Learning Dynamical Systems Using Standard Symbolic Regression

Sébastien Gaucel[1], Maarten Keijzer[2], Evelyne Lutton[1], and Alberto Tonda[1]

[1] INRA UMR 782 GMPA,1 Av. Brétignières, 78850, Thiverval-Grignon, France
{sebastien.gaucel,evelyne.lutton,alberto.tonda}@grignon.inra.fr
[2] Pegasystems Inc., Utrecht Area, Netherlands
maarten.keijzer@pega.com

Abstract. Symbolic regression has many successful applications in learning free-form regular equations from data. Trying to apply the same approach to differential equations is the logical next step: so far, however, results have not matched the quality obtained with regular equations, mainly due to additional constraints and dependencies between variables that make the problem extremely hard to tackle. In this paper we propose a new approach to dynamic systems learning. Symbolic regression is used to obtain a set of first-order Eulerian approximations of differential equations, and mathematical properties of the approximation are then exploited to reconstruct the original differential equations. Advantages of this technique include the de-coupling of systems of differential equations, that can now be learned independently; the possibility of exploiting established techniques for standard symbolic regression, after trivial operations on the original dataset; and the substantial reduction of computational effort, when compared to existing ad-hoc solutions for the same purpose. Experimental results show the efficacy of the proposed approach on an instance of the Lotka-Volterra model.

Keywords: Differential Equations, Dynamic Systems, Evolutionary Algorithms, Genetic Programming, Symbolic Regression

1 Introduction

In recent years, Genetic Programming (GP) gained popularity as an effective optimization technique [1], and its capabilities of automatically uncovering hidden relationships in datasets and producing rules to control complex systems haves been proved in several real-world applications [2] [3].

Differential equations are mathematical equations for an unknown function of one or several variables that relates the values of the function itself and its derivatives of various orders: they play a prominent role in engineering, physics, economics, biology, and other disciplines.

The idea of using symbolic regression to learn differential equations is present since the beginnings of GP [4]: given the great interest towards this topic, several

All authors contributed equally and their names are presented in alphabetical order.

research lines have followed. Babovic and Keijzer [5] propose a dimensionally-aware GP to learn dynamic systems in hydraulic engineering. Cao et al. [6] present a GP-based technique where an individual is a set of trees, representing a system of equations. Coefficients of the equations are optimized via a Genetic Algorithm, then the system is solved through a numerical integration method and the resulting equations are finally evaluated against training data. Iba [7] proposes an improvement over the previous approach, where coefficients are optimized through a least mean square technique, and a Runge-Kutta method of 4th order is used to build a solution. Bernardino and Barbos [8] use Grammar-Based Immune Programming to tackle the problem. It is important to notice that, while quite effective, all these concepts rely upon the use of ad-hoc individual construction, and significant computational costs to first solve the candidate equations and then compare them to experimental data.

We investigate a novel methodology for learning ordinary differential equations (ODE) through symbolic regression, whose original idea stems from an invited talk given by Maarten Keijzer during the GECCO conference in 2013 [9]. Given a system of ODEs, we show how the problem can be reduced to finding the first-order approximation of each ODE.We then apply the subsequent steps:

1. For each equation, standard symbolic regression is used to obtain a small group of candidate solutions that represent a trade-off between complexity and fitting;
2. A simple derivation procedure, following the properties of the first-order approximation of an ODE, is applied to each candidate solution, transforming them in ODEs;
3. Finally, corresponding equations are coupled in systems and examined with respect to dynamical behavior and fitting on the original data. The best system is returned to the user as the solution for the original problem.

Important advantages of our method are the possibility of learning differential equations using established symbolic regression techniques, instead of devising ad-hoc individual representations and fitness functions; the greatly reduced computational cost, since the most expensive procedures are performed *a posteriori* on a reduced set of candidate solutions; and the possibility of separately learning each differential equation in a target system, since the first-order approximation removes dependencies between variables.

Using the Lotka-Volterra model as a case study, we show the applicability of the proposed methodology through experimental validation. We find that the described approach is able to regularly find the correct structure of the original model, even in presence of noise. Results are discussed, and future works outlined.

The rest of the paper is structured as follows: Section 2 recalls a few necessary concepts related to symbolic regression and differential equations. The proposed approach is outlined in Section 3. The case study is presented in Section 4, while the experimental evaluation is described in Section 5. Results are discussed in Section 6, and finally Section 7 draws the conclusions and prospects future works.

2 Background

2.1 Genetic Programming and symbolic regression

Symbolic regression is an evolutionary technique able to extract free-form equations that correlate data from a given experimental dataset. The original idea is presented in [4]. Candidate solutions are encoded as trees, with terminal nodes corresponding to constants and variables of the problem, while intermediate nodes encode mathematical functions such as $\{+, -, *, /, ...\}$. The fitness function is usually proportional to the absolute or squared error between experimental data, with parsimony corrections to favor more compact solutions. An example of an individual for a symbolic regression problem is presented in Figure 1.

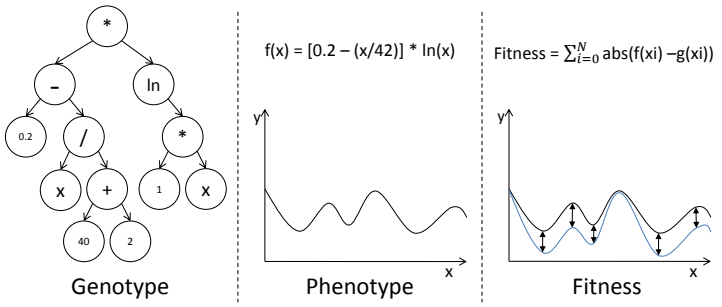


Fig. 1: A candidate solution in a typical symbolic regression problem. The internal representation (genotype) is a binary tree. The phenotype is the corresponding function, while the fitness to minimize is usually the absolute or squared error with respect to experimental points.

2.2 Differential equations and first-order approximation

In order to clarify the scope of our work, we briefly summarize a few basic concepts related to differential equations that will be extensively used in the following. A differential equation is defined as *an equation containing the derivatives of one or more dependent variables, with respect to one of more independent variables* [10]. We will focus on *ordinary differential equations* (ODE), that contain derivatives as a function of a single variable (e.g. the time). A classical example of a differential equation is the first-order ordinary differential equation :

$$y'(t) = f(t, y(t)) \qquad y(t_0) = y_0 \tag{1}$$

where $y(t)$ is a function and y_0 is an initial condition.

The *(Explicit) Euler method* [11] is a first-order numerical procedure for solving ordinary differential equations with a given initial value: it is the most

basic explicit method for numerical integration of ordinary differential equations. With reference to Equation 1, we use the finite difference formula to approximate $y'(t)$:

$$y'(t_n) = \lim_{\Delta_t \to 0} \frac{y(t_n + \Delta t) - y(t_n)}{\Delta_t} \cong \frac{y(t_n + \Delta t) - y(t_n)}{\Delta_t} \tag{2}$$

Choosing a value Δt for the size of every step and setting $t_n = t_0 + n \cdot \Delta t$, one step of the Euler method from t_n to $t_{n+\Delta t} = t_n + \Delta t$ is:

$$y_{n+\Delta t} = y_n + \Delta t \cdot f(t_n, y_n) \tag{3}$$

where the value of y_n is an approximation of the solution to the ODE at time t_n, so that $y_n \approx y(t_n)$. The error per step of this method is proportional to the square of the step size, while its error at a given time is proportional to the step size. It is important to notice how the selection of the step size plays a crucial role in the quality of the results.

A remarkable property of the Euler approximation is the possibility of reconstructing the initial ODE, under specific conditions. In particular, one can rewrite Equation 3 as follows:

$$y_{n+\Delta t} - y_n = F(t_n, y_n, \Delta_t) \tag{4}$$

where F is a function which allows to evaluate $y_{n+\Delta t}$ for any value Δ_t. From Equation 4 and looking at the derivative according to Δ_t around 0, we obtain

$$\lim_{\Delta t \to 0} \frac{y_{n+\Delta t} - y_n}{\Delta_t} = \lim_{\Delta t \to 0} \frac{F(t_n, y_n, \Delta_t) - F(t_n, y_n, 0)}{\Delta_t} \tag{5}$$

which can be rewritten as

$$f(t, y(t)) = y'(t) = \left. \frac{\partial F(t_n, y_n, \Delta_t)}{\partial \Delta_t} \right|_{\Delta_t = 0} \tag{6}$$

going back to Equation 1.

In a practical scenario, Equation 4 can be used to iteratively build the approximate solution of Equation 1. At the opposite, assuming that an analytical form of the approximate solution of Equation 1 is available, Equation 6 can be used to obtain function f.

3 Proposed approach

From Equation 6, we see how it is possible to return to the original ODE starting from the first-order approximation given in Equation 4. It is sufficient to find the classical function F in Equation 4.

In order to find F, additional data must be computed. Given a standard dataset with values of y for different values of time t, we need to add information to each line y_n, t_n, by computing the values of Δt and $y_n + 1$: in fact, in a real-world dataset, it is not given that $\Delta t = t_{n+1} - t_n$ will be constant for every n. Nevertheless, the procedure is trivial: an example is reported in Table 1. Once the new data are obtained, symbolic regression can be straightforwardly applied to the new dataset, to learn F.

t	y
0	20
1.8	16.1
3.5	13.2
5.4	10.9
7.4	8.8
...	...

$\Longrightarrow$

t	y	Δt	$F = y_{n+\Delta t} - y_n$
0	20	0	0
0	20	1.8	-3.9
1.8	16.1	0	0
1.8	16.1	1.7	-2.9
3.5	13.2	0	0
3.5	13.2	1.9	-2.3
5.4	10.9	0	0
5.4	10.9	2.0	-2.1
...	...	...	...

Table 1: An example on how the values of the additional variables (**right**) can be easily produced starting from the original dataset (**left**). In this case, for each line, we computed the values of Δt and F to the next point, only.

One of the known issues of symbolic regression and GP in general is the so-called *overfitting*: solutions that closely approximate training data often exploit exclusive features of the dataset, for example by including terms that model the noise as well. This leads to poor performances on validation sets. Overfitting is sometimes associated with *bloating*, that is, the tendency of GP algorithms to produce bigger and bigger solutions as the evolution goes on. Connections between overfitting and bloating are still being investigated [12] [13], but empirical evidence shows how it can be beneficial to add parsimony measurements in the fitness function or preserve solutions of different complexity, in order to contain the phenomenon.

While overfitting is always undesired, it is particularly deleterious for the proposed approach: even if the F found through symbolic regression performed reasonably well on validation data, when using our procedure to go back to the original ODE, terms with a limited influence on F could create degenerate solutions. For this reason, instead of just using the best solution obtained at the end of the process, we prefer to have a set of candidate equations, each one a different compromise on a Pareto front between complexity and fitting on data.

Dynamic systems are usually represented by a set of ODEs and our approach allows the user to run a symbolic regression algorithm independently on each equation: however, since we prefer to work with a set of candidate solutions for each equation, we need an extra step to choose the best combination to represent the original system. Thus, we apply the procedure described in Equation 6 to

every candidate solution of each set; we generate a set of n-uples, where n is the number of equations in the original system, by permuting solutions in all sets; we discard degenerate n-uples, showing a behavior dissimilar from the original data; and finally we choose the n-uple with the least absolute error with regards to the training data. The whole procedure is summarized in Figure 2.

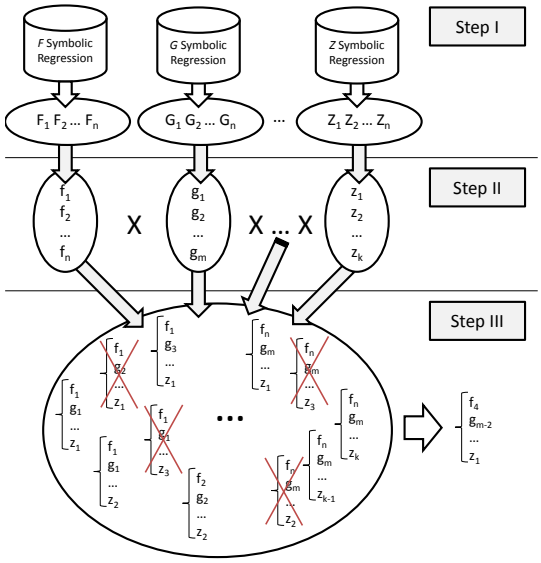


Fig. 2: Summary of the proposed approach. In **Step I**, standard symbolic regression is executed independently on each equation of the original dynamic system: each run returns a set of candidate solutions of variable size, representing different compromises between complexity and fitting on training data. During **Step II**, the obtained sets are transformed into sets of ODEs, following our methodology, and then permuted. Finally, in **Step III**, the resulting set of systems of ODEs is pruned of degenerate equations, the remaining candidate solutions are sorted by fitting on the original data, and the best solution is returned to the user.

4 Case study

In order to attest the viability of our approach, we choose the Lotka-Volterra model [14] as a case study. This model, also known as *predator-prey equations*, is a system composed of two first-order, non-linear, differential equations frequently used to describe the dynamics of biological systems in which two species interact, one as a predator and the other as prey. The equations have been extensively used in biology and other fields, such as economic theory [15]. Their form is:

$$\begin{cases} \frac{dx}{dt} = x(\alpha - \beta y) \\ \frac{dy}{dt} = -y(\gamma - \delta x) \end{cases} \tag{7}$$

where x is the number of prey, y is the number of predators, t represents time, $\frac{dx}{dt}$ and $\frac{dy}{dt}$ represent the growth rates of the two populations over time. α, β, γ and δ are parameters that describe the interaction between the two species.

We focus on a particular configuration of the Lotka-Volterra model, where the parameters' values have been chosen so that no population goes extinct, leading to periodic solutions: $\alpha = 0.04$, $\beta = 0.0005$, $\gamma = 0.2$ and $\delta = 0.004$. Initial populations were taken as $x_0 = y_0 = 20$. A plot of the chosen configuration is reported in Figure 3.

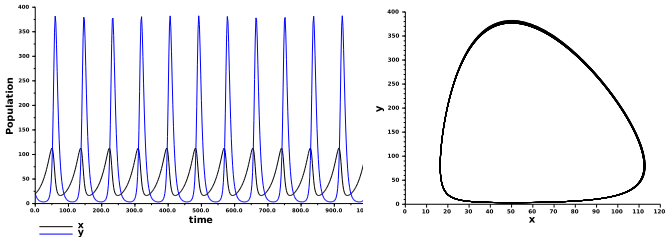


Fig. 3: Plots of the Lotka-Volterra model with parameters used in the experiments. On the left, the variation of the two population with respect to time (x in black, y in blue/light grey). On the right, the state plane with x on the horizontal axis and y on the vertical axis.

Following Equation 4, we are then interested in finding the two functions F and G, first-order approximations of the first and second differential equation of the Lotka-Volterra model, respectively:

$$x_{n+\Delta t} - x_n = F(\Delta t, x_n, y_n) \tag{8}$$
$$y_{n+\Delta t} - y_n = G(\Delta t, x_n, y_n) \tag{9}$$

A major feature of the proposed approach is the ability to learn the two functions in two separate and independent runs of the symbolic regression algorithm. Indeed, the reciprocal dependency of the Lotka-Volterra system has been removed.

5 Experimental results

Since one of the main advantages of the proposed approach is the possibility of exploiting existing tools for standard symbolic regression, for our study we

choose *Eureqa Formulize*[4] [1], considered a state-of-the-art software in the field. Eureqa has one feature of particular interest for our purpose: instead of returning a single solution per run, it presents the user a group of solutions that represent a Pareto front for the objectives of fitting and complexity: see Figure 5 for an example. In Eureqa, each symbol that can appear in a GP tree is associated with a weight, and the complexity of a candidate solution is simply the sum of all weights of terms appearing in it; fitting is computed with respect to the squared error with regards to the training data. It must be noted that, in principle, any GP-based technique able to preserve individuals of different complexity in the final population could be used for our methodology.

Each dataset is modified following the procedure described in Section 3: we use 200 points for the training set. We are interested in exploring the influence of noise and regularity of sampling on the quality of the final results, so for each experiment we use a first dataset sampled every 2 s, and a second one, where every point of data is sampled between 1.5 and 2.5 s from the previous one, following a uniform probability.

Eureqa is configured to employ its *Basic* set of functions {`+,-,*,/,negation`} and terminal symbols {`integer constant, float constant, variable`}. In each experiment Eureqa is run once to stagnation, that is, until the index for the maturity of the population hits the threshold value of 90%. On the machine used for the experiments, a laptop with an Intel i5-2430M CPU (2 cores, 2 threads per core) at 2.40 GHz and 4 GB of RAM, running to stagnation takes 15-20 minutes, and around 10^{10} total fitness evaluations. After each run, Eureqa typically returns about 20 solutions on its Pareto front.

5.1 Noise-free data

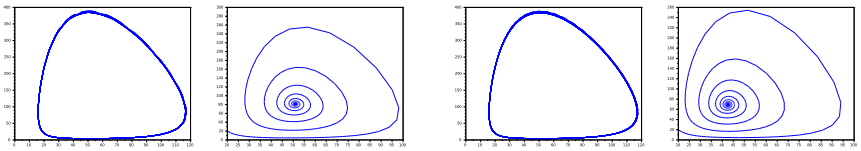
In the simplest scenario, we use datasets with no noise added. The first run, with data regularly sampled, returns 20 candidate solutions for F and 20 candidate solutions for G. Each equation is transformed into an ODE, following our proposed approach. The resulting 400 systems are then pruned of degenerate solutions, that is, solutions that converge towards a point in the x, y plane (see Figure 4 for an example). The remaining systems of ODEs are finally sorted by fitting on the original unmodified training data. The same procedure is followed for the dataset with irregular sampling. This time, 21 candidate solutions are produced for F and 25 for G. The best ODE systems are:

$$\begin{cases} \frac{dx}{dt} = 0.04114x - 0.0004946xy \\ \frac{dy}{dt} = 0.00367xy - 0.1861y \end{cases} \quad \begin{cases} \frac{dx}{dt} = 0.04116x - 0.0004924xy \\ \frac{dy}{dt} = 0.003599xy - 0.1826y \end{cases} \tag{10}$$

with the result for regular sampling on the left, and the result for irregular sampling on the right. Both show the same form of the original Lotka-Volterra model, and a remarkable approximation of the parameters' values. As a comparison, in Figure 4 the two systems found with the proposed approach are compared

[4] `http://formulize.nutonian.com/`

to the systems obtained by simply coupling the best fitting-wise candidate solutions produced in each run.



(a) Regular sampling, noise-free (b) Irregular sampling, noise-free

Fig. 4: Side-by-side comparison on the noise-free dataset, of the best system found through the proposed approach (**left**), and the system obtained by pairing the two fitting-wise best solutions of each run (**right**). It is easy to notice how simply pairing the best candidate solutions leads to degenerate forms or to a lowest fitting on the original training data.

5.2 Absolute noise

In a second trial, random noise (selected from the interval $(-5, 5)$ with uniform probability) is added to the x and y outputs of the model. On the regularly sampled dataset, Eureqa finds 17 candidate solutions for F and 20 for G. On the irregularly sampled dataset, 13 solutions for F and 19 for G are obtained. The best resulting systems are:

$$\begin{cases} \frac{dx}{dt} = 0.03992x - 0.0005548xy \\ \frac{dy}{dt} = 0.003525xy - 0.1916y \end{cases} \quad \begin{cases} \frac{dx}{dt} = 0.03946x - 0.0005354xy \\ \frac{dy}{dt} = 0.003662xy - 0.1948y \end{cases} \tag{11}$$

with the result for regular sampling on the left, and the result for irregular sampling on the right.

5.3 Noise 5%

In the third experimental run we add random noise proportional to the output value, ranging from -5% to +5% with uniform probability. On the regularly sampled dataset, Eureqa returns 16 candidate solutions for F and 15 for G. On the irregularly sampled dataset, we obtain 16 candidate solutions for F and 16 for G. The best resulting systems are:

$$\begin{cases} \frac{dx}{dt} = 0.03947x - 0.0004883xy \\ \frac{dy}{dt} = 0.003706xy - 0.1902y \end{cases} \quad \begin{cases} \frac{dx}{dt} = 0.03743x - 0.0004522xy \\ \frac{dy}{dt} = 0.003707xy - 0.1916y \end{cases} \tag{12}$$

with the result for regular sampling on the left, and the result for irregular sampling on the right.

5.4 Noise 10%

In the last experiment, we add random noise proportional to the output value, ranging from -10% to +10% with uniform probability. On the regularly sampled dataset, 23 candidate solutions for F and 20 for G are obtained. On the irregularly sampled dataset, Eureqa finds 17 candidate solutions for F and 18 for G. The best systems are:

$$\begin{cases} \frac{dx}{dt} = 0.0362x - 0.0004797xy \\ \frac{dy}{dt} = 0.003306xy - 0.1841y \end{cases} \quad \begin{cases} \frac{dx}{dt} = 0.03874x - 0.0004959xy \\ \frac{dy}{dt} = 0.003587xy - 0.1898y \end{cases} \tag{13}$$

with the result for regular sampling on the left, and the result for irregular sampling on the right.

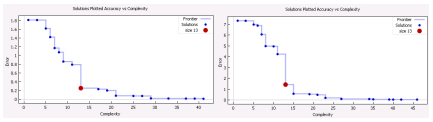
6 Results discussion

The proposed approach is able to find the correct model for the Lotka-Volterra function during each run, even if the parameters (α, β, γ, δ) might slightly differ, especially when dealing with noise. Remarkably, the irregularity of the sampling for the training set does not seem to influence the final outcome; while the presence of noise predictably returns results of lower quality.

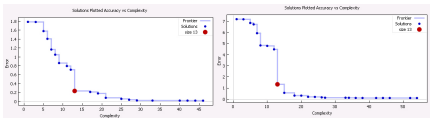
From the experimental evaluation, we can see how Eureqa consistently returns a set of candidate solutions in the order of 10^1: since there are only two differential equations in the model, the search space for coupling the candidate solutions and assessing the results in the second step of our process explores a search space of 10^2. However, when dealing with huge systems of differential equations, the complexity quickly explodes: if the GP routinely returns n solutions, the search space of possible systems of m equations would become $O(n^m)$. Thus, it would be beneficial to reduce the number of viable equations in each set before the coupling process. For example, all equations that, after the derivation process from Equation 6, are reduced to a constant, can be dismissed. This subset, however, includes only 1-2 candidate solutions per set: other methods to prune the Pareto front from uninteresting models should be explored. From the experimental results, we observe how most of the exact forms for the Lotka-Volterra equations always lie in the middle part of the Pareto front fitting/complexity provided by Eureqa (see Figure 5). It would be interesting to investigate whether this property can be generalized to all problems: in that case, the extremes of the Pareto front could be excluded; also, from the Pareto fronts, it looks that often the correct solution shows the biggest improvement with regards to the previous one. These considerations could be included in a heuristic coupling to reduce the number of associations.

7 Conclusions and future works

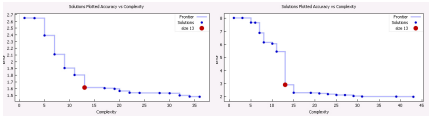
In this paper, we presented a GP-based methodology to learn ordinary differential equations starting from experimental data. The basic idea is reducing the



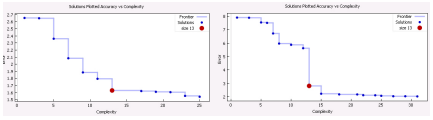
(a) Noise-free dataset, regular sampling: Pareto fronts for F (left) and G (right).



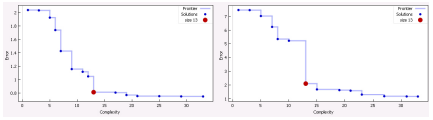
(b) Noise-free dataset, irregular sampling: Pareto fronts for F (left) and G (right).



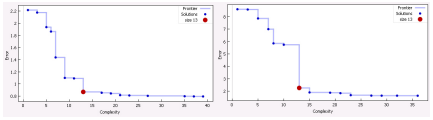
(c) Dataset with absolute noise, regular sampling: Pareto fronts for F (left) and G (right).



(d) Dataset with absolute noise, irregular sampling: Pareto fronts for F (left) and G (right).



(e) Dataset with 5% noise, regular sampling: Pareto fronts for F (left) and G (right).



(f) Dataset with 5% noise, irregular sampling: Pareto fronts for F (left) and G (right).

Fig. 5: Pareto fronts of the solutions found by Eureqa during some of the experiments. The individual with the correct form of the Lotka-Volterra function is highlighted in red, and it is noticeable how it almost always lies in the middle of the Pareto front, often showing the biggest improvement over the previous step.

problem to finding Euler's first-order approximation of an ODE, that is, a regular equation. Once the starting dataset is modified accordingly, we can apply a standard symbolic regression technique, obtaining a group of candidate solutions that represent a trade-off between complexity and fitting to data. Through an inverse procedure to reconstruct an ODE starting from its first-order approximation, used on the whole group of candidate solutions, we acquire a group of ODEs. Finally, by coupling the ODEs obtained, discarding degenerate solutions, and sorting the remaining ones by fitting on the training data, we are able to find a system of ODEs that solves the initial problem.

From the preliminary experiments, it is clear that the coupling step might lead to a combinatorial explosion for the systems to evaluate. Future works will explore an automated coupling of candidate solutions, using theoretical and heuristic measurements to return the best set of solutions. We are currently working on the application of the proposed methodology to a real-world problem for the modelling of processes in the food industry.

Acknowledgments

The authors would like to thank Luuk van Dijk of SpaceX for his interesting ideas and insightful discussions.

References

1. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. science **324**(5923) (2009) 81–85
2. Pickardt, C., Branke, J., Hildebrandt, T., Heger, J., Scholz-Reiter, B.: Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness. In: Simulation Conference (WSC), Proceedings of the 2010 Winter, IEEE (2010) 2504–2515
3. Soule, T., Heckendorn, R.B.: A practical platform for on-line genetic programming for robotics. In: Genetic Programming Theory and Practice X. Springer (2013) 15–29
4. Koza, J.R.: Genetic Programming: vol. 1, On the programming of computers by means of natural selection. Volume 1. MIT press (1992)
5. Babovic, V., Keijzer, M., Aguilera, D.R., Harrington, J.: An evolutionary approach to knowledge induction: Genetic programming in hydraulic engineering. In: Proceedings of the World Water and Environmental Resources Congress. Volume 111. (2001) 64–64
6. Cao, H., Kang, L., Chen, Y., Yu, J.: Evolutionary modeling of systems of ordinary differential equations with genetic programming. Genetic Programming and Evolvable Machines **1**(4) (2000) 309–337
7. Iba, H.: Inference of differential equation models by genetic programming. Information Sciences **178**(23) (2008) 4453–4468
8. Bernardino, H.S., Barbosa, H.J.: Inferring systems of ordinary differential equations via grammar-based immune programming. In Lio, P., Nicosia, G., Stibor, T., eds.: Artificial Immune Systems. Volume 6825 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 198–211
9. Keijzer, M.: Inducing differential / flow equations. Invited talk to the GECCO conference (July 2013)
10. Zill, D.G.: A First Course in Differential Equations: With Modeling Applications. Cengage Learning (2008)
11. Euler, L.: Institutionum calculi integralis. Volume 1. imp. Acad. imp. Saènt. (1768)
12. Vanneschi, L., Castelli, M., Silva, S.: Measuring bloat, overfitting and functional complexity in genetic programming. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation, ACM (2010) 877–884
13. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. Genetic Programming and Evolvable Machines **11**(3-4) (2010) 339–363
14. Lotka, A.J.: Contribution to the theory of periodic reactions. The Journal of Physical Chemistry **14**(3) (1910) 271–274
15. Goodwin, R.M.: A growth cycle. Socialism, capitalism and economic growth (1967) 54–58