A Method for Regularisation of Evolutionary Polynomial Regressions using Support Vector Machines

Francisco Coelho[a,c,*], João Pedro Neto[b,c]

[a] *Dept. Informática, Universidade de Évora, Rua Romão Ramalho 58, 7000-671 Évora*
[b] *Dept. Informática, Faculdade de Ciências da Universidade de Lisboa, Campo Grande 1749-016 Lisboa*
[c] *Laboratory of Agent Modelling (LabMAg)*

Abstract

Many applications require models that have no acceptable linear approximation and the simpler nonlinear models are defined by polynomials. The use of genetic algorithms to find polynomial models, Evolutionary Polynomial Regression (EPR), provides an automatic method to build such model from data but still poses challenges due to the complexity of the search and different definitions of optimal solution.

This paper describes a two-step method that uses genetic algorithms and linear regression to find empirical polynomial regressions. Experiments on common datasets show that, discounting the training computational effort, this method is quite competitive.

Keywords: evolutionary polynomial regression, regularization, feature extraction, dimensionality reduction

*Corresponding author
Email addresses: `fc@di.uevora.pt` (Francisco Coelho), `jpn@di.fc.ul.pt` (João Pedro Neto)

1. Introduction

With notable exceptions (*e.g.* neural networks) machine learning regression techniques are based on linear models. The linearity assumption has many advantages including reduced computational complexity and strong theoretical framework. However nonlinearity is unavoidable in many application scenarios, specially those with phase transitions or feedback loops, so common in engineering, ecology, cybernetics and other areas. The kernel trick in Support Vector Machines alleviates this problem by allowing a non-linear transformation of the feature-space. [**Find references.**]

Polynomials, one of the most studied subjects in mathematics, generalize linear functions and define, perhaps, the simplest and most used nonlinear models. Applications include colorimetric calibration (Mendes and Carvalho, 2005), explicit formulæ for turbulent pipe flows (Davidson et al., 1999), computational linguistics (Sánchez et al., 2009) and more recently, analytical techniques for cultural heritage materials (Cséfalvayová et al., 2010), liquid epoxy molding process (Chan et al., 2011), B-spline surface reconstruction (Gálvez et al., 2012), product design (Chan et al., 2012) or forecasting cyanotoxins presence in water reservoirs (García Nieto et al., 2013). These examples not only illustrate the wide spectrum of applications but, additionally, work in each one uses, at some point, a genetic algorithm.

Genetic algorithms (GA) where, arguably, one of the hottest topics of research in the recent decades and with good reason since they outline an optimization scheme easy to conceptualize and with very broad application. If a nonlinear (or otherwise) model requires parameterization GAs provide a simple and often effective approach to search for locally optimal parameters.

Research related to genetic algorithms abound and spans from the 1950s seminal work of Nils Aall Barricelli (Barricelli, 1962) in the Institute for Advanced Study of Princeton to today's principal area of study for thousands of researchers, covered in hundreds of conferences, workshops and other meetings. Perhaps the key impulse to GAs come from John Holland's work and his book "Adaptation in Natural and Artificial Systems" (Holland, 1975).

One interesting take on genetic algorithms, named *genetic programming* by John Koza (Koza, 1992), proposed the use of GAs to search the syntactic structure of complex functions. This syntactic structure search is keen to the central ideas of deep learning (Bengio et al., 2013; Bengio, 2009), a subarea of machine learning actually producing quite promising results (*e.g.* in Tarlow et al. (2013)). It is also related to the work presented in this paper in the sense that, unlike linear models that have a simple structure, $y = \sum_i \beta_i x_i$, nonlinear (in particular polynomial) models pose an additional "structure" search problem.

The idea of using GAs to find a polynomial regression is not new (Maertens et al., 2006; Yu and Lin, 2008; Wu et al., 2009) but still generates original research (Hofwing et al., 2011; Cetisli and Kalkan, 2011). [**Insert references and resume of EPR; Redo the purpose of this work.**] In line with that research this work describes a general method to find a polynomial regression of a given dataset. The optimal regression minimizes a cost function that accounts for both the root-mean-square error (error) and a regularization factor to avoid over-fitting.

It turns out that, discarding the computational cost of training, the polynomial regression method presented here, Genetic Algorithms for Polynomials (GAPoly), provides a quite competitive regression method. Indeed, it is only systematically out-performed by random forests, an *ensemble* method.

The remainder of this paper is organized as usual: the next section describes the details of our method and is followed by a presentation of some performance results. The last section draws some conclusions and points future research tasks.

2. Genetic Algorithms for Polynomials

This section is dedicated to the description of an algorithm to find a polynomial regression from a given dataset. It starts with a brief introduction and outline of the algorithm and proceeds into core details as the encoding used to represent individual polynomial instances in the GA populations and the regularization of the cost function.

An usual representation of polynomials is

$$p\left(x_1, \ldots, x_m\right) = \sum_i \theta_i q_i$$

where each q_i is a monomial, $q_i = \prod_j x_j^{\alpha_{ij}}$, the exponents are non-negative integers, $\alpha_{ij} \in \mathbb{N}_0$, and the coefficients are real valued, $\theta_i \in \mathbb{R}$. For example $p\left(x_1, x_2, x_3\right) = 2x_1 + x_2x_3 + \frac{1}{2}x_1^2x_3$ has monomials $q_1 = x_1, q_2 = x_2x_3$ and $q_3 = x_1^2x_3$, coefficients $\theta_1 = 2, \theta_2 = 1$ and $\theta_3 = 1/2$ and exponents $\alpha_{1,1} = 1, \alpha_{2,2} = 1, \alpha_{2,3} = 1, \alpha_{3,1} = 2, \alpha_{3,3} = 1$ and all other $\alpha_{ij} = 0$. The exponents alone are a matrix that defines the monomial structure of the polynomial, $A = [\alpha_{ij}]$.

For the example above the matrix of monomials is

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 2 & 0 & 1 \end{bmatrix} \sim \begin{bmatrix} x_1 \\ x_2 x_3 \\ x_1^2 x_3 \end{bmatrix}$$

where each row defines a monomial and each column represents a variable. Changing the order of the rows doesn't change the polynomial whereas changing the order of the columns corresponds to changing the respective variables.

This representation of polynomials makes the problem of structure search very clear: except for the trivial cases, the number of possible monomials given n variables and a maximum joint degree d grows exponentially with either n or d. But more importantly, the polynomial regression problem can be naturally split into two subproblems:

1. For a given set of monomials $\mathcal{P} = \{q_1, \ldots, q_k\}$, find the regression coefficients $\theta_1, \ldots, \theta_k$ that minimize the error on a given dataset;
2. Find the fittest set of monomials, *i.e.* the polynomial that minimizes the error on the same dataset;

More precisely, concerning the first problem, let $\mathcal{D}$ be a dataset with n observations of variables $Y, X_1, \ldots, X_m$ and $\mathcal{P} = \{q_1, \ldots, q_k\}$ a set of k monomial expressions over $X_1, \ldots, X_m$. Define the hypothesis[1]

$$h_{\Theta,\mathcal{P}}(x_1, \ldots, x_m) = \sum_{j=1}^{k} \theta_j q_j|_{X_i = x_i, \forall 1 \leq i \leq m}$$

and let the cost

$$J_{\text{fit}}(\Theta; \mathcal{P}, \mathcal{D}) =$$

[1]The expression $q|_{X=x}$ reads "*replace all instances of X by x in q*".

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - h_{\Theta,\mathcal{P}}\left(x_1^{(i)},\ldots,x_m^{(i)}\right)\right)^2} \tag{1}$$

be the usual root-mean-square error (error) function. Now the first problem can be stated as: *Given a dataset $\mathcal{D}$ and a set of monomials $\mathcal{P}$, find parameters Θ that minimize $J_{fit}(\Theta;\mathcal{P},\mathcal{D})$.*

It turns out that this problem can be solved as a usual linear regression problem by expanding $\mathcal{D}$ with columns that replicate the monomials in $\mathcal{M}$.

The second problem is treated in the GA setting: Let $\mathcal{D}$ be a dataset as above and Q a set of polynomials. For each polynomial $p \in Q$ let $\mathcal{P}_p$ be the set of monomials in p (without the coefficients) and compute the fitness

$$\phi_p \quad = \quad \min_{\Theta} J_{\text{fit}}(\Theta;\mathcal{P}_p,\mathcal{D})$$

by solving the first problem. With a fitness of every instance, a GA will apply genetic operators (usually mutation and crossover) to evolve the population Q until a reasonable approximation of a local minimum is found. Notice that the properties of GAs and linear regression entail that the composition of GAs with linear regression, as defined in Algorithm 1, converges to a polynomial that is a local minimum of the fitness function, encapsulated in the error function J_{fit}.

Subsection 2.1 describes the encoding of individual polynomial instances as chromosomes and other parameters of the utilized GA implementation. The regularization of the cost function is discussed in subsection 2.2.

2.1. Polynomial Encoding

The encoding for any polynomial will be as follows:

Algorithm 1 GAPoly uses linear regression to find monomial coefficients that minimize the error over a dataset and GAs to explore the space of polynomials. The role of the linear regression step is to produce a fitting error that sorts the population. At exit the error of the fittest instance is bounded by ϵ.

```
function GAPoly($D, pop_0, \epsilon$)
    $pop \leftarrow pop_0$; $err \leftarrow 1.0 + \epsilon$
    while $err > \epsilon$ do
        $pop \leftarrow$ IterateGA($pop$)
        $pop \leftarrow$ Sort($pop, key = J$)    ▷ Sort population by regression error
        $err \leftarrow J$ (First($pop$))
    end while
    return First($pop$)
end function
```

1. an initial segment detailing which monomials are active (the first monomial is always active), this is represented in unary description, i.e., each monomial is identified by a single bit
2. the remaining bits are split into k sets of equal size, each one representing a monomial
3. each monomial is split into m sets of d size each, *i.e.*, a variable
4. for each variable, the remaining bits are the binary description of the variable degree, *i.e.*, the maximum exponent is given by $2^d - 1$

Let's see an example: consider polynomial $x_1^3 x_3 + x_3^7 + x_1 x_2$ with $k = 4, m = 3$ and $d = 3$. One possible encoding would be:

```
110 - 011,000,001 ; 000,000,111 ; 001,001,000 ; 110,010,101
```

(for reading purposes the semicolons separate monomials, the commas separate variables)

The first three bits inform that the second and third monomials are active while the fourth is not (as said, the first monomial is always active). This last monomial does not enter neither in the polynomial regression nor in the error (fitness) evaluation. However, it acts as a kind of junk DNA, becoming active when, in a future mutation or crossover, the third bit of the entire sequence flips from 0 to 1.

Let's interpret the first monomial description, `011,000,001`. It is divided by three since $m = 3$. The first triple `011` is the binary description of the exponent of variable x_1 which is 3, so the first monomial includes x_1^3. The second triple, `000`, means that x_2 is not part of the monomial. The third triple `001` says that variable x_3 has exponent 1. The first monomial is $x_1^3 x_3$.

Notice that all binary descriptions give rise to valid polynomials. However this is not a bijective mapping. For each polynomial there are multiple representations. For example, $x_1 + x_2$ and $x_2 + x_1$ have different representations. The authors considered that more complex mappings in order to force a one to one mapping would impact negatively in the algorithm's performance.

2.2. Cost Function

The polynomial regression error considered so far is based on the ability to predict the test set after the polynomial regression has found the appropriate coefficients θ_i for each one of the monomials q_i.

This error function tends to prefer more complex polynomials, namely in the number of monomials which provides the regression algorithm for more

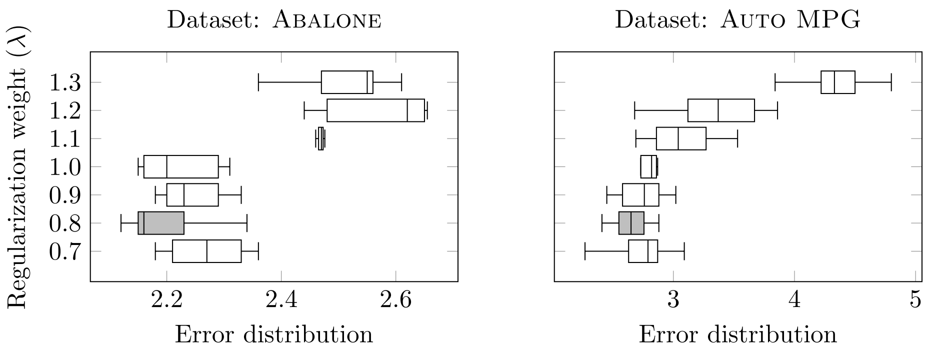

Figure 1: Error distribution by regularization exponent for the Abalone and Auto MPG datasets. The box plots summarize the error values of 10 runs for each value of λ. The smallest overall error is shown in gray and, in both cases, was achieved with $\lambda = 0.8$.

fitting possibilities. One way to balance this is to provide a regularization term into the error function. Our proposal is to include a multiplicative factor proportional to the number of monomials. Thus, the error from equation 1 defines

$$J_{\text{reg}}\left(\Theta;\mathcal{P},\mathcal{D}\right) \quad = \quad \lambda^{k} J_{\text{fit}}\left(\Theta;\mathcal{P},\mathcal{D}\right) \tag{2}$$

where k is the number of monomials in the polynomial. When $\lambda > 1$ polynomials with more monomials are penalized.

Somewhat unexpectedly after some experiences it was found that lower values for λ provide better, even if marginal, results. Figure 1 shows regularization results for the Abalone and Auto MPG datasets with ten runs for each λ. The following section includes information about these datasets.

The typical inflection point lies around $\lambda = 0.8$. The dataset results for applying the proposed regression method use the regularization parameter with this value.

2.3. Genetic Operators

To perform the genetic algorithm it was used the R package genalg (Willighagen, 2012). The operators were the standard ones: (a) crossover, *i.e.*, a pair of solutions from the previous generations are combined by splitting and mixing their respective representations, and (b) mutation, changing the values of single bits; the mutation chance applied in the datasets was 5%. There was also elitism between generations, *i.e.*, 20% of the best solutions survive to the next generation.

3. Experimental Results

The results were found using `R` programming language (R Core Team, 2013)[2]. To compare this paper's proposed algorithm we applied the exact same train and test samples using several well-known learning algorithms for regression, namely: Linear Regression, Support Vector Machines (Meyer et al., 2012), Regression Trees (Therneau et al., 2013), Conditional Inference Trees (Hothorn et al., 2006; Strobl et al., 2007, 2008) and Random Forests (Liaw and Wiener, 2002)

In order to train and test the performance of GAPoly several mainstream datasets were used. For each dataset, we selected 70% for training purposes and the remaining observations to make a test set in order to compute the estimated error. To achieve more robust results, each dataset were processed 25 times, each one with different samples for the train and test sets. For the datasets with attribute values of different magnitudes, a preliminary

[2]The datasets and the `R` code used to produce the results and plots in this paper are available online at `https://github.com/jpneto/GenAlgPoly`.

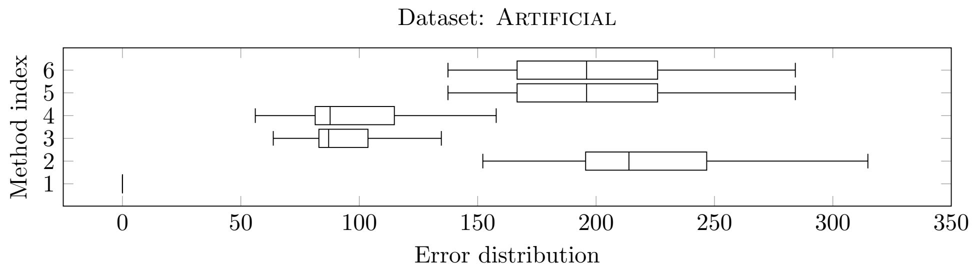

Figure 2: Results for Artificial dataset. As shown, GAPoly was able to find the exact polynomial structure that generates the dataset, reducing the test set prediction error to zero. The regression methods depicted are: 1. GAPoly, 2. Random Forests, 3. Linear Regression, 4. SVM, 5. Regression Trees and 6. Conditional Inference Trees

scaling was executed. The results below are box plots for the test set error predictions over these different runs.

Artificial this is an artificial dataset with four numeric features, $x_1, \dots x_4$, where x_1, x_3 are outcomes from Poisson random variables, and x_2, x_4 from Normal random variables. The dependent variable y is given by expression $x_2 x_4^2 + x_1^2 x_3 + 5$. The dataset includes $n = 50$ observations.

This dataset was used in order to verify if GAPoly was able to find the polynomial relation, which the algorithm did (cf. figure 2). The genetic algorithm run with a population of $n = 100$ solutions with 50 iterations for each run.

In the next datasets, the population of the genetic algorithm had size $n = 250$ with 100 iterations for each run.

Housing: This data set concerns the task of predicting housing values in areas of Boston. There are $m = 13$ continuous attributes and the

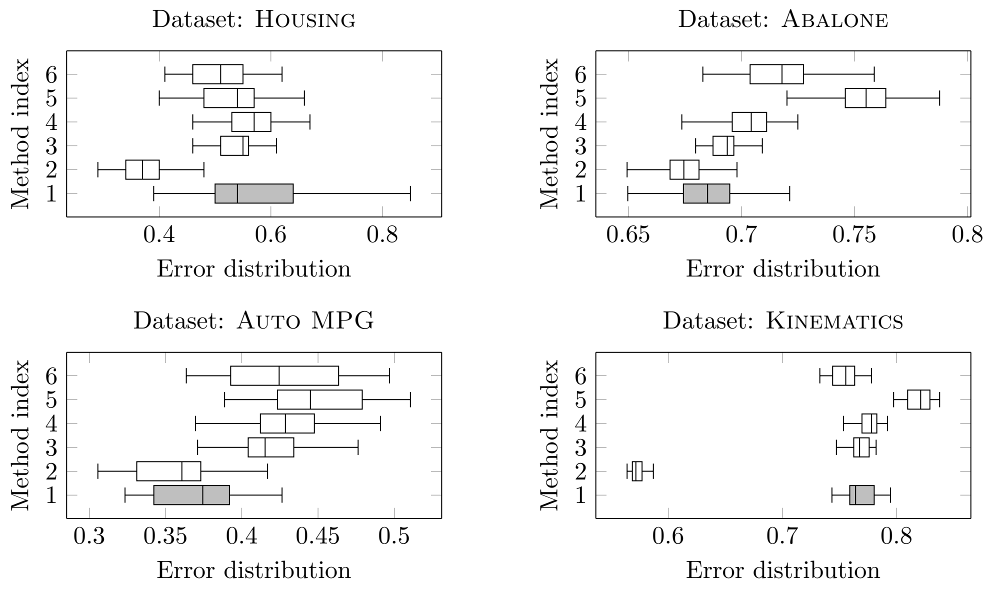

Figure 3: Summary results for different regression methods on the Housing, Abalone, Auto MPG and Kinematics datasets. Random Forests, an ensemble method ("2" in the plots), has the better results. However GAPoly achieves similar errors in two of the datasets. For the remaining two GAPoly produces similar errors to the other classic regression methods. The regression methods depicted in these figures are: 1. GAPoly, 2. Random Forests, 3. Linear Regression, 4. SVM, 5. Regression Trees and 6. Conditional Inference Trees

dependent variable is the median value of owner-occupied homes in \$1000's. There are $n = 506$ observations.

Just as an example of the model the GAPoly algorithm outputs: in this dataset the best polynomial,

$$y = -0.12x_6x_9^4 + 0.78x_6 - 0.4x_9^2x_{13} + 0.17x_{13}^2 - 0.044$$

where the attributes mean: x_6, RM average number of rooms per dwelling; x_9, RAD index of accessibility to radial highways; x_{13}, Lower status of the population.

For comparison, if we access the mean decrease in accuracy found by Random Forests, the most important attributes are — in decreasing order — x_{13}, x_6, x_5 (but x_5 is already considered 4 times less important than x_6). Both algorithms agree in two of their three most important attributes.

Abalone This dataset can be used to predict the age of a abalone shell using the given $m = 8$ numeric attributes concerning several physical measurements. There are $n = 4177$ observations.

Auto MPG This dataset is used to predict fuel consumption in miles per gallon, based on two discrete and five continuous attributes ($m = 7$). There are $n = 398$ observations.

Kinematics This dataset is concerned with the realistic simulation of the forward kinematics of an 8 link robot arm. The task is to predict the distance of the end-effector from a target using $m = 8$ continuous attributes. There are $n = 8192$ observations.

3.1. Convergence speed

The GA quickly proceeds in the first 50 to 100 generations to reasonable error rates. Then, it proceeds slower achieving best solutions with marginal error reduction. Since the entire learning process takes some time, in the current R implementation, placing a limit between 50 to 100 generations already achieves good results, relative to higher iteration values. Figure 4

Error evolution with number of iterations (Dataset: Abalone)

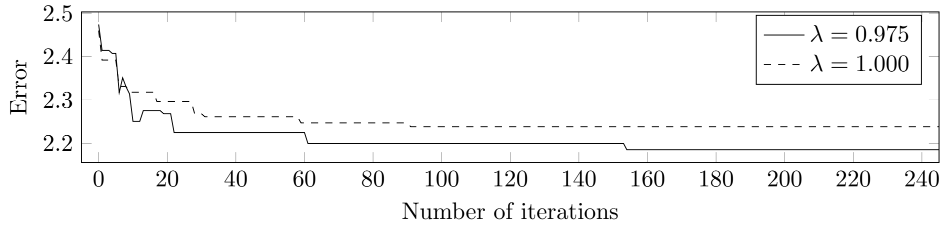

Figure 4: Error progress for Abalone dataset during a single execution of the genetic algorithm. The figure shows the fitness evolution for two different regularization values. The population for both consisted of 200 polynomials. The error values seem to stabilize around iteration 250.

shows a typical error evolution for the dataset Abalone given two different values for λ.

4. Conclusion and Future Work

Of all the non-ensemble regression methods considered, the proposed method is the one that shows, with one exception, the lowest error. Only Random Forest outperforms GAPoly systematically (it also outperforms all the other regression algorithms). The single non-ensemble exception — besides the artificial dataset that uses a straightforward polynomial relation — is the Auto MPG dataset where GAPoly has comparable results. This is evidence that applying standard genetic operators for polynomial model searching is a viable tool for regression purposes.

For complexity considerations GAPoly demands some processing time. On a current quad-core computer, processing the Kinematics dataset (with 8k observations) takes approximately 5 minutes. The processing time can probably be speeded by one to two orders in magnitude if the process is implemented in a low level programming language like `C++`. However, speed optimization was not the focus of this article.

A cross-validation procedure can be implemented to refine the appropriate parameter values to achieve better errors. Namely, the regularization parameter, λ, can be tested with several values, instead of being fixed at 0.8. Other parameters like mutation chance or the amount of elitism could also be tested. However, these type of tests need a low-level, fast implementation of GAPoly.

Acknowledgements

The authors are grateful to the Fundação para a Ciência e Tecnologia (FCT) and the R&D laboratory LabMAg for the financial support given to this work, under the strategic project PEst-OE/EEI/UI0434/2011.

The datasets used herein were selected from Luís Torgo's data repository, `http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html`. Most come originally from UCI ML repository, `http://archive.ics.uci.edu/ml/`.

The authors wish to thank professor André Falcão for motivation and useful discussions around the article.

Barricelli, N.A., 1962. Numerical testing of evolution theories. part i: Theoretical introduction and basic tests. Acta Biotheoretica 16, 69–98.

Bengio, Y., 2009. Learning deep architectures for AI. Foundations and trends in Machine Learning 2, 1–127.

Bengio, Y., Courville, A., Vincent, P., 2013. Representation learning: A review and new perspectives .

Cetisli, B., Kalkan, H., 2011. Polynomial curve fitting with varying real powers. Electronics and Electrical Engineering 112, 117–122.

Chan, K.Y., Dillon, T.S., Kwong, C.K., 2011. Modeling of a liquid epoxy molding process using a particle swarm optimization-based fuzzy regression approach. Industrial Informatics, IEEE Transactions on 7, 148–158.

Chan, K.Y., Kwong, C., Dillon, T.S., 2012. Development of product design models using fuzzy regression based genetic programming, in: Computational Intelligence Techniques for New Product Design. Springer, pp. 111–128.

Caéfalvayová, L., Pelikan, M., Kralj Cigić, I., Kolar, J., Strlič, M., 2010. Use of genetic algorithms with multivariate regression for determination of gelatine in historic papers based on FT-IR and NIR spectral data. Talanta 82, 1784–1790.

Davidson, J., Savic, D., Walters, G., 1999. Method for the identification of explicit polynomial formulae for the friction in turbulent pipe flow. Journal of Hydroinformatics 1, 115–126.

Gálvez, A., Iglesias, A., Puig-Pey, J., 2012. Iterative two-step genetic-algorithm-based method for efficient polynomial b-spline surface reconstruction. Information Sciences 182, 56–76.

García Nieto, P.J., Alonso Fernández, J., de Cos Juez, F., Sánchez Lasheras, F., Díaz Muñiz, C., 2013. Hybrid modelling based on support vector regression with genetic algorithms in forecasting the cyanotoxins presence in the trasona reservoir (northern spain). Environmental research .

Hofwing, M., Strömberg, N., Tapankov, M., 2011. Optimal polynomial regression models by using a genetic algorithm, in: Proceedings of the Second International Conference on Soft ComputingTechnology in Civil, Structural and Environmental Engineering Conference,(Crete, Greece), 2011009.

Holland, J.H., 1975. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press.

Hothorn, T., Hornik, K., Zeileis, A., 2006. Unbiased recursive partitioning: A conditional inference framework. Journal of Computational and Graphical Statistics 15, 651–674.

Koza, J.R., 1992. Genetic Programming: vol. 1, On the programming of computers by means of natural selection. volume 1. MIT press.

Liaw, A., Wiener, M., 2002. Classification and regression by randomforest. R News 2, 18–22. URL: `http://CRAN.R-project.org/doc/Rnews/`.

Maertens, K., De Baerdemaeker, J., Babuška, R., 2006. Genetic polynomial regression as input selection algorithm for non-linear identification. Soft Computing 10, 785–795.

Mendes, L., Carvalho, P.d., 2005. Adaptive polynomial regression for colorimetric scanner calibration using genetic algorithms, in: Intelligent Signal Processing, 2005 IEEE International Workshop on, IEEE. pp. 22–27.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., Leisch, F., 2012. e1071: Misc functions of the department of statistics (e1071), tu wien URL: `http://CRAN.R-project.org/package=e1071`. r package version 1.6-1.

R Core Team, 2013. R: A language and environment for statistical computing URL: `http://www.R-project.org/`.

Sánchez, L., Otero, J., Couso, I., 2009. Obtaining linguistic fuzzy rule-based regression models from imprecise data with multiobjective genetic algorithms. Soft Computing 13, 467–479.

Strobl, C., Boulesteix, A.L., Kneib, T., Augustin, T., Zeileis, A., 2008. Conditional variable importance for random forests. BMC Bioinformatics 9. URL: `http://www.biomedcentral.com/1471-2105/9/307`.

Strobl, C., Boulesteix, A.L., Zeileis, A., Hothorn, T., 2007. Bias in random forest variable importance measures: Illustrations, sources and a solution. BMC Bioinformatics 8. URL: `http://www.biomedcentral.com/1471-2105/8/25`.

Tarlow, D., Sutskever, I., Zemel, R.S., 2013. Stochastic k-neighborhood selection for supervised and unsupervised learning. Journal of Machine Learning Research .

Therneau, T., Atkinson, B., Ripley, B., 2013. rpart: Recursive partitioning R package version 4.1-1.

Willighagen, E., 2012. genalg: R based genetic algorithm.

Wu, C.H., Tzeng, G.H., Lin, R.H., 2009. A novel hybrid genetic algorithm for kernel function and parameter optimization in support vector regression. Expert Systems with Applications 36, 4725–4735.

Yu, T.L., Lin, W.K., 2008. Optimal sampling of genetic algorithms on polynomial regression, in: Proceedings of the 10th annual conference on Genetic and evolutionary computation, ACM. pp. 1089–1096.