Universidade Vigo

CITMAGA GALICIAN CENTRE FOR MATHEMATICAL RESEARCH AND TECHNOLOGY

STRIKE-GOLDD v4.2
User manual

Alejandro F. Villaverde

February 29, 2024

Contents

1 Introduction **1**
1.1 Theoretical foundations . . . 1
1.2 Version and publication history . . . 1
1.3 License . . . 2
1.4 Requirements, availability and installation . . . 2
1.5 Contact . . . 2

2 Quick start: using STRIKE-GOLDD in one minute **3**
2.1 Use as a MATLAB app . . . 3
2.2 Use as MATLAB scripts . . . 3

3 Software contents **5**

4 Usage **7**
4.1 Options . . . 7
4.1.1 Using other options files . . . 8
4.2 Input: entering a model . . . 9
4.2.1 Example: defining the MAPK model . . . 9
4.3 Analysing a model: known vs unknown inputs . . . 10
4.3.1 Example: two-compartment model with known input . . . 10
4.3.2 Example: two-compartment model with unknown input . . . 10
4.4 Searching for Lie symmetries . . . 10
4.5 Reparameterizing a model . . . 11
4.6 Output . . . 11
4.7 Analysing Accessibility and Controllability — NLcontrollability . . . 11

5 Acknowledgements **12**
5.1 Contributors . . . 12
5.2 Funding . . . 12

1 Introduction

STRIKE-GOLDD is a MATLAB toolbox that analyses a number of properties (local structural identifiability, observability, accessibility, and controllability) of nonlinear dynamic models, which can have multiple time-varying and possibly unknown inputs. It can also be used to find the symmetries in the model equations that lead to lack of identifiability or observability, and to automatically reparameterize the model in order to remove those symmetries.

1.1 Theoretical foundations

STRIKE-GOLDD adopts a differential geometry approach, recasting the identifiability problem as an observability problem. Essentially, the observability of the model variables (states, parameters, and inputs) is determined by calculating the rank of a generalized observability-identifiability matrix, which is built using Lie derivatives. When the matrix does *not* have full rank, there are some unobservable variables. If these variables are parameters, they are called (structurally) unidentifiable.

STRIKE-GOLDD determines the subset of identifiable parameters, observable states, and observable (also called reconstructible) inputs, thus performing a "Full Input-State-Parameter Observability" analysis (FISPO). STRIKE-GOLDD provides three different algorithms that perform this analysis:

- The FISPO algorithm presented in [10], which is implemented in the `STRIKE_GOLDD.m` file and can be applied to nonlinear models in general. Said procedure is directly applicable to many models of small and medium size. Larger systems can be analysed by building observability-identifiability matrices with a reduced number of Lie derivatives; in some cases this allows to determine the identifiability of every parameter in the model (complete case analysis); when such result cannot be achieved, at least partial results – i.e. identifiability of a subset of parameters – may be obtained.
- The ORC-DF algorithm [4] originally presented in [3], which can be used if the model is affine in the inputs.
- The Prob_Obs_Test algorithm originally presented in [7], which can be used if the model is rational. Certain non-rational models that can be automatically rewritten in rational form can also be analysed with this algorithm.

Furthermore, STRIKE-GOLDD can also perform certain analyses related to the existence of Lie symmetries in the model equations, which are a possible source of non-identifiability and non-observability:

- the Lie_symmetry function searches for the Lie symmetries that prevent a model from being FISPO.
- the AutoRepar function removes the symmetries found by Lie_symmetry procedure, thus reparameterizing the model so as to render it FISPO.

Since v4.1.0, STRIKE-GOLDD integrates the code of the NLcontrollability tool (https://github.com/afvillaverde/NLcontrollability), thus providing the following tests for accessibility and controllability:

- The Linearization Condition (LC) for controllability.
- Sussmann's General Sufficient Condition (GSC) for controllability.
- The Accessibility Rank Condition (ARC).
- The Lie Algebraic Rank Condition (LARC) for accessibility.

1.2 Version and publication history

List of main releases of STRIKE-GOLDD and their features:

- The first version of STRIKE-GOLDD was presented in [8].

- STRIKE-GOLDD v2.0 introduced the use of extended Lie derivatives [9]. This extension enabled the analysis of structural identifiability for *time-varying inputs* and, additionally, the characterization of the input profile required to make the parameters identifiable.
- STRIKE-GOLDD v2.1.1 incorporated the possibility of analysing models with *unknown time-varying inputs*, thus performing a FISPO analysis for the first time. The corresponding methodological details are given in [10].
- STRIKE-GOLDD v2.1.6 incorporated a procedure for finding *Lie symmetries* in a model, as well as for calculating the transformations that break them. The procedure is described in [6].
- STRIKE-GOLDD v2.2 incorporated the *ORC-DF* algorithm [3] for models that are affine in the inputs. A comparison between FISPO and ORC-DF is reported in [4]. STRIKE-GOLDD v2.2 also incorporated a multi-experiment analysis routine.
- STRIKE-GOLDD v3.0 incorporated an automatic reparameterization procedure, AutoRepar [5], which removes Lie symmetries to render the model FISPO.
- STRIKE-GOLDD v3.2 incorporated the Prob_Obs_Test algorithm [2], which is computationally more efficient than the FISPO algorithm and can be applied to rational models.
- STRIKE-GOLDD v4.0 incorporated a graphical user interface implemented as a MATLAB app.
- STRIKE-GOLDD v4.1 incorporated the NLcontrollability tool for accessibility and controllability analysis, which is described in [1].
- STRIKE-GOLDD v4.2 removed a number of outdated features.

1.3 License

STRIKE-GOLDD is licensed under the GNU General Public License version 3 (GPLv3), a free, copyleft license for software.

1.4 Requirements, availability and installation

STRIKE-GOLDD can run on any operating system. It only requires a computer with a MATLAB installation, including the Symbolic Math Toolbox. STRIKE-GOLDD has been tested with several MATLAB versions (R2017b–R2023b). Use of a recent version is recommended.

STRIKE-GOLDD can be obtained and installed in just two clicks:

1. Download STRIKE-GOLDD from https://github.com/afvillaverde/strike-goldd.
2. Unzip it.

1.5 Contact

STRIKE-GOLDD is maintained by Alejandro F. Villaverde (http://afvillaverde.webs.uvigo.gal/). For questions, send an email to afvillaverde@uvigo.gal.

2 Quick start: using STRIKE-GOLDD in one minute

To start using STRIKE-GOLDD, follow the installation instructions in Section 1.4, open a MATLAB session, and go to the STRIKE-GOLDD root directory ("STRIKE-GOLDD"). Then, choose one of the following ways[1]:

2.1 Use as a MATLAB app

1. Launch the MATLAB app (right-click on `STRIKE_GOLDD_APP.mlapp` → Run).
2. In the pop-up window, define the algorithm and model name (and, optionally, other options in their tabs).
   - QUICK DEMO EXAMPLE: select "C2M" in the "Model name" drop-down menu; STRIKE-GOLDD will analyse a two-compartment model with the FISPO algorithm and default options.
3. Press the "Run" button, and wait until the red sign turns green.

Done! Results will be reported in the MATLAB screen. A screenshot of the MATLAB app is shown in Fig. 1

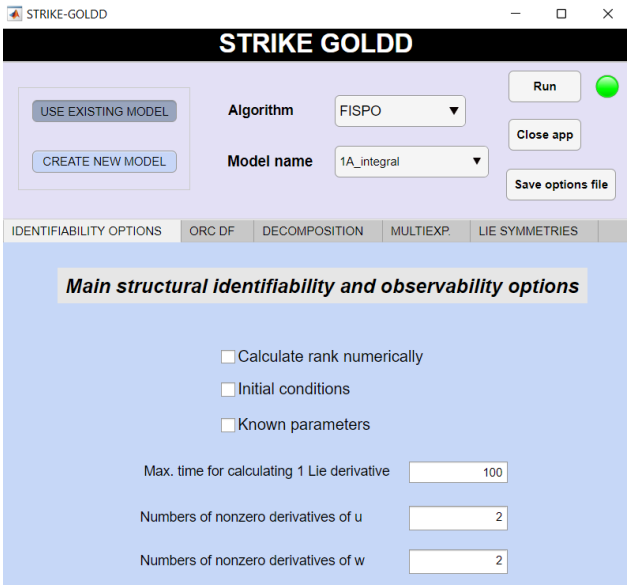


Figure 1: **Graphical interface of the STRIKE-GOLDD app**.

2.2 Use as MATLAB scripts

1. Run `install.m`.
2. Define the problem and options by editing the script `options.m` (see Section 4.2 for details).
   - QUICK DEMO EXAMPLE: If you leave `options.m` unedited, STRIKE-GOLDD will analyse a two-compartment model ("C2M") with the FISPO algorithm and default options.
3. Run `STRIKE_GOLDD.m` (to do this you can either type "STRIKE_GOLDD" in the command window, or right-click `STRIKE_GOLDD.m` in the "Current Directory" tab and select "Run").

Done! Results will be reported in the MATLAB screen. A screenshot of an execution is shown in Fig. 2.

[1]This section does not cover the accessibility and controllability tests provided with NLcontrollability. Information on how to perform them is given in Section 4.7

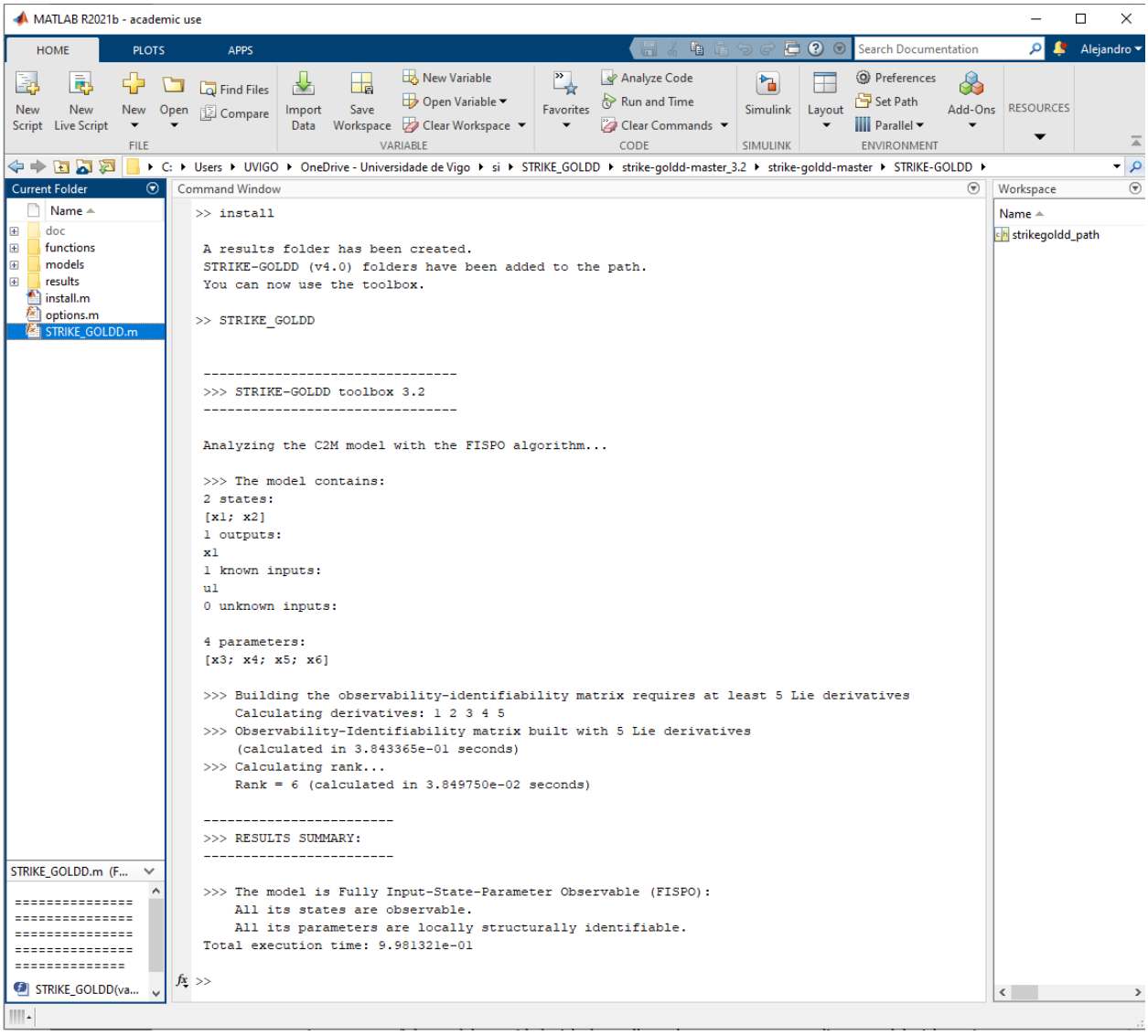


Figure 2: **Screen shot of a STRIKE-GOLDD run.**

3 Software contents

The STRIKE-GOLDD toolbox consists of several MATLAB files stored in several folders. One way of working with the toolbox is by executing the following file:

- `MainApp.mlapp`: MATLAB app; launching it executes STRIKE-GOLDD through a graphical user interface. Using this interface the user can specify the options and define new models.

Alternatively, it is possible to run STRIKE-GOLDD entirely by means of MATLAB scripts, without using the app. To this end the user may edit the following files:

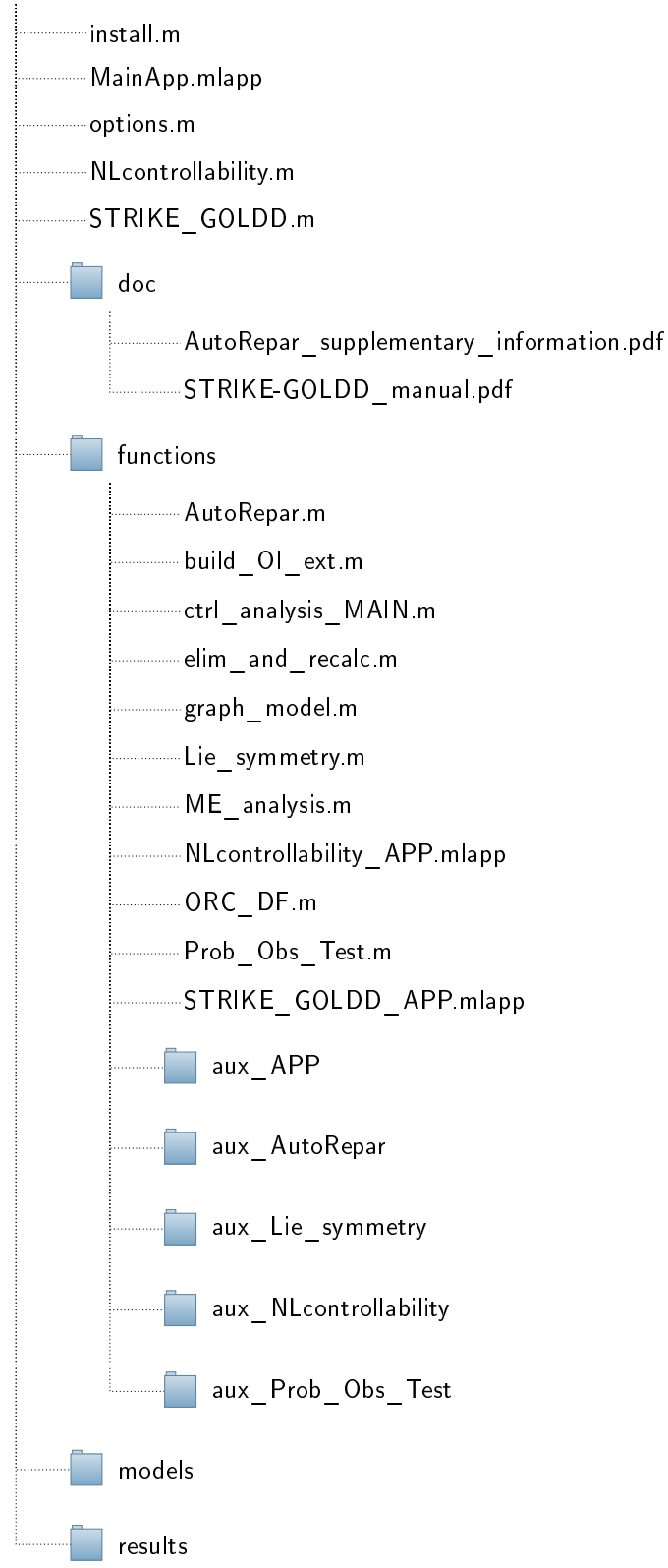
- `install.m`: installation file, which must be executed at the beginning of each session.
- `options.m`: file where the user enters the problem to solve and the options for solving it.
- `NLcontrollability.m`: main file for accessibility and controllability analyses.
- `STRIKE_GOLDD.m`: main file for structural identifiability and observability analyses, as well as for the related tasks (search for symmetries, reparameterization).

Additionally, a number of scripts are provided in the functions folder:

- `AutoRepar.m`: implements the AutoRepar method [5], which reparameterizes the model specified in the `options.m` file, removing any Lie symmetries found in order to make it fully identifiable and observable. This entails analysing the structural identifiability and observability of the model; if this has already been done, the results file can be specified in `options.m`.
- `build_OI_ext.m`: builds the generalized observability-identifiability matrix for a given number of Lie derivatives, which is passed as the argument. The resulting array is stored in a MAT file.
- `ctrl_analysis_MAIN.m` performs the accessibility and controllability analyses.
- `elim_and_recalc.m`: determines the identifiability of individual parameters one by one, by successive elimination of its column in the identifiability matrix and recalculation of its rank.
- `graph_model.m`: creates a graph showing the relations among model states, outputs, and parameters (i.e. which variables appear in which equations).
- `Lie_Symmetry.m`: searches for Lie symmetries using the method in [6]; if it finds symmetries admitted by the model, it calculates transformations of the variables in order to break them.
- `ME_analysis.m`: performs Multi-Experiment analysis; it modifies the model equations so that the analysis of the resulting model yields observability results from multiple experiments (by default, STRIKE-GOLDD considers a single experiment).
- `NLcontrollability.mlapp`: app that implements the analysis performed by the NLcontrollability script.
- `ORC_DF.m`: implements the ORC-DF algorithm [4] originally presented in [3], which can be used if the model is affine in the inputs.
- `prob_obs_test.m`: implements the Prob_Obs_Test algorithm originally presented in [7], which can be used if the model is rational.
- `STRIKE_GOLDD_APP.mlapp`: app that implements the analysis performed by the STRIKE-GOLDD script.

The folder tree structure of the toolbox files is as follows:

Root folder (/STRIKE-GOLDD/)



4 Usage

There are two ways of using the toolbox:

- By launching the **MATLAB app** `MainApp.mlapp` (introduced in STRIKE-GOLDD 4.0).
  In this way, the user can specify the options and define new models through a graphical interface.
- Entirely through **MATLAB scripts**, following three steps:
  1. **Run the installation file**, `install.m`.
  2. **Edit the options file**, `options.m`, to define the problem and the options for solving it.
     (You may also save your options file with a different name, e.g. `my_custom_options.m`)
  3. **Run the main file**: `STRIKE_GOLDD.m` for structural identifiability and observability analyses; or `NLcontrollability.m` for accessibility and controllability analyses.
     (If you wish to use STRIKE-GOLDD to analyse accessibility or controllability, jump to Section 4.7. For all other purposes, keep on reading.)
     Running `STRIKE_GOLDD` without arguments will use the options defined in `options.m`. To use the options defined in a different file, you must specify the file as an input argument to `STRIKE_GOLDD`, i.e. you must write in the command window something as e.g.:
     `STRIKE_GOLDD('my_custom_options.m')`

In the remainder of this section we describe the meaning of the available options, the model format, and the basic rationale behind the algorithms. We show how to use the toolbox using MATLAB scripts. Once the user is familiarised with these concepts, the use of the MATLAB app is self-explanatory.

4.1 Options

The model to analyse, as well as the options for performing the analysis, are entered in the `options.m` file. All options have default values that can be modified by the user. Their meaning is explained in the comments of the `options.m` file. In the `options.m` file the options are classified in the following blocks:

(1) **CHOOSE MODEL TO ANALYSE:** The first block consists of solely one option, the name of the model to analyse. By default it is set to one of the models provided with the toolbox, the two-compartment linear model with one input:

```
modelname = 'C2M';
```

The user may select other models provided with the toolbox – included in folder `/models` – or define a new model as explained in Section 4.2.

(2) **CHOOSE TYPE OF ANALYSIS:** In this block, the user can select among 5 analyses to perform:

```
opts.algorithm = 1;
```

Choose (1) for the FISPO algorithm, (2) for Prob_Obs_Test, (3) for ORC-DF, (4) for Lie_symmetry, and (5) for AutoRepar.

(3) **MAIN STRUCTURAL IDENTIFIABILITY & OBSERVABILITY (SIO) OPTIONS:** The third block consists of the following options:

`opts.maxLietime`, `opts.nnzDerU`, `opts.nnzDerW`, `prev_ident_pars`.
Their meaning is explained in the comments of the `options.m` file.

Note that, for models with several inputs, `opts.nnzDerU` and `opts.nnzDerW` must be row vectors with the same number of elements as inputs, e.g., for two inputs:

```
opts.nnzDerU = [0 1];
```

The optional vector `prev_ident_pars` is used for entering parameters that have already been classified as identifiable. This reduces the computational complexity of the calculations and may thus enable a deeper analysis, which can lead to more complete results. For example, if STRIKE-GOLDD has already determined that two parameters p_1 and p_2 are identifiable, we may enter:

```
syms p1 p2
prev_ident_pars = [p1 p2];
```

Otherwise, we must leave it blank:

```
prev_ident_pars = [];
```

This option can also be used to assume that some parameters are known, despite being entered as unknown in the model definition. This is useful to test what happens when fixing some parameters, without having to modify the model file.

(4) **OPTIONS FOR AFFINE SYSTEMS (apply only to the ORC-DF algorithm):** If the ORC-DF algorithm is selected *and* the model is indeed affine, it is possible to specify additional settings, including the use of parallelization.

(5) **MULTI-EXPERIMENT OPTIONS (apply to the 3 SIO algorithms):** If `opts.multiexp` is set to one, the observability of the model is analysed for multiple experiments. The number of experiments considered is set with `opts.multiexp_numexp`. For each experiment it is possible to specify different types of inputs.

(6) **LIE SYMMETRIES & REPARAMETERIZATION OPTIONS:** This block defines a number of options related to the search for Lie symmetries, and to the reparameterization of the model based on said symmetries: `opts.ansatz`, `opts.degree`, `opts.tmax`, and `opts.ode_n` tune the search for symmetries, while `opts.use_existing_results` and `opts.results_file` let the user specify if the structural identifiability and observability of the model has already been analysed.

(7) **PATHS:** These paths can be typically left unmodified.

(8) **DEPRECATED OPTIONS (FOR COMPATIBILITY WITH OLDER VERSIONS):** The final block lists a number of options that are no longer supported, but may be encountered in older versions.

4.1.1 Using other options files

It is possible to save additional options files and using them to perform analyses. This is useful for storing the options that were used for a particular analysis, and reusing them or editing with minimum effort. To this end, the user can tell STRIKE-GOLDD to use an options file different than `options.m`. This is done by passing the name of the options file as an argument, i.e.:

```
STRIKE_GOLDD('options_test1.m');
```

4.2 Input: entering a model

STRIKE-GOLDD reads models stored as MATLAB MAT-files (.mat). The model states, outputs, inputs, parameters, and dynamic equations must be defined as vectors of symbolic variables; the names of these vectors must follow the specific convention shown in Table 1.

Name	Reserved for:	Common mathematical notation:
x	state vector	$x(t)$
p	unknown parameter vector	θ
u	known input vector	$u(t)$
w	unknown input vector	$w(t)$
f	dynamic equations	$\dot{x}(t) = f(x(t), u(t), w(t), \theta)$
h	output function	$y = h(x(t), u(t), w(t), \theta)$

Table 1: **Reserved variable and function names.** Models analysed with STRIKE-GOLDD must follow the naming convention shown in the Table.

Note that **x, p, u, w, f, h** are reserved names, which *must be used* for the variable vectors and/or function vectors listed in Table 1 and *cannot be used* to name any other variables. However, it is possible to use variants of them to define the variables that are elements of those vectors, e.g. $x_1, x_2, p_{23}, xp, \ldots$

4.2.1 Example: defining the MAPK model

Here we illustrate how to define a model using the MAPK example included in the `models` folder. The file read by STRIKE-GOLDD is `MAPK.mat`. This file, which stores the model variables, can be created from the M-file `z_create_MAPK_model.m`. In the following lines we comment the different parts of the M-file, illustrating the process of defining a suitable model.

First, all the parameters, states, and any other entities (such as inputs or known constants) appearing in the model must be defined as symbolic variables:

```
syms k1 k2 k3 k4 k5 k6 ...
     ps1 ps2 ps3 ...
     s1t s2t s3t ...
     KK1 KK2 n1 n2 alpha  ...
```

Then we define the state variables, by creating a column vector named x:

```
x    = [ps1; ps2; ps3];
```

Similarly, we define the vector of output variables, which must be named h. In this case they coincide with the state variables:

```
h    = x;
```

Similarly, we define the known input vector, u, and the unknown input vector, w. If there are no inputs, enter blank vectors:

```
u  = [];
w  = [];
```

The vector of unknown parameters must be called p:

```
p    = [k1; k2; k3; k4; k5; k6;s1t; s2t; s3t; KK1; KK2; n1; n2; alpha];
```

The dynamic equations dx/dt must also be entered as a column vector, called f. It must have the same length as the state vector x:

```
f    = [k1*(s1t-ps1)*(KK1^n1)/(KK1^n1+ps3^n1)-k2*ps1;
           k3*(s2t-ps2)*ps1*(1+(alpha*ps3^n2)/(KK2^n2+ps3^n2))-k4*ps2 ;
           k5*(s3t-ps3)*ps2-k6*ps3 ];
```

Finally, save all the variables in a MAT-file:

```
save('MAPK','x','h','u','w','p','f');
```

> Note that most models provided with STRIKE-GOLDD define two vectors named `ics` and `known_ics`. Until STRIKE-GOLDD 4.1.0 these variables could be used to define specific initial conditions. This option was deprecated in v4.2, after it was found out that it could yield spurious results.

4.3 Analysing a model: known vs unknown inputs

The use of STRIKE-GOLDD for analysing a model was already illustrated in section 2. This section provides a few more details, basically about the use of models with known and unknown inputs.

4.3.1 Example: two-compartment model with known input

Section 2 showed how to analyse the default example, which is a two-compartment model with a known input, using default settings. By default, the option `opts.nnzDerU` is set to `opts.nnzDerU = 1` in the options file. This means that the model is analysed with exactly one non-zero derivative of the known input.

If we set `opts.nnzDerU = 0`, all input derivatives are set to zero. Running the two-compartment model example with this setting yields that the model is unidentifiable. Hence, this model requires a ramp or a higher-order polynomial input to be structurally identifiable and observable.

Note that for models with several inputs it is necessary to specify a vector, e.g. `opts.nnzDerU = [0 1]` for two inputs (or any other numbers, e.g. `opts.nnzDerU = [2 2]`).

4.3.2 Example: two-compartment model with unknown input

Let us now consider the two-compartment model with unknown input, and with the parameter b considered as known. This is already implemented in the model file `C2M_unknown_input_known_b` provided with the toolbox. The analysis of this model yields that all its parameters are structurally unidentifiable and its unmeasured state and input are unobservable. This is obtained for any choice of `opts.nnzDerW` (0, 1, 2, ...).

We now consider a version of this model in which both b and k_{1e} are considered known. This is implemented in the file `C2M_unknown_input_known_b_k1e`. In this case, the analysis yields that the model is fully observable (FISPO). This is obtained for any choice of `opts.nnzDerW` (0, 1, 2, 3, ...).

4.4 Searching for Lie symmetries

The existence of Lie symmetries amounts to lack of observability (in the generalized sense, that is, full input, state, and parameter observability, or 'FISPO'). Thus, determining whether a model admits a Lie group of transformations is a way of determining if it is observable and structurally identifiable.

The analysis of Lie symmetries can be performed by running the function `Lie_Symmetry`. It can be called in two different ways:

1. Without arguments, in which case it searches for symmetries in the model specified in the `options` file.
2. Specifying the model file as the argument, e.g.: `Lie_Symmetry('PK')`.

Furthermore, a number of options can be tuned. They are explained in the first lines of the script `Lie_Symmetry.m`. To modify them, directly edit the corresponding lines at the beginning of the script.

4.5 Reparameterizing a model

If a model is not fully identifiable or observable, the ideal solution is to reparameterize it so that it becomes FISPO. This can be done automatically with the `AutoRepar.m` function. By executing it, STRIKE-GOLDD performs a number of steps:

1. First it analyses the structural identifiability and observability of the model. If this analysis has already been done, this step may be skipped by setting `opts.use_existing_results = 1` in the options file, and specifying the mat-file that stores the results in `opts.results_file`.
2. Then, if the model is not FISPO, it searches for the Lie symmetries that cause this lack of identifiability (this step automatically calls `Lie_Symmetry.m`).
3. Finally, it uses the information provided by the Lie symmetries to introduce symmetry-breaking transformations that make the model FISPO.

In step 3 there may be several possible transformations. In this case, the toolbox lets the user decide which parameters or states can be transformed and which ones should remain intact if possible.

The AutoRepar methodology is described in [5], which provides several examples.

4.6 Output

STRIKE-GOLDD reports the main results of the analyses on screen. Additionally, it creates several MAT-files in the `results` folder:

- A file named `id_results_MODELNAME_DATE.mat`, with the results of the identifiability-observability analysis and most of the intermediate results. The main results are: `p_id` (list of identifiable parameters), `p_un` (unidentifiable parameters), `obs_states` (observable states), and `unobs_states` (unobservable states).
- One or several files named `obs_ident_matrix_MODEL_NUMBER_OF_Lie_deriv.mat`, with the generalized observability-identifiability matrices calculated with a given number of Lie derivatives. They are stored in separate files so that they can be reused in case a particular run is aborted due to excessive computation time.

4.7 Analysing Accessibility and Controllability — NLcontrollability

STRIKE-GOLDD was originally developed for analysing structural identifiability and observability. Hence, most of this manual is devoted to explaining its use for that purpose. However, since v4.1.0 it is also possible to perform accessibility and controllability analyses. To this end, follow these steps:

1. Run the installation file, `install.m`.
2. Edit `NLcontrollability.m` and run it .

When editing the script `NLcontrollability.m`, you must indicate the model that you want to analyse, and the analyses that you want to perform. Four different tests can be performed: the Linearization Condition (LC) for controllability, Sussmann's General Sufficient Condition (GSC) for controllability, the Accessibility Rank Condition (ARC), and the Lie Algebraic Rank Condition (LARC). The available options are explained in the comments of the script. The model should be created in the same format described in 4.2, although for the purpose of accessibility and controllability it is not necessary to define the output function.

5 Acknowledgements

5.1 Contributors

The development of STRIKE-GOLDD has been led by Alejandro F. Villaverde (Univ. Vigo, afvillaverde@uvigo.gal), who wrote the code up to version 2.1.1, and made later contributions. The code for finding Lie symmetries and reparameterizing the model based on them (AutoRepar) was written by Gemma Massonis Feixas (CSIC). The implementation of the ORC-DF algorithm and the automatic transformation for multi-experiment analysis was done by Nerea Martínez (Univ. Vigo & CSIC). The Prob_Obs_Test algorithm and the NLcontrollability tool were implemented by Sandra Díaz Seoane (Univ. Vigo). The graphical user interface was created by Xabier Rey Barreiro (Univ. Vigo). A number of collaborators have contributed to theoretical and/or application aspects: Antonio Barreiro Blas (Univ. Vigo), Antonis Papachristodoulou (Oxford Univ.), Neil D. Evans (Warwick Univ.), Michael J. Chappell (Warwick Univ.), Julio R. Banga (CSIC), and Nikolaos Tsiantis (CSIC). Alejandro F. Villaverde thanks Gleb Pogudin (École Polytechnique, Institut Polytechnique de Paris) for helpful suggestions.

5.2 Funding

STRIKE-GOLDD has received funding from the Xunta de Galicia, Consellería de Cultura, Educación e Universidade through the I2C postdoctoral program (fellowship ED481B2014/133-0) and through grant ED431F 2021/003; from the Spanish Ministry of Economy and Competitiveness (MINECO) through grants DPI2013-47100-C2-2-P and DPI2017-82896-C2-2-R; from the EPSRC projects EP/M002454/1 and EP/J012041/1; from the European Union's Horizon 2020 research and innovation programme under grant agreement No 686282 ("CANPATHPRO"); from the CSIC intramural project grant PIE 202070E062 ("MOEBIUS"); from grant RYC-2019-027537-I funded by MCIN/AEI/ 10.13039/501100011033 and by "ESF Investing in your future"; and from grant PID2020-113992RA-I00 funded by MCIN/AEI/ 10.13039/501100011033 (PREDYCT-BIO).

References

[1] Sandra Díaz-Seoane, Antonio Barreiro Blas, and Alejandro F Villaverde. Controllability and accessibility analysis of nonlinear biosystems. *Computer Methods and Programs in Biomedicine*, 242:107837, 2023.

[2] Sandra Díaz-Seoane, Xabier Rey Barreiro, and Alejandro F Villaverde. STRIKE-GOLDD 4.0: user-friendly, efficient analysis of structural identifiability and observability. *Bioinformatics*, 39(1):btac748, 2023.

[3] K Maes, MN Chatzis, and Geert Lombaert. Observability of nonlinear systems with unmeasured inputs. *Mechanical Systems and Signal Processing*, 130:378–394, 2019.

[4] Nerea Martínez and Alejandro F Villaverde. Nonlinear observability algorithms with known and unknown inputs: analysis and implementation. *Mathematics*, 8(11):1876, 2020.

[5] Gemma Massonis, Julio R Banga, and Alejandro F Villaverde. Autorepar: a method to obtain identifiable and observable reparameterizations of dynamic models with mechanistic insights. *International Journal of Robust and Nonlinear Control*, 33(9):5039–5057, 2023.

[6] Gemma Massonis and Alejandro F. Villaverde. Finding and breaking lie symmetries: implications for structural identifiability and observability in biological modelling. *Symmetry*, 12(3):469, 2020.

[7] Alexandre Sedoglavic. A probabilistic algorithm to test local algebraic observability in polynomial time. *Journal of Symbolic Computation*, 33:735–755, 2002.

[8] Alejandro F Villaverde, Antonio Barreiro, and Antonis Papachristodoulou. Structural identifiability of dynamic systems biology models. *PLoS Computational Biology*, 12(10):e1005153, 2016.

[9] Alejandro F Villaverde, Neil D Evans, Michael J Chappell, and Julio R Banga. Input-dependent structural identifiability of nonlinear systems. *IEEE Control Systems Letters*, 3(2):272–277, 2019.

[10] Alejandro F Villaverde, Nikolaos Tsiantis, and Julio R Banga. Full observability and estimation of unknown inputs, states and parameters of nonlinear biological models. *Journal of the Royal Society Interface*, 16(156):20190043, 2019.