NENE
International Conference
Nuclear Energy for New Europe

Burnup-dependent Group Constant Parametrization by Applying Different Machine Learning Methodologies

Dániel Sebestény
Centre for Energy Research
Konkoly-Thege Miklós út 29-33
1121 Budapest, Hungary
sebesteny.daniel@ek-cer.hu

István Panka, Bálint Batki
Centre for Energy Research
Konkoly-Thege Miklós út 29-33
1121 Budapest, Hungary
panka.istvan@ek-cer.hu, batki.balint@ek-cer.hu

ABSTRACT

In neutronics calculations, two-step calculation schemes are often used for computationally demanding tasks. This article concerns the challenge of group constant parametrization in these schemes by applying various machine learning methods. Several fitting methods have been examined and implemented in a package that conducts hyperparameter optimization and automatic model selection. Models were trained and evaluated on data from a VVER-1200 fuel assembly, calculated using the MULTICELL 2D transport code. An optimized and modified polynomial regression algorithm was found to be the best-performing model.

1 INTRODUCTION

The need for group constant parametrization arises from the two-step neutronics calculation schemes. The first code produces few-group constants spatially homogenized to assemblies or nodes, while the second code uses these for full core calculations. These values depend on several properties, including burnup and various operating conditions (e.g., temperatures, moderator density, boron concentration), these properties will be subsequently referred to as features. Since the first code can produce group constant values for a finite number of feature combinations, and the second code may need the values for any feature combination, a predictive model (also called the parametrized group constant library) is needed for each group constant.

The simplest solution is a linear interpolation between the calculated values (sample points). However, its fitting performance is suboptimal when the number of sample points is limited (e.g. when the first code uses a computationally expensive Monte Carlo method). To overcome this limitation, various regression-based algorithms have been examined in the literature, including stepwise regression [1], [2]. Recent studies have also investigated other models, such as a combination of neural networks and decision trees whose performance

exceeded that of the polynomial regression [3], and a support vector regression model that also includes a novel optimization procedure [4].

In this paper, a testing and optimization methodology will be presented with various algorithms, and their respective fitting performances will be compared on an assembly model.

2 METHODOLOGY

In this section, the whole testing and fitting pipeline will be discussed, starting from sample point generation through testing and optimization steps to the description of the fitting algorithms used.

2.1 Group constant generation

A goal of the research this study is part of is to determine the best method specifically for Monte Carlo-generated group constants. This requires the algorithm to be robust for stochastic variation of the values and to perform well even on smaller datasets since Monte Carlo methods are much more time- and resource-consuming than traditional, deterministic methods. Because of this limitation, a deterministic code called MULTICELL was used for testing purposes in this study. MULTICELL was developed at the Centre for Energy Research, which is a part of KARATE-1200 code system [5], an improved version of KARATE-440 [6].

The physical model used in this study was an assembly of the VVER-1200-type pressurized water reactor, which consists of hexagonally arranged fuel pins containing uranium dioxide with 4.0% enrichment. It contains guide and instrumentation tubes but no fuel rods with burnable poison.

2.2 Data point distribution and dataset splitting

After a set of data points (homogenized cross-section values and their respective feature vectors) have been generated, the next step is the division of these into training and test datasets. The training dataset is what we refer to as the set of points on which the fitting algorithm is performed, while the test dataset is the set of points on which we evaluate the performance of the fitted model using an error metric. The test points should never be used during fitting since the evaluated performance has to show how well the model performs on new, previously unseen data. This split must be done in a way that the two resulting datasets are independent of each other in structure. Otherwise, the evaluated performance can be unrealistically good, as the model can pick up information true for that specific structure of the datasets but not for the whole feature space.

A third set of points is needed for performance evaluation during hyperparameter optimization, which will be discussed later in section 2.4. It is referred to as the validation dataset and should be independent of both the training and test datasets.

A simple dataset distribution and division methodology is to generate a regular N-dimensional grid of sample points (N is the number of features, including burnup) and divide them randomly into training, validation, and test datasets. However, the aforementioned independence is not fulfilled this way since the test points fill the holes in the grid. Thus, much more information is known about them than a point somewhere in the middle of a grid cell.

A solution would be to generate points randomly distributed in the feature space and then split them randomly into the three desired sets. The independence of the datasets' structure is ensured this way, as all sample points are completely independently distributed from each other, and there is zero correlation between the features [1]. In our case, however,

MULTICELL can only generate random points with respect to the non-burnup features, while burnup can be changed in discrete steps (depletion calculation). Therefore the sample point distribution is still grid-like, albeit only in one dimension.

To avoid this, our sampling methodology was to generate three different datasets instead of splitting a single one. The burnup points are different in all three sets. The base point of the depletion calculation (where the burnup steps are calculated, technically a line along the burnup axis in the N-dimensional feature space), as well as all material and geometry input data, remains the same among the three runs. It is important to note that apart from random points inside the N-dimensional feature space, the training dataset also contains extremal points at the edge of the feature space so that no extrapolation is needed during evaluation. The limits of the feature space in each dimension are shown in Table 1. The table also shows the respective base point of depletion calculation. The burnup steps are heuristically selected in a way that steps are more frequent for small values since the most complex burnup dependence appears at the beginning of a fuel cycle. The number of points in the training, validation, and test datasets is 1211, 239, and 239, respectively.

Table 1. Range of each feature and the base point of the depletion calculation

Feature	**Unit**	**Lower limit**	**Upper limit**	**Base point**
Burnup	MWd/kgU	0	23.96	-
Boric acid concentration	g/kg	0.0	8.0	8.0
Fuel temperature	K	300.0	1200.0	874.0
Coolant temperature	K	300.0	586.7	586.7
Coolant density	g/cm^3	0.791	1.000	0.791

It is important to mention that scaling of the feature space should be undertaken before performing a fit. It is a mapping to a space where the sample point distribution has a mean of 0 and a standard deviation of 1 in every dimension (standardization). The purpose of this procedure is to avoid numerical errors that can emerge in some of the algorithms when different features have vastly different magnitudes. For example, when expressed in atoms per cubic centimeters, atomic densities are usually several orders of magnitudes larger than all other features.

2.3 Testing

A model's performance is evaluated by comparing the predicted values to the test set's target values at the points in the feature space where the test set has its samples. Several error metrics can be used, most commonly the root of mean square error (RMS) and the mean absolute error (MAE). We used RMS during the study because it gives a higher loss for a few large errors coming from a badly working model than for numerous small errors that can result from stochastics in the case of Monte Carlo.

A very small training error coinciding with a high test error is a hallmark of overfitting, which means, our model is overtrained on the data and learns some structure that is only true for the specific part of the feature space where the training samples are located. The opposite effect is called underfitting when the applied model is too simple to grasp the complexity of the problem to be fitted (e.g., fitting a linear on a nonlinear dependence). One of the methodology's biggest challenges is finding the balance between over- and underfitting.

2.4 Hyperparameter optimization

An essential step in the pipeline is the optimization of the model. For a fitting algorithm, there are parameters that are specific not for the particular fit but to the model used, and they are called hyperparameters. Just like performance evaluation with the training set is undertaken in order to determine the best fitting parameters, hyperparameter optimization with the validation set is to determine the best combination of hyperparameters. As an example, in polynomial regression, the model's parameters are the coefficients of the fitted polynomial, while the maximum degree of the polynomial is the model's hyperparameter. In other algorithms, there can be multiple hyperparameters for which the model's performance can drastically differ.

Determining the best combination of hyperparameters can be simply done by performing a fit with each combination and choosing the best-performing one. This is called grid search optimization. However, this can be done only with hyperparameters that have discrete values, and a larger number of hyperparameters results in an exponentially larger number of potential combinations. Some of the algorithms used in this study have many hyperparameters, so another approach called Bayesian optimization is used. This means that the optimizer algorithm searches iteratively, every guess is based on knowledge acquired by the previous guesses. This procedure cannot be parallelized like grid search, however, it finds a well-performing combination in much fewer steps.

We used the Optuna [7] framework, which is a flexible framework that can use several algorithms and black-box optimization. It means the optimizer framework knows nothing about the problem to be optimized, it simply provides the client code with a set of hyperparameters and receives a simple number (error value) from the client code after it has run. Due to this principle, one can arbitrarily modify the returned error function in a way that more complex models get a punishment (e.g., a multiplication factor that scales with the model's complexity). This can steer the optimization procedure to simpler models, thus preventing overfitting. Optuna can dynamically construct the hyperparameter space with conditionals and loops. This gives huge flexibility in adding further hyperparameters for better optimization. Optuna uses a Bayesian algorithm called Parzen tree estimator search for the iterative search for the best hyperparameter combination.

2.5 Models

In this section, the implemented fitting algorithms are presented, ranging from simpler to more complex methods. The SciPy, Scikit-learn, and Keras libraries were used [8]–[10].

The simplest ways to handle this problem are the various interpolation methods, such as linear interpolation. This method approximates a point by fitting it on a linear function connecting two (or, in a multidimensional case, more) neighbouring points. These algorithms are unable to extrapolate the data and are heavily limited by the density of data points, more so than other algorithms. Hyperparameter optimization is not needed for these methods.

Linear regression algorithms are one of the most commonly used approaches for these kinds of problems. The simplest linear regression is fitting a linear function to the dataset using the ordinary least square (OLS) method. Fitting any other function (e.g., a polynomial) can be equalled to multiple linear regression if one applies the function to the features and adds the results to the dataset as new features. A generalized polynomial regression would take not just the powers of each feature but also the mixed products into account. In such an algorithm, there are multiple ways to construct the hyperparameter space. One approach is that the single hyperparameter is the maximum degree of any term in the polynomial function. Another approach to this problem is to give a hyperparameter for each feature that denote the maximum degrees the features can be raised to in any term (pure power or mixed product as well) of the polynomial.

In this study, these two approaches were combined in order to prevent the emergence of very high-degree mixed terms, which can lead to overfitting. The hyperparameter space consists of maximum degrees for each feature and an overall limit to the rank of terms that excludes any mixed terms, which would be otherwise allowed according to the features' degree limits. To add even more flexibility, taking square roots of the features into account were also added as boolean (0 or 1) hyperparameters for each feature. This is sensible not only in mathematical but also in physical sense, as the neutronic Doppler effect depends on the square root of fuel temperature. This gives overall $N = 2n + 1$ hyperparameters for which one should find the optimal combination. As a larger number of raw features results in an exponentially larger number of possible combinations, it gets practically impossible to find the global minimum. The goal with the optimization is instead to find a reasonably good combination with performance close to the optimum. We named this method 'CORK' regressor for convenience, and it is referred to as such in the further sections.

Instead of the OLS method, other fitting methods can be considered for polynomial regression. Ridge regression takes into consideration the magnitude of the fitting coefficients (regularization), as the OLS method may overfit, resulting in very high coefficients. The ridge method minimizes the sum of the OLS error function and $\alpha|\mathbf{c}|^2$, where α is called the regularization parameter and $\mathbf{c}$ is the vectorized form of the coefficients. Thus, fittings with smaller coefficients are preferred by the algorithm. Lasso regressor uses the 1-norm (sum of absolute values) of the coefficients in a similar manner. Choosing from these three methods is also added to CORK regressor as a discrete hyperparameter, with the respective regularization parameters added as conditional, continuous hyperparameters.

Random forest is a widely used algorithm for both classification and regression problems. It consists of an ensemble of decision trees, which themselves are non-parametric models trained by successively dividing ("branching") data points into groups ("leaves") according to a determined criterion, thus getting into a tree structure, this is where its name stems from. The best splitting criterion (the feature and its value where the division happens) is itself determined by an entropy calculation. Training several trees with a random selection of sample points and features and taking the average of each one's prediction as the whole model's prediction (thus "planting a forest from randomized trees") serves as protection from overfitting. Pruning techniques can also be used to avoid this. In this study, Scikit-learn's random forest implementation was used [8].

Support-vector machines (SVMs) are commonly used in problems with high dimensionality (a large number of features) and for small and medium-sized datasets. For regression problems, the goal is to find a hyperplane that best fits the data being learned. In this case, the support vectors are the data points at a given distance (ε) from the hyperplane. During training, we minimize the distance between the hyperplane and the data points, ignoring the points between the hyperplane and the support vectors. The advantage of SVM is that it allows transforming the data by nonlinear projection using the kernel trick, which enables the solution of nonlinear problems. In this study, Scikit-learn's support vector regression model (ε-SVR) is used with the radial basis function kernel [8].

Artificial neural networks have been arguably the hottest areas of research in the field of machine learning during the last few decades. In this study, a simple (dense) neural network was experimented with using the Keras framework [10]. The number and size of layers, solver algorithm, dropout, and learning rate were the hyperparameters used in this study.

3 RESULTS

The whole pipeline has been run for each algorithm, and the results are shown and evaluated in this section. In this study, the infinite multiplication factor (k_{inf}) was used as an

example of the fitted target value. Its value ranges between 0.96 and 1.41 in the sample dataset, and the values presented are absolute errors. However, various group constants can be fitted the same way.

It is important to keep in mind that some algorithm's performance is not deterministic, it can vary with each run. This comes from several factors, including the randomness of the Bayesian search in hyperparameter optimization, the inherent randomness of some of the algorithms (e.g., Random Forest), and the dataset splitting (in those cases where it is part of the pipeline). Initialization of fitting weights is also a source of stochasticity, specifically in neural networks. Thus, the results will be presented as a mean of error values of several (6-7) runs with their sample standard deviation using Bessel's correction.

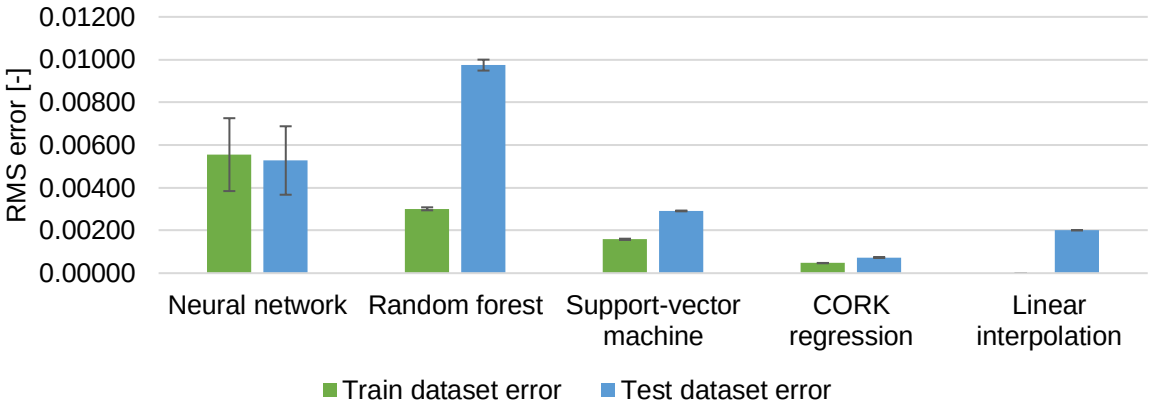

Figure 2. Mean and corrected sample standard deviation of root mean square error at the fitting of k_{inf}, shown for different models

One can make several observations by comparing the average training and test errors for each method (Figure 2):

Random Forest has the largest (0.00974 ± 0.00026) test error. The training error is substantially smaller (0.00301 ± 0.00007), which means the algorithm was not able to generalize well to new data. Random Forest is a widely used algorithm for large and complex, noisy datasets; however, in this problem, the dataset is neither large enough (that is, the distance between data points is large) nor sufficiently complex to justify the usage of this algorithm.

The artificial neural network also has notable errors for both training (0.00555 ± 0.00171) and test (0.00528 ± 0.00160) datasets. The two errors' similar magnitude implies this is not a case of overfitting but simply the model's inability to approximate this problem better. While deep neural networks are incredibly useful and widely used tools that can outperform all other methods for complex and nonlinear machine learning tasks, this problem does not seem to be where these algorithms can shine. However, neural networks are an extensive topic, so other subtypes, as well as feature engineering, further hyperparameters, and training techniques, should be considered before declaring this algorithm family suboptimal for our problem.

Support vector machines are widely used for problems of similar size and complexity to our task. Their test error (0.00291 ± 0.00002) is substantially better than the two already mentioned algorithms, it was not able to outperform a simple linear interpolation (0.00201) for our case, however. This algorithm should also be considered with other kernels and hyperparameters before we deem it suboptimal.

Among regression algorithms, simple polynomial regression (not shown) was able to outperform the linear interpolation slightly (with a test error of 0.00197). The best-performing

algorithm was the modified CORK regression, the performance of which was 0.00073 ± 0.00001, while the training error was 0.00047 ± 0.00001, which is somewhat, but not drastically smaller than the test error, so overfitting was successfully avoided for this method.

The best (completely perfect) performance for the training dataset was reached by linear interpolation since this algorithm fits perfectly on training points by definition.

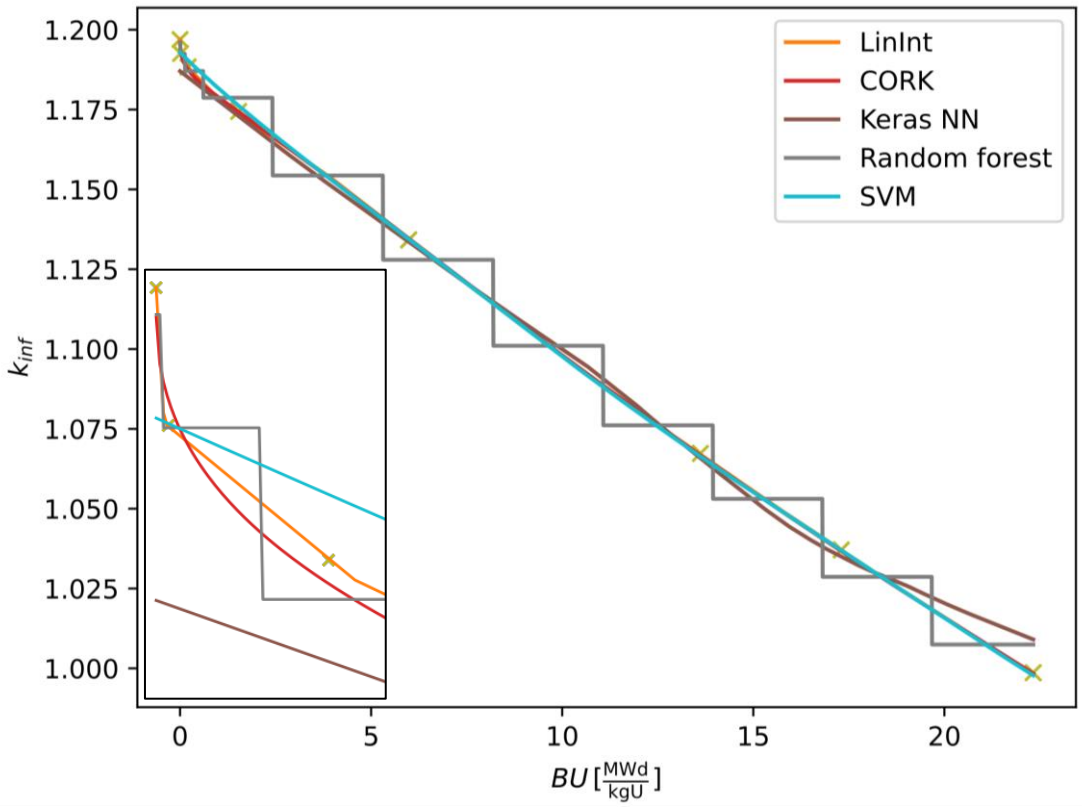

Figure 3. Fitting of each method on points along a BU-dependent line in the 5-dimensional feature space. The fittings at the beginning are shown enlarged in the bottom left corner.

In Figure 3, the fits of each algorithm are shown in the function of burnup. It is important to note that the plotted points (shown with 'x' markers) are part of neither the training nor the test dataset. Similar plots have been made along all dimensions, but only burnup dependence is shown here as it is the most nonlinear of all. Since points in both training, validation, and test datasets are randomly distributed along all dimensions except for burnup, and burnup points are also irregularly spaced, another set of points was generated where just one feature is changed, while the rest is fixed on a base value. This 4th dataset is referred to as the plotting dataset and is shown in the figure with x markers. The random forest approximates the least credibly. As seen in the figure, it has a step-like prediction curve when projected in one dimension. The neural network model also has larger errors, especially at the two ends of the curve. SVM cannot catch the initial sharp nonlinear changes in the multiplication factor. Linear interpolation can not easily be distinguished from the CORK regressor, the performance of these two methods can only be compared by the errors in Figure 2, for which the CORK regressor performs better.

4 CONCLUSION

In this study, a cross-section fitting and testing code package was implemented, and the various fitting algorithms were compared with the newly developed CORK regressor. It

was found that the CORK regressor has the best performance on the used VVER-1200 assembly model with an average RMS error of 0.00073 for the fitted k_{inf}. The method's stable performance is demonstrated by the estimated standard deviation of the error of ~1%. Random forest was found to be ineffective for this problem, while other methods showed suboptimal performance. However, running the pipeline with outputs from other assemblies and fitting different group constants might have slightly different conclusions.

The topic of neural networks is vast, more complex neural network-based models need to be studied, which can be the goal of further research. Other further directions of this research include fitting Monte Carlo-generated group constants and investigating the effect of stochastic errors of the data points. Finally, other features, such as concentrations of reactor poison isotopes (Xe-135 and Sm-149), can also be included in the fitting methodology.

REFERENCES

[1] V. G. Zimin and A. A. Semenov, "Building neutron cross-section dependencies for few-group reactor calculations using stepwise regression," *Annals of Nuclear Energy*, vol. 32, no. 1, pp. 119–136, Jan. 2005, doi: 10.1016/j.anucene.2004.06.009.

[2] Z. Feng, C. Jia, S. Huang, N. An, and K. Wang, "Polynomial interpolation cross-section parameterization method with the RMC Monte Carlo code," *Annals of Nuclear Energy*, vol. 174, p. 109161, Sep. 2022, doi: 10.1016/j.anucene.2022.109161.

[3] Z. Li, J. Sun, C. Wei, Z. Sui, and X. Qian, "A new Cross-section calculation method in HTGR engineering simulator system based on Machine learning methods," *Annals of Nuclear Energy*, vol. 145, p. 107553, Sep. 2020, doi: 10.1016/j.anucene.2020.107553.

[4] K. Tan, Z. Liu, C. Wei, and M. Liu, "The assembly homogeneous few-group constants generation method for PWR based on machine learning," *Annals of Nuclear Energy*, vol. 165, p. 108772, Jan. 2022, doi: 10.1016/j.anucene.2021.108772.

[5] E. Temesvári, G. Hegyi, and C. Maráczy, "Analysis of the startup physics tests of a VVER-1200 reactor with the KARATE- 1200 code system," *Kerntechnik*, vol. 85, no. 4, pp. 250–256, Apr. 2020, doi: 10.3139/124.200010.

[6] A. Kereszturi, Gy. Hegyi, L. Korpas, Cs. Maraczy, M. Makai, and M. Telbisz, "General features and validation of the recent KARATE-440 code system," *International Journal of Nuclear Energy Science and Technology*, vol. 5, no. 3, pp. 207–238, Jan. 2010, doi: 10.1504/IJNEST.2010.033476.

[7] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, in KDD '19. New York, NY, USA: Association for Computing Machinery, 0 2019, pp. 2623–2631. doi: 10.1145/3292500.3330701.

[8] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011.

[9] P. Virtanen *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nat Methods*, vol. 17, no. 3, Art. no. 3, Mar. 2020, doi: 10.1038/s41592-019-0686-2.

[10] F. Chollet and others, "Keras." 2015. [Online]. Available: https://keras.io