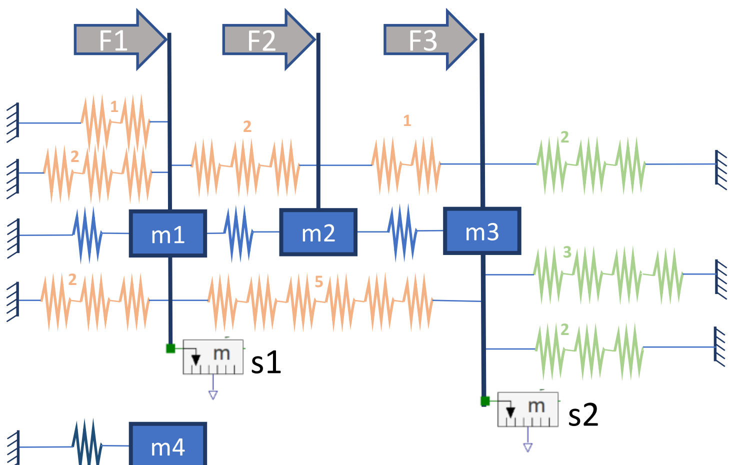
Scalable Translation Statistics

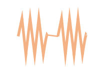
- Sophisticated model for testing
- **P**roper **H**ybrid **M**odels for **S**marter **V**ehicles
  https://phymos.de/
- Project partners LTX Simulation GmbH provided one

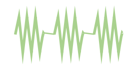
Modelica model with scalable translation statistics

- <u>Example of a scaled mass-spring system</u>

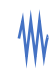
<u>Parametrization:</u>
num_masses=4
NL_equations={2,1,5,1,2,2}
Lin_equations={2,3,2}

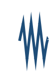


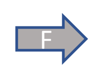

Nonlinear spring chain with 2 springs: gives a nonlinear equation system with one unknown


Linear spring chain with 3 springs: gives a linear equation system with two unknowns

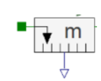
Mass with two state variables: position and velocity


Linear spring without equations: Default connection of the masses


Sleepy stiff linear spring: spring with a different stiffness to manipulate the stiffness of the whole system; contains a sleeping function to imitate longer simulation times



External Force, acting as input



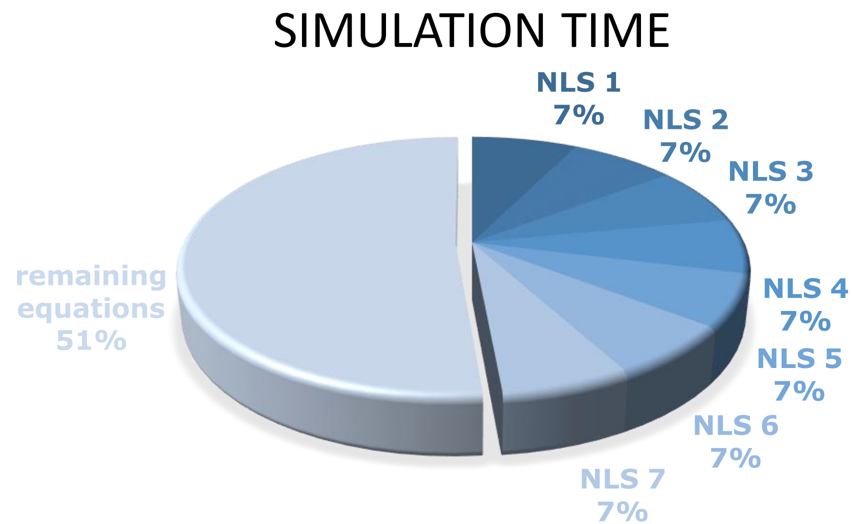
Position measurement, acting as output

LTX Simulation GmbH, Wohlfartstraße 21b, 80939 München Deutschland

Profiling Simulation Time

ScaleTranslationStatistics

- Linear torn systems: 6
- Non-linear torn systems: 8
- Single equations: 483



FH Bielefeld University of Applied Sciences

Replacing Strong Components

Why replace non-linear algebraic loops?

- Expensive
- Error control possible
- Improve ODE solver step size

Artificial Neural Network Surrogates

Artificial Neural Surrogates

We are investigating different approaches for ODE / DAE systems

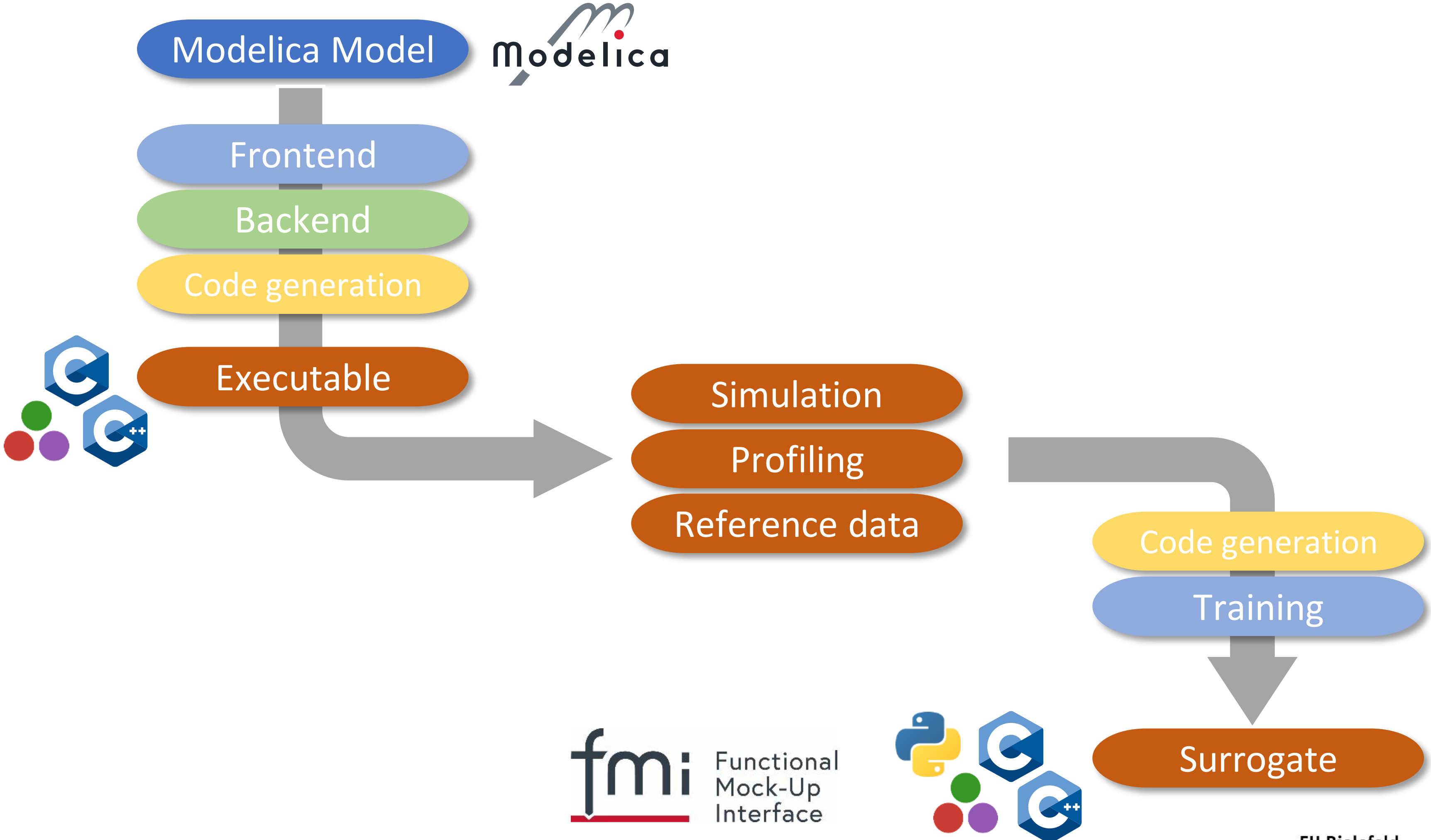
- Echo State Networks (ESN)
- Continuous-Time Echo State Networks (CTESN)
- Recurrent Neural Networks (RNN)
  - Long-Short Term Memory (LSTM)
- Polynomial Neural Networks (PNN)

Workflow

Automated Surrogate Generation

General Workflow

1. Identify relevant equation set
2. Generate training data
3. Train surrogate
4. Replace equation set with surrogate


Modelica Model
Modelica
Frontend
Backend
Code generation
Executable
Simulation
Profiling
Reference data
Code generation
Training
Surrogate
fmi
Functional
Mock-Up
Interface

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 threashold

3. Process info JSON file

- Get dependent variables of equation

4. Process reference results

- Get min/max values of relevant variables

Generation of Training Data

1. Generate 2.0 ME C Source-Code FMU

2. Add FMI-like 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 random input
- Save training data to CSV

4. Train ANN

- Using Flux.jl

FH Bielefeld
University of
Applied Sciences

FMI Extension: fmi2EvaluateEq

```
fmi2Status fmi2EvaluateEq(fmi2Component c, const size_t eqNumber) {
    ModelInstance *comp = (ModelInstance *)c;
    DATA* data = comp->fmuData;
    threadData_t *threadData = comp->threadData;

    FILTERED_LOG(comp, fmi2OK, LOG_FMI2_CALL, "myfmi2evaluateEq: Evaluating equation %u", eqNumber)

    switch (eqNumber) {
        case 14:
            simpleLoop_eqFunction_14(data, threadData);
            comp->_need_update = 0;
            break;
        default:
            return fmi2Error;
  }
  return fmi2OK;
}
```

Generation of Training Data

1. Generate 2.0 ME C Source-Code FMU

2. Add FMI-like 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 random input
- Save training data to CSV

4. Train ANN
- Using Flux.jl

FH Bielefeld
University of
Applied Sciences

Replace Strong Component Equation

Add C wrapper to embed Julia

- Add binary files and sources to FMU
- Re-compile FMU

Alternatives to embedding Julia

- Use PackageCompiler.jl to create C library bundle from Julia code
- Provide callbacks with C-compatible function pointers to Julia function @cfunction

```c
/**
 * @brief Initialize Julia instance.
 *
 * Load simpleLoop.jl from FMI resources directory.
 *
 * @param resourcesDir
 */
void initJulia(const char* resourcesDir) {
  jl_function_t* cd = NULL;
  jl_value_t* jl_resourcesDir = NULL;

  /* Setup Julia context */
  jl_init();

  /* GC roots */
  JL_GC_PUSH1(&jl_resourcesDir);

  /* Protect variables over scopes inside refs to prevent deallocation by GC */
  jl_value_t* refs = jl_eval_string("refs = IdDict()");

  /* Cd into resources */
  cd = jl_get_function(jl_base_module, "cd");
  jl_resourcesDir = jl_cstr_to_string(resourcesDir);
  jl_call1(cd, jl_resourcesDir);
  jl_eval_string("@info \"Julia running in $(pwd())\"");

  jl_eval_string("Base.include(Main, \"simpleLoop.jl\")");
  jl_eval_string("using Main.SimpleLoop");
  jl_eval_string("@info \"Package SimpleLoop loaded\"");

  JL_GC_POP();
  return;
}
```

Different Replacement Strategies

1. Replace total solver

2. Improve initial guess of solver

3. Replace Jacobian

Generalization:
Replace arbitrary sets of equations

```
model intersectionPoints
  Real r, s;
  Real x(start=1.0), y(start=0.5);
equation
  r = 1+time;
  s = sqrt((2-time)*0.9);

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


FH Bielefeld
University of
Applied Sciences

Different Replacement Strategies

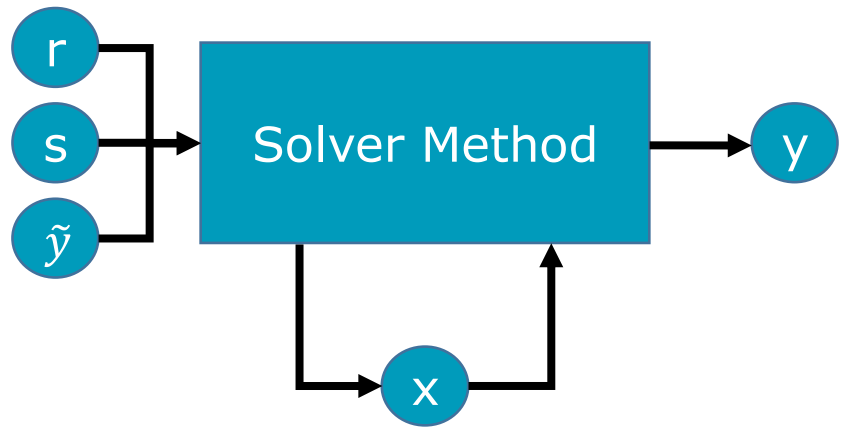
```
x = r*s - y
res = x² + y² - r²
```

1. Replace total solver

2. Improve initial guess of solver

3. Replace Jacobian

Generalization:
Replace arbitrary sets of equations



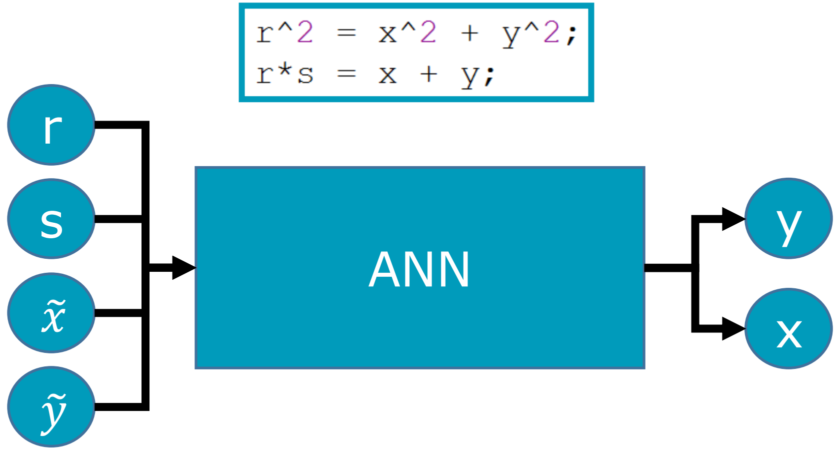
Different Replacement Strategies

1. Replace total solver

2. Improve initial guess of solver

3. Replace Jacobian

Generalization:
Replace arbitrary sets of equations



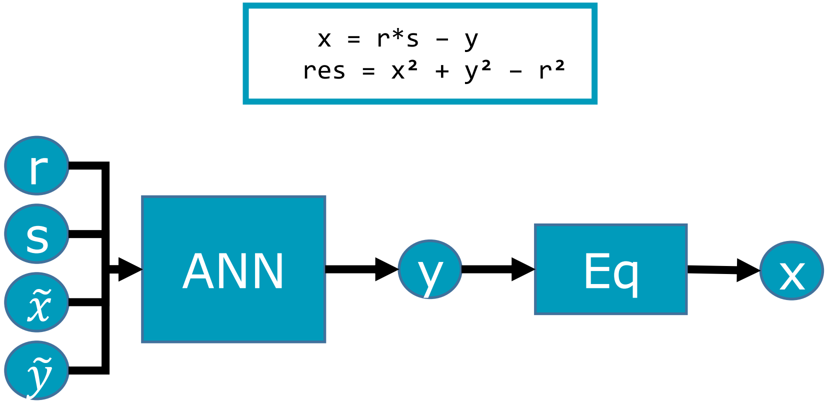
Different Replacement Strategies

1. Replace total solver

2. Improve initial guess of solver

3. Replace Jacobian

Generalization:
Replace arbitrary sets of equations



Different Replacement Strategies

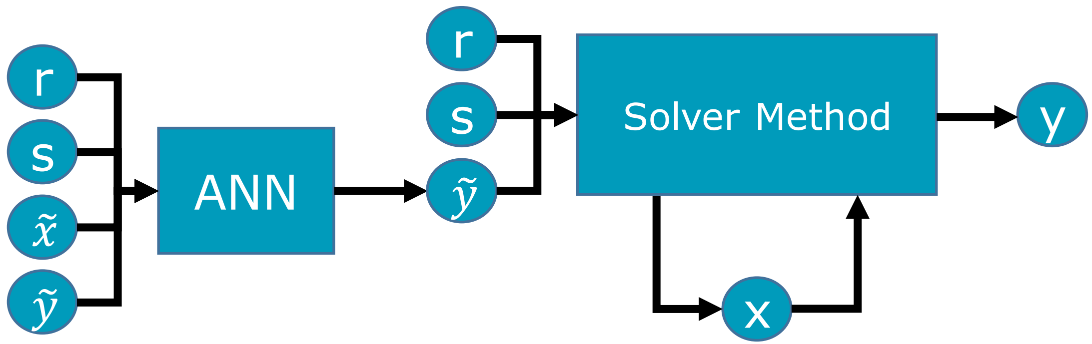
1. Replace total solver

2. Improve initial guess of solver

3. Replace Jacobian

Generalization:
Replace arbitrary sets of equations

```
x = r*s - y
res = x² + y² - r²
```



FH Bielefeld
University of
Applied Sciences

Different Replacement Strategies

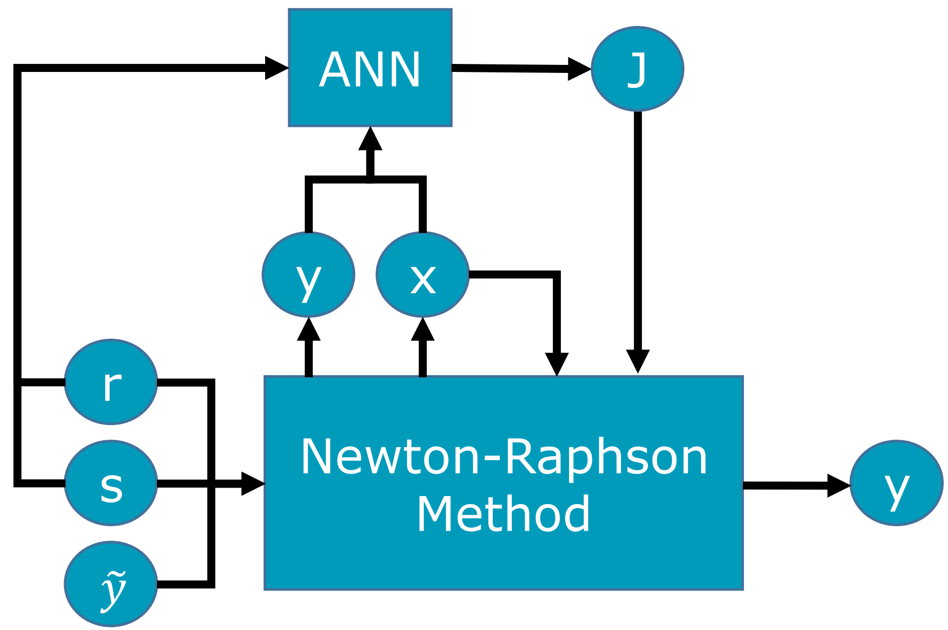
```
x = r*s - y
res = x² + y² - r²
```

1. Replace total solver

2. Improve initial guess of solver

3. Replace Jacobian

Generalization:
Replace arbitrary sets of equations



FH Bielefeld University of Applied Sciences

Replace Strong Component Equation

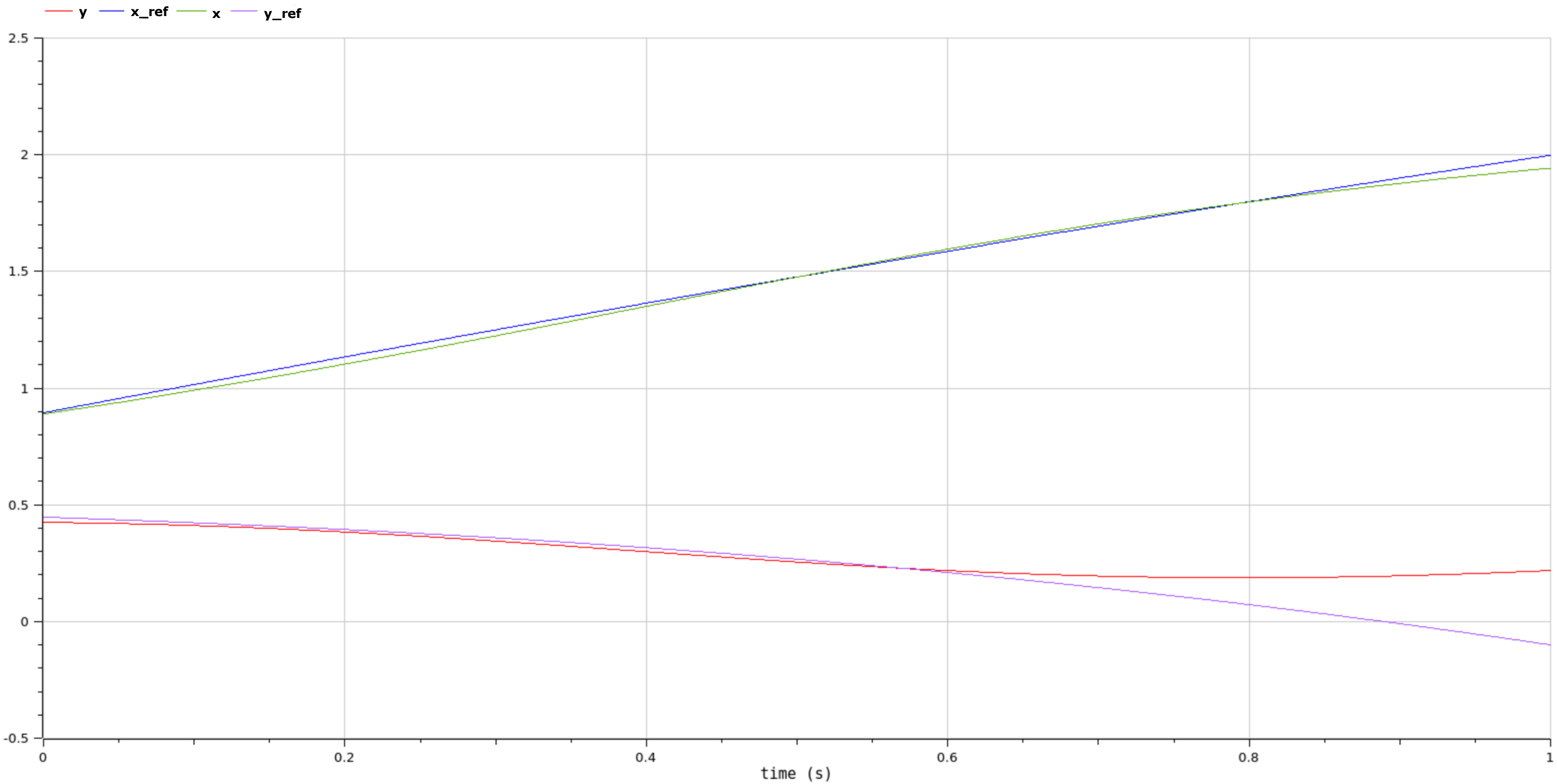
```c
void simpleLoop_eqFunction_14(DATA *data, threadData_t *threadData)
{
  TRACE_PUSH
  const int equationIndexes[2] = {1,14};
  int retValue;
  if(ACTIVE_STREAM(LOG_DT))
  {
    infoStreamPrint(LOG_DT, 1, "Solving nonlinear system 14 (STRICT TEARING SET if tearing enabled) at time = %18.10e", data->localData[0]->timeValue);
    messageClose(LOG_DT);
  }

  /* Evaluate NN */
#ifdef JULIA_FMU
  julia_pointers* juliaNNData = data->simulationInfo->nonlinearSystemData[1].juliaNNData;
  double* input = inputDataPtr(juliaNNData);
  double* output = outputDataPtr(juliaNNData);

  input[0] = data->localData[0]->realVars[0] /* r variable */;
  input[1] = data->localData[0]->realVars[1] /* s variable */;

  evalNN(juliaNNData);
  data->localData[0]->realVars[4] /* y variable */ = output[0];
  data->localData[0]->realVars[2] /* x variable */ = output[1];
#else
  /* get old value */
  data->simulationInfo->nonlinearSystemData[1].nlsxOld[0] = data->localData[0]->realVars[4] /* y variable */;
  retValue = solve_nonlinear_system(data, threadData, 1);
  /* check if solution process was successful */
  if (retValue > 0){
    const int indexes[2] = {1,14};
    throwStreamPrintWithEquationIndexes(threadData, indexes, "Solving non-linear system 14 failed at time=%.15g.\n\
                                    For more information please use -lv LOG_NLS.", data->localData[0]->timeValue);
  }
  /* write solution */
  data->localData[0]->realVars[4] /* y variable */ = data->simulationInfo->nonlinearSystemData[1].nlsx[0];
  printf("Loop solution: [x,y] = [%.4f, %.4f]\n", data->localData[0]->realVars[2], data->localData[0]->realVars[4]);
#endif
  TRACE_POP
}
```

Result for Dummy-NN



FH Bielefeld
University of
Applied Sciences

Next Steps

- Finish prototype implementation
  - Test different (ANN) methods
  - Balancing performance, accuracy and training effort

- Re-evaluate approach
  - Skip FMI or more FMI?
  - Julia vs. C/C++ vs. Python
  - Tool specific method or tool unspecific?
  - What equations / parts of a Modelica model should be replaced?

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 Energy.

Homepage: https://phymos.de/

Supported by:

Federal Ministry for Economic Affairs and Energy

on the basis of a decision by the German Bundestag

Project number: 19I20022G

PHyMoS

Questions Remarks Comments

FH Bielefeld
University of
Applied Sciences