Empirical Emulators for Process Monitoring and Optimization

Arthur K. Kordon, Alex N. Kalos, and Bryan Adams

Abstract-- A novel methodology for the development of empirical emulators based on analytic neural networks and genetic programming is proposed. The capability of the analytic neural net-based emulator to detect unreliable predictions with a model disagreement indicator is of critical importance to process monitoring and optimization. Genetic programming-based emulators generate explicit functional models that are more convenient for optimization and on-line implementation than black-box solutions. The advantages of the proposed methodology are illustrated with an industrial application in the chemical industry.

Index Terms-- Empirical emulators, neural networks, genetic programming, process monitoring and optimization

I. Introduction

Empirical emulators mimic the performance of first principle models by using various data-driven modeling techniques. The driving force to develop empirical emulators is the push for reducing the time and cost for new product or process development [1]. Empirical emulators are especially effective when hard real-time optimization of a variety of complex fundamental models is needed [2]. The increased robustness of modern data-driven techniques such as analytic neural networks [3], support vector machines [4], genetic programming [5], etc. is a reliable basis for accurate representation of the behavior of fundamental models. This makes it possible to substitute fundamental models with their empirical emulators and provides many opportunities for effective synergy of these two key modeling approaches.

Most of the known empirical emulators are implemented as "classical" neural networks based on a back-propagation learning algorithm [1], [6]. Their property of being universal approximators is a key theoretical result for successful emulation [7]. However, "classical" neural networks suffer from a number of problems such as: long computational time for training, convergence to local minima, sensitivity to weight initialization, too many tunable parameters, etc. These problems put serious limitations on the quality of the developed empirical model, increase development time, and require experienced model developers.

All authors are affiliated with The Dow Chemical Company. A. K. Kordon and A. N. Kalos are located at 2301 N. Brazosport Bvd., Bldg B-1217, Freeport Texas, 77541. B. Adams is at Louisiana Hwy 1, PO Box 150 Plaquemine, Louisiana, 70765.

An alternative empirical emulator can be based on analytic neural networks [3]. A key advantage of analytic neural networks is that the function to be optimized is a quadratic function of the error and has one global optimum. This type of neural network has been successfully implemented in several industrial applications in The Dow Chemical Company [8].

Another approach for building a successful empirical emulator uses genetic programming [5]. By mimicking natural evolution and using genetic operators such as crossover and mutation, genetic programming delivers empirical models in a form of explicit analytic functions mapping process inputs to outputs. The main advantage of this type of emulators is its ease of implementation.

The development of empirical emulators for process monitoring and optimization based on analytic neural networks and genetic programming is described in this paper. They are used in a real chemical process in The Dow Chemical Company.

II. Motivation for developing Empirical Emulators

The primary motivation for developing an empirical emulator of a first principle model is to facilitate the on-line implementation of a model for process monitoring and control. Often times it may prove difficult or impractical to incorporate a first principles model directly within an optimization framework. For example, the complexity of the model may preclude wrapping an optimization layer around it. Or, the model may be implemented in a different software/hardware platform than the Distributed Control System (DCS) of the process, again preventing its on-line use. In other occasions, the source code of the model may not even be available. In such circumstances, an empirical emulator of the fundamental model can be an attractive alternative. An additional benefit is that there is a significant acceleration of the execution of the on-line model: the computational gain is on the order of $10^3 - 10^5$ times faster.

III. Empirical Emulators Structures

The most obvious scheme for utilization of empirical emulators is for complete "replacement" of a fundamental model. The key feature of this scheme, shown in Figure 1, is that the emulator represents the fundamental model entirely and is used as a stand-alone on-line application. This scheme is appropriate when the fundamental model does not include too many input variables and a robust and

parsimonious empirical model can be built from the available data generated by design of experiments (DOE).

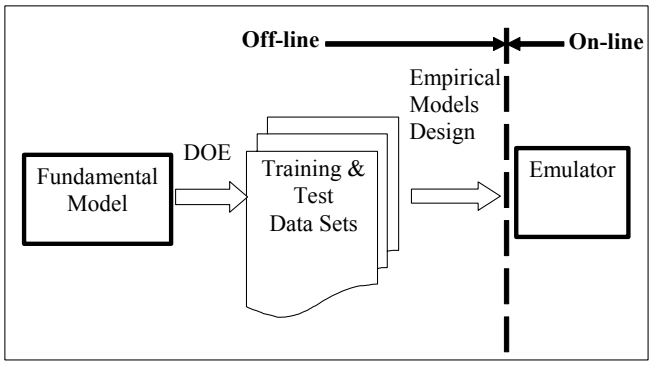

Figure 1. Empirical emulator as accelerator of fundamental models.

In case of higher dimensionality and model complexity, a hybrid scheme of fundamental model and emulator integration is recommended (see Fig.2). Emulators based only on sub-models with high computational load are developed off-line using different training data sets. These emulators substitute the related sub-models in on-line operation and enhance the speed of execution of the original fundamental model. This scheme is of particular interest when process dynamics have to be taken into account in the modeling process.

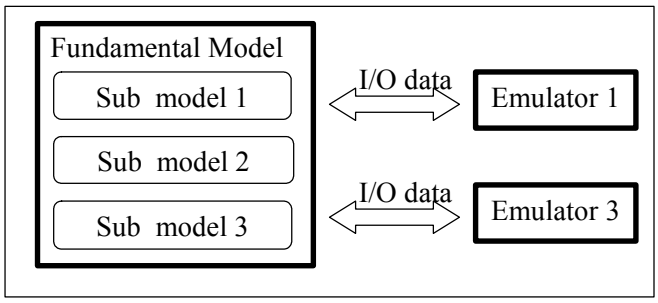

Figure 2. Hybrid scheme of empirical emulators and fundamental models.

Finally, an item of special importance to on-line optimization is the scheme (shown in Fig.3) where the empirical emulator is used as an integrator of different types of fundamental models (steady-state, dynamic, fluid, kinetic, thermal, etc).

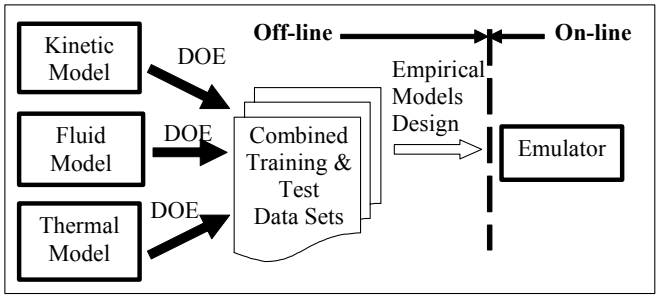

Figure 3. Empirical emulator as integrator and accelerator of fundamental models.

In this structure, data from several fundamental models can be merged and a single empirical model can be developed on the combined data. The empirical emulator, as integrator of different fundamental models, offers two main advantages for on-line implementation [6]. The first advantage is that it is simpler to interface only the inputs and outputs from the models than the models themselves. More importantly, when constructing the data sets by DOE, the developer selects only those inputs/outputs that are significant to the optimization. Hence, the emulator is a compact empirical representation of only the information that is pertinent to the optimization.

The second advantage is that one optimizer can address the whole problem, rather than trying to interface several separate optimizers. The optimization objectives, costs, constraints, algorithm, and parameters are more consistent, allowing the multi-model problem to be solved more efficiently.

IV. Contemporary Machine Learning Techniques for Building Empirical Emulators

There are different modeling techniques to build empirical emulators (for current survey, see [9]). The majority of applications discussed in the literature are based on "classical" back-propagation neural nets [1], [2], [6]. Regardless of illustrating the validity of the approach, the delivered solutions have inefficient structures and cannot extrapolate well outside the training range. The last problem is of critical importance if scale-up of the emulator is necessary.

Three contemporary machine learning techniques permit building empirical models (e.g., emulators) with robust performance and potential for good generalization outside the training range. The first technique is the analytic neural network [3]. Analytic neural nets are based on a collection of individual, feedforward, single layer neural networks, where the weights of the input to hidden layer have been initialized according to a fixed distribution such that all hidden nodes are active. The weights of the hidden to output layer can then be calculated directly using least squares minimization techniques. The advantage of this method is that it is fast and each neural network has a well-defined, single, global optimum. Since the global optimum is guaranteed by design, it is no longer possible to get stuck in local minima and the learning algorithm is not iterative. As a result, the data-driven modeling process is significantly reduced and the developed empirical models are parsimonious. In addition, the use of a collection of networks gives more robust models, which includes a model disagreement indicator, based on the standard deviation of the output of the stacked neural nets. Such indicators enable the emulators to be "aware" of their own performance which is essential for any data-driven model, especially for on-line, real-time applications.

The second technique for robust emulator building is symbolic regression performed by genetic programming (GP). GP-based symbolic regression is the mathematical analogue of natural selection, where numerous potential mathematical expressions are evolved, eventually resulting in a list of "the best and the brightest" analytical forms, according to a fitness selection objective function. The

following unique features of GP are of special importance to industry [8]:

1. no *a priori* modeling assumptions
2. derivative-free optimization
3. "natural" selection of the most important process inputs
4. parsimonious analytical functions as a final result.

The last feature has double benefits. On one hand, a simple empirical emulator often has better generalization capability and is more reliable for operation outside the training range. On the other hand, process engineers and developers prefer to use non-black box empirical models and are much more open to take the risk to implement emulators based on functional relationships. An additional advantage is the low implementation cost of such type of emulators, because they can be deployed directly in the existing DCS avoiding additional specialized software packages, which is typical for many commercially available neural network-based models.

The third technique for robust emulator building is support vector machines (SVM). Some recent results show that by using a mixture between global (polynomial) and local (Radial Basis Function) kernels, empirical models with extremely good generalization capability can be built [10]. However, SVMs are still an area of active research and will not be covered in detail in this paper.

V. A Methodology for Empirical Emulators Development

The objective of the proposed methodology is to optimize the advantages of the different technologies in order to build empirical emulators with high prediction quality with minimal development effort. The key steps of the methodology are as follows.

A. Step 1: Define empirical emulator performance

This step specifies the requirements for a successful empirical emulator such as expected accuracy of prediction relative to the fundamental model, detection of areas with unreliable model prediction, optimization constraints, off-line or on-line model implementation, software environment, maintenance and support.

B. Step2: Identify critical process variables

Since the purpose of the empirical emulator is to capture the behavior of a complex fundamental model, only the most significant process variables have to be taken into account. However, reducing the complexity must not be at the expense of reduced accuracy of prediction. The selection of the critical variables is based on the nature of the problem and the existing fundamental model. In addition, factors such as the possibility for wide-range variable changes, influence on profit, availability of measurements, and statistical measures (correlation coefficients, R^2, or variable importance in multivariate analysis) are taken into consideration.

C. Step3: Planned Design Of Experiments (DOE)

One advantage of empirical emulators is that the training data for building the emulator can be generated using design of experiments (DOE) strategies. This allows a high degree of freedom in developing reliable data-driven models. Even though the "experiments" are performed in a virtual rather than a physical plant, executing the runs still demands time, resources and money; so judicious experiment selection is important. Still, we have the luxury to do runs in regions where we may never be able to push a physical plant because of safety or economic constraints.

The type of DOE (two-level, three-level, mixed-level, or fractional factorial designs [11]) depends on the number of critical variables and the expected nonlinear behavior of the response surface. It is also possible that the fundamental model cannot converge in all edges of the designed space and a new DOE sequence may need to be defined.

D. Step4: Generate data sets

This step delivers the source of data for empirical emulator development. In order to improve the quality and robustness of the designed model, the generation of the following three data sets is recommended: The "training data set" is used for training the neural net (classical or analytic) and for performing symbolic regression with GP. Usually, this is the data set that is generated by DOE and covers the maximal range of input variables. The second data set is the "test data set" which is used to validate the interpolation quality of the emulator and it is generated by random selection of input values within the training range. This data set is also used during the neural net development for selection of the optimal structure (the number of the neurons in the hidden layer), and it is used to validate the performance of the GP-generated analytical functions. The third data set is the "extreme-range data set" that checks the extrapolation capability of the emulator. Usually it includes several data points with critical inputs 10-20% outside the training range. This data set is necessary to test if the model disagreement indicator (a type of confidence limit, discussed in more detail later) is able to detect "uncharted" territory and to raise alarm flags for invalid predictions outside the training range. Another purpose of the extreme data set is to test how the modeling performance degrades, especially in operational regions where possible future changes are expected.

E. Step5: Emulator design

The key step of the methodology includes two options:

1) Emulator design based on neural networks

This type of emulator is in the form of a black-box model. The designed neural network can be based on a variety of architectures (feedforward, recurrent, analytic, etc.) and numerous learning/optimization algorithms (back propagation, Levenberg-Marquadt, Gauss-Newton, etc.) [3], [7]. The critical design parameter that controls the prediction quality is the model complexity or the number of the neurons in the hidden layer. Usually this is determined empirically by selecting a neural network structure with the minimal prediction error on the test data set [7]. Due to the

availability of a balanced data set from DOE and the good interpolation properties of the neural networks, this type of empirical emulator mimics the fundamental model with high fitness (R^2 ~ 0.95-0.99). The development process is relatively fast, however, it requires specialized knowledge of neural networks.

2) Emulator design based on symbolic regression

The second option to mimic the behavior of fundamental models is by a set of nonlinear analytical functions. In contrast to the neural-network-based emulator the model is not a black-box but an explicit analytical function. It is a result of a simulated evolution based on the genetic programming algorithm [5]. The convergence of the algorithm toward high fitness functions can be improved if the initial set of functions includes transformations that reflect the chemistry and physics of the process. Since the source of the emulator is a fundamental model based on physico-chemical laws, appropriate functional dependencies (like the Arrhenius law) can be extracted and used in the symbolic regression generation.

Due to the random nature of the simulated evolution, it is necessary to repeat the design several times. The development process requires significant computational resources and the manipulation of several design parameters (population size, number of generations, number of reproductions per generation, probability for function selection as next node, etc.) Of special importance is the parsimony pressure parameter that controls the complexity of the generated functions.

F. Step6: Off-line emulator validation

The performance of the derived empirical model is tested off-line in as many scenarios as possible. Usually, at this stage of development, the emulator is integrated with the optimization package. It is recommended that a complete set of test cases is performed, particulary with inputs on the edges of the training range and with different constraints.

G. Step7: On-line emulator implementation

In many cases the final implementation is an on-line empirical model. If the emulator is based on neural networks, specialized software may be required for on-line deployment (such as Process Insights from Pavilion Technologies, Inc., or NeurOnline from Gensym Corporation). This can add additional cost to the emulator development and maintenance. In contrast, an empirical emulator based on symbolic regression can be directly implemented in the control system, precluding the need of specialized packages.

VI. A CASE STUDY: AN EMPIRICAL EMULATOR FOR OPTIMIZATION OF AN INDUSTRIAL CHEMICAL PROCESS

A. Problem definition

A significant problem in the chemical industry is the optimal handling of intermediate products. Of special interest are cases where intermediate products from one process can be used as raw materials for another process in different geographical locations. The case study is based on a real industrial application of intermediate products optimization between two plants in the Dow Chemical Company, one in Freeport, Texas and the other in Plaquemine, Louisiana. The objective is to maximize the intermediate product flow from the plant in Texas and to use it effectively as a feed in the plant in Louisiana. The experience of using a huge fundamental model for "what-if" scenarios in planning the production schedule was not favorable because of the specialized knowledge required and the slow execution speed (~20-25 min/prediction). Empirical emulators are a viable alternative to solve this problem. The objective is to develop an empirical model which emulates the existing fundamental model with acceptable accuracy (with R^2 ~ 0.9) and which can significantly speed up the calculation time (< 1 sec).

B. Data preparation

Ten input variables (different product flows) were selected by the experts from several hundred parameters in the fundamental model. There are 12 output variables that need to be predicted and used in process optimization. The assumption was that the behavior of the process can be captured with these most significant variables and that a representative empirical model could be built for each output. A 32-run Plackett-Burman experimental design with 10 factors at four levels was used as the DOE strategy [11]. The training data set consisted of 320 data points. For 15 of these the fundamental model did not converge for three of the outputs. The test data set included 275 data points where the inputs were randomly generated within the training range. The extreme-range data set consisted of 181 data points with some of the inputs at 8-10% outside the training range.

C. Empirical emulator based on analytic neural networks

Several runs with different numbers of hidden nodes were done and the results for all 12 emulators are summarized in Table 1.

Table 1. Performance of all emulators on training and test data.

Output	R^2 NN Training	R^2 NN Test	# Hidden nodes
Y1	0.91	0.89	30
Y2	0.994	0.989	20
Y3	0.984	0.979	20
Y4	0.987	0.981	20
Y5	0.991	0.967	30
Y6	0.999	0.999	1
Y7	0.995	0.999	1
Y8	0.995	0.993	10
Y9	0.994	0.992	10
Y10	0.992	0.993	1
Y11	1	1	1
Y12	0.997	0.989	20

The structure of the neural network for each emulator includes 10 inputs and one output. The same set of inputs is used for all emulators. Since single hidden-layer analytic neural networks are based on direct optimization, the only design parameter to be adjusted is the number of neurons in the hidden layer. A number of different structures (with between 1 and 50 hidden nodes) were constructed and each neural net was optimized based on the training data set. The optimal number of hidden nodes was then determined by applying each neural network to the test data set and selecting the structure with the minimal R^2 value. This procedure was repeated for each emulator.

One special feature of the analytical neural network is the method of initializing the random weights between the input and the hidden layer. In order to minimize the effect of randomization, it is possible to use a stack of many neural networks with the same complexity (i.e., with the same number of hidden nodes) for each emulator. In principle, combined predictors have better properties than individual models [12]. An advantage of this approach is that the final prediction is based on the average of all models in the ensemble. Of even greater practical importance is that the standard deviation between the individual predictors can be used to develop a model disagreement measure which is a type of a confidence indicator for the stacked neural net models and also adds some self-assessment capability to the emulator.

As it is shown in Table 1, all emulators have acceptable accuracy on the training and test data. An example of emulator performance for emulator Y5 is shown in Figure 4 for the training data set and in Figure 5 for the test data set. In both cases, the stacked ensemble model disagreement is shown (at the bottom of the figures) on the same scale as the actual data and thus the range of its magnitude is relatively small. As expected, it is somewhat larger for the testing data.

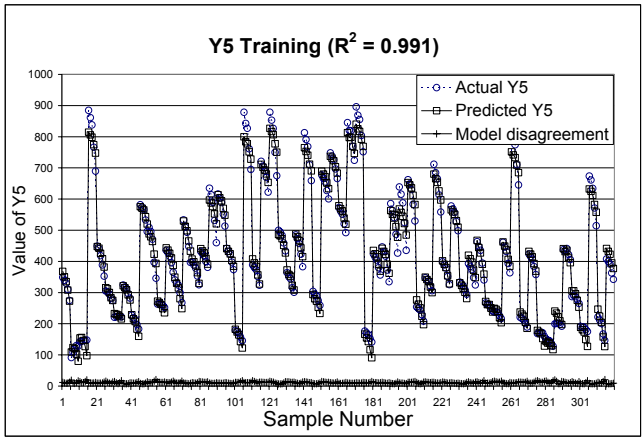

Figure 4. Emulator Y5 actual and predicted values and model disagreement indicator based on training data.

The performance varies between R^2 0.89 for Y1 and the perfect fit for Y11. The neural network complexity also varies – from an almost linear structure of 1 hidden node for Y6, Y7, Y10, and Y11 to a structure with 30 hidden nodes for Y1 and Y5. The prediction quality is good in all ranges and the model disagreement indicator is low.

The role of the model disagreement indicator is to detect areas where the predictions are unacceptable due to the degradation of neural net performance outside the training range. This is illustrated in Fig. 6.

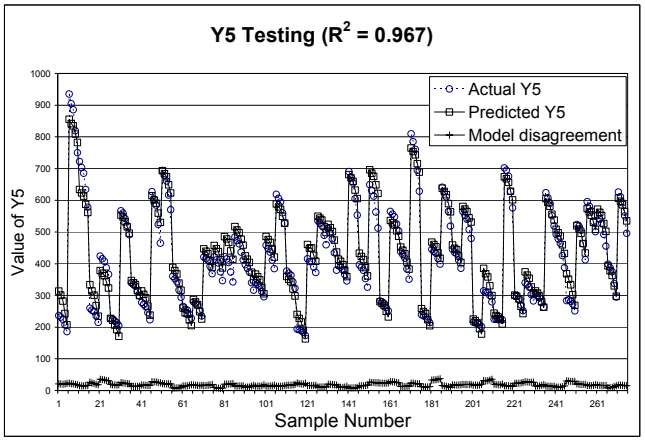

Figure 5. Emulator Y5 actual and predicted values and model disagreement indicator based on test data.

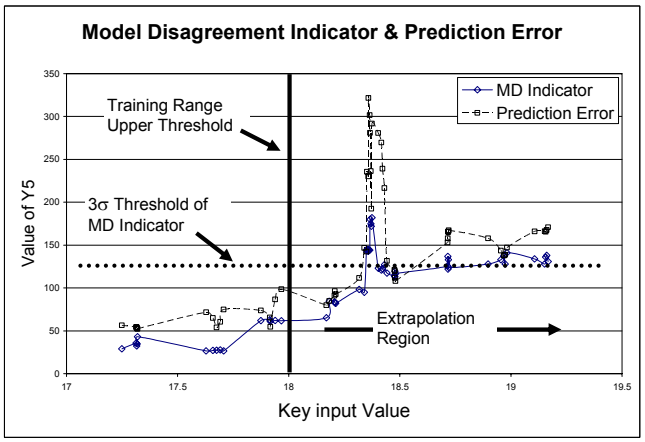

Figure 6. Model disagreement (MD) indicator and prediction error on the threshold of training data.

The figure shows the trends of the model disagreement indicator (based on the standard deviation of 30 stacked neural networks each with 30 hidden nodes) and the prediction error around the threshold of the training data range for one of the key inputs. The upper threshold of the training range for this key input is 18 while the 3σ threshold for the model disagreement indicator is 124 (based on the model disagreement on the test data). It is observed that below the upper threshold of the training range for this input, the model prediction is relatively low. The prediction error is still acceptable even up to about 6% above this limit, where there is a sharp spike in the error. At the same time, the model disagreement indicator tracks the prediction error very well (the error spike is above the 3σ threshold for the indicator). Therefore, this shows that the model disagreement threshold is a good metric of emulator performance. Furthermore, such an indicator can

be easily implemented on-line, along with the emulator itself.

D. *Empirical emulators based on symbolic regression*

Due to the statistical nature of GP, several sets of simulated evolution runs need to be executed. The symbolic regression model is derived from a population size of 200 potential functions that evolve during 300 generations with 0.5 probability for random crossover, 0.3 probability for mutation of functions and terminals, 4 reproductions per generation, 0.6 probability for selecting a function as the next node, and correlation coefficient as the optimization criterion. The initial functional set for the GP includes: {addition, subtraction, multiplication, division, square, square root, sign change, natural logarithm, exponential, and power}. The complexity of the final solution can be controlled through a parsimony pressure parameter that penalizes more complex functions with lower goodness of fit. A value of 0.05 is a good compromise between the complexity of the derived functions and their fitness. An example of a GP-based symbolic regression emulator for Y5 is shown below where, x_1 to x_{10} are the emulator inputs.

(Equation 1):

$$Y5 = 3x_9 + \frac{6x_3 + x_4 + x_5 + 2x_6 + x_2x_9 - 3x_{10} - \frac{x_2 - 3x_3 - x_5 + \sqrt{x_6} - 2x_7 - x_9 + x_{10}}{\mathrm{Log}[x_2]^2}}{x_2}$$

The performance on the training data set (shown in Figure 7, R^2 =0.94) and on the test data set are comparable.

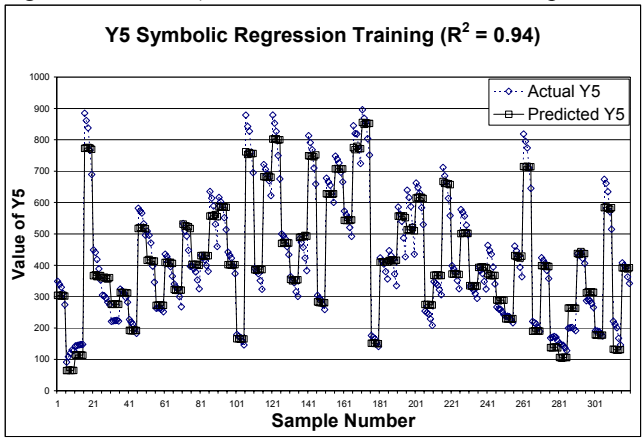

Figure 7. Emulator Y5 actual and predicted values from symbolic regression on training data.

In summary, the performance of stacked analytic neural nets is generally better than that of GP-generated emulators (training R^2 0.99 vs. 0.94; test R^2 0.97 vs. 0.94). This is consistent with the theoretical property of neural nets being universal approximators. Still, the performance of GP-generated functions is within acceptable accuracy of prediction. Deciding which method to use depends on the application: Stacked analytic neural nets offer the potential for self-assessment of unreliable model predictions and the model disagreement indicator, which are of critical importance for on-line process monitoring and optimization. Although they are faster to develop, they are somewhat slower in execution due to the fact that an ensemble of models needs to be calculated. On the other hand, symbolic regression-based emulators require much longer development time due to the computationally intensive GP algorithm and nontrivial model selection. However, end users are more comfortable optimizing the process with an analytical function, such as equation (1), than with black-box models.

VII. CONCLUSIONS

Empirical emulators mimic well the behavior of large fundamental models and effectively represent them in time-critical applications in process monitoring and control. A novel methodology for the development of empirical emulators based on analytic neural networks and genetic programming is presented in this paper. The methodology systematizes the key steps for effective emulator development such as the identification of critical process variables, planning the fundamental model-based DOE, generation of training, test and "extreme" data sets, as well as issues regarding emulator design, off-line validation, and on-line implementation. The advantages of empirical emulators based on the proposed methodology, demonstrated in a real industrial application for intermediate product optimization between two chemical plants in The Dow Chemical Company are:

1. Analytic neural network-based emulators have a black-box structure derived from a direct optimization with guaranteed global optimum;
2. The analytic neural network-based emulator development is very fast in comparison to back-propagation neural networks;
3. The use of stacked analytical neural networks improves the prediction quality and gives self-assessment capability based on a model disagreement indicator;
4. GP-based symbolic regression generates emulators that represent the fundamental model with explicit functions which can be directly included in optimization packages for on-line implementation.

REFERENCES

[1] Wynn H. and R. Bates, *Emulator Technology in Engineering Design*, Proc. Instn Mech Engrs, **213 B**, pp. 305-309, 1999.

[2] Bates R. *et al*, *Fast Optimization of Mechanical Designs Using CAD Emulation: A Case Study*, Proc. Instn Mech Engrs, **213 B**, pp. 27-35, 1999

[3] Smits G., personal communication, 1999.

[4] Vapnik, V. *Statistical Learning Theory*, Wiley, NY, 1998.

[5] Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.

[6] Thompson W., G. Martin, and N. Bhat, *How Neural Network Modeling Methods Complement Those of Physical Modeling*, NPRA Computer Conference, Atlanta, GA, 1996.

[7] Haykin, S. *Neural Networks: A Comprehensive Foundation,* Prentice Hall, New York, 1998

[8] Kordon A. G.Smits, E. Jordaan and E. Rightor, "Robust Soft Sensors Based on Integration of Genetic Programming, Analytical Neural Networks, and Support Vector Machines" *In Proceedings of WCCI 2002*, Honolulu, HW: IEEE Press, pp. 896 – 901, 2002.

[9] Berthold M. and D. Hand, *Intelligent Data Analysis: An Introduction*, Springer, Berlin, 1999.

[10] Smits G. and E. Jordaan, "Using Mixtures of Polynomial and RBF Kernels for Support Vector Regression", *In Proceedings of WCCI'2002*, Honolulu, HW: IEEE Press, 2002, pp. 2785 – 2790.

[11] Montgomery D. *Design and Analysis of Experiments*, John Wiley&Sons, NY, 2001.

[12] Sharkely A. (Editor), *Combining Artificial Neural Nets*, Springer, London, 1999.