Overview of the Repeatability Package for "RINO: Robust INner and Outer Approximated Reachability of Neural Networks Controlled Systems"

Eric Goubault and Sylvie Putot

LIX, Ecole Polytechnique, CNRS and Institut Polytechnique de Paris,
91128 Palaiseau, France
{Name.Surname}@polytechnique.edu

1 Introduction

This document provides an overview of the repeatability package for the paper "RINO: Robust INner and Outer Approximated Reachability of Neural Networks Controlled Systems". This paper presents a verification tool, called RINO, for computing over and under approximations of reachable sets, for neural net based controlled systems.

In summary, the package contains the following components:

- RINO: the tool we present in the related CAV paper[1]
- Verisig 2.0 and ReachNN*: the two tools we are using in the comparison evaluation
- 14 verification benchmarks, including dynamics models and neural network controllers, plus various configuration files for comparing the influence of verifiers' parameters on the analyses
- scripts to reproduce the computational components of the paper

2 Installation

The repeatability package is fully contained in a Docker image. This image uses the repeatability package of Verisig 2.0 presented with [7], so as to compare it with RINO.

The repeatability package can be found at

`https://github.com/cosynus-lix/RINO`

which contains the official distribution of RINO, together with the Docker file `Dockerfile-CAV22` that constitutes the repeatability package. Instructions for

[1] Although we improved since then the interpretation of trigonometric functions, explaining a better precision for `TORA` and `ACC`, that will be reflected in the final version of our paper.

building the container from the Docker file are given in Section 2.1. More general instructions and details about RINO are given on RINO's github page, in particular, a manual that is regularly updated[2] at:

`https://github.com/cosynus-lix/RINO/blob/master/Manual/Manual.pdf`

RINO is distributed under GNU Lesser General Public License v3.0, see the license file

`https://github.com/cosynus-lix/RINO/blob/master/LICENSE`

For those who prefer to use a pre-built Docker image (about 3.7Gb), it can be downloaded from:
`https://drive.google.com/file/d/10a-iyTACiRP6Mn4NF-9OQAf869ok3RH8/view?usp=sharing`
Corresponding instructions are given in Section 2.2.

2.1 Building and running the Docker image from RINO's git

Download the `github` repository (in fact, the code that has been tagged as the CAV 2022 repeatability version) by:

```
> git clone --depth 1 --branch CAV22_final_rep https://github.com/cosynus-lix/RINO
```

Get into the RINO directory and run:

```
> cd RINO
> docker build . -f Dockerfile-CAV22
```

An image shaxyz... is built which you can run by:

```
> docker run -it --name rino shaxyz....
```

```
--------                                  ---------------
___  __/__________________________________  ____/__  /________      __
__  /  _  _ \_  __ \_  ___/  __ \_  ___/_  /_   __  /_  __ \_ | /| / /
_  /   /  __/  / / /(__  )/ /_/ /  /    _  __/   _  / / /_/ /_ |/ |/ /
/_/    \___//_/ /_//____/ \____//_/     /_/      /_/  \____/____/|__/


WARNING: You are running this container as root, which can cause new files in
mounted volumes to be created as the root user on your host machine.

To avoid this, run the container by specifying your user's userid:

$ docker run -u $(id -u):$(id -g) args...

root@c19cb082c32e:/#
```

[2] So we advise to get it from the main branch in github, and not on the frozen distribution "CAV22_final_rep".

2.2 Loading and running the pre-built Docker image

Once you downloaded the pre-built Docker image `rino.tar.gz` from . Its SHA256 checksum is:

```
> shasum -a 256 rino.tar.gz
91b2df559c6ecf30199b2766ef3633fac01ef4aed258c2999814023755983222  rino.tar.gz
```

You can load it on the docker machine (running on your host) by:

```
> docker load -i rino.tar.gz
Loaded image ID: sha256:xxx
```

and then run it by (in which sha256:xxx has to be replaced by the name that was produced by the build command):

```
> docker run -it --name rino sha256:xxx
```

```
________                                _______________
___  __/__________________________________  ____/__  /________      __
__  /  _  _ \_  __ \_  ___/  __ \_  ___/_  /_   __  /_  __ \_ | /| / /
_  /   /  __/  / / /(__  )/ /_/ /  /    _  __/   _  / / /_/ /_ |/ |/ /
/_/    \___//_/ /_//____/ \____//_/     /_/      /_/  \____/____/|__/


WARNING: You are running this container as root, which can cause new files in
mounted volumes to be created as the root user on your host machine.

To avoid this, run the container by specifying your user's userid:

$ docker run -u $(id -u):$(id -g) args...

root@c19cb082c32e:/#
```

2.3 Organization of the container

You can then execute RINO from directory `/home/RINO` as indicated in Section 3. `Verisig 2.0` and `ReachNNStar` are in the CAV 2021 repeatability package from [7], within directory `/CAV21_repeatability_package`, their usage for comparison purpose is described in Sections 4.2 and 4.3.

3 RINO overview

As mentioned in the full paper, enclosed in the distribution, RINO implements all ideas presented in [4, 3, 5, 6] for the joint computation of inner and outer approximations of robustly reachable sets of differentiable nonlinear discrete-time ([3, 5]) or continuous-time systems ([4, 3]), possibly with constant delays

([6]). For experiments with systems without neural networks, we refer to the results presented in these works, obtained with a previous version of RINO. All details for running RINO on these systems, and not only the neural network analysis we present in the accompanying paper submitted to CAV, can be found on the corresponding github page: `https://github.com/cosynus-lix/RINO`.

RINO is written in C++. Intervals and zonotopes are used for set representation: the tool relies on the FILIB++ library [8] for interval computations and the aaflib library[3] for affine arithmetic [1]. Ole Stauning's FADBAD++ library[4] is used for automatic differentiation: its implementation with template enables us to easily evaluate the differentiation in the set representation of our choice (affine forms or zonotopes mostly). The tool takes as inputs:

- an open-loop or closed loop system, either discrete time or continuous-time, which for now is hard-coded in C++,
- an optional neural network, provided to the tool in a format directly inspired from the format analyzed by Sherlock [2], which can be used as some inputs of the closed-loop system,
- an optional configuration file to set initial values, input and disturbances ranges, and some parameter of the analysis (such as time step, order of Taylor expansion in time)

It computes inner and outer-approximations of the projection on each component of ranges, as well as joint 2D and 3D inner-approximations (provided as yaml file and Jupyter/python-produced figures). Additionally to the classical ranges, RINO computes approximations of output ranges that are reachable robustly or adversarially with respect to disturbances, specified as a subset of inputs. In the experiments presented herafter, we consider examples only of classical reachability, for which comparisons with existing work are available, but the extension to robust reachability based on our previous work is straightforward.

We apply for a "Artifact Reusable" badge as we believe that:

- It is made publicly available at `https://github.com/cosynus-lix/RINO`
- It is fully functional and all results of the accompanying paper can be checked, all examples (and more) can be run by RINO
- This can be done under docker, or by compiling it natively on your host machine: every detail is given again on the same github page
- RINO does much more than what is described in the accompanying paper (and related papers we mentionned above). You can also test new examples and modify all parameters through the configuration files, and by adding new plant models within the respective `ode_def.h`, `ode_def.cpp` for continuous-time systems, or `discrete_system.h`, `discrete_system.cpp` for discrete-time systems. The README.md file provides more details.

[3] `http://aaflib.sourceforge.net`
[4] `http://www.fadbad.com`

4 Reproducing Table 2 of the paper

In order to produce Table 2, we need to construct Tables 4, 6, and 8 reported in the appendix, which consists of the results of the 3 analysers, RINO, Verisig 2.0 and ReachNNStar, with respectively, integration steps of 0.05s and integration steps that were defined for Verisig 2.0 and ReachNNStar in the CAV 2021 repeatability of Verisig 2.0.

4.1 Reproducing RINO's results (Table 4)

All benchmarks of Table 4 can be run all together by:

```
cd /home/RINO
./script_RINO.sh
```

or it can be run step by step by copying the corresponding lines from this documentation.

The script `script_RINO.sh` is as follows:

```
#! /bin/sh
# TORA
./rino -configfile Examples/ConfigFiles/cfg_tora_tanh.txt
mv output output_tora_tanh
./rino -configfile Examples/ConfigFiles/cfg_tora_sigmoid.txt
mv output output_tora_sigmoid

# B1
./rino -configfile Examples/ConfigFiles/cfg_B1_tanh.txt
mv output output_B1_tanh
./rino -configfile Examples/ConfigFiles/cfg_B1_sigmoid.txt
mv output output_B1_sigmoid

# B2
./rino -configfile Examples/ConfigFiles/cfg_B2_sigmoid.txt
mv output output_B2_sigmoid

# B3
./rino -configfile Examples/ConfigFiles/cfg_B3_tanh.txt
mv output output_B3_tanh
./rino -configfile Examples/ConfigFiles/cfg_B3_sigmoid.txt
mv output output_B3_sigmoid

# B4:
./rino -configfile Examples/ConfigFiles/cfg_B4_tanh.txt
mv output output_B4_tanh
./rino -configfile Examples/ConfigFiles/cfg_B4_sigmoid.txt
mv output output_B4_sigmoid
```

```
# B5:
./rino -configfile Examples/ConfigFiles/cfg_B5_tanh.txt
mv output output_B5_tanh
./rino -configfile Examples/ConfigFiles/cfg_B5_sigmoid.txt
mv output output_B5_sigmoid

# ACC:
./rino -configfile Examples/ConfigFiles/cfg_acc_tanh.txt
mv output output_ACC_tanh

# Continuous-time Mountain Car
./rino -configfile Examples/ConfigFiles/cfg_MC_sigmoid.txt
mv output output_MC_sigmoid

# Discrete-time Mountain Car
./rino -configfile Examples/ConfigFiles/cfg_discrete_mc.txt
mv output output_discrete_mc
```

The config files indicate all options you can use during the analysis, as well as a pointer to the equations for the plant, and the neural network used as a controller, we refer the reader to the README section of RINO's github. For instance, the analysis for the continuous-time mountain car, as in the paper, is (as above):

```
# Continuous-time Mountain Car
> ./rino -configfile Examples/ConfigFiles/cfg_MC_sigmoid.txt
```

Each analysis produces a number of files in `/home/RINO/output`:

```
approxreachset.yaml  samplesreachset.yaml  sumup.txt  sumup.yaml
```

The results at final time can be read from files `sumup.txt` (and `sumup.yaml`, less human friendly, but with more decimals printed for lower and upper bounds of estimates for each variable):

```
> cd /home/RINO/output
# (change this to cd /home/RINO/output_MC_sigmoid if you have run script_RINO)
> more sumup.txt

Summary
Command-line   ./rino -configfile Examples/ConfigFiles/cfg_MC_sigmoid.txt

Systype  ode
Dynamics  451
Initial conditions  [-0.5, -0.48] [0, 0.001]
```

```
Inputs
Time interval  [0,75]
Control step  1
Time step  1
System dimension  2
Taylor Models order  3
Neural network file  Examples/Networks/MC\_sigmoid.sfx

Sampled estimate of final reachset  [-0.768, -0.661] [-0.018, -0.0108]
Over-approximation  [-0.782, -0.647] [-0.0194, -0.0094]
Under-approximation  [ EMPTY ] [ EMPTY ]

Elapsed analysis time (sec)  40.4141
Elapsed sampling time (sec)  1.01293

# more sumup.yaml
systype: 0
sysdim: 2
control_period: 1
dt: 1
nb_subdiv_init: 1
zouter:
  - -0.7819687873122952
  - -0.647042743625322
  - -0.01938709604483971
  - -0.009397465467241786
zinner:
  - nan
  - nan
  - nan
  - nan
elapsed-secs: 40.414087
elapsed-secs-sampling: 1.012927
```

At the console:

```
> ./rino -configfile Examples/ConfigFiles/cfg_discrete_mc.txt
****** Reading system choice from file Examples/ConfigFiles
/cfg_discrete_mc.txt ******
reading network fromExamples/Networks/MC_sigmoid.sfx
nb_inputs layer 0 = 2 nb_outputs = 200 activation function = 1
nb_inputs layer 1 = 200 nb_outputs = 200 activation function = 1
nb_inputs layer 2 = 200 nb_outputs = 1 activation function = 2
****** Reading initial conditions and parameters from file Examples
/ConfigFiles/cfg_discrete_mc.txt ******
initial_value=[-0.5, -0.48]
```

```
initial_value=[0, 0.001]
****** End initial conditions reading ******

rm: cannot remove 'output': No such file or directory
Estimate reachset:
Initial condition x: [-0.5, -0.48] [0, 0.001]
(...)
nncontrol[i]=[-0.421, 0.00231]
setting new bf at time 73
without quadrature: Xouter=z[0]=[-0.815, -0.584] z[1]=[-0.0242, -0.00655]
Xinner=z[0]=[ EMPTY ] z[1]=[ EMPTY ]
with intersection with direct solution: t=74
Xouter_maximal[0]=[-0.764338, -0.634861]  Xinner_maximal[0]=[nan, nan]
Sampled estim.[0]=[-0.751755, -0.647755]   eta_o[0]=0.803227  eta_i[0]=nan
gamma[0]=nan
Xouter_maximal[1]=[-0.020197, -0.0106099]  Xinner_maximal[1]=[nan, nan]
Sampled estim.[1]=[-0.0189066, -0.0118965]   eta_o[1]=0.731198  eta_i[1]=nan
gamma[1]=nan

(......)

Result files are in the output directory
Visualize them with by : cd GUI; python3 Visu_output.py --interactive=0
--printvar=all; cd ..
The python script will save files as png files (in same directory output)
-1-2-
cd GUI; python3 Visu_output.py --interactive=0 --printvar=all; cd ..
/home/RINO
```

4.2 Reproducing Verisig 2.0 results (Table 6)

We provide, still in directory `/home/RINO`, a script `script_Verisig2.sh`, that produces the results reported in Table 6:

```
# with the original time steps from CAV 2021

cd /CAV21_repeatability_package

./tmp/flowstar -p ./verisig_models/ex1_tanh/tanh20x20.yml
< ./verisig_models/ex1_tanh/ex1_tanh_tmp.model

./tmp/flowstar -p ./verisig_models/ex1_sig/sig20x20.yml
< ./verisig_models/ex1_sig/ex1_sig_tmp.model

./tmp/flowstar -p ./verisig_models/ex2_tanh/tanh20x20.yml
< ./verisig_models/ex2_tanh/ex2_tanh_tmp.model
```

```
./tmp/flowstar -p ./verisig_models/ex2_sig/sig20x20.yml
< ./verisig_models/ex2_sig/ex2_sig_tmp.model

./tmp/flowstar -p ./verisig_models/ex3_tanh/tanh20x20.yml
< ./verisig_models/ex3_tanh/ex3_tanh_tmp.model

./tmp/flowstar -p ./verisig_models/ex3_sig/sig20x20.yml
< ./verisig_models/ex3_sig/ex3_sig_tmp.model

./tmp/flowstar -p ./verisig_models/ex4_tanh/tanh20x20.yml
< ./verisig_models/ex4_tanh/ex4_tanh_tmp.model

./tmp/flowstar -p ./verisig_models/ex4_sig/sig20x20.yml
< ./verisig_models/ex4_sig/ex4_sig_tmp.model

./tmp/flowstar -p ./verisig_models/ex5_tanh/tanh100x100x100.yml
<  ./verisig_models/ex5_tanh/ex5_tanh_tmp.model

./tmp/flowstar -p ./verisig_models/ex5_sig/sig100x100x100.yml
< ./verisig_models/ex5_sig/ex5_sig_tmp.model

./tmp/flowstar -p ./verisig_models/tora_tanh/tanh20x20x20.yml
< ./verisig_models/tora_tanh/tora_tanh_tmp.model

./tmp/flowstar -p ./verisig_models/tora_sig/sig20x20x20.yml
< ./verisig_models/tora_sig/tora_sig_tmp.model

./tmp/flowstar -p ./verisig_models/acc/tanh20x20x20.yml
< ./verisig_models/acc/ACC_tanh_tmp.model
```

It can be run by:

```
> cd /home/RINO
> ./script_Verisig2.sh
```

or step by step by copying the corresponding lines from this documentation. More details about the usage of `Verisig 2.0` can be found in

`/CAV21_repeatability_package/UserManual.pdf`

and

`/CAV21_repeatability_package/documentation.pdf`

Output files are in `/CAV21_repeatability_package/outputs`. For instance, after analysing (first line above) TORA with tanh activation function (with a network of 3 layers of 20 neurons each), we get the file `tora_tanh_tmp.m`. Table 6 is built from the last values that are in this file (equivalently, these values can be found during execution of Verisig 2.0 at the console):

```
> tail tora_tanh_tmp.m
clear;
plot( [ 0.031943 , 0.025975 , 0.013349 , 0.005595 ,
0.005595 , 0.011563 , 0.024189 , 0.031943 , 0.031943 ] ,
[ -0.779824 , -0.773857 , -0.773857 , -0.781611 , -0.794606 ,
-0.800574 , -0.800574 , -0.792820 , -0.779824 ] ,
'color' , '[0 0.4 0]');
hold on;
clear;
plot( [ 0.031943 , 0.025975 , 0.013349 , 0.005595 ,
0.005595 , 0.011563 , 0.024189 , 0.031943 , 0.031943 ] ,
[ -0.779824 , -0.773857 , -0.773857 , -0.781611 , -0.794606 ,
-0.800574 , -0.800574 , -0.792820 , -0.779824 ] ,
'color' , '[0 0.4 0]');
hold on;
clear;
plot( [ 0.031943 , 0.025975 , 0.009582 , 0.001591 ,
0.001591 , 0.007607 , 0.023827 , 0.031943 , 0.031943 ] ,
[ -0.779824 , -0.773856 , -0.773856 , -0.781848 ,
-0.794920 , -0.800936 , -0.800936 , -0.792820 , -0.779824 ] ,
'color' , '[0 0.4 0]');
hold on;
clear;
```

For some strange reason, in the CAV 2021 repeatability file that we based our distribution of Verisig 2.0 on, `ex4_tanh` and `ex5_tanh` produce gnuplot files instead of matlab files (you have to change explicitely the corresponding flowstar file if you want to modify this), but you can retrieve in a similar manner the min and max values for each variable, looking at the last lines of the corresponding `.plt` files.

As described in the flowstar model file

`/CAV21_repeatability_package/verisig_models/tora_tanh/`

`tora_tanh_tmp.model`, line 11: `matlab octagon x1, x2`. This means that the coordinates that are plotted by verisig 2.0 are the ones for the first two variables x_1 and x_2.

From the last line of the file `tora_tanh_tmp.m` we get the min and max values for x_1 ([0.001591, 0.031943]) and for x_2 ([-0.800936, -0.773856]).

In order to get the two other ones, we should replace line 11 by `matlab octagon x3, x4` and rerun the analysis to get the min and max of the two other state variables. Note that the magnitudes of the variables are also shown during the analysis on the console.

The total time of computation, as indicated in Figure 6, is taken from the console output of each execution of Verisig 2.0. Note that we do not use multicore executions for fairness of comparison with RINO.

4.3 Reproducing ReachNNStar results (Table 8)

As indicated in
`/CAV21_repeatability_package/ReachNNStar/ReachNNStar_Usage.pdf`, the following commands run `ReachNN*` on examples `B1`, `B2`, `B3`, `B4`, `B5` and `TORA`, either with a network based on sigmoid or on tanh activation. For convenience, we did put all these commands in the script file `/home/RINO/script_ReachNNStar.sh`, but they can be run one at a time as well:

```
cd /CAV21_repeatability_package/ReachNNStar/ReachNN

./run_ex1_tanh.sh
./run_ex1_sig.sh
./run_ex2_tanh.sh
./run_ex2_sig.sh
./run_ex3_tanh.sh
./run_ex3_sig.sh
./run_ex4_tanh.sh
./run_ex4_sig.sh
./run_ex5_tanh.sh
./run_ex5_sig.sh
./run_tora_tanh.sh
./run_tora_sig.sh
```

More indications about `ReachNNStar` usage can be found at
`/CAV21_repeatability_package/ReachNNStar/ReachNNStar_Usage.pdf`.

Results of the analyses are in directory
`/CAV21_repeatability_package/ReachNNStar/ReachNN/outputs`
and contain, for each examples, a `.txt` file containing a abstract of the analysis results (in particular the time of the analysis we use in our comparison tables), and either a `.plt` or a `.m` file for producing figures (see Section 5.3):

```
> cd /CAV21_repeatability_package/ReachNNStar/ReachNN/outputs
> ls
nn_1_sigmoid.m    nn_2_sigmoid.m
nn_4_sigmoid.m    nn_5_sigmoid.m
nn_tora_sigmoid.m nn_1_sigmoid.txt
nn_2_sigmoid.txt  nn_4_sigmoid.txt
nn_5_sigmoid.txt  nn_tora_sigmoid.txt
nn_1_tanh.m       nn_2_tanh.m
nn_4_tanh.m       nn_5_tanh.m
nn_tora_tanh.m    nn_1_tanh.txt
nn_2_tanh.txt     nn_4_tanh.txt
nn_5_tanh.txt     nn_tora_tanh.txt
> more nn_1_tanh.txt
Verification result: No(35)
Max Error: 12.000000
Running Time: 15.000000 seconds
```

For some reason, the analysis of `nn_3` (that we could run independently) seems to crash in the docker version of the CAV 21 repeatability of Verisig 2.0, and does indeed crash within our docker version which builds on the CAV 21 Verisig 2.0 repeatability.

Beyond the running time, we read the min and max values of all variables at the last time step within the last lines of the `.m` files that are created:

```
> cd /CAV21_repeatability_package/ReachNNStar/ReachNN/outputs/
> tail nn_1_tanh.m .
clear;
plot( [ 9.336566e-01 , 9.291770e-01 , 8.203373e-01 , 8.121455e-01 ,
8.121455e-01 , 8.166251e-01 , 9.254648e-01 , 9.336566e-01 , 9.336566e-01 ] ,
[ 1.247732e+00 , 1.252212e+00 , 1.252212e+00 , 1.244020e+00 , 9.788182e-02 ,
9.340226e-02 , 9.340226e-02 , 1.015941e-01 , 1.247732e+00 ] , 'color' , '[0 0.4 0]');
hold on;
clear;
plot( [ 9.403955e-01 , 9.350552e-01 , 8.213966e-01 , 8.122410e-01 , 8.122410e-01 ,
8.175813e-01 , 9.312399e-01 , 9.403955e-01 , 9.403955e-01 ] , [
1.455116e+00 , 1.460456e+00 , 1.460456e+00 , 1.451301e+00 , -6.605215e-02 ,
-7.139239e-02 , -7.139239e-02 , -6.223684e-02 , 1.455116e+00 ] , 'color' , '[0 0.4 0]');
hold on;
clear;
plot( [ 9.483966e-01 , 9.421161e-01 , 8.215015e-01 , 8.113053e-01 , 8.113053e-01 ,
8.175858e-01 , 9.382004e-01 , 9.483966e-01 , 9.483966e-01 ] ,
[ 1.751739e+00 , 1.758019e+00 , 1.758019e+00 , 1.747823e+00 ,
-3.144514e-01 , -3.207320e-01 , -3.207320e-01 , -3.105357e-01 ,
1.751739e+00 ] , 'color' , '[0 0.4 0]');
hold on;
clear;
```

From which we read that $x_1 \in [8.113053e-01, 9.483966e-01]$ and $x_2 \in [-3.207320e-1, 1.758019]$.

5 Reproducing the figures and extra tables of the appendix (Tables 5, 7, 8)

5.1 Reproducing RINO's results for time step=0.01s (Table 5)

The corresponding script to be run is `/home/RINO/script_RINO_001.sh` which is as follows:

```
# Now with time step 0.01s

# TORA
./rino -configfile Examples/ConfigFiles/cfg_tora_tanh_001.txt
mv output output_tora_tanh_001
```

```
./rino -configfile Examples/ConfigFiles/cfg_tora_sigmoid_001.txt
mv output output_tora_sigmoid_001

# B1
./rino -configfile Examples/ConfigFiles/cfg_B1_tanh_001.txt
mv output output_B1_tanh_001
./rino -configfile Examples/ConfigFiles/cfg_B1_sigmoid_001.txt
mv output output_B1_sigmoid_001

# B2
./rino -configfile Examples/ConfigFiles/cfg_B2_sigmoid_001.txt
mv output output_B2_sigmoid_001

# B3
./rino -configfile Examples/ConfigFiles/cfg_B3_tanh_001.txt
mv output output_B3_tanh_001
./rino -configfile Examples/ConfigFiles/cfg_B3_sigmoid_001.txt
mv output output_B3_sigmoid_001

# B4:
./rino -configfile Examples/ConfigFiles/cfg_B4_tanh_001.txt
mv output output_B4_tanh_001
./rino -configfile Examples/ConfigFiles/cfg_B4_sigmoid_001.txt
mv output output_B4_sigmoid_001

# B5:
./rino -configfile Examples/ConfigFiles/cfg_B5_tanh_001.txt
mv output output_B5_tanh_001
./rino -configfile Examples/ConfigFiles/cfg_B5_sigmoid_001.txt
mv output output_B5_sigmoid_001

# ACC:
./rino -configfile Examples/ConfigFiles/cfg_acc_tanh_001.txt
mv output output_acc_tanh_001
```

Then the way to extract the relevant information from the output directory is the same as in Section 4.1.

5.2 Reproducing Verisig 2.0 results for time step=0.05s (Table 7)

The corresponding script to be run is `/home/RINO/script_Verisig2_005.sh` which is as follows:

```
# with time step of 0.05s

./tmp/flowstar -p ./verisig_models/ex1_tanh/tanh20x20.yml
```

```
< ./verisig_models/ex1_tanh/ex1_tanh_tmp_005.model

./tmp/flowstar -p ./verisig_models/ex1_sig/sig20x20.yml
< ./verisig_models/ex1_sig/ex1_sig_tmp_005.model

./tmp/flowstar -p ./verisig_models/ex2_tanh/tanh20x20.yml
< ./verisig_models/ex2_tanh/ex2_tanh_tmp_005.model

./tmp/flowstar -p ./verisig_models/ex2_sig/sig20x20.yml
< ./verisig_models/ex2_sig/ex2_sig_tmp_005.model

./tmp/flowstar -p ./verisig_models/ex3_tanh/tanh20x20.yml
< ./verisig_models/ex3_tanh/ex3_tanh_tmp_005.model

./tmp/flowstar -p ./verisig_models/ex3_sig/sig20x20.yml
< ./verisig_models/ex3_sig/ex3_sig_tmp_005.model

./tmp/flowstar -p ./verisig_models/ex4_tanh/tanh20x20.yml
< ./verisig_models/ex4_tanh/ex4_tanh_tmp_005.model

./tmp/flowstar -p ./verisig_models/ex4_sig/sig20x20.yml
< ./verisig_models/ex4_sig/ex4_sig_tmp_005.model

./tmp/flowstar -p ./verisig_models/ex5_tanh/tanh100x100x100.yml
< ./verisig_models/ex5_tanh/ex5_tanh_tmp_005.model

./tmp/flowstar -p ./verisig_models/ex5_sig/sig100x100x100.yml
< ./verisig_models/ex5_sig/ex5_sig_tmp_005.model

./tmp/flowstar -p ./verisig_models/tora_tanh/tanh20x20x20.yml
< ./verisig_models/tora_tanh/tora_tanh_tmp_005.model

./tmp/flowstar -p ./verisig_models/tora_sig/sig20x20x20.yml
< ./verisig_models/tora_sig/tora_sig_tmp_005.model

./tmp/flowstar -p ./verisig_models/acc/tanh20x20x20.yml
< ./verisig_models/acc/ACC_tanh_tmp_005.model
```

Then the way to extract the relevant information from the output directory is the same as in Section 4.2.

5.3 Reproducing Figures

RINO After running an example, all results are in the subdirectory "`output`" (moved to `output_$NAME_OF_EXAMPLE` by `script_RINO.sh`). The data are provided in the following files :

- `sumup.txt`: summary of configuration, running time and ranges at the final state of the analysis (part of this information can also be found with more significant digits in `sumup.yaml`)
- `samplesreachset.yaml`: sampled trajectories (used to assess accuracy of reachability results)
- `approxreachset.yaml`: over and under-approximated reachset (projected, robust, joint ranges) and accuracy measures (eta, gamma) at each time step.

A python visualization file `Visu_output.py` is available in the GUI directory, as explained in the `README` of RINO's distribution under github. The python script `Visu_output.py` will save files as `.png` files in the directory `$HOME/output`.

Under docker, it is rather difficult to ensure that the visualization on the host machine will go correctly without further machine-dependant local installations. We chose to turn off the automatic visualization in the configuration files for all examples of the paper, and rely on a local execution of a python script file on the host machine for generating the figures, as having a python installation working on the host machine is much easier to ensure than the link between docker and a X11 interface.

First, you have to make sure that you have an `output` directory under the `$HOME/RINO` directory on the host machine (not under the docker container):

```
> mkdir output
```

Then for instance for retrieving on the host machine the elements after analysis, needed for visualization on the host machine (here for instance, for the analysis of ACC with tanh activation)[5], you will need to type in a console on the host machine:

```
> cd $HOME/RINO/output
> docker cp rino:/home/RINO/output_ACC_tanh/approxreachset.yaml .
> docker cp rino:/home/RINO/output_ACC_tanh/samplesreachset.yaml .
> docker cp rino:/home/RINO/output_ACC_tanh/sumup.txt .
> docker cp rino:/home/RINO/output_ACC_tanh/sumup.yaml .
```

Then for visualization (make sure `python3` is installed on your host machine), you can e.g. write:

```
> cd ../GUI
> python3 Visu_output.py --interactive=0 --printvar=-1-2
> cd ../output
> ls
eta.png
gamma.png
x1.png
x1_max.png
```

[5] You may need to change in the path below to `approxreachset.yaml` if you did not use `script_RINO.sh output_"NAMEOFEXAMPLE"` to `output`

```
x1_max_sample.png
x1_sample.png
xi_max.png
xi_subplots_max.png
(...)
```

that will produce `.png` files only for variables `x[1]` and `x[2]`.

There are many more ways to visualize results, as described in the README file of RINO. In particular, for the k ranging from 1 to system dimension, the following results files print the projected ranges on dimension k as function of time:

- `xk_max.png` (e.g. `x1_max.png`) and `xk_max_sample.png`: the maximal inner and outer-approximations, with and without sampled trajectories

For any pair (k, l) we also print 2-dimensional projections :

- `xkxl.png`: maximal inner and outer-approximations of the joint range (xk, xl) as skewed boxes (see e.g. [3]),
- `xkxl_sample.png`: sampled trajectories for `(xk,xl)`,
- `xkxl_approx_sample.png`: on the same graph the inner and outer-approximations of the joint range `(xk,xl)` as skewed boxes and sampled trajectories,
- `xkxl_box_sample.png`: same as above but the approximations are printed as boxes (useful in a few cases where the skewed boxes have a bad behavior),
- `xkxl_finalstate.png`: box and skewed box inner and outer-approximations, and sampled points at the final state of the analysis, which we used in the appendix of our CAV paper.

Three-dimensional projections when relevant are also printed, only the corners of boxes are printed for more lisibility.

We also provide `xi_max.png` and `xi_subplots_min_max.png`: reachset for all variables on one graph `eta.png`, `gamma.png`: error measures (eta_o = (width of sampled set)/(width of over_approx) ; eta_i = (width of under_approx)/(width of sampled set); gamma = (width of under_approx)/(width of over_approx).

These different type of inner and outer approximations are those described in [4].

Note that the files produced can slightly vary depending on the system type (ode, dde, discrete-time) Also you need a version of python between 3.6 and 3.8, it was only tested with python version 3.8.8, and does not work with python 3.9. You may need to install `pyenv` on your host computer if you want to install one of these versions of python together with yours. In that case, you should also `matplotlib` and `pyyaml`, with, for instance, the following lines:

```
> pip3 install matplotlib
> pip3 install pyyaml
```

From there on, you can use your favorite png file viewer to reproduce the figures of the paper.

5.4 Verisig 2.0

The figures in the Appendix were retrieved using first the command (here, as an example, for the discrete mountain car example):

```
> docker cp rino:/CAV21_repeatability_package/outputs/MC_large_sig_tmp.m .
```

We then use `Matlab` (that is not provided in the repeatability package due to obvious license problems) to load the corresponding `.m` and run it, producing the corresponding picture. This, and more details on using Verisig 2.0 can be found in particular in Section 5 of `/CAV21_repeatability_package/documentation.pdf`.

5.5 ReachNNStar

As with the other tools, we first need to retrieve the plot files on the host machine:

```
docker cp rino:/CAV21_repeatability_package/ReachNNStar/ReachNN/outputs/nn_1_tanh.m .
```

We then use `Matlab` to load the corresponding `.m` and run it, producing the corresponding picture. This, and more details on using ReachNNStar can be found in particular in Section 5 of `/CAV21_repeatability_package/documentation.pdf`.

References

1. João Comba and Jorge Stolfi. Affine arithmetic and its applications to computer graphics. *In SIBGRAPI*, 1993.
2. S. Dutta, T. Kushner, S. Jha, S. Sankaranarayanana, N. Shankar, and A. Tiwari. Sherlock : A tool for verification of deep neural networks. 2019.
3. E. Goubault and S. Putot. Robust under-approximations and application to reachability of non-linear control systems with disturbances. *IEEE Control Systems Letters*, 4(4):928–933, 2020.
4. Eric Goubault and Sylvie Putot. Inner and outer reachability for the verification of control systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, pages 11–22. ACM, 2019.
5. Eric Goubault and Sylvie Putot. Tractable higher-order under-approximating AE extensions for non-linear systems. In Raphaël M. Jungers, Necmiye Ozay, and Alessandro Abate, editors, *ADHS 2021*, volume 54 of *IFAC-PapersOnLine*, pages 235–240. Elsevier, 2021.
6. Eric Goubault, Sylvie Putot, and Lorenz Sahlmann. Inner and Outer Approximating Flowpipes for Delay Differential Equations. In *Proceedings of the 30th International Conference on Computer Aided Verification, Part II*, pages 523–541. Springer, 2018.
7. Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *Computer Aided Verification*, pages 249–262. Springer International Publishing, 2021.
8. M. Lerch, G. Tischler, J. W. von Gudenberg, W. Hofschuster, and W. Kramer. filib++, a fast interval library supporting containment computations. *ACM Trans. Math. Soft.*, 2006.