

## SOFTWARE METAPAPER

# Glyph: Symbolic Regression Tools

Markus Quade, Julien Gout and Markus Abel

Ambrosys GmbH, Potsdam, DE

Corresponding author: Markus Quade ([markus.quade@ambrosys.de](mailto:markus.quade@ambrosys.de))

We present Glyph – a Python package for genetic programming based symbolic regression. Glyph is designed for usage in numerical simulations as well as real world experiments. For experimentalists, glyph-remote provides a separation of tasks: a ZeroMQ interface splits the genetic programming optimization task from the evaluation of an experimental (or numerical) run. Glyph can be accessed at <https://github.com/Ambrosys/glyph>. Domain experts are able to employ symbolic regression in their experiments with ease, even if they are not expert programmers. The reuse potential is kept high by a generic interface design. Glyph is available on PyPI and Github.

**Keywords:** Symbolic Regression; Genetic Programming; Machine Learning Control; MOGP; Python

**Funding statement:** This work has been partially supported by the German Science Foundation via SFB 880. MQ was supported by a fellowship within the FITweltweit program of the German Academic Exchange Service (DAAD).

## (1) Overview

### Introduction

Symbolic regression [1] is an optimization method to find an optimal representation of a function. The method is “symbolic”, because building blocks of the functions, i.e. variables, primitive functions, and operators, are represented symbolically on the computer. Genetic programming (GP) [2] can be implemented to find such a function for system identification [3, 4] or fluid dynamical control [5, 6]. Glyph is an effort to separate optimization method and optimization task allowing domain-experts without special programming skills to employ symbolic regression in their experiments. We adopt this separation of concerns implementing a client-server architecture; a minimal communication protocol eases its use. Throughout this paper “experiment” is meant as a synonym for any symbolic regression task including a lab-experiment, a numerical simulation or data fitting.

Previous work on system identification and reverse engineering of conservation laws was reported in [1, 7]. As a substantial extension, we put the algorithms in the context of control problems and in contrast to these works, we publish the implementation. Modern algorithms also include multi-objective optimization [4] and advances like age fitness based genetic programming [8] or epigenetic local search [9]. There exist various approaches to the representation of multi IO problems, including stack- or graph-based representations and pointers [9, 10]. As a special feature we extended our implementation such that it can be easily used for finding an optimal control

law for a given, measured and actuated system. This is, however just one application of our software.

### Implementation and architecture

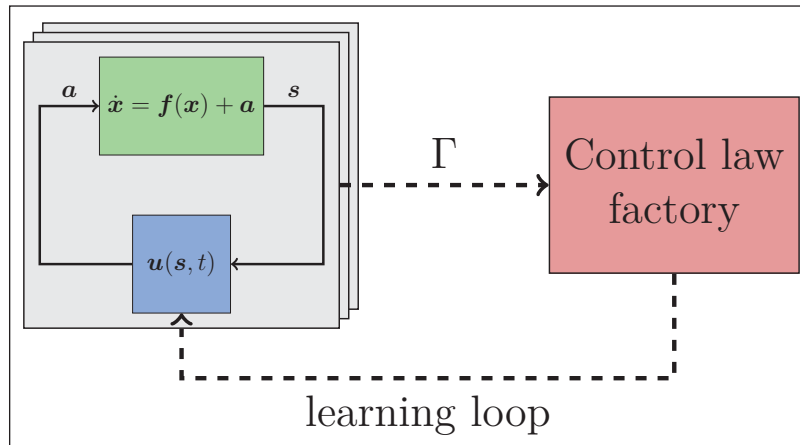
Glyph is intended as a lightweight framework to build an application finding an optimal system representation given measurement data. The main application is intended as system control, consequently a control law is determined and returned. Glyph is built on the idea of loose coupling such that dependencies can be released if wanted.

A typical control application consists of a system and its controller, possibly separated, cf. **Figure 1**. Glyph has three main module to build such an application: i) the assessment, which holds all methods and data structures belonging to the experiment, ii) the GP which is responsible for the system identification, and iii) the application components, which constitute an application.

### Abstraction Layers

Glyph is conceptualized around different layers of abstraction:<sup>1</sup>

1. Representation layer: This layer consists of a data-structure representing a GP solution, i.e. an individual, and methods to manipulate such data-structures. This abstraction layer also allows for an interchangeable representation.
2. Algorithm layer: This layers encapsulates the selection and tweaking of individuals.



**Figure 1:** Left: A typical closed loop control task is sketched. The system  $\dot{x} = f(x)$  is observed by some measurements  $s$ ; it is controlled by adding the actuation  $a = u(s, t)$ . The corresponding control law  $a = u(s, t)$  is determined by symbolic regression. Right: gp-based symbolic regression finds different candidate control laws. Each candidate solution is given a fitness score  $\Gamma$  which is used to compare different solutions and to advance the search in function space. Figure adapted from [5] with permission.

3. AssessmentRunner layer: Abstracts calculation of cost functions.
4. GPRunner layer: This is an integration layer which lets the user iterate the evolutionary algorithm calculating statistics after each iteration.
5. Application layer: This layer sets the context and control flow.

The first three layers are sufficient to formulate and optimize a symbolic regression problem. The GPRunner and Application let you gap the bridge between prototypes and real-world-problem-solving applications.

### Building an Application

An application consists of a GP callable, the `gp_runner`, an assessment callable for input, the `assessment_runner`, and the application which uses both of these classes and holds all application-relevant details. A command-line application is built by

```
assessment_runner = AssessmentRunner (assess_args)
gp_runner = glyph.application.GPRunner (gp_args)
app = command_line_application (app_args)
```

The `assessment_runner` has one argument, the `parallel_factory` which implements a `map()` method, possibly parallel. For an application one needs to implement `setup`, `assign_fitness`, and `measure`: `setup` is self-explaining, `measure` is a key method which takes as input a set of measurement functions and combines them into a tuple of callable measures for multiobjective optimization. The measures are used eventually in `assign_fitness` where the return values are used to assign a fitness to an individual from GP. The interface is freely extensible. A `gp_runner` forwards the evolutionary iteration. It takes as arguments, `gp_args`, an individual class, a `gp_algorithm`, and an `assessment_runner`. The individual class contains the representation of a function, the individual; it is based on DEAP's tree-based implementation. The `gp_algorithm` takes care for the breeding and selection steps, its principles are described in [2].

The application is run in the main function with `app.run()`.

In the application and `gp_runner`, the user has freedom to add functionality using the list of callbacks in the arguments, say, to implement other logging or streaming options. This allows for very flexible programming. We constructed the components that way to allow users to specialize for their particular experiments and possibly increase performance or extend the symbolic regression, e.g. by replacing the DEAP tree-based representation of an individual.

### Remote Control

One main objective of glyph is its use in a real experiment. In this case, the GP loop is separated from the experimental loop in a client-server setup using ZeroMQ [11], cf. **Figure 2**.

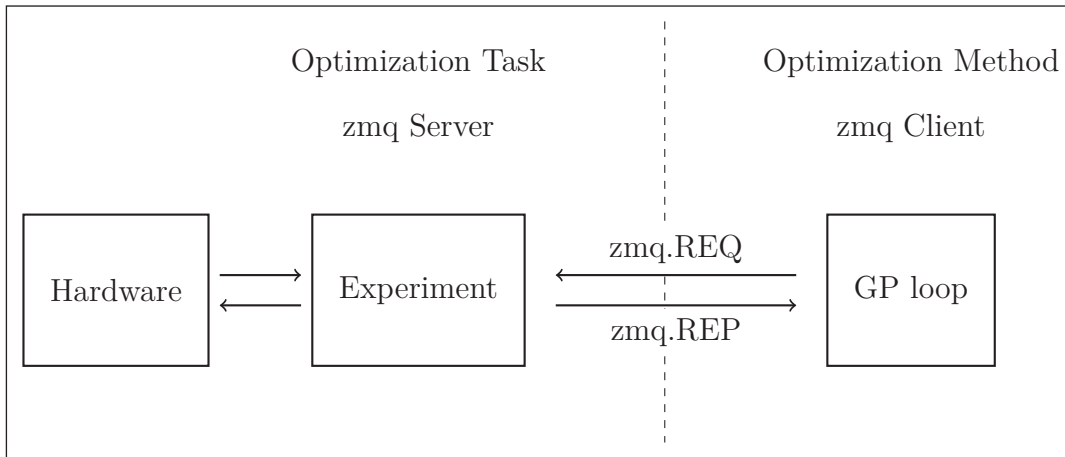
Consequently, one should implement the interface to the experiment using the protocol described in the following subsection. Having the implementation of the experiment, the server, one needs to implement the client, i.e. the interface to the `gp_runner`. In essence this means connecting the correct sockets with ZeroMQ and ensuring that the `gp_runner` and the `assessment_runner` use the corresponding sockets. Then, the main application is assembled as before, now using a `RemoteApp` for the main application, which in turn uses a `gp_runner`, which then uses now a `RemoteAssessmentRunner`. That is it, we can run remotely our GP evaluation from some client and the experiment in place of the experiment.

### Communication Protocol

The communication is encoded in json [12]. A message is a json object with two members:

```
{
  "action": "value",
  "payload": "value"
}
```

The possible values are listed in **Table 1**. The config action is performed prior to the evolutionary loop. Entering



**Figure 2:** Sketch of the implementation of the experiment – GP communication as client-server pattern. Left: single experiment server plus event handler. Right: GP client. Both parts are interfaced using ZeroMQ. As described in Sec. the GP program performs requests, e.g. the evaluation of a candidate solution. The event handler takes care of these requests and eventually forwards them to the hardware.

**Table 1:** Communication protocol. The *config* action contains the options to be set, the *experiment* action contains a list of expressions, the *shutdown* action terminates the application.

Action name	Payload	Expected return Value
<i>CONFIG</i>	–	config settings
<i>EXPERIMENT</i>	list of expressions	list of fitness value(s)
<i>SHUTDOWN</i>	–	–

the loop, discovered solutions will be batched and a *experiment* action will be requested. You can configure optional caching for re-discovered solutions. This includes persistent caching between different runs. The *shutdown* action will let the experiment program know that the GP loop is finished and you can safely stop the hardware.

Configuration settings are sent as a json object in key:value form, where the keys contain the option to be set, there is only one mandatory option: the primitive set. To configure the primitive set, the primitive names are passed as content of the key *config*, whose values specify the corresponding arities, both fields described again as json object.

The *experiment* action sends a list of expressions, encoded as string in prefix (also: polish) notation [13]. For each expression sent, the experiment returns a fitness tuple.

Additionally, one can define the type of algorithm, error metric, representation, hyper-parameters, etc. A comprehensive up to date list can be found at [http://glyph.readthedocs.io/en/latest/usr/glyph\\_remote/](http://glyph.readthedocs.io/en/latest/usr/glyph_remote/).

**Application example: control of the chaotic Lorenz System**

In the following, we demonstrate the application and use of Glyph by the determination of an unknown optimal control law for a chaotic system. As an example, we study the control of the potentially chaotic Lorenz system. Chaotic systems are very hard to predict and control in

practice due to their sensitivity towards small changes in the initial state which may lead to exponential divergence of trajectories. The Lorenz model [14] consists of a system of three ordinary differential equations:

$$\begin{aligned}
 \dot{x} &= s(y - x) \\
 \dot{y} &= rx - y - xz \\
 \dot{z} &= xy - bz,
 \end{aligned}
 \tag{1}$$

with two nonlinearities,  $xy$  and  $xz$ . Here  $x, y,$  and  $z$  make up the system state and  $s, r, b$  are parameters:  $s$  is the Prandtl number,  $r$  is the Rayleigh number, and  $b$  is related to the aspect ratio of the air rolls. For a certain choice of parameters and initial conditions chaotic behavior emerges.

We present two examples where the target is to learn control of bring a chaotic Lorenz system to a complete stop, that is,  $(x, y, z) = 0 \ (t \in \mathbb{R})$ . In the first example, “control in  $y$ ”, the actuator term is applied to  $\dot{y}$ . This allows for a more direct control of the system, since  $y$  appears in every equation of (1) and, thus, influence all three state components,  $x, y,$  and  $z$ . In the second example, “control in  $z$ ”, the actuator term is applied to  $\dot{z}$ , which leads to a more indirect control, since the flow of information from  $z$  to  $x$  is only through  $y$ .

The system setup is summarized in **Tables 2** and **3**. When  $r = 28, s = 10,$  and  $b = 8/3,$  the Lorenz system produces chaotic solutions (not all solutions are chaotic). Almost all initial points will tend to an invariant set – the Lorenz attractor – a strange attractor and a fractal. When plotted the chaotic trajectory of the Lorenz system resembles a butterfly (blue graph in **Figure 3**). The target of control is, again, formulated as  $RMSE^2$  of the system state with respect to zero (separately for each component).

$$RMSE_x := RMSE(x, 0), \quad RMSE_y := RMSE(y, 0), \quad RMSE_z := RMSE(z, 0).$$

The control function  $u$  can make use of ideal measurements of the state components. Additionally, we allow for a single symbolic constant  $k$  to be used as argument for

the control law. Given an expression, this constant is optimized separately, see also [4]. The respective GP runs for control in  $y$  and control in  $z$  are conducted with the corresponding random seeds labeled “in  $y$ ” and “in  $z$ ”.

**Table 2:** General setup of the GP runs.

population size	500
max. generations	20
MOO algorithm	NSGA-II
tree generation	halfandhalf
min. height	1
max. height	4
selection	selTournament
tournament size	2
breeding	varOr
recombination	cxOnePoint
crossover probability	0.5
crossover max. height	20
mutation	mutUniform
mutation probability	0.2
mutation max. height	20
constant optimization	leastsq

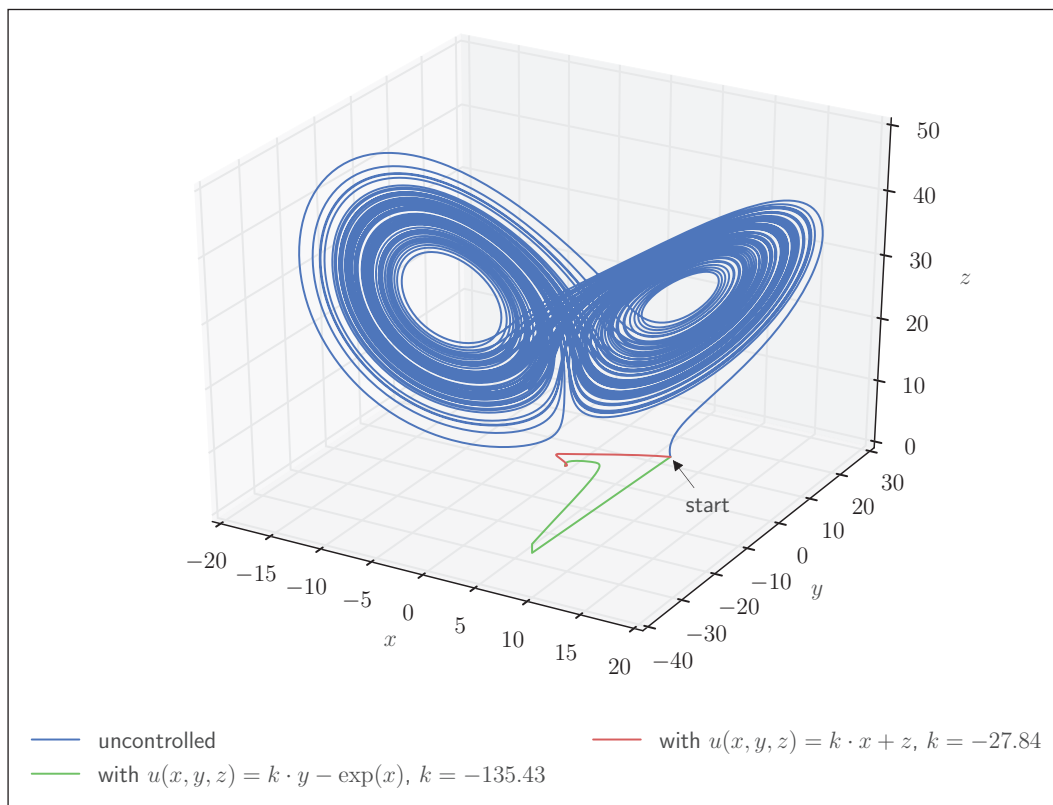
**Control in  $y$ :** For control in  $y$  the actuator term  $u$  is added to the left side of the equation for  $\dot{y}$  in the uncontrolled system (1):

$$\dot{y} = rx - y - xz + u(x, y, z)$$

The Pareto solutions from the GP run are shown in **Table 4**. The wide spread of the RMSE values is a sign of conflicting objectives that are hard to satisfy in conjunction. Interestingly, almost all solutions,  $u$ , commonly introduce a negative growth rate into  $\dot{y}$ . This effectively drives  $y$  to zero and suppresses the growth terms,  $sy$  and  $xy$ , in the equations for  $\dot{x}$  and  $\dot{z}$  respectively, in turn, driving  $x$  and  $z$  to zero as well. As would be

**Table 3:** Control of the Lorenz system: system setup.

Dynamic system		GP
$s$	10	cost functionals RMSE ( $x, 0$ )
$r$	28	RMSE ( $y, 0$ )
$b$	8/3	RMSE ( $z, 0$ )
$x(t_0)$	10.0	length ( $u$ )
$y(t_0)$	1.0	argument set $\{x, y, z\}$
$z(t_0)$	5.0	constant set $\{k\}$
$t_0, t_n$	0, 100	seed (in $y$ ) 4360036820278701581
$n$	5000	seed (in $z$ ) 2480329230996732981



**Figure 3:** Phase portrait of the forced Lorenz system with control exerted in  $\dot{y}$ . (Green and red: The system trajectories when controlled by two particular Pareto-front solutions. Blue: the uncontrolled chaotic system).

expected, minimal expressions, of length 1 or 2, cannot compete in terms of the RMSE. For example, the simple solution,  $u(x, y, z) = -ky$  (fourth row), is almost as good as the lengthier one,  $u(x, y, z) = -\exp(x) + ky$  (first row), and even better in  $RMSE_y$ .

**Table 4** shows the results from the GP run. One solution immediately stands out:  $u = k \cdot x + z$ , with  $k = -27.84$  (second row). It is exactly what one might expect as a control term for the chaotic Lorenz system with control in  $y$ . This control law effectively reduces the Rayleigh number  $r$  to a value close to zero ( $k \approx r$ ), pushing the Lorenz system past the first pitchfork bifurcation, at  $r = 1$ , back into the stable-origin regime. If  $r < 1$  then there is only one equilibrium point, which is at the origin. This point corresponds to no convection. All orbits converge to the origin, which is a global attractor, when  $r < 1$ .

The phase portrait of the solution from the first and second row of **Table 4** are illustrated in **Figure 3**. After a short excursion in negative  $y$  direction ( $t \approx 5$ ), the green trajectory quickly converges to zero. The red trajectory seems to take a shorter path in phase space, but, it is actually slower to converge to the origin. This is verified

by a plot of the trajectories for the separate dimensions  $x$ ,  $y$  and  $z$  over time **Figure 4**.

**Control in z:** For control in  $z$  the actuator term  $u$  is added to the left side of the equation for  $\dot{z}$  in the uncontrolled system (1)

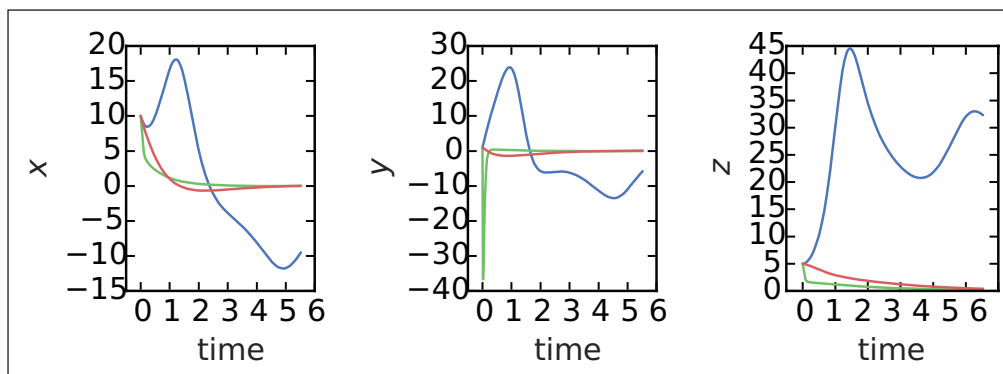
$$\dot{z} = xy - bz + u(x, y, z)$$

Selected Pareto-front individuals from the GP run are displayed in **Table 5**. As mentioned at the beginning of this section, effective control is hindered by the indirect influence of  $z$  on the other state variables, hence, it is not surprising that the control laws here are more involved than in the previous case. Also, they generally do not perform well in the control of  $z$ , which is expressed by the relatively high values in  $RMSE_z$ . This is confirmed by the phase portrait of the solution  $u(x, y, z) = -(k \cdot (-y) + x \cdot z + y + z)$  shown in figure **Figure 5**: While going straight to the origin in the  $xy$ -plane there are strong oscillations of the trajectory along the  $z$ -axis.

The dynamics caused by the actuation, e.g. for the best control law found, can be explained qualitatively: there

**Table 4:** Control of the Lorenz system in  $y$ : Pareto-front solutions.

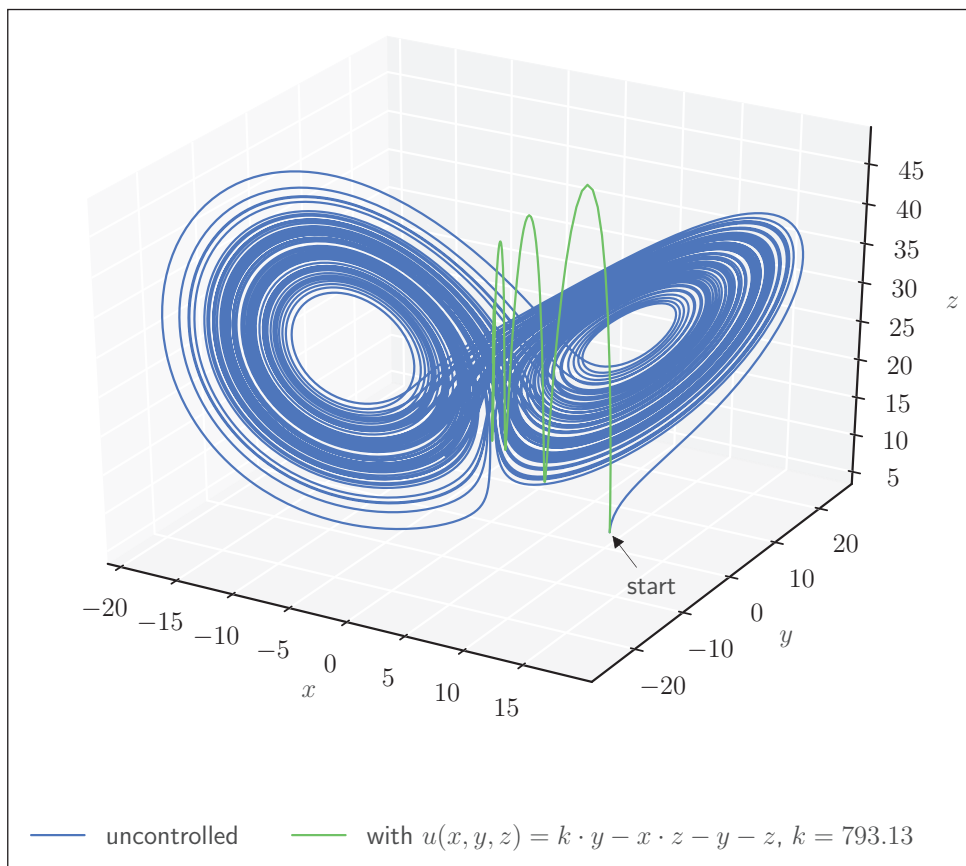
$RMSE_x$	$RMSE_y$	$RMSE_z$	length	expression	constants
0.178884	0.087476	0.105256	7	$-\exp(x) + k \cdot y$	$k = -135.43$
0.241226	0.069896	0.213063	5	$k \cdot x + z$	$k = -27.84$
0.246315	0.014142	0.222345	6	$-z + k \cdot y$	$k = -75590.65$
0.246316	0.014142	0.222347	4	$-k \cdot y$	$k = 75608.50$
0.246367	0.028851	0.220426	10	$-x \cdot (k + y) \cdot \exp(\exp(y))$	$k = 9.62$
0.246729	0.118439	0.211212	6	$-x \cdot (k + y)$	$k = 29.21$
0.246850	0.031747	0.220726	9	$-x \cdot (k + y) \cdot \exp(y)$	$k = 26.12$
4.476902	4.468534	7.488516	3	$-\exp(y)$	
7.783655	8.820086	24.122441	2	$-x$	
7.931978	9.066296	25.047630	1	$k$	$k = 1.0$
8.319191	8.371462	25.932887	2	$-y$	
8.994685	9.042226	30.300641	1	$z$	



**Figure 4:** Detailed view of the single trajectories in  $x$ ,  $y$ , and  $z$  dimension. (blue: uncontrolled; green:  $u(x, y, z) = -\exp(x) + k \cdot y$ ,  $k = -135.43$ ; red:  $u(x, y, z) = k \cdot x + z$ ,  $k = -27.84$ ).

**Table 5:** Control of the Lorenz system in  $z$ : selected Pareto-front solutions.

RMSE <sub>x</sub>	RMSE <sub>y</sub>	RMSE <sub>z</sub>	length	expression	constants
0.289289	0.139652	26.994070	13	$-(k \cdot (-y) + x \cdot z + y + z)$	$k = 793.129676$
0.327926	0.267043	27.070289	8	$\exp(-k + y \cdot \sin(y))$	$k = -4.254574$
0.431993	0.508829	32.116326	7	$(k + x) \cdot (y + z)$	$k = 2.638069$
0.471535	0.525010	26.986321	5	$k + x \cdot z$	$k = 67.137183$
0.637056	0.605686	26.895493	7	$\exp(k + y \cdot \sin(y))$	$k = 3.964478$
0.677204	0.703577	27.019308	4	$y + \exp(k)$	$k = 4.276256$
0.930668	0.952734	26.895126	5	$x + \exp(\exp(k))$	$k = 1.448198$
1.764030	1.860288	26.766383	6	$(k + x) \cdot \exp(y)$	$k = 21.783557$



**Figure 5:** Control of the Lorenz system in  $\dot{z}$ .

is a strong damping in all variables but  $y$ . This reflects the tendency to suppress  $z$ -oscillations and, at the same time, to add damping in  $y$  through the  $xz$  term: if  $y$  grows, the  $z$  contribution to damping on the right hand side of the Lorenz equations (1) grows and, in turn, damps  $y$ . This is, however, only possible to some extent, hence, the oscillations observed in figure **Figure 5**.

We conclude the demonstration with a short summary: Using Glyph we can find complex control laws, even for unknown systems. This cannot be easily achieved with other frameworks. The control laws found can be studied analytically in contrast to several other methods which have black-box character. The usage is straightforward,

as we have described above. The above example can be found online as an example.

**Other symbolic regression libraries**

Due to its popularity, symbolic regression is implemented by most genetic programming libraries. A semi-curated list can be found at <http://geneticprogramming.com/software/>. In contrast to other implementations, Glyph implements higher concepts, such as symbolic constant optimization, and also offers parallel execution for complex examples (control simulation, system identification). This is very important for practical applications. Glyph is well tested, cf. **Table 6** and currently applied in two

**Table 6:** Comparison of Glyph and openMLC features. MOGP refers to multi-objective optimization. MO means multiple outputs. SCO means symbolic constant optimization.

	CI/tests	doc	caching	checkpointing	MOGP	SCO	MO
openMLC	✓	✗	✓	✓	✗	✗	✗
Glyph	✓	✓	✓	✓	✓	✓	✓

experiments and several numerical problems. For control, apart from our implementation, there exists only one more alternative as a dedicated matlab toolbox (with python interface), openMLC [15], which contains much of the material treated in [6]. Apart from the fact that matlab itself is no free software, one difference is that a toolbox is not as powerful as a library made for developers to extend into various directions. Our first application is control, however the library is built in an extensible way and now more and different applications are ongoing. Further, we offer multi-objectivity, multi-output and constant optimization which make symbolic regression applicable and useful for practical tasks.

### Quality control

Continuous Integration tests are conducted for Mac, Linux and Windows using Travis and AppVeyor. The tests consider Python version 3.6. Unit test coverage is around 85% as reported by codecov. Additionally, tests specifically cover the stochastic parts of the optimization to ensure reproducibility. Along with the software tests are shipped which guarantee the correct execution of the examples. The user can reuse these tests for further development. Locally, tests can be executed via the pytest command.

## (2) Availability

### Operating system

Glyph is compatible with Mac, Linux and Windows.

### Programming language

Python 3.6+

### Dependencies

Glyph is based on *DEAP* [16], an evolutionary computation framework adopting a toolbox-like structure for rapid prototyping. Further dependencies are found up-to-date at <https://github.com/Ambrosys/glyph/blob/master/requirements.txt>. To run test, examples and to build the documentation, one needs to install <https://github.com/Ambrosys/glyph/blob/master/requirements-dev.txt>.

### List of contributors

Core contributors (prior to open source): Markus Quade  
Julien Gout  
Open source contributors can be found at <https://github.com/Ambrosys/glyph/graphs/contributors>.

### Software location

#### Archive Zenodo

**Name:** Ambrosys/glyph

**Persistent identifier:** <http://doi.org/10.5281/zenodo.2572859>

**Licence:** LGPL

**Publisher:** Markus Quade

**Version published:** 0.5.2

**Date published:** 19.02.19

### Code repository Github

**Name:** glyph

**Persistent identifier:** <https://github.com/Ambrosys/glyph>

**Licence:** LGPL

**Date published:** 08.12.16

### Language

English

## (3) Reuse potential

The potential to use Glyph is twofold: on one hand applications can be easily written and the elegant core functionality can be extended; on the other hand, researchers can use the code as core for symbolic regression and extend its functionality in a very generic way. With respect to applications, currently two main directions are targeted: modeling using genetic programming- based symbolic regression and the control of complex system, where a control law can be found generically, using genetic programming. The detailed examples and tutorial allow usage from beginner to experienced level, i.e. undergraduate research projects to faculty research. The design of Glyph is such that generic interfaces are provided allowing for very flexible extension.

### Notes

<sup>1</sup> See also <https://glyph.readthedocs.io/en/latest/usr/concepts/>.

<sup>2</sup> 
$$\text{RMSE}(a, b) = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} (a(t_i) - b(t_i))^2}$$

### Acknowledgements

We acknowledge very fruitful discussions with respect to applications of MLC with S. Brunton, B. Noack, A. Pikovsky, M. Rosenblum, R. Semaan, and B. Strom and coding gossip with V. Mittal and F. Meckel.

### Competing Interests

The authors have no competing interests to declare.

### References

1. **Schmidt, M** and **Lipson, H** "Distilling Free-Form Natural Laws from Experimental Data". In: *Science*, 324(5923): 81–85. Apr. 2009. DOI: <https://doi.org/10.1126/science.1165893>

2. **Koza, J R** *Genetic programming: On the programming of computers by means of natural selection*, 1. MIT press, 1992.
3. **Vladislavleva, E**, et al. "Predicting the energy output of wind farms based on weather data: Important variables and their correlation". In: *Re-newable Energy*, 50: 236–243. Feb. 2013. DOI: <https://doi.org/10.1016/j.renene.2012.06.036>
4. **Quade, M**, et al. "Prediction of dynamical systems by symbolic regression". In: *Physical Review E*, 94(1). July 2016. DOI: <https://doi.org/10.1103/PhysRevE.94.012214>
5. **Gout, J**, et al. "Synchronization control of oscillator networks using symbolic regression". In: *Nonlinear Dyn*, 91(2): 1001–1021. Nov. 2017. DOI: <https://doi.org/10.1007/s11071-017-3925-z>
6. **Duriez, T, Brunton, S L and Noack, B R** *Machine Learning Control – Taming Nonlinear Dynamics and Turbulence*. Springer International Publishing, 2017. DOI: <https://doi.org/10.1007/978-3-319-40624-4>
7. **Schmidt, M D**, et al. "Automated refinement and inference of analytical models for metabolic networks". In: *Physical Biology*, 8(5). Aug. 2011. DOI: <https://doi.org/10.1088/1478-3975/8/5/055011>
8. **Schmidt, M and Lipson, H** "Age-Fitness Pareto Optimization". In: *Genetic Programming Theory and Practice VIII*, 129–146. New York: Springer, Oct. 2010. DOI: [https://doi.org/10.1007/978-1-4419-7747-2\\_8](https://doi.org/10.1007/978-1-4419-7747-2_8)
9. **La Cava, W, Danai, K and Spector, L** "Inference of compact nonlinear dynamic models by epigenetic local search". In: *Engineering Applications of Artificial Intelligence*, 55: 292–306. Oct. 2016. DOI: <https://doi.org/10.1016/j.engappai.2016.07.004>
10. **Galvan-Lopez, E** "Efficient graph-based genetic programming representation with multiple outputs". In: *International Journal of Automation and Computing*, 5(1): 81–89. Jan. 2008. DOI: <https://doi.org/10.1007/s11633-008-0081-4>
11. **Akgul, F** *ZeroMQ*. Packt Publishing, 2013.
12. **Ecma International**. "The JSON Data Interchange Format". In: *Standard ECMA-404*, 9(2013).
13. **Jorke, G, Lampe, B and Wengel, N** *Arithmetische Algorithmen der Mikrorechentchnik*. Verlag Technik, 1989.
14. **Lorenz, E N** "Deterministic Nonperiodic Flow". In: *Journal of the Atmospheric Sciences*, 20(2): 130–141. 1963. DOI: [https://doi.org/10.1175/1520-0469\(1963\)020<0130:DNF>2.0.CO;2](https://doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2)
15. **MachineLearningControl**. *OpenMLC-Python*. Aug. 2017. URL: <https://github.com/MachineLearningControl/OpenMLC-Python>.
16. **De Rainville, F-M**, et al. "DEAP". In: *ACM SIGEVOlution*, 6(2): 17–26. (Feb. 2014). DOI: <https://doi.org/10.1145/2597453.2597455>


**How to cite this article:** Quade, M, Gout, J and Abel, M 2019 Glyph: Symbolic Regression Tools. *Journal of Open Research Software*, 7: 19. DOI: <https://doi.org/10.5334/jors.192>

**Submitted:** 12 September 2017

**Accepted:** 20 May 2019

**Published:** 17 June 2019

**Copyright:** © 2019 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <http://creativecommons.org/licenses/by/4.0/>.

 *Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press.

**OPEN ACCESS** 