JuliaHEP 2023 Workshop

Using Symbolic Regression in Julia to find analytic functions to model low statistics

Guru Reddy

8th Nov, 2023

Introduction

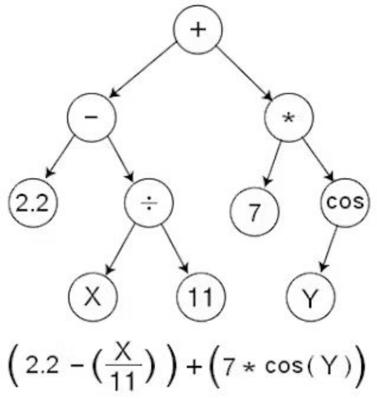
- In High Energy Physics (HEP) analyses, limited statistics in Monte Carlo simulations sometimes pose a challenge because a significant portion of events are discarded to define a region of interest.
- This can compromise the reliability of computing confidence intervals using maximum likelihood techniques.
- Generating more events is not always an option but we can easily fit the low statistics into a function which can be used to model the samples in our region of interest.
- Finding the optimal fit isn't always straightforward through guesswork alone. That's why we utilize symbolic regression approach to obtain an approximate function.

What is Symbolic Regression

- A computational technique used in machine learning and optimization.
- It involves finding a mathematical expression or formula that best describes the relationship between input variables and the output variable in a given dataset.
- Unlike traditional regression techniques that typically rely on predefined functional forms (e.g., linear, polynomial), symbolic regression searches for an equation without making any assumptions about the underlying mathematical structure. This makes it particularly useful for discovering complex and non-linear relationships in data.
- Symbolic regression explores a space of mathematical expressions, combining basic mathematical operations (e.g. addition, subtraction, multiplication, division) and functions (e.g. logarithm, exponential) to construct models.
- It uses optimization algorithms (such as genetic algorithm) to iteratively improve the fit of the generated expressions to the data.

Genetic algorithms

- In nature, individuals of a population compete for basic resources. Those with superior survival capabilities have higher probabilities of producing descendants.
- Emulating nature, the genetic algorithm (GA) methods replicate the competitive and selective processes observed in nature. They apply the principles of natural selection and reproduction to optimization.
- The GAs takes different predefined genes (functions and operators) and makes individuals. Each individual represents a possible solution to a problem.
- The quality of each individual (possible solution) is evaluated in terms of a fitness function (loss). A higher probability to have descendants is assigned to individuals with better fitness functions.


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

Symbolic regression in Julia

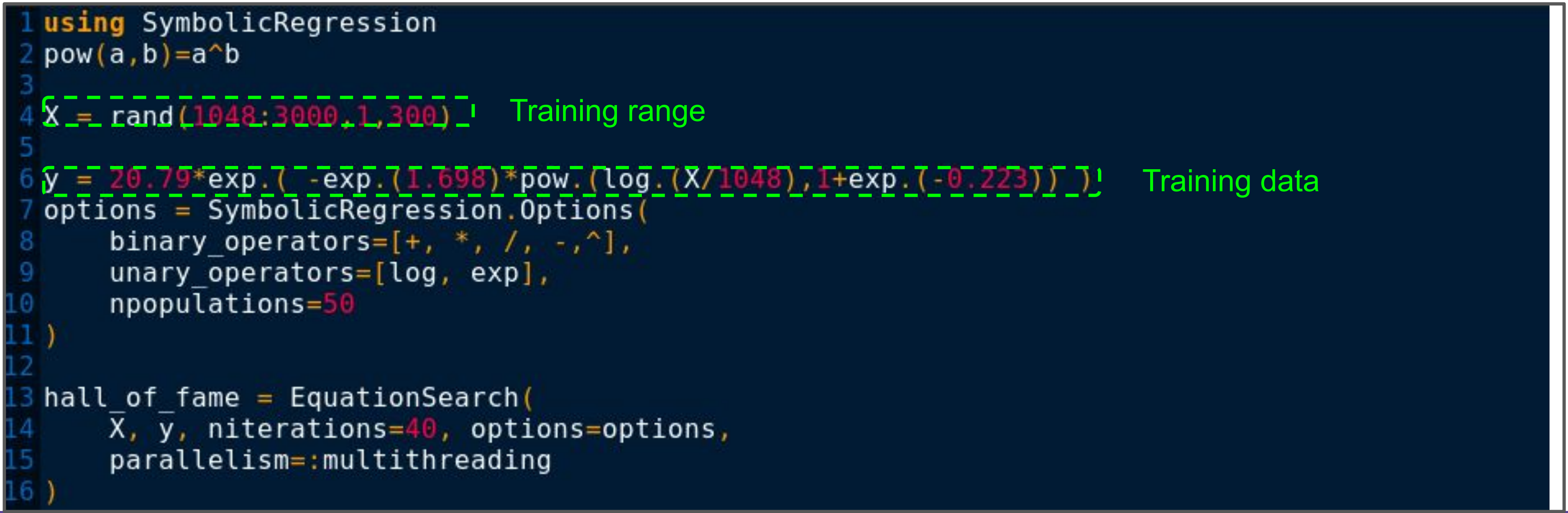
- There are many python packages (eg. fastfx, gplearn, PySr, etc) available to perform symbolic regression.
- I used SymbolicRegression.jl package in Julia due to its versatility and ease of use.

A simple example!

(made on the tail of my distribution)

Symbolic regression in Julia

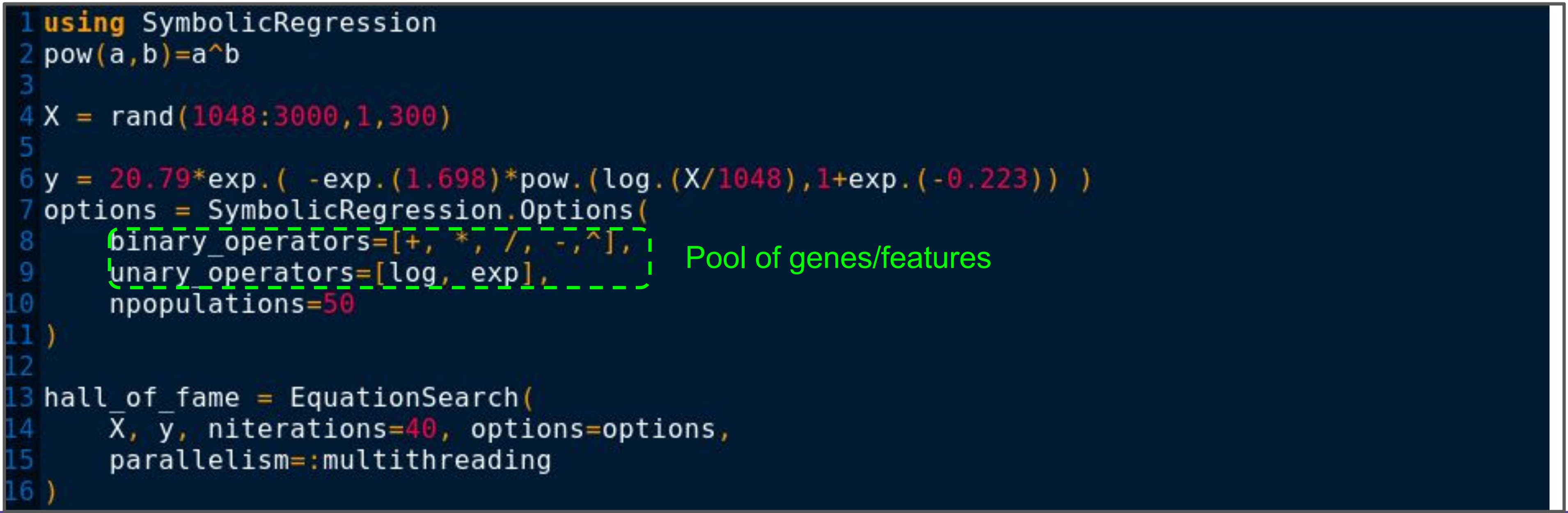
- There are many python packages (eg. fastfx, gplearn, PySr, etc) available to perform symbolic regression.
- I used SymbolicRegression.jl package in Julia due to its versatility and ease of use.



```julia
using SymbolicRegression
pow(a,b)=a^b

X = rand(1048:3000,1,300)

y = 20.79*exp.( -exp.(1.698)*pow.(log.(X/1048),1+exp.(-0.223)) )
options = SymbolicRegression.Options(
    binary_operators=[+, *, /, -,^],
    unary_operators=[log, exp],
    npopulations=50
)

hall_of_fame = EquationSearch(
    X, y, niterations=40, options=options,
    parallelism=:multithreading
)
```

Symbolic regression in Julia

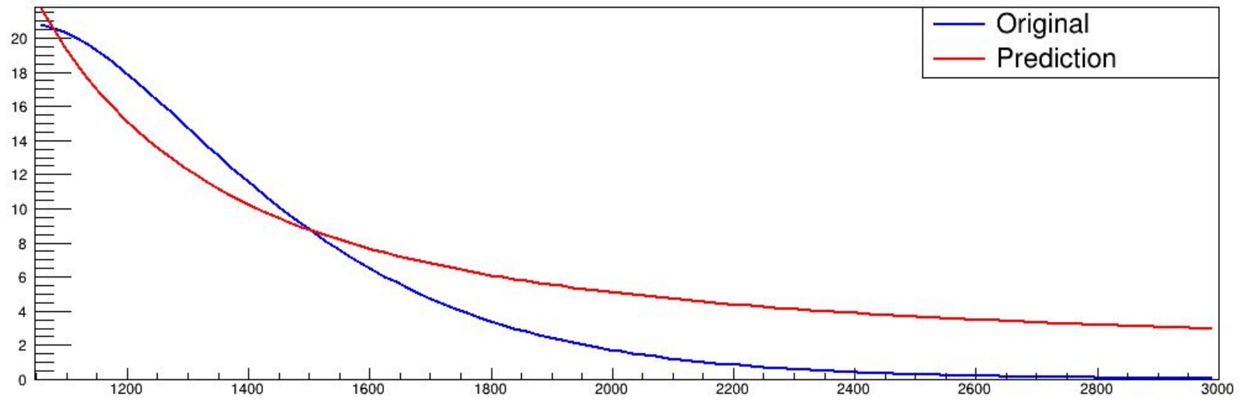
- There are many python packages (eg. fastfx, gplearn, PySR, etc) available to perform symbolic regression.
- I used SymbolicRegression.jl package in Julia due to its versatility and ease of use.



```julia
using SymbolicRegression
pow(a,b)=a^b

X = rand(1048:3000,1,300)

y = 20.79*exp.( -exp.(1.698)*pow.(log.(X/1048),1+exp.(-0.223)) )
options = SymbolicRegression.Options(
    binary_operators=[+, *, /, -,^],
    unary_operators=[log, exp],
    npopulations=50
)

hall_of_fame = EquationSearch(
    X, y, niterations=40, options=options,
    parallelism=:multithreading
)
```

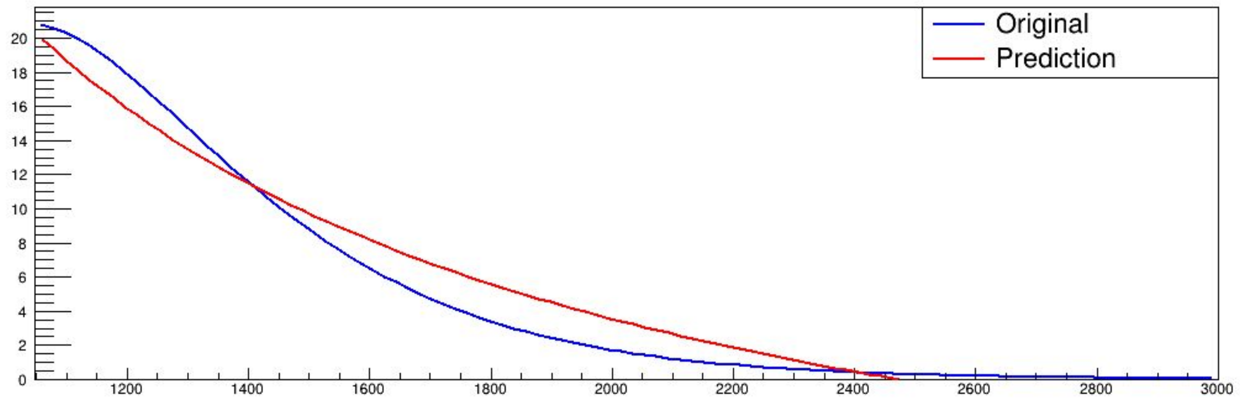
Example result

```
Complexity  Loss           Score     Equation
1         4.101e+01  -1.000e-10  5.0821548832161545
3         2.200e+01  3.115e-01  (11806.401291294773 / x1)
4         7.870e+00  1.028e+00  exp(3257.7993250962613 / x1)
5         2.338e+00  1.214e+00  ((37007.8870357095 / x1) - 14.9656066544245)
6         1.318e+00  5.730e-01  (exp(3603.8010947077087 / x1) + -3.5535019191799138)
7         8.183e-01  4.767e-01  ((3.112070405674172e7 / (x1 * x1)) - 4.929115552456605)
9         7.750e-01  2.720e-02  (((2.803973687249781e7 / (x1 * x1)) ^ 1.0322984835484592) - 4.660739584678003)
11        5.516e-01  1.700e-01  ((((x1 * -21902.74178635805) + 4.900886833153066e7) / (x1 * x1)) - -1.1814753786068186)
12        1.350e-01  1.407e+00  (exp(sin(log(x1 / 1.884273308322849) / 0.31213484385144724) - -0.6128988723018063) ^ 1.8420235679110728)
14        5.903e-02  4.137e-01  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) + -0.2911449618737346) ^ 1.7119499720559461)
16        5.842e-02  5.223e-03  ((exp(sin(log(x1 / (2.1255287562098677 / x1)) / -1.179208922104723) / 0.5486514363188524) ^ 1.7242316038328118) + -0.6557802132348866)
17        3.079e-02  6.404e-01  ((exp(sin(log((x1 ^ exp(0.002987539148310622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
-0.5337585084045818)
18        3.074e-02  1.748e-03  ((exp(sin(log((x1 ^ exp(0.002987539148310622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
sin(-0.559505775722691))
20        2.686e-02  6.741e-02  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5486514363188524) ^ 1.6971302727139999) - sin(log((x1 / 2.1255287562098677)
^ 2.1273824593164528)))
```



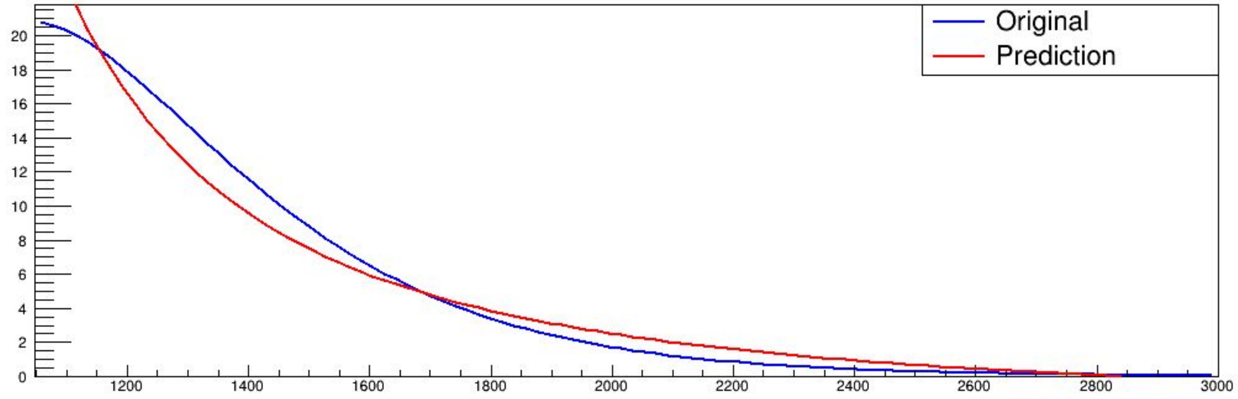
Example result

```
Complexity  Loss           Score     Equation
1         4.101e+01  -1.000e-10  5.0821548832161545
3         2.200e+01  3.115e-01  (11806.401291294773 / x1)
4         7.870e+00  1.028e+00  exp(3257.7993250962613 / x1)
5         2.338e+00  1.214e+00  ((37007.8870357095 / x1) - 14.9656066544245)
6         1.318e+00  5.730e-01  (exp(3603.8010947077087 / x1) + -3.5535019191799138)
7         8.183e-01  4.767e-01  ((3.112070405674172e7 / (x1 * x1)) - 4.929115552456605)
9         7.750e-01  2.720e-02  (((2.803973687249781e7 / (x1 * x1)) ^ 1.0322984835484592) - 4.660739584678003)
11        5.516e-01  1.700e-01  ((((x1 * -21902.74178635805) + 4.900886833153066e7) / (x1 * x1)) - -1.1814753786068186)
12        1.350e-01  1.407e+00  (exp(sin(log(x1 / 1.884273308322849) / 0.31213484385144724) - -0.6128988723018063) ^ 1.8420235679110728)
14        5.903e-02  4.137e-01  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) + -0.2911449618737346) ^ 1.7119499720559461)
16        5.842e-02  5.223e-03  ((exp(sin(log(x1 / (2.1255287562098677 / x1)) / -1.179208922104723) / 0.5486514363188524) ^ 1.7242316038328118) + -0.6557802132348866)
17        3.079e-02  6.404e-01  ((exp(sin(log((x1 ^ exp(0.0029875391483106 22)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
-0.5337585084045818)
18        3.074e-02  1.748e-03  ((exp(sin(log((x1 ^ exp(0.002987539148310622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
sin(-0.559505775722691))
20        2.686e-02  6.741e-02  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5486514363188524) ^ 1.6971302727139999) - sin(log((x1 / 2.1255287562098677)
^ 2.1273824593164528)))
```



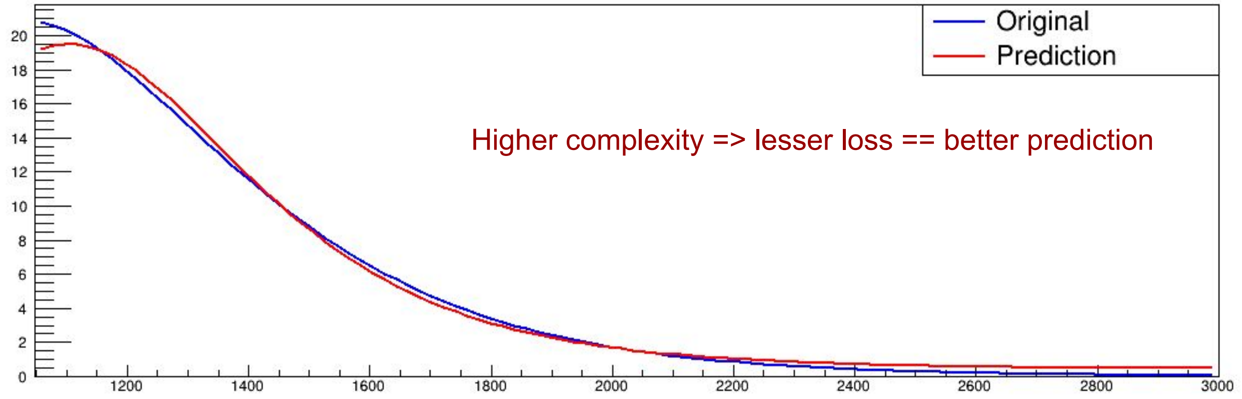
Example result

```
Complexity  Loss           Score     Equation
1        4.101e+01  -1.000e-10  5.0821548832161545
3        2.200e+01  3.115e-01  (11806.401291294773 / x1)
4        7.870e+00  1.028e+00  exp(3257.7993250962613 / x1)
5        2.338e+00  1.214e+00  ((37007.8870357095 / x1) - 14.9656066544245)
6        1.318e+00  5.730e-01  (exp(3603.8010947077087 / x1) + -3.5535019191799138)
7        8.183e-01  4.767e-01  ((3.112070405674172e7 / (x1 * x1)) - 4.929115552456605)
9        7.750e-01  2.720e-02  (((2.8039736872497811e7 / (x1 * x1)) ^ 1.0322984835484592) - 4.660739584678003)
11       5.516e-01  1.700e-01  ((((x1 * -21902.74178635805) + 4.900886833153066e7) / (x1 * x1)) - -1.1814753786068186)
12       1.350e-01  1.407e+00  (exp(sin(log(x1 / 1.884273308322849) / 0.31213484385144724) - -0.6128988723018063) ^ 1.8420235679110728)
14       5.903e-02  4.137e-01  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) + -0.2911449618737346) ^ 1.7119499720559461)
16       5.842e-02  5.223e-03  ((exp(sin(log(x1 / (2.1255287562098677 / x1)) / -1.179208922104723) / 0.5486514363188524) ^ 1.7242316038328118) + -0.6557802132348866)
17       3.079e-02  6.404e-01  ((exp(sin(log((x1 ^ exp(0.0029875391483106 22)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
-0.5337585084045818)
18       3.074e-02  1.748e-03  ((exp(sin(log((x1 ^ exp(0.002987539148310622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
sin(-0.559505775722691))
20       2.686e-02  6.741e-02  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5486514363188524) ^ 1.6971302727139999) - sin(log((x1 / 2.1255287562098677)
^ 2.1273824593164528)))
```



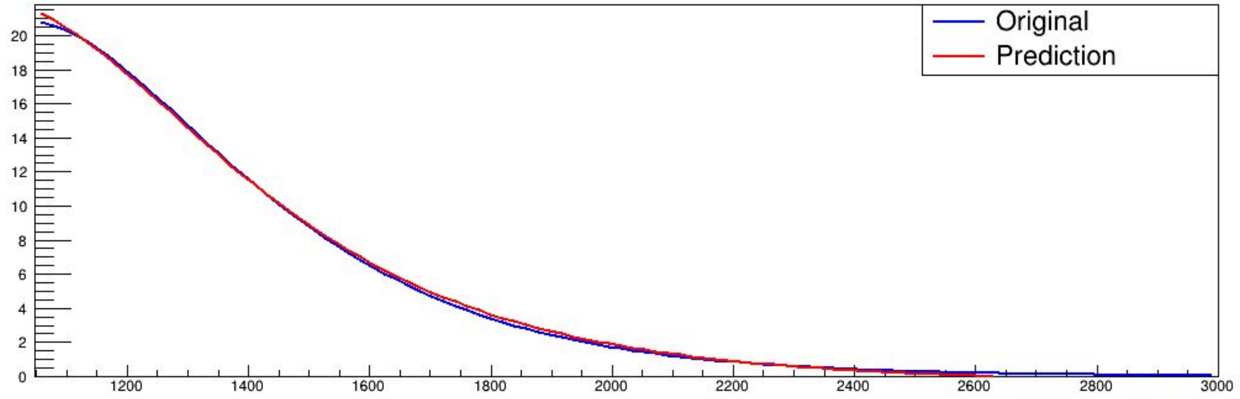
Example result

```
Complexity  Loss           Score     Equation
1         4.101e+01  -1.000e-10  5.0821548832161545
3         2.200e+01  3.115e-01  (11806.401291294773 / x1)
4         7.870e+00  1.028e+00  exp(3257.7993250962613 / x1)
5         2.338e+00  1.214e+00  ((37007.8870357095 / x1) - 14.9656066544245)
6         1.318e+00  5.730e-01  (exp(3603.8010947077087 / x1) + -3.5535019191799138)
7         8.183e-01  4.767e-01  ((3.112070405674172e7 / (x1 * x1)) - 4.929115552456605)
9         7.750e-01  2.720e-02  (((2.803973687249781e7 / (x1 * x1)) ^ 1.0322984835484592) - 4.660739584678003)
11        5.516e-01  1.700e-01  ((((x1 * -21902.74178635805) + 4.900886833153066e7) / (x1 * x1)) - -1.1814753786068186)
12        1.350e-01  1.407e+00  (exp(sin(log(x1 / 1.884273308322849) / 0.31213484385144724) - -0.6128988723018063) ^ 1.8420235679110728)
14        5.903e-02  4.137e-01  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) + -0.2911449618737346) ^ 1.7119499720559461)
16        5.842e-02  5.223e-03  ((exp(sin(log(x1 / (2.1255287562098677 / x1)) / -1.179208922104723) / 0.5486514363188524) ^ 1.7242316038328118) + -0.6557802132348866)
17        3.079e-02  6.404e-01  ((exp(sin(log((x1 ^ exp(0.002987539148310622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
-0.5337585084045818)
18        3.074e-02  1.748e-03  ((exp(sin(log((x1 ^ exp(0.002987539148310622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
sin(-0.559505775722691))
20        2.686e-02  6.741e-02  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5486514363188524) ^ 1.6971302727139999) - sin(log((x1 / 2.1255287562098677)
^ 2.1273824593164528)))
```



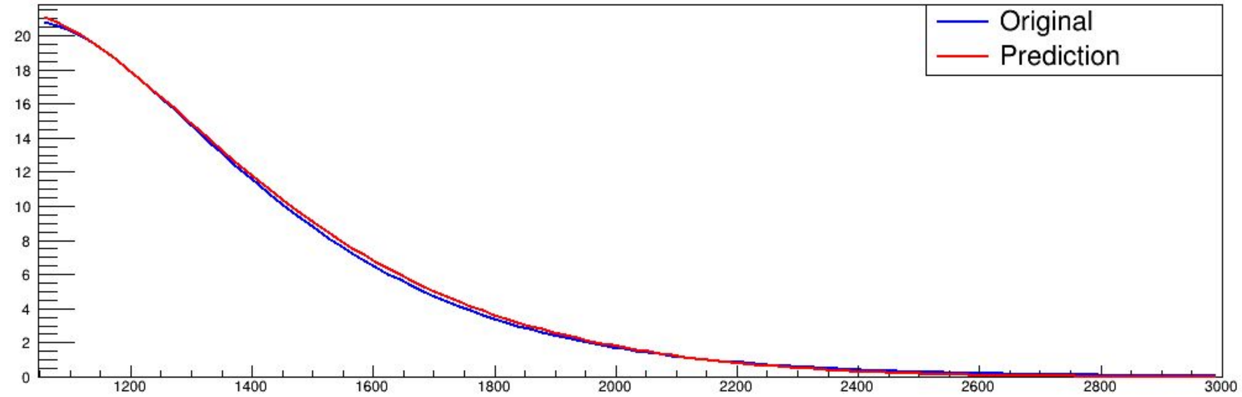
Example result

```
Complexity  Loss           Score     Equation
1         4.101e+01  -1.000e-10  5.0821548832161545
3         2.200e+01  3.115e-01  (11806.401291294773 / x1)
4         7.870e+00  1.028e+00  exp(3257.7993250962613 / x1)
5         2.338e+00  1.214e+00  ((37007.8870357095 / x1) - 14.9656066544245)
6         1.318e+00  5.730e-01  (exp(3603.8010947077087 / x1) + -3.5535019191799138)
7         8.183e-01  4.767e-01  ((3.112070405674172e7 / (x1 * x1)) - 4.929115552456605)
9         7.750e-01  2.720e-02  (((2.803973687249781e7 / (x1 * x1)) ^ 1.0322984835484592) - 4.660739584678003)
11        5.516e-01  1.700e-01  ((((x1 * -21902.74178635805) + 4.900886833153066e7) / (x1 * x1)) - -1.1814753786068186)
12        1.350e-01  1.407e+00  (exp(sin(log(x1 / 1.884273308322849) / 0.31213484385144724) - -0.6128988723018063) ^ 1.8420235679110728)
14        5.903e-02  4.137e-01  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) + -0.2911449618737346) ^ 1.7119499720559461)
16        5.842e-02  5.223e-03  ((exp(sin(log(x1 / (2.1255287562098677 / x1)) / -1.179208922104723) / 0.5486514363188524) ^ 1.7242316038328118) + -0.6557802132348866)
17        3.079e-02  6.404e-01  ((exp(sin(log((x1 ^ exp(0.002987539148310622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
-0.5337585084045818)
18        3.074e-02  1.748e-03  ((exp(sin(log((x1 ^ exp(0.002987539148310622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
sin(-0.559505775722691))
20        2.686e-02  6.741e-02  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5486514363188524) ^ 1.6971302727139999) - sin(log((x1 / 2.1255287562098677)
^ 2.1273824593164528)))
```

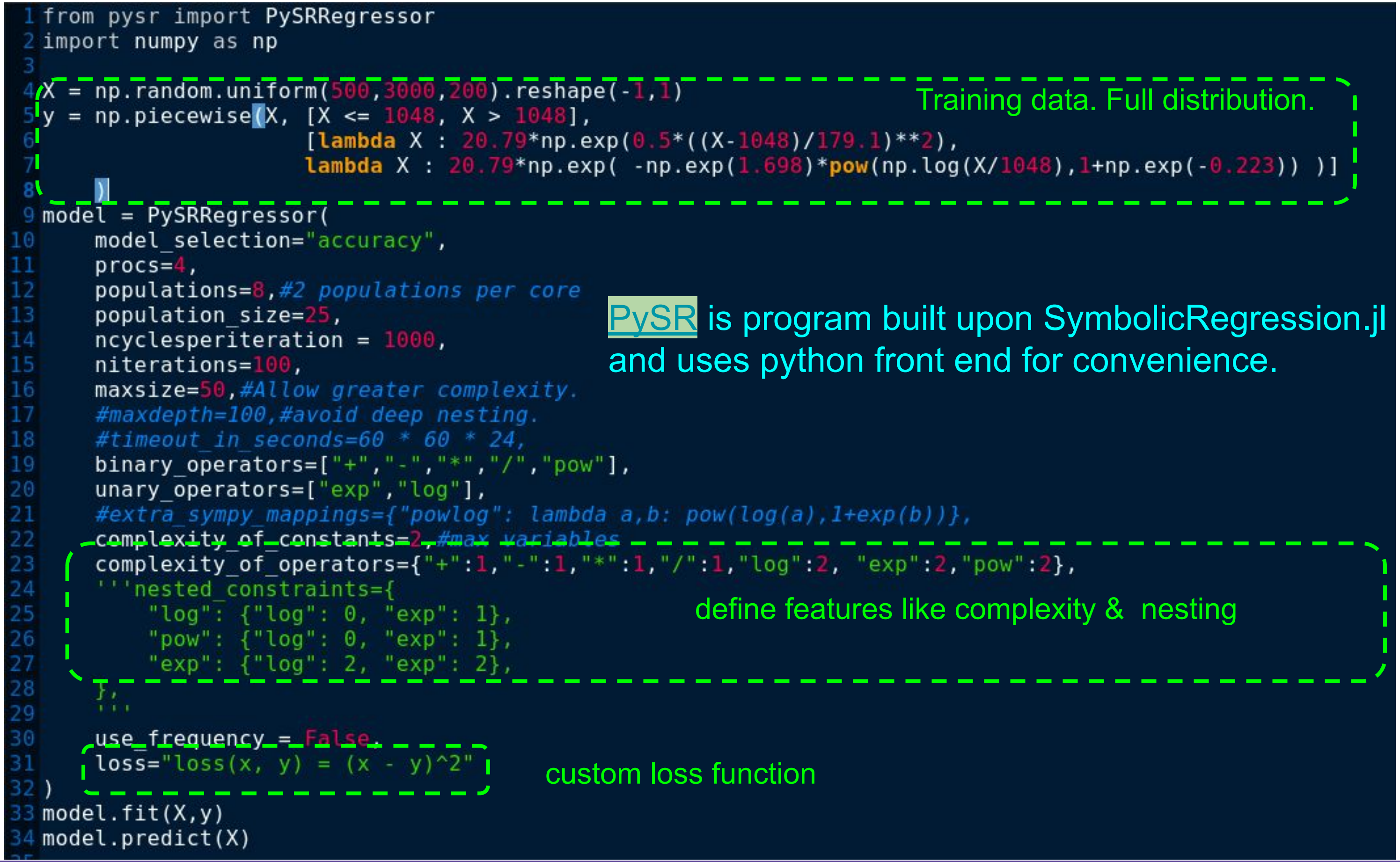


Example result

```
Complexity  Loss           Score     Equation
1         4.101e+01  -1.000e-10  5.0821548832161545
3         2.200e+01  3.115e-01  (11806.401291294773 / x1)
4         7.870e+00  1.028e+00  exp(3257.7993250962613 / x1)
5         2.338e+00  1.214e+00  ((37007.8870357095 / x1) - 14.9656066544245)
6         1.318e+00  5.730e-01  (exp(3603.8010947077087 / x1) + -3.5535019191799138)
7         8.183e-01  4.767e-01  ((3.112070405674172e7 / (x1 * x1)) - 4.929115552456605)
9         7.750e-01  2.720e-02  (((2.803973687249781e7 / (x1 * x1)) ^ 1.0322984835484592) - 4.660739584678003)
11        5.516e-01  1.700e-01  ((((x1 * -21902.74178635805) + 4.900886833153066e7) / (x1 * x1)) - -1.1814753786068186)
12        1.350e-01  1.407e+00  (exp(sin(log(x1 / 1.884273308322849) / 0.31213484385144724) - -0.6128988723018063) ^ 1.8420235679110728)
14        5.903e-02  4.137e-01  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) + -0.2911449618737346) ^ 1.7119499720559461)
16        5.842e-02  5.223e-03  ((exp(sin(log(x1 / (2.1255287562098677 / x1)) / -1.179208922104723) / 0.5486514363188524) ^ 1.7242316038328118) + -0.6557802132348866)
17        3.079e-02  6.404e-01  ((exp(sin(log((x1 ^ exp(0.0029875391483106622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
-0.5337585084045818)
18        3.074e-02  1.748e-03  ((exp(sin(log((x1 ^ exp(0.0029875391483106622)) / 2.1255287562098677) / -0.559505775722691) / 0.5451910029780047) ^ 1.6971302727139999) +
sin(-0.559505775722691))
20        2.686e-02  6.741e-02  ((exp(sin(log(x1 / 2.1255287562098677) / -0.559505775722691) / 0.5486514363188524) ^ 1.6971302727139999) - sin(log((x1 / 2.1255287562098677)
^ 2.1273824593164528)))
```



PySR program



```
from pysr import PySRRegressor
import numpy as np

X = np.random.uniform(500,3000,200).reshape(-1,1)
y = np.piecewise(X, [X <= 1048, X > 1048],
                    [lambda X : 20.79*np.exp(0.5*((X-1048)/179.1)**2),
                    lambda X : 20.79*np.exp( -np.exp(1.698)*pow(np.log(X/1048),1+np.exp(-0.223)) )]
    )
model = PySRRegressor(
    model_selection="accuracy",
    procs=4,
    populations=8,#2 populations per core
    population_size=25,
    ncyclesperiteration = 1000,
    niterations=100,
    maxsize=50,#Allow greater complexity.
    #maxdepth=100,#avoid deep nesting.
    #timeout_in_seconds=60 * 60 * 24,
    binary_operators=["+","-","*","/","pow"],
    unary_operators=["exp","log"],
    #extra_sympy_mappings={"powlog": lambda a,b: pow(log(a),1+exp(b))},
    complexity_of_constants=2,#max variables
    complexity_of_operators={"+":1,"-":1,"*":1,"/":1,"log":2, "exp":2,"pow":2},
    '''nested_constraints={
        "log": {"log": 0, "exp": 1},
        "pow": {"log": 0, "exp": 1},
        "exp": {"log": 2, "exp": 2},
    },
    '''
    use_frequency = False,
    loss="loss(x, y) = (x - y)^2"
)
model.fit(X,y)
model.predict(X)
```

Training data. Full distribution.

PySR is program built upon SymbolicRegression.jl and uses python front end for convenience.

define features like complexity & nesting

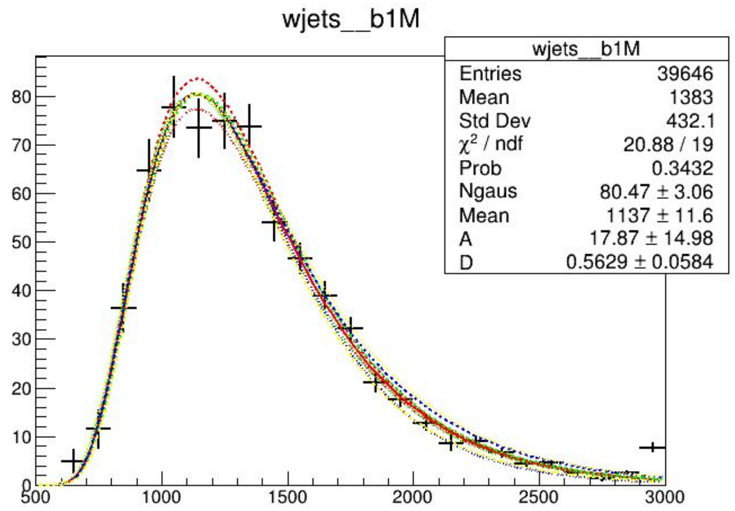
custom loss function

PySR results

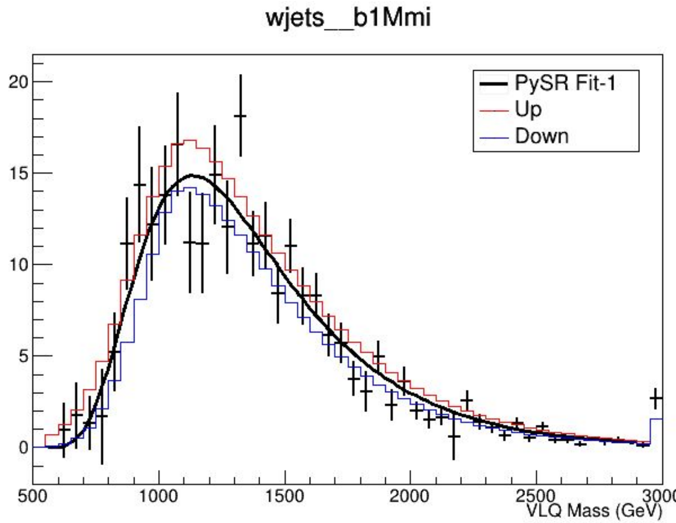
```
Complexity  Loss     Score  Equation
1             4.245e+06  -1.763e+01  x0
2             9.398e-02  1.762e+01  0.24086297
4             5.177e-02  2.982e-01  (556.1356 / x0)
5             1.207e-02  1.456e+00  (7.7977037 - log(x0))
6             1.167e-02  3.327e-02  ((2296.5872 - x0) / x0)
9             5.337e-03  2.609e-01  ((2387.473 - x0) / (x0 / 0.77058065))
10            1.720e-03  1.132e+00  (0.6057233 ^ ((x0 * 0.0046957773) + -5.1249743))
13            1.482e-03  4.966e-02  (0.6057233 ^ ((x0 * 0.0046957773) + (-5.1249743 - 0.104726955)))
15            8.756e-04  2.632e-01  (((22.0733 / x0) ^ ((x0 * 0.0046957773) + -5.1891212)) ^ 0.1161151)
18            1.746e-04  5.374e-01  (((0.7572757 ^ (x0 * 0.009739149)) ^ ((x0 * 0.0046957773) + -5.0826964)) ^ 0.10586759)
21            1.471e-04  5.717e-02  (((0.7572757 ^ ((x0 * 0.010256053) - 1.1168501)) ^ ((x0 * 0.0046957773) + -5.0826964)) ^
0.108118005)
24            1.387e-04  1.951e-02  ((((0.75753784 ^ (x0 * 0.010256053)) / 0.57127213) ^ (((x0 * 0.0046957773) + -5.0826964) *
1.085235)) ^ 0.108118005)
25            1.303e-04  6.272e-02  ((((0.7572757 ^ ((x0 * 0.010256053) - exp(1.2106702))) ^ ((x0 * 0.0046957773) +
-5.0826964)) ^ 0.108118005) ^ 1.1969104)
26            1.904e-05  1.924e+00  (((exp(x0 ^ (1.3265928 / (0.0046957773 * x0))) / x0) ^ ((x0 * 0.0046957773) + (-5.0826964
+ 0.17363225))) ^ 0.12238044)
29            4.446e-06  4.848e-01  ((((exp(x0 ^ (1.3265928 / (0.0046957773 * x0))) - -1.2470548) / x0) ^ (((x0 *
0.0046957773) + -5.0826964) + 0.16093823)) ^ 0.12238044)
```

- The final function can be simplified into 20.79*(((1.24705 + exp(x^(282.508/x)))/x)^(0.00469578*x - 4.92176))^0.12238
- Or into a general form $f(x) = N \cdot (A + e^{x^{B/x}})^{Cx-D}$ where B can be written as a function of A,C and D.

Creating nuisance parameters from the fit



wjets__b1M	
Entries	39646
Mean	1383
Std Dev	432.1
χ^2 / ndf	20.88 / 19
Prob	0.3432
Ngaus	80.47 ± 3.06
Mean	1137 ± 11.6
A	17.87 ± 14.98
D	0.5629 ± 0.0584



- We could use the same function to fit a sample in different regions (b1M, b2M, charge category, etc) and systematic uncertainties.
- Vary the fit parameters by ± σ to get an envelop of analytic shapes. These up/down and left/right extremes of these shapes can be traced out to make new shape systematics.
- This way we can create more nuisance parameters for the CLs method.

Final thoughts

- Symbolic regression via genetic programming is a useful technique to identify the functional form of models without a priori assumptions.
- Once we have a function, we can use it’s form as an “a priori” knowledge to find alternate functions and perform the F-statistic or residual tests to optimize the fit.
- My only trouble with this package is that a random seed couldn’t be set. This made the function evolution take a different route every time the program was run.

Thank you!