Symbolic Regression with Fast Function Extraction and Nonlinear Least Squares Optimization

Lukas Kammerer[1,2][0000−0001−8236−4294], Gabriel Kronberger[1][0000−0002−3012−3189], and Michael Kommenda[1][0000−0003−2049−723X]

[1] Josef Ressel Center for Symbolic Regression
Heuristic and Evolutionary Algorithms Laboratory
University of Applied Sciences Upper Austria, Hagenberg, Austria

[2] Department of Computer Science
Johannes Kepler University, Linz, Austria
lukas.kammerer@fh-hagenberg.at

Abstract. Fast Function Extraction (FFX) is a deterministic algorithm for solving symbolic regression problems. We improve the accuracy of FFX by adding parameters to the arguments of nonlinear functions. Instead of only optimizing linear parameters, we optimize these additional nonlinear parameters with separable nonlinear least squared optimization using a variable projection algorithm. Both FFX and our new algorithm is applied on the PennML benchmark suite. We show that the proposed extensions of FFX leads to higher accuracy while providing models of similar length and with only a small increase in runtime on the given data. Our results are compared to a large set of regression methods that were already published for the given benchmark suite.

Keywords: Symbolic Regression · Machine Learning.

1 Symbolic Regression and FFX

Symbolic regression is a machine learning task in which we try to identify mathematical formulas that cover linear and nonlinear relations within given data. The most common algorithm for solving symbolic regression is genetic programming (GP), which optimizes a population of mathematical models using crossover and mutation. GP is in theory capable of finding models of any syntactical structure and complexity. However, disadvantages of GP are its stochasticity, its long algorithm runtime for nontrivial problems and its complex hyperparameter settings. These characteristics led to the development of non-evolutionary

Submitted manuscript to be published in *Computer Aided Systems Theory - EUROCAST 2022: 18th International Conference, Las Palmas de Gran Canaria, Feb. 2022.*

algorithms which produce more restricted models but have advantages in determinism, runtime or complexity of hyperparameters [3,6,9].

One of the first algorithms that was developed to tackle the shortages of GP is Fast Function Extraction (FFX) [9]. FFX generates a large set of *base functions* first, then it learns a regularized linear model using these base functions as terms. The set of base functions is a combination of the original features with several nonlinear functions such as $\exp(\dots)$ or $\log(\dots)$ and a predefined set of real-valued exponent values. Examples of base functions for problems with features $\{x_1, x_2\}$ are x_1, x_2, ${x_1}^2$, $\exp(x_1)$, $\exp({x_1}^2)$, $\log(x_1)$ or $\log({x_2}^{0.5})$.

The structure of FFX models with n features $\{x_1 ... x_n\}$ is outlined in Equation 1. Parameters $\{c_0, c_1, \dots, c_m\}$ are linear parameters of a model with m base functions. They are learned with ElasticNet regression [4]. ElasticNet regression identifies only the most relevant base functions due to its regularization by setting linear parameters c_i of irrelevant base functions to zero (sparsification). Therefore, learned models contain only a subset of the original base functions.

$$\begin{gathered}\hat{f}(\mathbf{x}) = c_0 + c_1 \ \mathrm{func}_1({x_1}^{e_1}) + \dots + c_m \ \mathrm{func}_m({x_n}^{e_m}) \\ \text{with } \mathrm{func}_1, \dots, \mathrm{func}_m \in \{\mathrm{abs}(), \log(), \dots\} \text{ and } e_1, \dots, e_m \in \{0.5, 1, 2\}\end{gathered} \tag{1}$$

1.1 Motivation and Objectives

A disadvantage of FFX is that only linear parameters are optimized. Parameters within nonlinear functions are not present. For example functions with feature x and nonlinear parameters k_i such as $\log(x + k_i)$ or $\exp(k_i x)$ have to be approximated by FFX with a linear model of several base functions. In this work, we extend the capabilities of FFX by adding such a real-valued parameter k to each generated base functions. Adding several nonlinear parameters to the possible model structure should allow to fit additive models with fewer base functions, as we have more degrees of freedom per base function.

The introduced nonlinear parameters are optimized in combination with the linear parameters by separable nonlinear least squares optmization (NLS) using a variable projection algorithm [2]. We call this new algorithm *FFX with Nonlinear Least Squares Optimization* (FFX NLS). We test whether FFX NLS leads to higher accuracy than the original implementation on the *PennML* benchmark suite [10] and compare the complexity of the generated models. Given, that we have more degrees of freedom to fit data with a single base function, we expect to find that it produces models with a lower number of base functions and therefore simpler models than the original FFX algorithm.

2 Algorithm Description

Similar to FFX, FFX NLS runs in several steps. First, base functions are generated. Then the most relevant base functions and the model's parameters are determined. In difference to FFX, the selection of relevant base function and the

optimization of parameters are separate steps. FFX NLS performs the following four steps that are described in detail in the next sections:

1. Generate a list of all univariate base functions $\boldsymbol{f}$ (cf. Section 2.1).
2. Optimize all parameters $\boldsymbol{k}$ and $\boldsymbol{l}$ of a nonlinear model $l_0 + \sum_i l_i f_i(\boldsymbol{k}, \boldsymbol{x})$ that consists of the base functions as terms. $\boldsymbol{k}$ is a vector of all nonlinear parameters. $\boldsymbol{l}$ are linear parameters, $\boldsymbol{x}$ are features (cf. Section 2.2).
3. Select most important base functions of $\boldsymbol{f}$ with a regularized linear model and the nonlinear parameters $\boldsymbol{k}$ from the previous step (cf. Section 2.3).
4. The final model is created by optimizing all parameters again with nonlinear least squares optimization but only using the most important base function (cf. Section 2.4).

2.1 Base Functions

In step 1, we generate all univariate base functions $\boldsymbol{f}$ with placeholders for nonlinear parameters. For each feature x_i we create base functions of structure $func(ax_i^p + b)$ with $func \in \{\mathrm{id}, \log, \exp, \mathrm{sqrt}\}$, $p \in \{1, 2\}$ and two scaling values a and b with $\mathrm{id}(x) = x$. The scaling values a and b are placeholder for nonlinear parameters that will be optimized later on. We also include bivariate base functions, which are described in Section 2.4.

We utilize the linear structure in the final model as well as mathematical identities of the used nonlinear functions to reduce the number of nonlinear parameters. Due to mathematical identities such as $\log(ax_i) = \log(a) + \log(x_i)$ with a being a parameter that is trained later on, we can skip certain scaling values. In the case of logarithm, we can just use $\log(x_i + b)$ instead of $\log(ax_i + b)$ as base function as we can rewrite it to $log(c)\log(x_i + d)$. Then we can skip $log(a)$ as it is constant in the final model and therefore summed up by the final model's intercept. The same applies to the exp-function, in which we can skip the multiplicative scaling value in the argument as we can rewrite $\exp(x + a) = \exp(x)\exp(a)$ and skip $\exp(a)$ in the final model.

2.2 Parameter Optimization with Variable Projection

The generated base functions $\boldsymbol{f}$ are combined to one large linear model $\hat{\Theta}(\boldsymbol{x}) = l_0 + \sum_i l_i f_i(\boldsymbol{k}, \boldsymbol{x})$ with $\boldsymbol{l}$ as vector of linear parameters and intercept and $\boldsymbol{k}$ as vector of nonlinear parameters. We use NLS to find the values in $\boldsymbol{l}$ and $\boldsymbol{k}$ that minimize the mean squared error (MSE) for given training data.

Since nonlinear least squares optimization is computationally expensive, we use a variable projection (VP) algorithm initially developed by Golub and Pereya [5] for optimizing all parameters $\boldsymbol{l}$ and $\boldsymbol{k}$. The advantage of VP over plain NLS optimization is that VP utilizes the generated model's structure – a nonlinear model with several linear parameters $\boldsymbol{l}$. Golub and Pereya call the given model structure a *separable least squares problem*. This setting is common in engineering domains. VP optimizes nonlinear parameters in $\boldsymbol{k}$ iteratively, while optimal linear parameters are solved exactly via ordinary least squares. Given the high number

of linear parameters in the model (due to the large number of base functions), the use of VP is an important performance aspect of FFX NLS. We use the effieicennt VP algorithm by Krogh [8].

2.3 Base Function Selection

As we generate a very large number of base functions, we need to select the most relevant ones in order to create a both interpretable and well-generalizing model. The number of selected base functions is thereby a hyperparameter of FFX NLS. In plain FFX, the selection of relevant base functions and the optimization of parameters is done in one step with a ElasticNet regression [4], as only linear parameters need to be optimized.

Since we also need to optimize nonlinear parameters, no simple way to combine NLS optimization and regularization of linear parameters is available. The VP algorithm by Chen et al. [2] uses Tikhonov regularization [4]. However, in initial experiments this algorithm was not beneficial for FFX NLS to identify relevant terms. Tikhonov regularization shrinks linear parameters, however, it did not provide the necessary sparsification of linear parameters and the algorithm in [2] is computationally more expensive. Alternatively, we use the already computed vector of nonlinear parameters $\boldsymbol{k}$ of the previous step as fixed constants and optimize the linear parameters $\boldsymbol{l}$ with a lasso regression [4]. Terms with a linear parameter $\neq 0$ are selected as most important base functions. We get the desired number of base functions by iteratively increasing the lasso regression's λ value in order to shrink more parameters to zero. Although this method ignores dependencies between linear and nonlinear parameters, it is still effective for selecting base functions both regarding runtime and further modelling accuracy.

2.4 Training of Final Model and Bivariate Base Functions

Similar to the parameter optimization in Section 2.2, we combine base functions to one single model and optimize all parameters again. In difference to the previous NLS optimization step, we take in this step only the most relevant base functions from Section 2.3. As parameters are not independent from each other, we repeat this step to get optimal parameters for a subset of base functions.

To cover interactions between features, we also include bivariate base functions. For this we take pairwise combinations of the previously generated univariate base function and multiply them to get one new base function for each pair. To prevent a combinatorial explosion of base functions and nonlinear parameters in highly multidimensional problems, we only consider the ten most important univariate base functions from Section 2.3 for to create bivariate base functions. We take the resulting $\binom{10}{2} = 45$ bivariate functions and append them to our existing list of univariate base functions. Then we repeat the steps described in Section 2.2 and 2.3 to determine the most important base functions across both univariate and bivariate base function for the final model. We consider only ten univariate base functions as a larger number did not provide any benefit in achieving our objectives.

3 Experimental Setup

We use *PennML* benchmark suite [10] for all experiments. This benchmark consist of over 90 regression problem and provides a performance overview of several common regression algorithms. We apply both FFX and FFX NLS on these problems to compare the algorithms' accuracies with each other and with other common regression algorithms. Thereby we take the experimental results from [7] for a comprehensive comparison as this work shows the results of the currently most accurate regression algorithms like GP CoOp as implemented in [1] for the PennML data.

We apply the same modelling workflow as in [10] in our experiments. For every regression problem in the benchmark suite, we repeat the modelling ten times. We shuffle the dataset in every repetition and take the first 75% of observations as training set and the remaining ones as test set. We perform a grid search with a 5-fold cross validation for each shuffled dataset for hyperparameter tuning and train one final model with these best hyperparameters. Eventually, we get ten different models for each original regression problem. Hyperparameters for all other algorithms are described in [7]. We use the following hyperparameter sets for both FFX and FFX NLS:

- Max. number of base function in final model $\in \{3, 5, 10, 20, 30, 50\}$
- Use bivariate base functions $\in \{\text{true}, \text{false}\}$
- Use nonlinear functions $\in \{\text{true}, \text{false}\}$
- FFX only: L1 ratio $\in \{0, 0.5, 1\}$

4 Results

As in the original experiments for this benchmark suite [10], we first calculate the median (mean squared) error of the ten models of each regression problem. Then we rank all regression methods by their median error for each problem. Figure 1 and 2 show the distribution of ranks for each algorithm for training and test across all regression problems in the benchmark suite. E.g. in Figure 1, gradient boosting was for most problems the most accurate algorithm in training.

Figure 1 and 2 show the distribution of rankings of each algorithm's median rank on the training set and test set. The optimization of nonlinear parameters in FFX NLS provided to more accurate models than FFX both in training and test. In comparison to other algorithms, FFX as well as FFX NLS perform better than linear models, which are on the right of Figure 1 (like linear regression or lasso regression). This is expected given both FFX methods can cover nonlinear dependencies and interactions in the data in contrast to purely linear methods. However, both algorithms perform worse than boosting methods or GP with NLS (GP CoOp) by Kommenda et al. [7]. Also this is plausible, as both methods have a more powerful search algorithm and a larger hypothesis space than FFX methods with their restricted model structures.

To analyze the size of models, we count the number of syntactical symbols in a model. E.g. the model $c_0 + \log x_1 + c_2$ has a complexity of six. We use this

measure because it takes the added scaling terms within function arguments in FFX NLS into account. Figure 3a shows that both the size of models and the number of base function within models produced by FFX NLS and FFX are similar. FFX NLS models are just slightly shorter.

Figure 4 shows the median runtime across all problems per algorithm. While one FFX run takes less than a second for most problems, the runtime of FFX NLS ranges between one and a few seconds. Although the runtime increase is relatively large, it is still feasible to perform large grid searches in reasonable amount of time with FFX NLS. Both algorithms beat some GP-based algorithms and are on a similar level as boosting algorithms.

5 Conclusion

We proposed an extension of the model structure of the FFX algorithm. We added nonlinear scaling parameter which were optimized using the variable projection algorithm by Krogh [8] and separate base function selection, which we called FFX NLS. We achieved a large improvement in test accuracy in comparison to plain FFX on the PennML benchmark suite while providing models of similar complexity. Although the runtime of FFX NLS is higher than the one of FFX, it still finishes training within seconds.

However, FFX and FFX NLS still perform worse than many symbolic regression algorithms. Big advantages of FFX NLS towards GP-based algorithms are its short runtime, the low number of hyperparameters and the simple, comprehensible model structure. Compared to boosting algorithms, FFX NLS performs

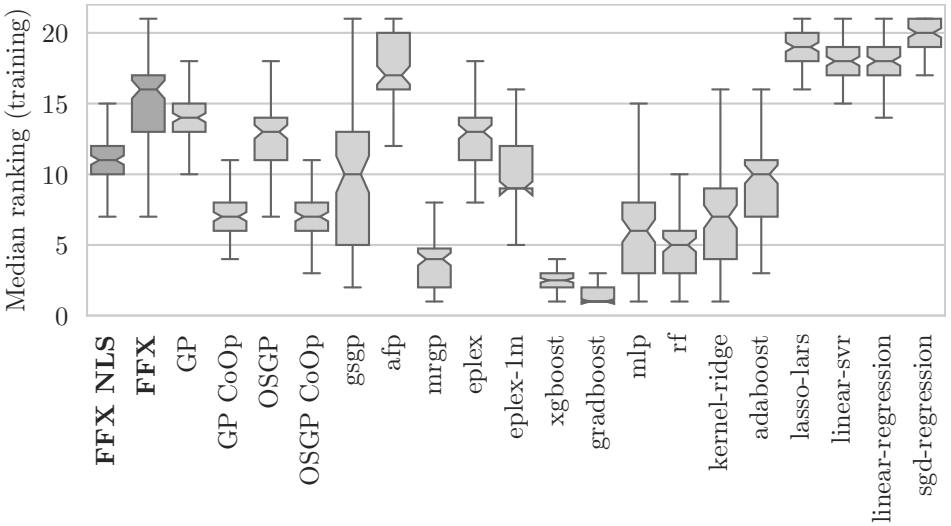


Fig. 1: Distribution of median rankings of the mean squared error on the training set.

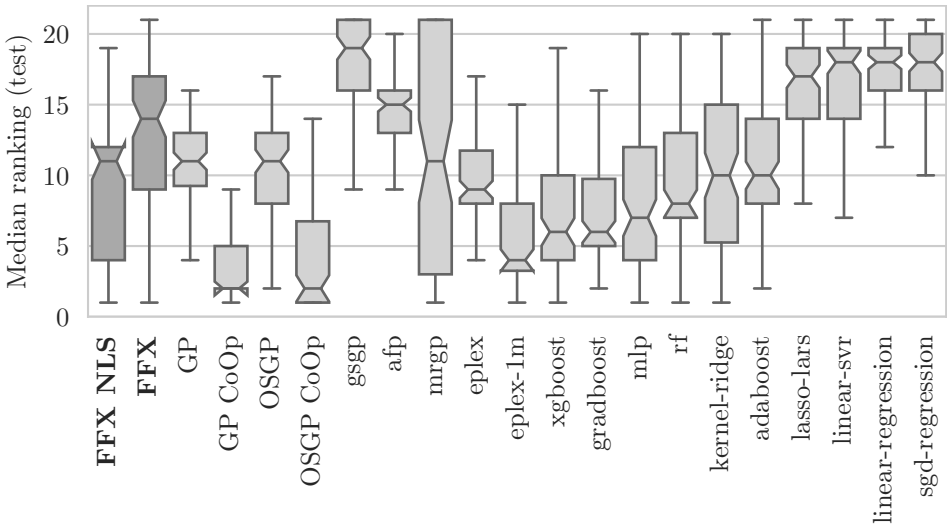
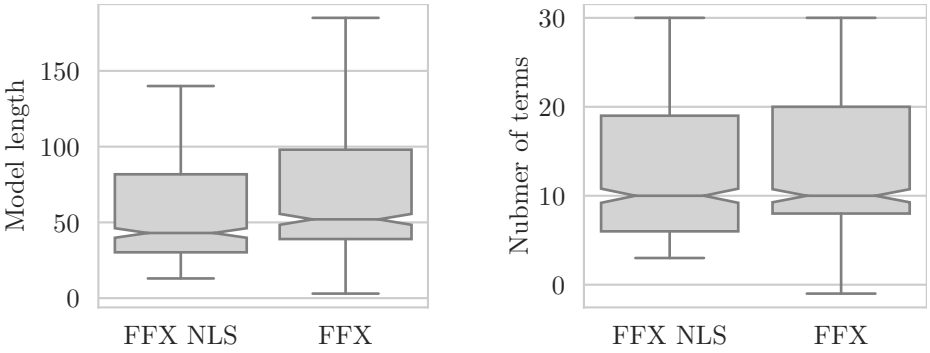


Fig. 2: Distribution of median rankings of the mean squared error on the test set.



(a) Length per model.

(b) Number of base functions per model.

Fig. 3: Distribution complexity of all models from FFX and FFX NLS.

worse in accuracy but similar in runtime. However, boosting algorithms are considered black box methods and allow no readability of its models.

To sum it up, FFX is a promising tool for quick data exploration. It identifies interactions as well as nonlinear relations with simple hyperparameter configuration and quick execution. However, further improvements regarding accuracy are needed. Potential improvements are the combination of regularization and parameter optimization, as this are separate steps right now that deliberately ignore dependencies between linear and nonlinear parameters.

Acknowledgements The authors gratefully acknowledge support by the Christian Doppler Research Association and the Federal Ministry for Digital and Economic Affairs within the *Josef Ressel Center for Symbolic Regression*.

References

1. Burlacu, B., Kronberger, G., Kommenda, M.: Operon c++ an efficient genetic programming framework for symbolic regression. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion. pp. 1562–1570 (2020)
2. Chen, G.Y., Gan, M., Chen, C.P., Li, H.X.: A regularized variable projection algorithm for separable nonlinear least-squares problems. IEEE Transactions on Automatic Control **64**(2), 526–537 (2018)
3. de França, F.O.: A greedy search tree heuristic for symbolic regression. Information Sciences **442**, 18–32 (2018)
4. Friedman, J.H.: The elements of statistical learning: Data mining, inference, and prediction. springer open (2017)
5. Golub, G.H., Pereyra, V.: The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. SIAM Journal on numerical analysis **10**(2), 413–432 (1973)
6. Kammerer, L., Kronberger, G., Burlacu, B., Winkler, S.M., Kommenda, M., Affenzeller, M.: Symbolic regression by exhaustive search: reducing the search space using syntactical constraints and efficient semantic structure deduplication. Genetic programming theory and practice **17**, 79–99 (2020)
7. Kommenda, M., Burlacu, B., Kronberger, G., Affenzeller, M.: Parameter identification for symbolic regression using nonlinear least squares. Genetic Programming and Evolvable Machines **21**(3), 471–501 (2020)
8. Krogh, F.T.: Efficient implementation of a variable projection algorithm for nonlinear least squares problems. Communications of the ACM **17**(3), 167–169 (1974)

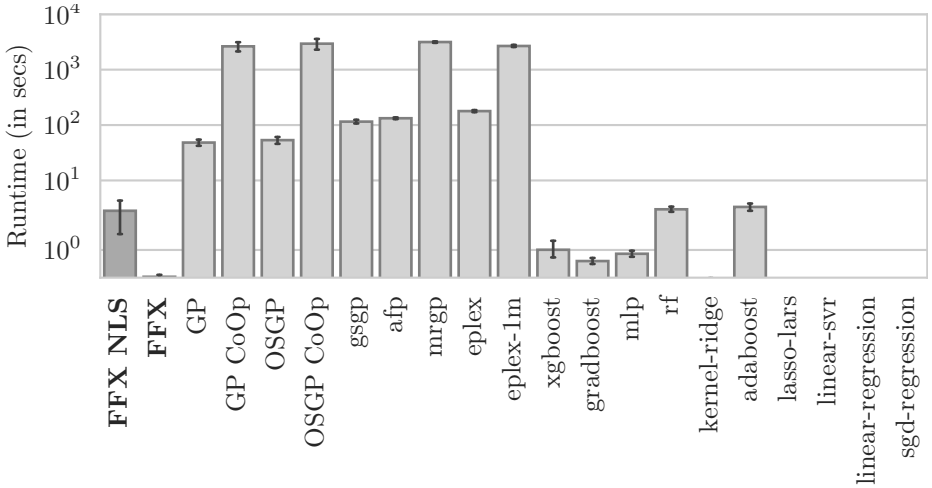


Fig. 4: Median runtime of all algorithms.

9. McConaghy, T.: Ffx: Fast, scalable, deterministic symbolic regression technology. In: Genetic Programming Theory and Practice IX, pp. 235–260. Springer (2011)
10. Orzechowski, P., La Cava, W., Moore, J.H.: Where are we now? a large benchmark study of recent symbolic regression methods. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1183–1190 (2018)