NonLinearSystemNeuralNetworkFMU.jl

August 22, 2022

Contents

Contents ii

I Home 1

1 NonLinearSystemNeuralNetworkFMU.jl 2

1.1 Table of Contents . . . 2

1.2 Overview . . . 2

1.3 Installation . . . 3

II Profiling 4

2 Profiling Modelica Models 5

2.1 Functions . . . 5

2.2 Structures . . . 6

2.3 Examples . . . 7

III Data Generation 9

3 Training Data Generation 10

3.1 Functions . . . 10

3.2 Examples . . . 12

Part I

Home

Chapter 1

NonLinearSystemNeuralNetworkFMU.jl

Generate Neural Networks to replace non-linear systems inside OpenModelica 2.0 FMUs.

1.1 Table of Contents

- Training Data Generation
  - Functions
  - Examples
- NonLinearSystemNeuralNetworkFMU.jl
  - Table of Contents
  - Overview
  - Installation
- Profiling Modelica Models
  - Functions
  - Structures
  - Examples

1.2 Overview

The package generates an FMU from a modelica file in 3 steps (+ 1 user step):

1. Find non-linear equation systems to replace.
   - Simulate and profile Modelica model with OpenModelica using OMJulia.jl.
   - Find slowest equations below given threshold.
   - Find depending variables specifying input and output for every non-linear equation system.
   - Find min-max ranges for input variables by analyzing the simulation results.
2. Generate training data.
   - Generate 2.0 Model Exchange FMU with OpenModelica.

- Add C interface to evaluate single non-linear equation system without evaluating anything else.
- Re-compile FMU.
- Initialize FMU using FMI.jl.
- Generate training data for each equation system by calling new interface.

3. Train neural network.

   - Step performed by user.

4. Integrate neural network into FMU

   - Replace equations with neural network in generated C code.
   - Re-compile FMU.

1.3 Installation

Clone this repository to your machine and use the package manager Pkg to develop this package.

```
(@v1.7) pkg> dev /path/to/NonLinearSystemNeuralNetworkFMU
julia> using NonLinearSystemNeuralNetworkFMU
```

Part II

Profiling

Chapter 2

Profiling Modelica Models

2.1 Functions

`NonLinearSystemNeuralNetworkFMU.profiling` – Function.

```
profiling(modelName, pathToMo, pathToOmc, workingDir; threshold = 0.03)
```

Find equations of Modelica model that are slower then threashold.

Arguments

- `modelName::String`: Name of the Modelica model.
- `pathToMo::String`: Path to the *.mo file containing the model.
- `pathToOm::Stringc`: Path to omc used for simulating the model.

Keywords

- `workingDir::String = pwd()`: Working directory for omc. Defaults to the current directory.
- `threshold = 0.01`: Slowest equations that need more then `threshold` of total simulation time.

Returns

- `profilingInfo::Vector{ProfilingInfo}`: Profiling information with non-linear equation systems slower than `threshold`.

source

`NonLinearSystemNeuralNetworkFMU.minMaxValuesReSim` – Function.

```
minMaxValuesReSim(vars::Array{String}, modelName::String, pathToMo::String, pathToOmc::String;
↪  workingDir::String = pwd())
```

(Re-)simulate Modelica model and find miminum and maximum value each variable has during simulation.

Arguments

- `vars::Array{String}`: Array of variables to get min-max values for.
- `modelName::String`: Name of Modelica model to simulate.
- `pathToMo::String`: Path to .mo file.

- `pathToOm::Stringc`: Path to OpenModelica Compiler omc.

Keywords

- `workingDir::String = pwd()`: Working directory for omc. Defaults to the current directory.

Returns

- `min::Array{Float64}`: Minimum values for each variable listed in `vars`, minus some small epsilon.
- `max::Array{Float64}`: Maximum values for each variable listed in `vars`, plus some small epsilon.

See also `profiling`.

`source`

2.2 Structures

`NonLinearSystemNeuralNetworkFMU.ProfilingInfo` – Type.

```
ProfilingInfo <: Any
```

Profiling information for single non-linear equation.

- `eqInfo::EqInfo`
  Non-linear equation
- `iterationVariables::Array{String}`
  Iteration (output) variables of non-linear system
- `innerEquations::Array{Int64}`
  Inner (torn) equations of non-linear system.
- `usingVars::Array{String}`
  Used (input) variables of non-linear system.

`source`

`NonLinearSystemNeuralNetworkFMU.EqInfo` – Type.

```
EqInfo <: Any
```

Equation info struct.

- `id::Int64`
  Unique equation id
- `ncall::Int64`
  Number of calls during simulation
- `time::Float64`
  Total time [s] spend on evaluating this equation.
- `maxTime::Float64`
  Maximum time [s] needed for single evaluation of equation.
- `fraction::Float64`
  Fraction of total simulation time spend on evaluating this equation.

`source`

2.3 Examples

Find Slowest Non-linear Equation Systems

We have a Modelica model `SimpleLoop`, see test/simpleLoop.mo with some non-linear equation system

$$\begin{aligned} r^2 &= x^2 + y^2 \\ rs &= x + y \end{aligned}$$

We want to see how much simulation time is spend solving this equation. So let's start `profiling`:

```
julia> using NonLinearSystemNeuralNetworkFMU


julia> modelName = "simpleLoop";


julia> pathToMo = joinpath("test","simpleLoop.mo");



julia> profilingInfo = profiling(modelName, pathToMo, omc; threshold=0)
[ Info: Path to zmq file="/tmp/openmodelica.aheuermann.port.julia.JT5GM7Vzra"
[ Info: setCommandLineOptions
[ Info: simulate
[ Info: Slowest eq 14: ncall: 2512, time: 0.00097956, maxTime: 2.7668e-5, fraction:
↪  0.1644691221631024
2-element Vector{ProfilingInfo}:
 ProfilingInfo(EqInfo(14, 2512, 0.00097956, 2.7668e-5, 0.1644691221631024), iterationVariables:
↪  ["y"], innerEquations: [11], usingVars: ["s", "r"])
 ProfilingInfo(EqInfo(6, 8, 1.281e-5, 1.295e-5, 0.002150812053278351), iterationVariables: ["y"],
↪  innerEquations: [3], usingVars: ["s", "r"])
```

We can see that non-linear equation system 14 is using variables `s` and `r` as input and has iteration variable `y`. `x` will be computed in the inner equation.

```
julia> profilingInfo[1].usingVars
2-element Vector{String}:
 "s"
 "r"

julia> profilingInfo[1].iterationVariables
1-element Vector{String}:
 "y"
```

So we can see, that equations 14 is the slowest non-linear equation system. It is called 2512 times and needs around 15% of the total simulation time, in this case that is around 592 μs.

If we want to get the minimal and maximal values for the used variables `s` and `r` can get we can use `minMaxValuesReSim`. This will re-simulate the Modelica model and read the simulation results to find the smallest and largest values for each given variable.

```
julia> (min, max)  = minMaxValuesReSim(profilingInfo[1].usingVars, modelName, pathToMo, omc)
[ Info: Path to zmq file="/tmp/openmodelica.aheuermann.port.julia.t9pc6V8DUn"
[ Info: setCommandLineOptions
[ Info: simulate
([0.0, 0.95], [1.4087228258248679, 3.15])
```

Part III

Data Generation

Chapter 3

Training Data Generation

To generate training data for the slowest non-linear equations found during Profiling Modelica Models we now simulate the equations multiple time and save in- and outputs.

We will use the Functional Mock-up Interface (FMI) standard to generate FMU that we extend with some function to evaluate single equations without the need to simulate the rest of the model.

3.1 Functions

`NonLinearSystemNeuralNetworkFMU.generateFMU` – Function.

```
generateFMU(;modelName::String,
             pathToMo::String,
             pathToOmc::String,
             tempDir::String,
             clean::Bool = false)
```

Generate 2.0 Model Exchange FMU for Modelica model using OMJulia.

Keywords

- `modelName::String`: Name of Modelica model to export as FMU.
- `pathToMo::String`: Path to Modelica file.
- `pathToOmc::String`: Path to OpenModlica Compiler.
- `tempDir::String`: Path to temp directory in which FMU will be saved to.
- `clean::Bool=false`: True if tempDir should be removed and re-created before working in it.

Returns

- Path to generated FMU `tempDir/<modelName>.fmu`.

See also `addEqInterface2FMU`, `generateTrainingData`.

source

`NonLinearSystemNeuralNetworkFMU.addEqInterface2FMU` – Function.

```
addEqInterface2FMU(;modelName::String,
                    pathToFmu::String,
                    pathToFmiHeader::String,
                    eqIndices::Array{Int64},
                    tempDir::String)
```

Create extendedFMU with special_interface to evalaute single equations.

Keywords

- `modelName::String`: Name of Modelica model to export as FMU.
- `pathToFmu::String`: Path to FMU to extend.
- `pathToFmiHeader::String`: Path to FMI headers. They are part of this repository in FMI-Standard-2.0.3/headers.
- `eqIndices::Array{Int64}`: Array with equation indices to add equiation interface for.
- `tempDir::String`:

Returns

- Path to generated FMU `tempDir/<modelName>.interface.fmu`.

See also `profiling`, `generateFMU`, `generateTrainingData`.

source

`NonLinearSystemNeuralNetworkFMU.generateTrainingData` – Function.

```
generateTrainingData(fmuPath::String,
                     fname::String,
                     eqId::Int64,
                     inputVars::Array{String},
                     min::AbstractVector{<:Number}
                     max::AbstractVector{<:Number},
                     outputVars::Array{String};
                     N::Integer=1000)
```

Generate training data for given equation of FMU.

Generate random inputs between `min` and `max`, evalaute equation and compute output. All input-output pairs are saved in `fname`.

Arguments

- `fmuPath::String`: Path to FMU.
- `fname::String`: File name to save training data to.
- `eqId::Int64`: Index of equation to generate training data for.
- `inputVars::Array{String}`: Array with names of input variables.
- `min::AbstractVector{<:Number}`: Array with minimum value for each input variable.
- `max::AbstractVector{<:Number}`: Array with maximum value for each input variable.
- `outputVars::Array{String}`: Array with names of output variables.

Keywords

- `N::Integer = 1000`: Number of input-output pairs to generate.

See also `generateFMU`, `generateFMU`.

source

3.2 Examples

First we need to create a Model-Exchange 2.0 FMU with OpenModelica.

This can be done directly from OpenModelica or with `generateFMU`:

```
fmu = generateFMU(modelName = "simpleLoop",
                  pathToMo = "test/simpleLoop.mo",
                  pathToOmc = omc,
                  tempDir = "tempDir")
```

```
"tempDir/simpleLoop.fmu"
```

Next we need to add non-standard C function

```
fmi2Status myfmi2evaluateEq(fmi2Component c, const size_t eqNumber)
```

that will call `<modelname>_eqFunction_<eqIndex>(DATA* data, threadData_t *threadData)` for all non-linear equations we want to generate data for.

Using `addEqInterface2FMU` this C code will be generated and added to the FMU.

```
interfaceFmu = addEqInterface2FMU(modelName = "simpleLoop",
                                  pathToFmu = fmu,
                                  pathToFmiHeader = joinpath("FMI-Standard-2.0.3", "headers"),
                                  eqIndices = [14],
                                  tempDir = "tempDir")
```

```
"tempDir/simpleLoop.interface.fmu"
```

Now we can create evaluate equation 14 for random values and save the outputs to generate training data.

```
using CSV
using DataFrames
generateTrainingData(interfaceFmu,
                     "simpleLoop_data.csv",
                     14,
                     ["s", "r"],
                     [0.0, 0.95],
                     [1.5, 3.15],
                     ["y"];
                     N = 10)
df =  CSV.File("simpleLoop_data.csv")
```

```
10-element CSV.File:
 CSV.Row: (s = 0.6458724364830406, r = 1.6708754421371936, y = -0.5114890188473764)
 CSV.Row: (s = 1.2213200994258078, r = 2.1609630370877095, y = 0.5492242486441208)
 CSV.Row: (s = 0.023402225179326386, r = 1.2336024457632182, y = 0.8866037373144551)
 CSV.Row: (s = 0.10509568511475881, r = 3.05189019938354, y = 2.312415384362115)
 CSV.Row: (s = 0.9007999001291488, r = 2.0509603345846856, y = 2.041741963034728)
 CSV.Row: (s = 1.3370050009461372, r = 1.5197961938351419, y = 1.366215370048221)
 CSV.Row: (s = 0.34381367648894495, r = 2.325821507983602, y = 1.9950873322031204)
 CSV.Row: (s = 0.07274282269772464, r = 1.1521498358721378, y = 0.855519823029226)
 CSV.Row: (s = 0.35214327036874976, r = 2.5115850973159524, y = 2.162239912141065)
 CSV.Row: (s = 1.2223326262812095, r = 2.3811860769830897, y = 2.302132100125795)
```