A Unified Framework for Robust Discovery of Hybrid Physical Dynamics from Data TODO: change title

Eymeric Chauchat,[1*] A. Collaborator,[1,2] Lead Scientist[1]

[1]Department of Robotics, Graduate School of Engineering, Tohoku University, Sendai, Japan.

[2]Another Department, Different Institution, City & Postal Code, Country.

[*]Corresponding author. Email: [Your Email Here]

[†]These authors contributed equally to this work. TODO: correctly format, add correct people

Automated discovery of governing equations for complex mechanical systems is often hindered by methods that cannot handle hybrid physical formalisms or implicit dynamics. To address this, we introduce PY-XL-SINDY, a unified computational framework that extends Sparse Identification of Nonlinear Dynamics (SINDy). Our framework's key innovation is a modular catalog that seamlessly combines Lagrangian, Newtonian, and other modeling paradigms to identify hybrid systems from a single data source. It further incorporates a robust, constrained-optimization solver to accurately identify implicit dynamics, even in systems with unforced coordinates. We demonstrate that PY-XL-SINDY successfully discovers the governing equations for a series of increasingly complex mechanical systems, with models that show high-fidelity agreement with the MuJoCo physics simulator. This work provides a powerful and extensible tool for the automated discovery of complex physical laws from experimental data.

TODO: mainly correct need to refine

TEASER : A new system identification framework that can handle hybrid physical formalism and implicit/explicit experiments.

The ability to distill natural laws from observational data into concise mathematical equations is a cornerstone of scientific progress. Recently, data-driven methods, particularly the Sparse Identification of Nonlinear Dynamics (SINDy) framework, have shown great promise in automating this discovery process (*1*). SINDy operates on the assumption of parsimony: that most physical systems are governed by equations with only a few active terms. However, the original SINDy formulation is best suited for explicit, ordinary differential equations in the form $\dot{\mathbf{x}} = f(\mathbf{x})$.

This has led to the development of specialized variants to handle different physical paradigms. For instance, Lagrangian SINDy focuses on identifying the scalar Lagrangian of a system, a more concise representation for many mechanical systems (***?****, 2*). Other variants like SINDy-PI were developed to handle implicit dynamics ($\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}) = 0$), which are common in constrained mechanical systems or models with rational dynamics, though they can be sensitive to noise (*3*). These methods, while powerful, exist as distinct algorithms, making it difficult to model hybrid systems that might involve, for example, a core Lagrangian structure with additional non-conservative forces best described in a classical framework. In addition, it is likely to find on system an implicit and explict subspace, specificaly concerning robot where not all coordinate are activated. Furthermore, scaling these discovery methods to handle large datasets and enabling their use in modern applications like reinforcement learning requires a focus on computational efficiency.

Here, we introduce PY-XL-SINDY, a comprehensive and extensible Python framework that unifies these disparate approaches and addresses their limitations. Our main contributions are:

- **A Unified Hybrid Modeling Framework:** We introduce a modular catalog architecture that allows for the seamless combination of different physical models. Users can construct a single regression problem from ‘Lagrange‘, ‘Classical‘ (Newtonian), and ‘ExternalForces‘ components, enabling the discovery of hybrid dynamics.

- **Robust Implicit Dynamics Identification:** We implement a robust solver for implicit dynamics based on constrained optimization, inspired by SINDy-PI, and introduce a novel post-processing algorithm to automatically cluster and identify the true sparse solution from the resulting coefficient matrix.

- **High-Performance Parallelization:** The framework is integrated with JAX, enabling JIT-compilation and auto-vectorization (‘vmap‘) of the dynamics functions. This allows for

massive parallelization of simulations, critical for large-scale hyperparameter searches, uncertainty quantification, and integration with machine learning pipelines.**TODO: Not clear and not really true**

- **End-to-End Validation Pipeline:** We present a complete workflow for generating data from the high-fidelity MuJoCo simulator, performing the system identification, and validating the discovered models against the simulator's trajectory, providing a robust metric for real-world performance.

TODO: Overall, clear but need to be refined...

Methods

This work is a combinaison of previously developped algorithm and new addition from PY-XL-SINDY. Before diving into the newer method, we will give indepth explanation of the past method. The new unified algorithm has been developped by mixing Sindy and Lagrangian (*2*) catalog and also mixing the Explicit and Implicit discovery.

SINDy catalog

Before talking about the unified catalog framework, it is important to understand how the classical SINDy catalog is working. When talking about SINDy we are refering to the original SINDy algorithm developped in (*1*). The main idea of SINDy is to discover the governing equation of a system from data. The system is assumed to be govern by an ordinary differential equation of the form :

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) \tag{1}$$

To recover this ordinary differential equation from data, SINDy is using a library of candidate function to build an experimental matrix $\boldsymbol{\Theta}(\mathbf{X})$ where each column represent the evaluation of a candidate function on the data. The goal is then to find the sparse vector of coefficient $\boldsymbol{\Xi}$ that verify the following equation :

$$\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi} \tag{2}$$

From this catalog definition we can also infer the implicit catalog definition where the goal is to find the sparse vector of coefficient $\boldsymbol{\Xi}$ that verify the following equation :

$$\mathbf{0} = \boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})\boldsymbol{\Xi} \tag{3}$$

The Unified Catalog Framework

From the base SINDy catalog definition, we could create an unified catalog system able to mix different physical formalism.

By applying a transformation step between the candidate function library and the experimental matrix construction, we can define different catalog category that will handle differently the construction of the experimental matrix. Using this approach, we can take advantage of the strenght of each physical formalism.

The core of PY-XL-SINDY is its modular catalog system. We implemented three main categories:

- ***Lagrange*:** This category takes a library of candidate symbolic functions and constructs the experimental matrix by applying the Euler-Lagrange operator, $\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i}$.
- ***Classical*:** This category represents standard Newtonian or state-space models. It takes a library of functions of the state variables (e.g., polynomials, trigonometric functions) and constructs the corresponding columns in the experimental matrix directly.
- ***ExternalForces*:** This category explicitly models external forces, typically serving as the known right-hand-side of the final governing equation.

The ‘CatalogRepartition‘ class combines instances of these categories into a single, cohesive experimental matrix $\boldsymbol{\Theta}$. This allows for the construction of a hybrid equation, for example $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \tau + \mathbf{F}_{friction}(\dot{\mathbf{q}})$, where the Lagrangian terms are handled by the ‘Lagrange‘ catalog and the friction terms by the ‘Classical‘ catalog.

Explicit regression

After building the experimental matrix using the unified catalog framework, we can now perform the regression to find the sparse coefficient vector $\boldsymbol{\Xi}$ that satisfy the following equation :

$$\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi} \tag{4}$$

In order to perform this operation multiple sparse regression optimizer are available :

- Lasso : Lasso regression and LassoCV an enhanced version which automatically find the best regularization parameter.
- Hard tresholding : The method mainly studied by (*?*) where at each iteration the small coefficient are tresholded to zero and a least square regression is performed on the remaining coefficient.
- Proximal gradient method : Method that enable parallelization of the regression step (*2*).

Lasso is theoretically the most robust method but it is also the slowest one. However in our case, since we used compact catalog using our unified catalog framework, we could use Lasso without running into performance issue.

Implicit regression

In the same manner after building the experimental matrix using the unified catalog framework, we can also perform the regression to find the sparse coefficient vector $\boldsymbol{\Xi}$ that satisfy the following equation :

$$\mathbf{0} = \boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})\boldsymbol{\Xi} \tag{5}$$

For this implicit regression we have implemented the method used in SINdy-PI (*3*), using the CVxPY library to solve the constrained optimization problem. We have however changed the solution identification phase to make it more efficient on our system. The original paper retrieve the sparse solution by looking at the column of the solution matrix with the lowest l0 norm. We have decided to implement a post-processing step that will look for cluster of homothetic column in the solution matrix. The main idea is that the true solution will be represented by multiple homothetic column in the solution matrix, because of the noise and the constrained optimization problem. By looking at the cluster of homothetic column we can average them to retrieve a robust solution. This decision has been motivated by the fact that in mechanical system, multiple independent sub system can exist, leading to multiple independent implicit equation. By looking only at the sparset columns we usually missed some of these independent equation.

TODO: Graph to explain solution clustering

mixed regression

In addition to the explicit and implicit regression method, we have also implemented a mixed regression method that can handle both explicit and implicit dynamic in the same time. This provides to the end user a unified framework for regression. The main idea is to split the experimental matrix into two part : one for the explicit dynamic and one for the implicit dynamic.

TODO: Create a graph to explain the force propagation

Results

Catalog generation

In order to compare regression that uses different base catalog it is important to generate equivalent catalog following a rule. For the xlsindy catalog we have generated all the combination of the following polynome : **TODO: list polynome**

Concerning the sindy catalog, we have decided to help it as much as possible and we have decided to choose the minimum polynome generate that lead to a catalog that include all the polynome formed by euler lagrange equation in the xlsindy case (we have the catalog of the mininum size that include the catalog resulted by the euler lagrange transform of the xlsindy catalog)**TODO: not really clear right now.....**

Experiment Results

In order to validate the capabilities of PY-XL-SINDY, we conducted a series of experiments on mechanical systems of increasing complexity. All data were generated using the MuJoCo physics simulator, with random time-varying external forces applied to ensure rich exploration of the state space. The generated datasets were then used to identify the governing equations using our framework, and the discovered models were validated by comparing their simulated trajectories against those from MuJoCo.

We have focused on three main systems:

1. Cart-pole

2. Double pendulum (point mass)

3. Double pendulum (point mass) on a cart

On these three systems, we have generated multiple trajectory with different friction coefficient, and different input forces profile. We have divided the input forces into implicit (no forces on the system, only initial condition), explicit (forces on every actuated joint) and mixed (forces on a subset of the joint). On this dataset we have added multiple level of gaussian noise to the position, velocity and acceleration. Finally we have run the different configuration of PY-XL-SINDY to identify the system. The same level of resources has been allocated to each configuration and a timeout of 1h has been set for each run. The list of framework is the following:

1. mixed catalog (Lagrange + classical) X mixed implicit/explicit solver

2. mixed catalog (Lagrange + classical) X explicit solver

3. mixed catalog (Lagrange + classical) X implicit solver

4. Lagrange catalog X mixed implicit/explicit solver

5. Lagrange catalog X explicit solver

6. Lagrange catalog X implicit solver

7. classical catalog X mixed implicit/explicit solver

8. classical catalog X explicit solver

9. classical catalog X implicit solver

After obtaining the different models, we have generated a validation trajectory with Mujoco and compared the trajectory of the discovered model with the one of Mujoco. Using this metric we have ranked the different configuration of PY-XL-SINDY.

We will now study in detail some of the results obtained from the different configuration before giving an overall performance comparison.

Explicit identification of a cart-pole

Other results

All the other experiments results are available on the visualisation website `https://eymeric65.github.io/py-xl-sindy-data-visualisation/`. On this website you can find the detailed results of each training and validation trajectory. There is also the coefficient of each discovered model.

Overall Performance Comparison

For each of the trajectory (with a fixed noise level) the different framework configuration are compared with each other, and ranked from best to worst. A special award is given if a configuration is the only one to find a valid model (able to create a validation trajectory that doesn't diverge) called "Wins without competition" (Wins WC). The results are summarized in Table 1 and ranked by number of 1st place wins.

Table 1: Overall Performance Ranking (by number of 1st place wins)

combo type	invalid	timeout	uncompleted	1st	2nd	2nd>	Wins WC	success rate
mixed x mixed (our)	25	0	20	59	12	0	45	0.612069
mixed x explicit (our)	14	0	17	53	8	0	42	0.663043
sindy x mixed	81	43	11	12	8	0	3	0.178571
sindy x explicit	61	16	11	12	8	0	3	0.217391
mixed x implicit (our)	1	0	1	10	2	2	6	0.875000
xlsindy x mixed	59	0	43	3	6	4	3	0.113043
xlsindy x explicit	38	0	43	3	3	5	3	0.119565
xlsindy x implicit	0	0	1	2	1	1	0	0.800000
sindy x implicit	24	23	0	0	0	0	0	0.000000

Our developped method (mixed catalog X mixed implicit/explicit solver) is the best performing one, winning 69 out of 116 trajectory (59.5% win rate) and being the only one to find a valid model on 26 trajectory. The second best method (mixed catalog X explicit solver) is also using the mixed catalog but only the explicit solver. The third best method (classical catalog X explicit solver) is using only the classical catalog, as it can been seen the greater size of catalog usually leads to more

timeouts. The main reason of why xlsindy is not performing well is because it is not able to handle well the friction term and majority of the experiment have friction.

Discussion

In this work, we presented PY-XL-SINDY, a novel framework that unifies and extends the capabilities of Sparse Identification of Nonlinear Dynamics. By introducing a modular, hybrid catalog system, we have bridged the gap between Lagrangian and Newtonian formulations, allowing for the discovery of more realistic physical models that include both conservative and dissipative forces. Our implementation of a robust implicit solver with automated post-processing expands the class of systems that can be identified to include those with complex constraints.

The integration with JAX for high-performance computing is a key practical advance, making it feasible to apply these discovery techniques to large-scale problems. Our end-to-end pipeline, which uses the MuJoCo simulator for data generation and validation, ensures that the discovered models are not just theoretically sound but are benchmarked against a realistic physics engine.

Limitations of our current framework include the need for the user to design the initial candidate library of functions, a common challenge in all SINDy-based methods. Future work will focus on automating this library construction process and integrating the high-performance dynamics models into model-based reinforcement learning algorithms, paving the way for agents that can not only learn to control a system but also discover its underlying physical laws. **TODO: don't talk about jax, it is too implementation related...**

Table 2: **Performance benchmark of JAX-based parallelized dynamics function.** Comparison of average computation time per simulation step for a double pendulum model. The JAX implementation is benchmarked on both CPU and GPU, showing orders-of-magnitude speedup when simulating large batches in parallel.

Implementation	Batch Size	Hardware	Avg. Time per Step (ms)
NumPy (sequential)	1	CPU	~0.15
JAX (sequential)	1	CPU	~0.16
JAX (parallelized)	10,000	GPU	~0.0002

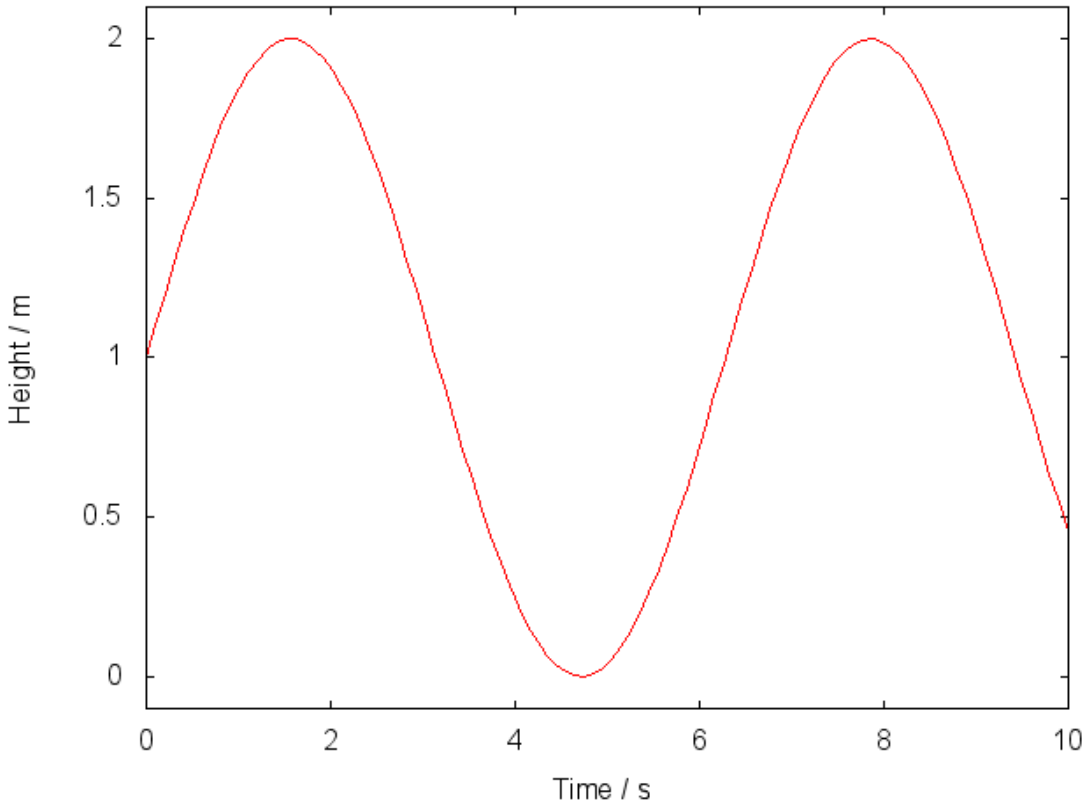

Figure 1: **Validation of a discovered hybrid model for a cart-pole with friction.** (**A**) Comparison of trajectories for the generalized coordinates (position and angle) between the ground-truth MuJoCo simulation (black dashed line) and the model discovered by PY-XL-SINDY (blue solid line). (**B**) Residuals between the MuJoCo and discovered model trajectories, showing high fidelity. (**C**) The discovered coefficient vector, showing sparse non-zero terms corresponding to the correct Lagrangian and friction functions.

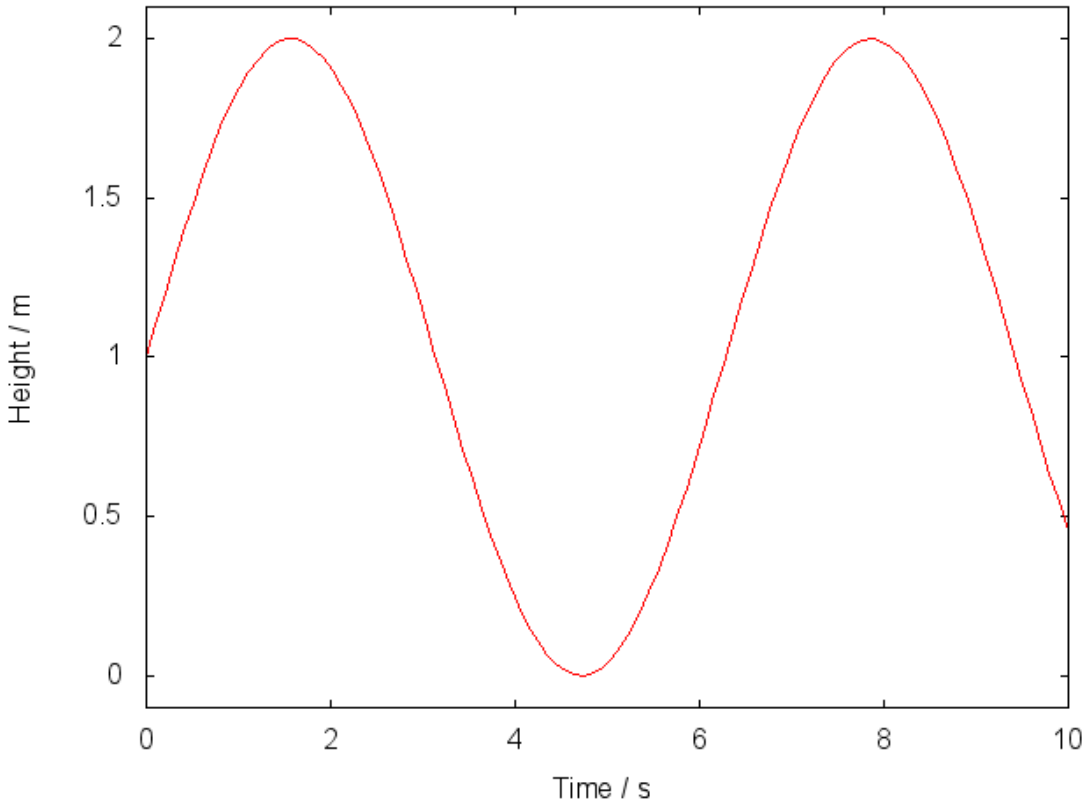

Figure 2: **Implicit identification of a double pendulum on a cart.** (**A**) The solution matrix 'X' from the constrained optimization problem. Most columns are sparse. (**B**) Post-processing identifies a cluster of homothetic column vectors (highlighted in red) that correspond to the true physical law. (**C**) The final, single solution vector extracted from the cluster, accurately representing the system's implicit dynamics.

References and Notes

1. S. L. Brunton, J. L. Proctor, J. N. Kutz, Discovering Governing Equations from Data: Sparse Identification of Nonlinear Dynamical Systems. Proceedings of the National Academy of Sciences **113** (15), 3932–3937 (2016), doi:10.1073/pnas.1517384113.

2. A. Purnomo, M. Hayashibe, Sparse Identification of Lagrangian for Nonlinear Dynamical Systems via Proximal Gradient Method. Scientific Reports **13** (1), 7919 (2023), doi: 10.1038/s41598-023-34931-0.

3. K. Kaheman, J. N. Kutz, S. L. Brunton, SINDy-PI: A Robust Algorithm for Parallel Implicit Sparse Identification of Nonlinear Dynamics. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **476** (2242), 20200279 (2020), doi:10.1098/rspa.2020.0279.

Acknowledgments

Funding: [TODO: List your funding sources here. Example: This work was supported by the JSPS Grant-in-Aid for Scientific Research (B) under Grant 18H01399.]

Author contributions: E.C. conceived the idea, developed the software, performed the experiments, and wrote the manuscript. A.C. assisted with... L.S. supervised the project... [TODO: Fill out contributions.]

Competing interests: The authors declare no competing interests.

Data and materials availability: All code for the PY-XL-SINDY framework and the scripts to reproduce the experiments are publicly available on GitHub at `https://github.com/Eymeric65/py-xl-sindy`. All data used in this study was generated using the provided code and the MuJoCo simulator. **TODO: add information about the .sh script used for generating the material.**

Supplementary materials

Materials and Methods

Supplementary Text

Figs. S1 to S2

Tables S1 to S2

Supplementary Materials for

A Unified Framework for Robust Discovery of Hybrid Physical Dynamics from Data TODO: change title

Eymeric Chauchat,* A. Collaborator, Lead Scientist

*Corresponding author. Email: [Your Email Here]

This PDF file includes:

Materials and Methods

Supplementary Text

Figures S1 to S2

Tables S1 to S2

Materials and Methods

The Unified Catalog Framework

The core of PY-XL-SINDY is its modular catalog system, defined in 'src/xlsindy/catalog.py'. The abstract base class 'CatalogCategory' defines a standard interface for different physical paradigms. We implemented three main categories:

- ***Lagrange*:** This category takes a library of candidate symbolic functions and constructs the experimental matrix by applying the Euler-Lagrange operator, $\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i}$.
- ***Classical*:** This category represents standard Newtonian or state-space models. It takes a library of functions of the state variables (e.g., polynomials, trigonometric functions) and constructs the corresponding columns in the experimental matrix directly.
- ***ExternalForces*:** This category explicitly models external forces, typically serving as the known right-hand-side of the final governing equation.

The 'CatalogRepartition' class combines instances of these categories into a single, cohesive experimental matrix $\mathbf{\Theta}$. This allows for the construction of a hybrid equation, for example $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \tau + \mathbf{F}_{friction}(\dot{\mathbf{q}})$, where the Lagrangian terms are handled by the 'Lagrange' catalog and the friction terms by the 'Classical' catalog.

Sparse Regression Techniques

The framework solves the sparse regression problem $\mathbf{b} = \mathbf{\Theta\Xi}$ to find the coefficient vector $\mathbf{\Xi}$. Two primary solvers are implemented in 'src/xlsindy/optimization.py':

- **Hard Thresholding (STLSQ):** An iterative method where a standard least-squares solution is computed, and then coefficients below a certain threshold are pruned. The process is repeated on the reduced library until the solution converges.
- **Lasso Regression:** Utilizes the L1 regularizer to promote sparsity, implemented via Scikit-learn's 'LassoCV' to automatically select the optimal regularization parameter α.

Implicit Dynamics Identification

For implicit systems, where no term can be isolated on the left-hand side, we seek a sparse vector $\boldsymbol{\Xi}$ in the null space of $\boldsymbol{\Theta}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = 0$. Following the robust approach of SINDy-PI (*?*), we solve a constrained optimization problem using ‘cvxpy‘. The problem is formulated as:

$$\min_{\mathbf{X}} ||\boldsymbol{\Theta}\mathbf{X} - \boldsymbol{\Theta}||_F + \lambda||\mathbf{X}||_1 \quad \text{s.t.} \quad \text{diag}(\mathbf{X}) = 0. \tag{S1}$$

The resulting solution matrix $\mathbf{X}$ ideally contains multiple column vectors that are sparse and homothetic (i.e., pointing in the same direction), representing the same physical law. Our post-processing algorithm in ‘_implicit_post_treatment‘ clusters these vectors based on angular similarity and extracts the principal direction via SVD to yield a single, robust solution vector.

Data Generation from MuJoCo Simulator

All training and validation data were generated using the MuJoCo physics simulator. The scripts in the ‘util/‘ directory manage this process.

1. **Data Generation (‘mujoco_generate_data.py‘):** A simulation is run for a specified duration (‘max_time‘). At each step, a randomly generated, time-varying external force is applied to the system’s actuators. The force is generated using a sum of sinusoids with varying frequencies and amplitudes to ensure sufficient exploration of the state space.

2. **Coordinate Transformation (‘xlsindy_gen.py‘ files):** MuJoCo often uses a different set of generalized coordinates than the standard textbook Lagrangian formulation (e.g., cumulative angles). Each experiment folder contains a ‘mujoco_transform‘ function to convert the simulator’s output (‘qpos‘, ‘qvel‘, ‘qacc‘) into the coordinate system used for the symbolic model.

3. **Data Storage:** The generated time-series data is saved to ‘.npz‘ files, while simulation metadata is saved to corresponding ‘.json‘ files. A validation database (‘validation_database.pkl‘) is created by subsampling from all generated experiments to test the generalization of discovered models.