Symbolic regression

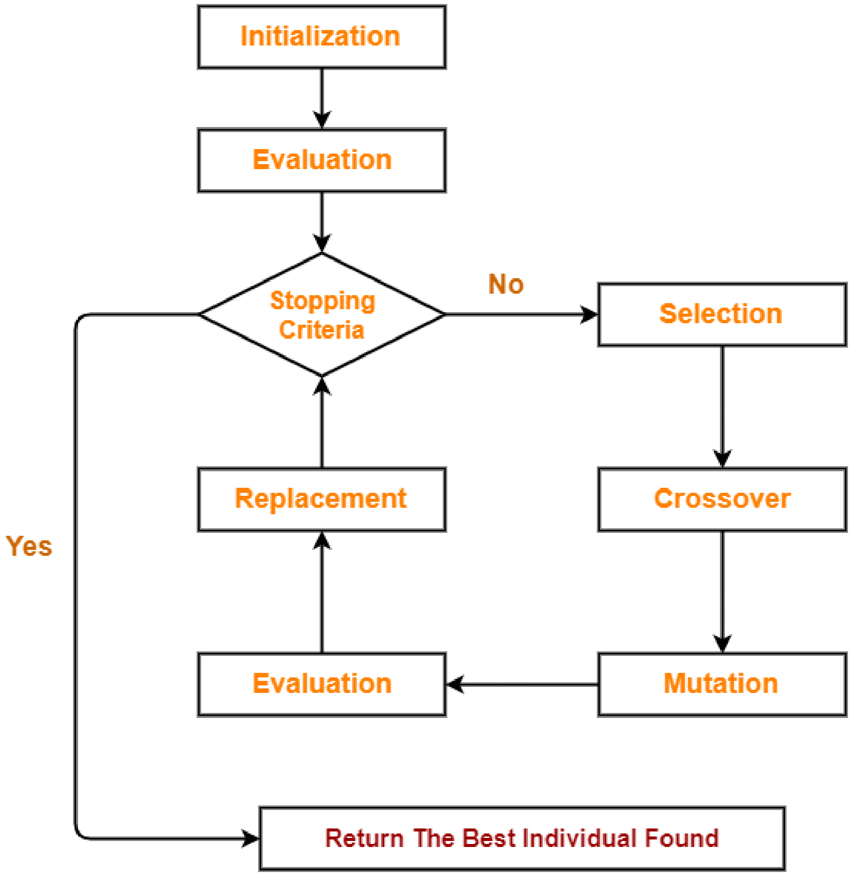
What we want now?

Analytical soluto

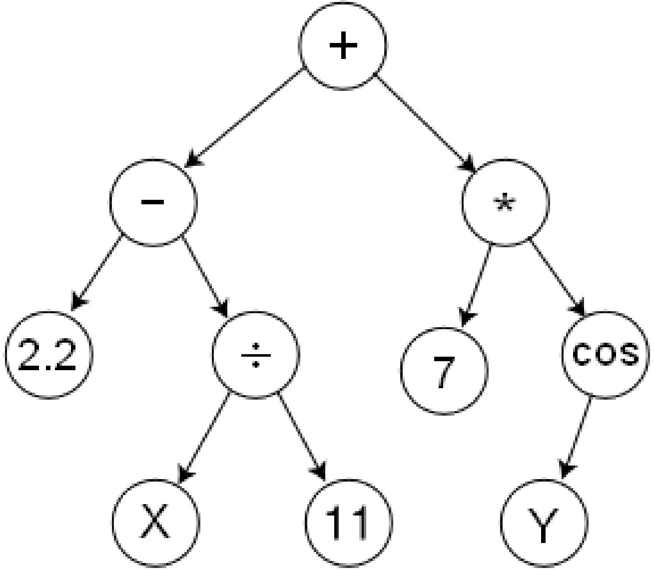
Genetic algorithms

Simple Genetic Algorithm

STEP 1. INITIALIZATION
Generate initial population $\mathcal{P}$ at random or with prior knowledge
STEP 2. FITNESS EVALUATION
Evaluate the fitness for all individuals in $\mathcal{P}$
STEP 3. SELECTION
Select a set of promising candidates $\mathcal{S}$ from $\mathcal{P}$
STEP 4. CROSSOVER
Apply crossover to the mating pool $\mathcal{S}$ for generating a set of offspring $\mathcal{O}$
STEP 5. MUTATION
Apply mutation to the offspring set $\mathcal{O}$ for obtaining its perturbed set $\mathcal{O}'$
STEP 6. REPLACEMENT
Replace the current population $\mathcal{P}$ with the set of offspring $\mathcal{O}'$
STEP 7. TERMINATION
If the termination criteria are not met, go to STEP 2

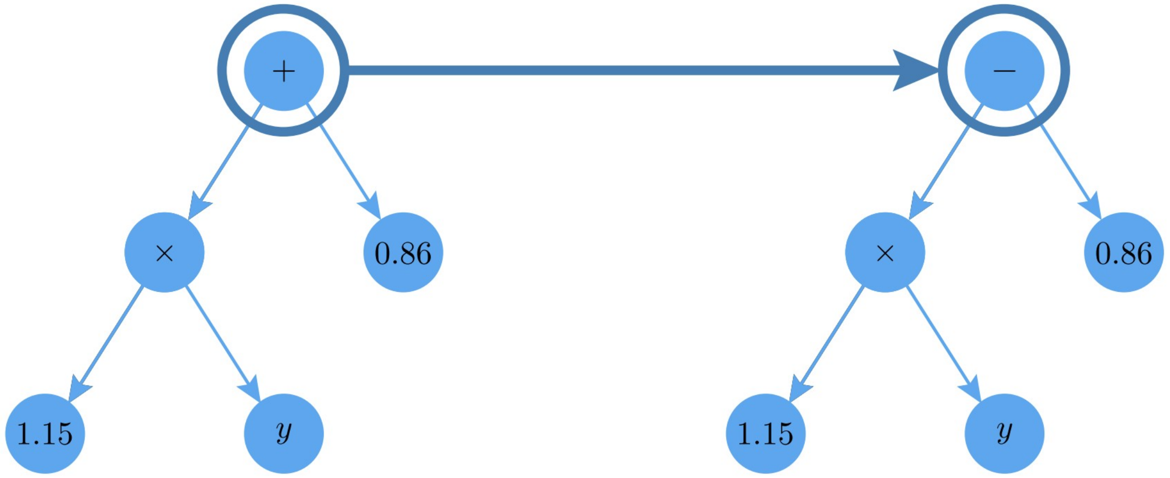


Arithmetic trees



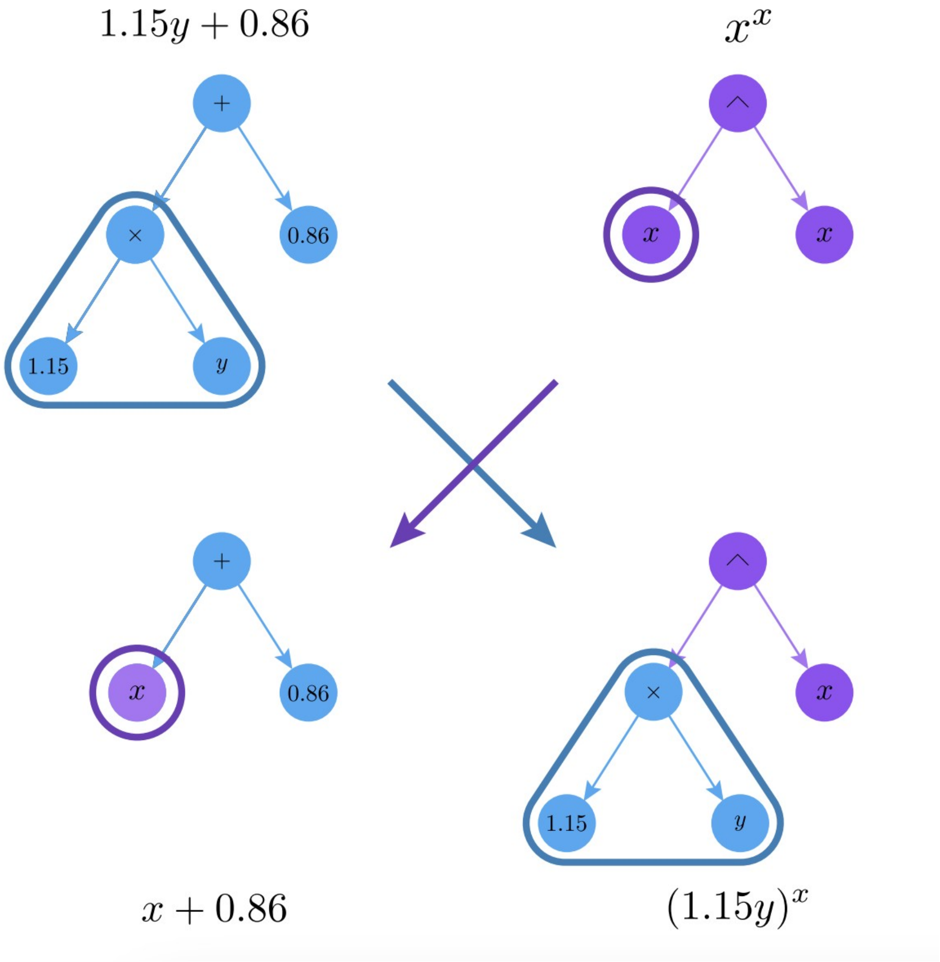
$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$

Mutation

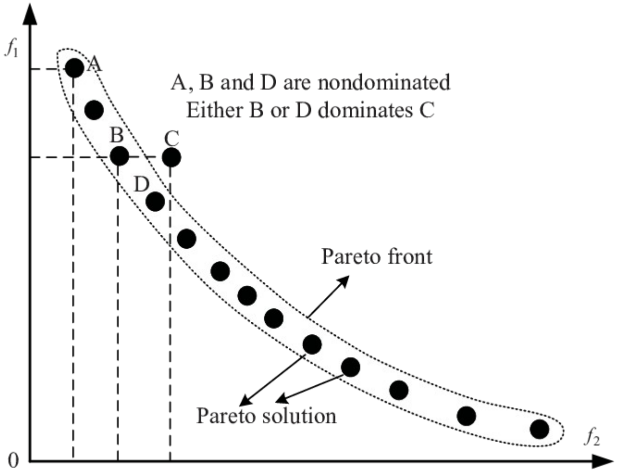


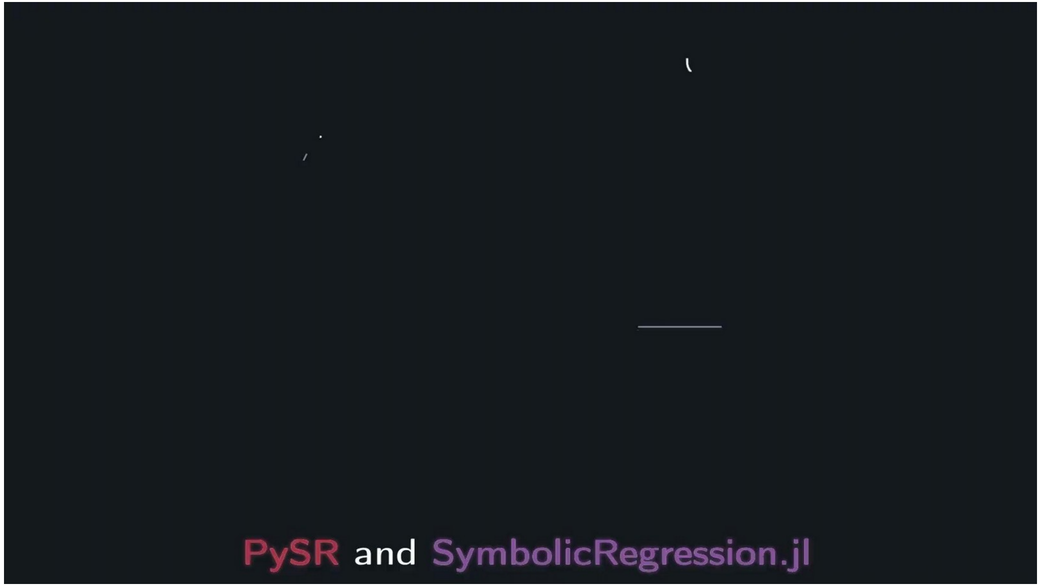
$$1.15y + 0.86 \qquad\qquad 1.15y - 0.86$$

Crossover



Fitness function





$$\ell(E) = \ell_{\text{pred}}(E) + (\text{parsimony}) \cdot C(E)$$

PySR

PySR & SymbolicRegression.jl github.com/MilesCranmer/pysr_paper

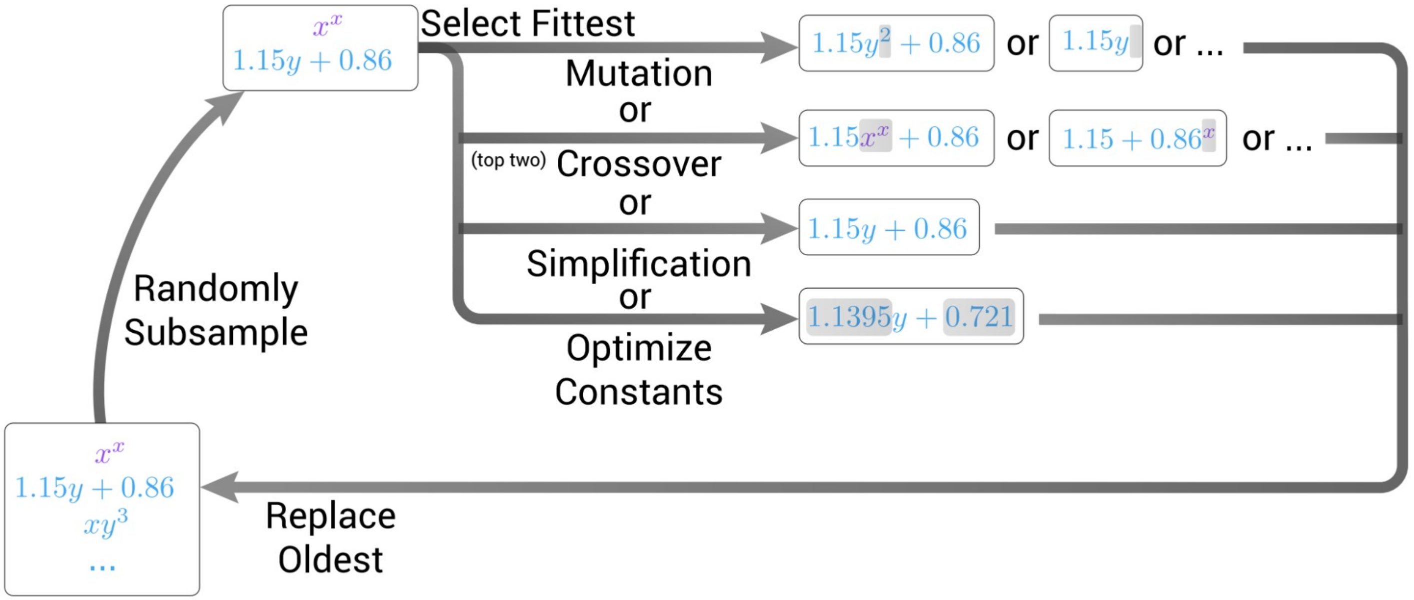
Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl

Miles Cranmer[1,2]

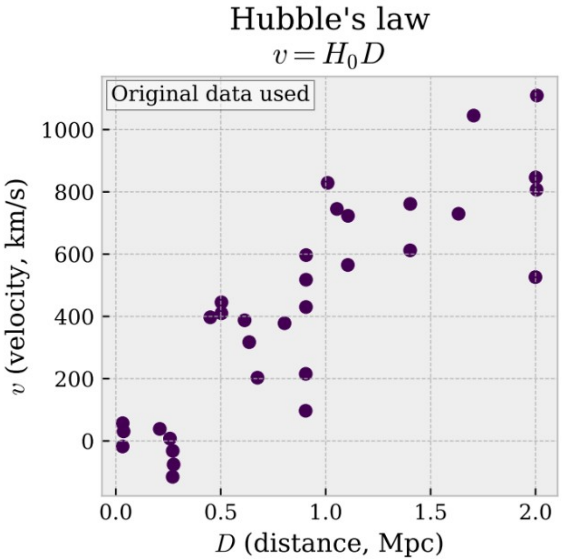
[1] *Princeton University, Princeton, NJ, USA*
[2] *Flatiron Institute, New York, NY, USA*

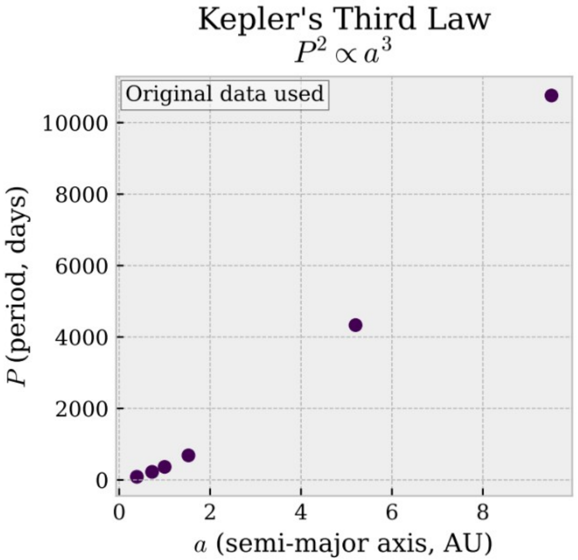
May 2, 2023

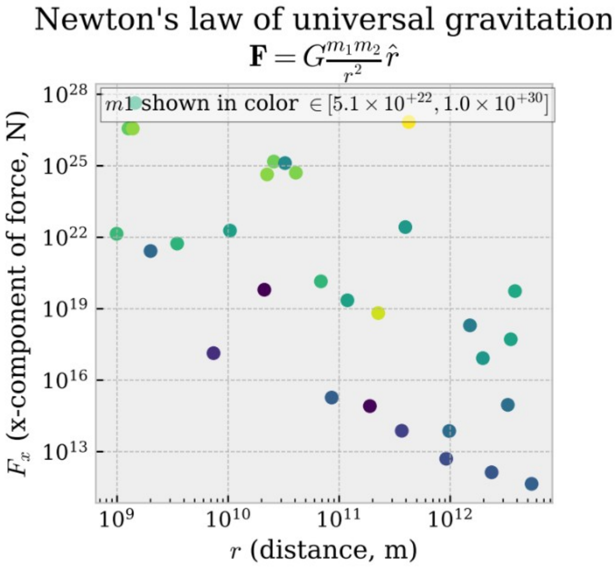
The symbolic regression loop



Use cases


Hubble's law
$v = H_0 D$
Original data used
v (velocity, km/s)
D (distance, Mpc)


Kepler's Third Law
$P^2 \propto a^3$
Original data used
P (period, days)
a (semi-major axis, AU)


Newton's law of universal gravitation
$\mathbf{F} = G\frac{m_1 m_2}{r^2}\hat{r}$
$m1$ shown in color $\in [5.1 \times 10^{+22}, 1.0 \times 10^{+30}]$
F_x (x-component of force, N)
r (distance, m)

Benchmark

	PySR	Operon	DSR	EQL	QLattice	SR-Transformer
Hubble	5/5 (5, 0, 0, 0)	0/5 (0, 5, 0, 0)	1/5 (1, 0, 4, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 5, 0, 0)	0/5 (0, 0, 0, 5)
Kepler	5/5 (5, 0, 0, 0)	0/5 (0, 5, 0, 0)	4/5 (4, 1, 0, 0)	0/5 (0, 0, 2, 3)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
Newton	5/5 (5, 0, 0, 0)	1/5 (1, 2, 0, 2)	1/5 (1, 0, 4, 0)	0/5 (0, 0, 5, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
Planck	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 1, 4)	0/5 (0, 0, 5, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
Leavitt	5/5 (5, 0, 0, 0)	0/5 (0, 0, 0, 5)	5/5 (5, 0, 0, 0)	0/5 (0, 0, 5, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
Schechter	5/5 (5, 0, 0, 0)	5/5 (5, 0, 0, 0)	5/5 (5, 0, 0, 0)	0/5 (0, 0, 4, 1)	5/5 (5, 0, 0, 0)	0/5 (0, 0, 0, 5)
Bode	5/5 (5, 0, 0, 0)	3/5 (3, 0, 0, 2)	1/5 (1, 0, 3, 1)	0/5 (0, 0, 4, 1)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
Ideal Gas	5/5 (5, 0, 0, 0)	0/5 (0, 0, 0, 5)	5/5 (5, 0, 0, 0)	0/5 (0, 0, 4, 1)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)
Rydberg	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 5, 0)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)	0/5 (0, 0, 0, 5)

Code example

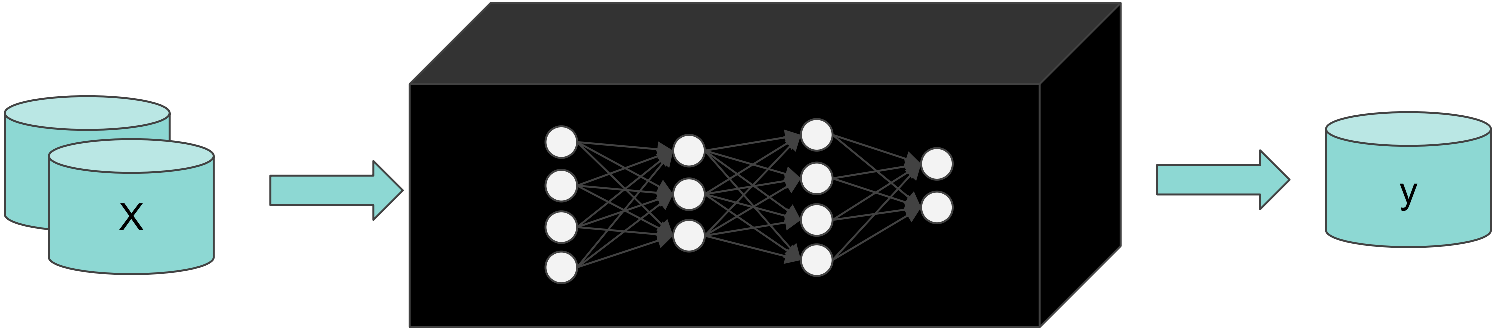
We can use custom operators!

```
op = "special(x, y) = cos(x) * (x + y)"
model = PySRRegressor(binary_operators=[op])
```

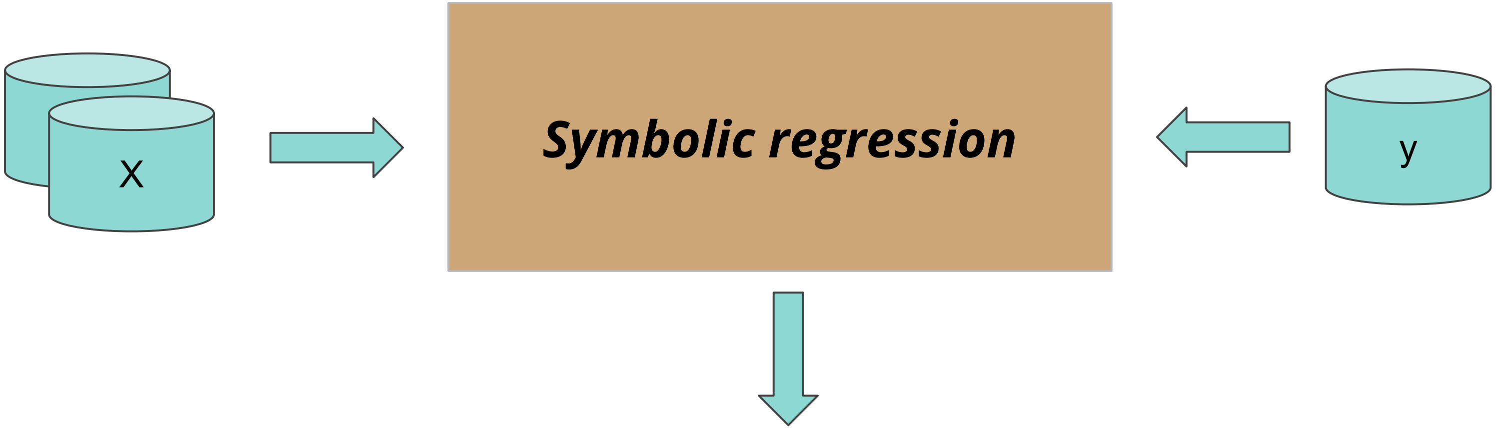
[PySRRegressor Reference - PySR]

```
from pysr import PySRRegressor

# Declare search options:
model = PySRRegressor(
  model_selection="best",
  unary_operators=["cos", "sin"],
  binary_operators=["+", "-", "/", "*"],
)

# Load the data
X, y = load_data()
# X shape: (n_rows, n_features)
# y shape: (n_rows) or (n_rows, n_targets)

# Run the search:
model.fit(X, y)

# View the discovered expressions:
print(model)

# Evaluate, using the 5th expression along
# the Pareto front:
y_predicted = model.predict(X, 5)
# (Without specify `5`, it will select an
↪  expression
# which balances complexity and error)
```

Coupling with neural networks



Coupling with neural networks



complexity	pick	score	equation	loss
1	0	0.000000	4.4324794	42.354317
3	1	1.255691	(x0 * x0)	3.437307
5	2	0.011629	((x0 * x0) + -0.28087974)	3.358285
	3	0.897855	((x0 * x0) + cos(x3))	1.368308

Exercises

1. Add Gaussian noise to `y_sr` before passing it to PySR. How robust is the symbolic regression?
2. Change the PINN to solve the Burgers' equation (from Day 1) and try to distill the shockwave equation.
3. Remove "sin" from the unary_operators list. Can PySR approximate the sine wave using Taylor expansion terms (polynomials)?