Approximate Reductions of Rational Dynamical Systems in CLUE

Antonio Jiménez-Pastor[1], Alexander Leguizamon-Robayo[1(✉)], Max Tschaikowski[1], and Andrea Vandin[2,3]

[1] Aalborg University, Aalborg, Denmark
alexanderlr@cs.aau.dk
[2] Sant'Anna School of Advanced Studies, Pisa, Italy
[3] DTU Technical University of Denmark, Kongens Lyngby, Denmark

Abstract. In life sciences, deriving insights from dynamical systems can be challenging due to the large number of state variables involved. To address this, model reduction techniques can be used to project the system onto a lower-dimensional state space. CLUE is a tool that computes exact reductions for rational systems of ordinary differential equations. In this paper, we present an extension of CLUE to include approximate reductions which allow for larger aggregating power at the expense of a bounded error. Additionally, our extension includes new functionalities such as an interface to the model database ODEBase repository and simulation techniques for exploratory analyses.

Keywords: Approximate reduction · Dynamical systems · Constrained lumping

1 Introduction

Dynamical systems are used to model biochemical systems [9,29], performance models [12,31,32,34], electrical circuits [7] and even neural networks [14]. Model reduction is a family of techniques that that provides a modeler with a lower-dimensional dynamical system, while preserving properties of interest of the original system. In biology, e.g., it is key to preserve physical interpretability as these models validate mechanistic insights [2,6,22,26,28].

Lumping is a set of model reduction techniques for different mathematical formalisms [1,4,6,10,11], including ODEs. Given an original model, lumping produces a self-consistent system of ODEs involving macro-variables, each given in terms of combinations of the original ones [9,13,23,27]. In linear lumping, the reduction is a linear transformation of the original state variables. To avoid a complete loss of physical intelligibility, *constrained lumping* restricts the reduced state space so that previously defined linear combinations of state variables are preserved in the reduction [20].

R. Gori et al. (Eds.): CMSB 2024, LNBI 14971, pp. 108–116, 2024.
https://doi.org/10.1007/978-3-031-71671-3_9

The tool CLUE[1] was introduced along with a theoretical framework to efficiently compute *the smallest* constrained linear lumping for polynomial ODEs (i.e., ODEs with polynomial derivatives) that preserves the evolution of an arbitrary combination of state variables given by the user [24]. The efficiency of this framework with respect to previous work [19,20] stems from the fact that it does not rely on the symbolic computation of eigenvalues of a non-constant matrix. By leveraging this, the first version of CLUE (`v1.0`) was used to analyse models with several thousands of equations on standard hardware [24]. While it is possible to apply the polynomial theory to rational systems by means of symbolic computations, this approach quickly becomes computationally unfeasible [16]. This problem can be avoided by efficiently creating a representation of the dynamics by sampling it at different points using automatic differentiation [16], as implemented in the current version of CLUE (`v1.5`).

In this paper, we introduce a new version of CLUE (`v1.7`), extending [16, 24]. We add support for approximate reductions from [18] which have greater aggregation power at the expense of the exactness of the reduction. Moreover, we provide importing support from the public online repository ODEBase [21], as well as analysis capabilities by means of model simulation for exploratory purposes. Last but not least, we present the updated architecture of the new version of CLUE, and illustrate how the tool can be applied on a concrete model from the literature. The presented version of CLUE is publicly available and can be installed via `pip` with the following command

```
$ pip install git+https://github.com/clue-developers/CLUE@v1.7
```

2 Preliminaries

Constrained lumping is a reduction method that allows to reduce an ODE system in a way that it preserves a given linear combination of variables of interest. Suppose we are interested in the evolution of $x_{obs} = x_2 + 2x_3$ for the following system of ODEs

$$\dot{x}_1 = \frac{x_2^2 + 4x_2x_3 + 4x_3^2}{x_1^2 + 1}, \qquad \dot{x}_2 = \frac{2x_1 - 4x_3}{x_2 + 2x_3 + 1}, \qquad \dot{x}_3 = \frac{-x_1 - x_2}{x_2 + 2x_3 + 1}. \quad (1)$$

The matrix $L = (1\ 0\ 0, 0\ 1\ 2)^T$ is an *exact constrained lumping* of dimension 2, since it allows the construction a smaller self-consistent system in terms of the two *macro-variables* $y_1 = x_1$ and $y_2 = x_2 + 2x_3$, given by $(\dot{y}_1, \dot{y}_2)^T = (y_2^2/(y_1^2 + 1), -2y_2/(y_2 + 1))^T$.

The matrix L is a *constrained* lumping since the evolution of x_{obs} can be directly recovered from the evolution of y_2. It is *exact*, as the evolution of x_{obs} can be recovered exactly without any errors by using the lower dimensional system. In general, given a system of ODEs $\dot{x} = f(x)$ and an exact lumping L, a self-consistent reduced system is given by $y = Lf(\bar{L}y)$, where $y = Lx$, and $\bar{L}$ is a right-pseudo inverse of L. In this case, $\bar{L} = (1\ 0, 0\ 0.2, 0\ 0.4)^T$.

[1] https://github.com/clue-developers/CLUE.

Now, suppose that the evolution of x_{obs} is described by a slightly modified system with $4.05x_2x_3$ in the numerator of the first equation:

$$\dot{x}_1 = \frac{x_2^2 + 4.05x_2x_3 + 4x_3^2}{x_1^2 + 1}, \quad \dot{x}_2 = \frac{2x_1 - 4x_3}{x_2 + 2x_3 + 1}, \quad \dot{x}_3 = \frac{-x_1 - x_2}{x_2 + 2x_3 + 1}. \tag{2}$$

In this case, it is not possible to construct an exact approximate constrained lumping. However, by relaxing the conditions for lumping, the matrix $L = (1\ 0\ 0, 0\ 1\ 2)^T$ can be used to compute a smaller system that preserves x_{obs} up to an error, given by $(\dot{y}_1, \dot{y}_2)^T = (1.004y_2^2/(y_1^2 + 1), -2y_2/(y_2 + 1))^T$.

This means that the evolution of x_{obs} will be recovered up to an error. Figure 1 shows the evolution of x_{obs} computed via the original and reduced system for initial conditions $x = (1, 1, 1)^T$. Note that, by using an approximate reduction, it is possible to reduce a system that was not exactly reducible while still obtaining a simulation with low error. The detailed theory of approximate constrained lumping for polynomial systems, including the algorithm to compute them and how to bound the introduced errors, can be found in [18].

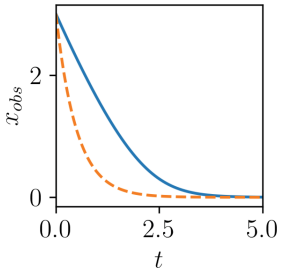

Fig. 1. x_{obs} in original (blue) and reduced (orange) model. (Color figure online)

3 Implementation

CLUE is designed as an open source python library to boost model analysis and exploration. Its architecture, summarized in Fig. 2, has been planned for efficient interaction and data handling through three main components: *model input*, *core functionalities*, and *outputs*. The *model input* component handles the steps of model acquisition and processing, while the *core functionalities* component, instead, contains all the logic to apply model reduction or approximation onto the models. The *outputs* component conducts simulations and allows for preliminary data analysis. We now proceed to explain each component in detail.

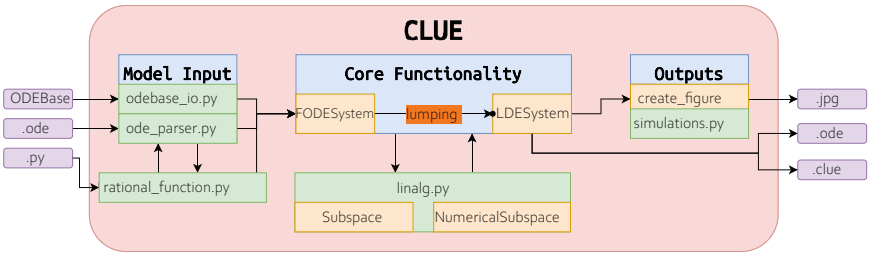

Fig. 2. Architecture of CLUE (red). The arrows indicate data flow. The main components of CLUE (blue) are composed of modules (green) of which the main classes and methods (e.g., `lumping`) are reported (orange) .External files and sources are displayed in purple. (Color figure online)

Model Input. The main goal of this component is to construct instances of the `FODEsystem` class from the supported sources: ODEBase [21], whose support has been added in this version of the tool, and ERODE [8] `.ode` files, and systems of ODEs symbolically written using `sympy`. The modules realizing the import functionalities of aforementioned formats are, respectively, `odebase_io.py`, `ode_parser.py` and `rational_function.py`. Regardless of the format, the creation of `FODESystem` instances relies on the efficient representation of polynomials and rational functions provided in the module `rational_function.py`.

Core Functionalities. The core functionalities of CLUE are available in the module `clue.py`. In this module, the main class is `FODEsystem` which contains all necessary information to represent a model, e.g., equations, observables, parameters and initial conditions. Simulations can be computed via the `simulate` method. Additionally, `FODESystem` offers export functionality to `.ode` files and to serialized `.clue` files.

Given an observable, a constrained lumping is computed by finding the smallest invariant subspace from which the evolution of the observable can be recovered. Following the theory presented in [16,18], lumpings are computed using the `lumping` and `app_lumping` methods which find exact and approximate lumpings, respectively. The outputs of these methods are instances of `LDESystem`. This class inherits from `FODEsystem`, while including lumping information, e.g., the lumping subspace and mappings from the original to the lumped variables.

Exact lumping subspaces are instances of the `Subspace` class, which stores them as matrices in row-echelon form. In contrast, approximate lumping subspaces are instances of the `NumericalSubspace` class, which stores them as orthonormal matrices. Both these classes are described in the module `linalg.py`. It should be noted that all exact computations are carried out using the rational numbers implementation provided by `sympy`. For both numerical and exact computations, we use our own implementation of matrix arithmetic. Matrices are stored as hash tables where the keys are the nonzero rows and the values are `SparseVectors` representing the actual row. Instances of `SparseVector` are hash tables storing the number of the nonzero value as keys and the actual vector values as values.

Outputs. Utilities to handle simulations are provided in the `simulations.py` module. Basic manipulation of simulations and data is supported. This corresponds to merging, comparing and applying matrices to simulation results. Similarly, basic plot functionality is supported, namely, exporting plots (e.g., Fig. 1), and simulation data (as CSV files).

4 Illustration of Model Workflow Using CLUE

In this section, we show an example workflow displaying the main features of CLUE on a simple, yet informative, model from the literature. A full listing of

this section is available as a Jupyter notebook[2]. Suppose we want to study the *signaling mechanism in apoptosis*, as described by the kinetic model in [17].

Step 1: Model Retrieval. To access the required model from ODEBase, we use the following line

```
>>> model = ode_scrapper(name="BIOMD0000000102")
>>> model = model.remove_parameters_ic()
```

In general, CLUE treats parameters as variables of the differential system. We can replace each known parameter by its value using the method `remove_parameters_ic`.

Step 2: Model Validation. Having retrieved the model, we can examine its attributes:

```
>>> print(model.name)        # Outputs the name of the model
>>> print(model.size)        # Outputs the size of the model
>>> print(model.equations)   # Shows the model equations
```

In particular, we infer that the original model size is 13.

Step 3: Analyses and Lumping. Following the original paper presenting the model [17], we are interested in studying the evolution of the observable C3, formally given by the variable `x7` in the model. As a first step, we compute the respective exact lumping which has `x7` as observable:

```
>>> exact_lump = model.lumping(['x7'])
```

This command raises a warning that the system could not be reduced.

```
[lumping] Warning: lumped size (13)
and original size (13) are the same.
```

This means that it was impossible to find an exact reduction.

We can use the approximate lumping functionality of CLUE to increase the aggregation power with the following command:

```
>>> app_lump_1 = model.app_lumping(['x7'])
```

This command relaxes the conditions of exact lumping until it finds a reduction that is smaller than the exact one. In this case, the resulting reduction is of size 12. To find more compact reductions, it is possible to add a size limit as follows:

```
>>> app_lump_2 = model.app_lumping(['x7'], max_size=10)
```

The output will be the largest reduction possible having at most 10 lumped variables (species), following the approach of [18]. The resulting lumped model in this case is of size 9.

[2] https://github.com/clue-developers/CLUE/tree/v1.7/notebooks/ODEBase_example.ipynb.

Step 4: Simulating and Comparing Models. To evaluate the effectiveness of the lumpings, we can simulate the original model and its reduction with the initial conditions and the time horizon from the original paper [17].

```
>>> exact_sim = model.simulate(0, 2500, view=["x7"])
>>> app_sim_1 = app_lump_1.simulate(0, 2500,
                        view=app_lump_1.observe(["x7"]))
>>> app_sim_2 = app_lump_2.simulate(0, 2500,
                        view=app_lump_2.observe(["x7"])
```

It is possible to choose the observable of interest with the third argument in the `simulate` method. The method `observe` computes the value of a given observable based on the lumped variables. We now merge all simulations into one.

```
>>> merged = merge_simulations(exact_sim,app_sim_1,app_sim_2)
```

Step 5: Visualizing the Results. To visualize the merged results (Fig. 3), we use the following line.

```
>>> create_figure(merged,names=['Exact','App_12','App_9'])
```

Figure 3 shows that it was possible to find further reductions beyond the exact one. We have two exemplary situations: on one hand, the reduction with 12 species provides a small error. On the other hand, the one with 9 is too aggressive, being close to the original only for a few units of time. This example shows that approximate reductions provide a trade-off between the aggressiveness of the reduction and the error of the reduced model. Depending on the modeller's goals, any of the reduced models can be further analyzed in Python or ERODE [8].

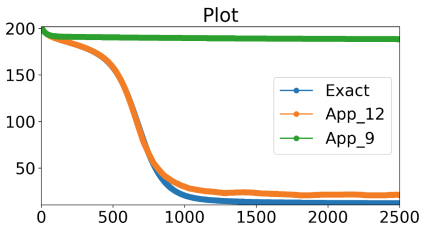

Fig. 3. Time evolution of C3 using approximate constrained lumping.

5 Conclusion

We presented an extension of the tool CLUE [16,24] to support approximate reductions. The underlying theory increases the aggregation power of exact lumping approaches [3,15,30,33] by relaxing the exactness criteria of aggregations. The extension also included simulation functionalities which allow for the

exploration of different model reductions, and importing capabilities for popular online model repositories. Future work will extend the functionalities of the CLUE tool to include analytic functions and to support importing and exporting other formats such as `SBML` [25] and BioNetGen [5].

Acknowledgement. This work was partially supported by Poul Due Jensen Foundation grant no. 883901, the Villum Investigator Grant S4OS, the Fsc regional Tuscan project AUTOXAI2 J53D21003810008, the project SERICS (PE00000014) and the Tuscany Health Ecosystem (THE) project with CUP B83C22003920001, under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

1. Abate, A., Andriushchenko, R., Ceska, M., Kwiatkowska, M.: Adaptive formal approximations of Markov chains. Perf. Eval. **148**, 102207 (2021)
2. Apri, M., de Gee, M., Molenaar, J.: Complexity reduction preserving dynamical behavior of biochemical networks. J. Theor. Biol. **304**, 16–26 (2012)
3. Bacci, G., Bacci, G., Larsen, K.G., Mardare, R.: On-the-fly exact computation of bisimilarity distances. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 1–15. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_1
4. Beica, A., Feret, J., Petrov, T.: Tropical abstraction of biochemical reaction networks with guarantees. Electron. Notes Theor. Comput. Sci. **350**, 3–32 (2020)
5. Blinov, M.L., Faeder, J.R., Goldstein, B., Hlavacek, W.S.: BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Bioinformatics **20**(17), 3289–3291 (2004)
6. Cardelli, L., Perez-Verona, I.C., Tribastone, M., Tschaikowski, M., Vandin, A., Waizmann, T.: Exact maximal reduction of stochastic reaction networks by species lumping. Bioinformatics **37**(15), 2175–2182 (2021)
7. Cardelli, L., Tribastone, M., Tschaikowski, M.: From electric circuits to chemical networks. Nat. Comput. **19**(1), 237–248 (2020)
8. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: ERODE: a tool for the evaluation and reduction of ordinary differential equations. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 310–328. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_19
9. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Maximal aggregation of polynomial dynamical systems. PNAS **114**(38), 10029–10034 (2017)
10. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Syntactic Markovian bisimulation for chemical reaction networks (2017)
11. Feret, J., Danos, V., Krivine, J., Harmer, R., Fontana, W.: Internal coarse-graining of molecular systems. Proc. Natl. Acad. Sci. **106**(16), 6453–6458 (2009)
12. Gast, N., Bortolussi, L., Tribastone, M.: Size expansions of mean field approximation: transient and steady-state analysis. ACM SIGMETRICS Perf. Eval. Rev. **46**(3), 25–26 (2019)
13. Großmann, G., Kyriakopoulos, C., Bortolussi, L., Wolf, V.: Lumping the approximate master equation for multistate processes on complex networks. In: McIver, A., Horváth, A. (eds.) QEST, vol. 11024, pp. 157–172 (2018)

14. Hasani, R., et al.: Closed-form continuous-depth models. arXiv preprint arXiv: 2106.13898 (2021)
15. Hillston, J., Tribastone, M., Gilmore, S.: Stochastic process algebras: from individuals to populations. Comput. J. **55**(7), 866–881 (2011)
16. Antonio, J.-P., Jacob, J.P., Pogudin, G.: Exact linear reduction for rational dynamical systems. In: Petre, I., Paun, A. (eds.) CMSB 2022. LNCS, vol. 13447, pp. 198–216. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-15034-0_10
17. Legewie, S., Blüthgen, N., Herzel, H.: Mathematical modeling identifies inhibitors of apoptosis as mediators of positive feedback and bistability. PLoS Comput. Biol. **2**(9), e120 (2006)
18. Leguizamon-Robayo, A., Jiménez-Pastor, A., Tribastone, M., Tschaikowski, M., Vandin, A.: Approximate constrained lumping of polynomial differential equations. In: CMSB, pp. 106–123 (2023)
19. Li, G., Rabitz, H.: A general analysis of exact lumping in chemical kinetics. Chem. Eng. Sci. **44**(6), 1413–1430 (1989)
20. Li, G., Rabitz, H.: New approaches to determination of constrained lumping schemes for a reaction system in the whole composition space. Chem. Eng. Sci. **46**(1), 95–111 (1991)
21. Lüders, C., Sturm, T., Radulescu, O.: ODEbase: a repository of ODE systems for systems biology. Bioinf. Adv. **2**(1), vbac027 (2022)
22. Maus, C., Rybacki, S., Uhrmacher, A.M.: Rule-based multi-level modeling of cell biological systems. BMC Syst. Biol. **5**, 1–20 (2011)
23. Okino, M., Mavrovouniotis, M.: Simplification of mathematical models of chemical reaction systems. Chem. Rev. **2**(98), 391–408 (1998)
24. Ovchinnikov, A., Pérez Verona, I., Pogudin, G., Tribastone, M.: CLUE: exact maximal reduction of kinetic models by constrained lumping of differential equations. Bioinformatics **37**(12), 1732–1738 (2021)
25. Pérez-Verona, I.C., Tribastone, M., Vandin, A.: A large-scale assessment of exact model reduction in the biomodels repository. In: Bortolussi, L., Sanguinetti, G. (eds.) CMSB 2019. LNCS, vol. 11773, pp. 248–265. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31304-3_13
26. Schmidt, H., Madsen, M., Danø, S., Cedersund, G.: Complexity reduction of biochemical rate expressions. Bioinformatics **24**(6), 848–854 (2008)
27. Snowden, T., van der Graaf, P., Tindall, M.: Methods of model reduction for large-scale biological systems: a survey of current methods and trends. Bull. Math. Biol. **79**(7), 1449–1486 (2017)
28. Sunnaker, M., Cedersund, G., Jirstrand, M.: A method for zooming of nonlinear models of biochemical systems. BMC Syst. Biol. **5**(1), 140 (2011)
29. Tognazzi, S., Tribastone, M., Tschaikowski, M., Vandin, A.: EGAC: a genetic algorithm to compare chemical reaction networks. In: Genetic and Evolutionary Computation Conference, GECCO, pp. 833–840 (2017)
30. Tribastone, M.: Behavioral relations in a process algebra for variants. In: Gnesi, S., Fantechi, A., Heymans, P., Rubin, J., Czarnecki, K., Dhungana, D. (eds.) SPLC, pp. 82–91. ACM (2014)
31. Tribastone, M., Mayer, P., Wirsing, M.: Performance prediction of service-oriented systems with layered queueing networks. In: International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA), pp. 51–65 (2010)
32. Tschaikowski, M., Tribastone, M.: Tackling continuous state-space explosion in a Markovian process algebra. Theoret. Comput. Sci. **517**, 1–33 (2014)

33. Tschaikowski, M., Tribastone, M.: Spatial fluid limits for stochastic mobile networks. Perf. Eval. **109**, 52–76 (2017)
34. Wirsing, M., et al.: Sensoria patterns: augmenting service engineering with formal analysis, transformation and dynamicity. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation, pp. 170–190 (2008)