Chapter 4
Response Surface Modeling for Design Space Exploration of Embedded Systems

Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, Enrico Rigoni, Carlos Kavka, Alessandro Turco, and Giovanni Mariani

Abstract A typical design space exploration flow involves an event-based simulator in the loop, often leading to an actual evaluation time that can exceed practical limits for realistic applications. Chip multi-processor architectures further exacerbate this problem given that the actual simulation speed decreases by increasing the number of cores of the chip. Traditional design space exploration lacks of efficient techniques that reduce the number of architectural alternatives to be analyzed. In this chapter, we introduce a set of statistical and machine learning techniques that can be used to predict system level metrics by using closed-form analytical expressions instead of lengthy simulations; the latter are called *Response Surface Models* (RSM). The principle of RSM is to exploit a set of simulations generated by one or more Design of Experiments strategies to build a *surrogate model* to predict the system-level metrics. The response model has the same input and output features of the original simulation-based model but offers significant speed-up by leveraging analytical, closed-form functions which are tuned during *model training.* The techniques presented in this chapter can be used to improve the performance of traditional design space exploration algorithms such as those presented in Chap. 3.

4.1 Introduction

Nowadays, Multi-Processor Systems-on-Chip (MPSoCs) and Chip-Multi-Processors (CMPs) [5] represent the *de facto* standard for both embedded and general-purpose architectures. In particular, programmable MPSoCs have become the dominant computing paradigm for application-specific processors. In fact, they represent the best compromise in terms of a stable hardware platform that is software programmable, thus customizable, upgradable and extensible. In this sense, the MPSoC paradigm minimizes the risk of missing the time-to-market deadline

V. Zaccaria (✉)
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: zaccaria@elet.polimi.it

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_4,

while ensuring greater efficiency due to architecture customization and software compilation techniques.

Design space exploration involves an event-based simulator in the loop. Event-based simulation still represents a fundamental tool to predict performance of candidate architectural design points. If we consider cycle-accurate system-level simulation (where the *event* corresponds to the completion of a processor clock cycle), we can have several high-complexity mathematical models to be evaluated during each event (or clock cycle), leading to an actual evaluation time that can exceed practical limits for realistic applications. Chip multi-processor architectures further exacerbate this problem given that the actual simulation speed decreases by increasing the number of cores of the chip (as shown in Fig. 4.1).

While statistical sampling techniques have already been proposed for a single simulation [12], design space exploration still lacks of efficient techniques that reduce the number of architectural alternatives to be analyzed. To face this problem, we decided to adopt statistical and machine learning techniques to create a prediction of the system level metrics for the whole simulation by using closed-form analytical expressions; the latter are called *Response Surface Models* (RSM) since they represent a suitable approximation of the system response under a specific instance of the input set of parameters (e.g. the system configuration).

This chapter is organized as follows: Sect. 4.2 presents some background on response surface models while Sect. 4.3 analyzes some of the very peculiar problems that arise when modeling embedded design spaces. Section 4.4 presents a detailed

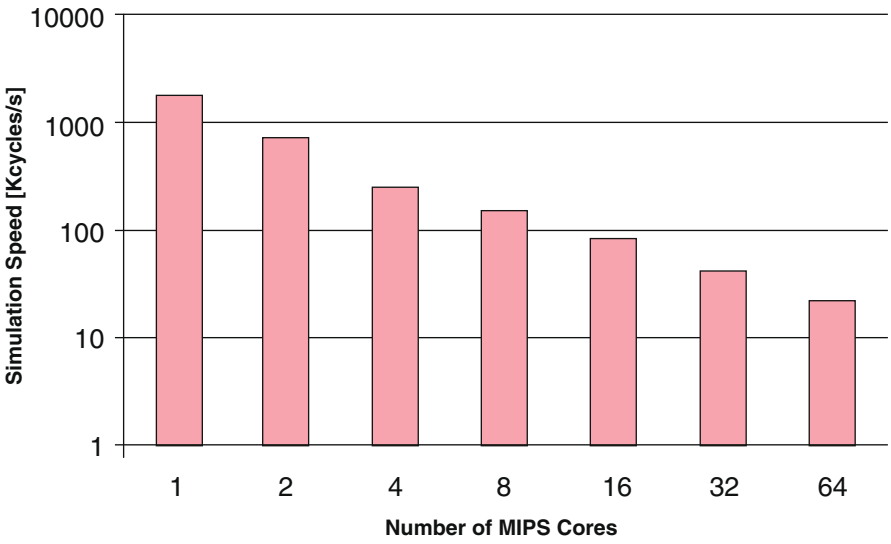

Fig. 4.1 SESC [15] simulation speed when executing the FFT kernel from [18] by varying the number of MIPS cores from 1 to 64 (Host machine: two Intel Xeons quad-core at 3 GHz)

descriptions of the RSM algorithms while Sect. 4.5 presents the general validation flow. Section 4.6 summarizes the main content of this chapter.

4.2 Background on Response Surface Models

RSM techniques are typically introduced to decrease the time due to the evaluation of a system-level metric $f(\mathbf{x})$ for each architecture configuration $\mathbf{x}$[1]. In fact, for applications of commercial interest, evaluating $f(\mathbf{x})$ can involve one or more simulations which can take several hours, depending on the platform complexity and the system resources dedicated to the simulation.

The principle of RSM is to exploit a set of simulations generated by a Design of Experiment strategy to build a *surrogate model* to predict the system-level metrics. The response model has the same input and output features of the original simulation-based model but offers dramatic speed-up in terms of evaluation since it consists of an analytical, closed-form function.

A typical RSM-based flow involves a *training phase*, in which simulation data (or *training set*) is used to tune the RSM, and a *prediction phase* in which the RSM is used to forecast unknown system response (see Fig. 4.2).

Given a system-level metric $f(\mathbf{x})$ associated with an architectural configuration $\mathbf{x}$, a response surface model $\rho(\mathbf{x})$ is defined such that:

$$f(\mathbf{x}) = \rho(\mathbf{x}) + \varepsilon \tag{4.1}$$

where ε is an ideally negligible estimation error. The prediction $\rho(\mathbf{x})$ is a function of the target architecture $\mathbf{x}$.

In simple cases, $\rho(\mathbf{x})$ may consists of a *closure*[2] of a more general function $\Phi(\mathbf{x}, \mathbf{w})$ where the vector of parameters $\mathbf{w}$ has been fixed to $\mathbf{w}_0$:

$$\rho(\mathbf{x}) = \Phi(\mathbf{x}, \mathbf{w}_0) \tag{4.2}$$

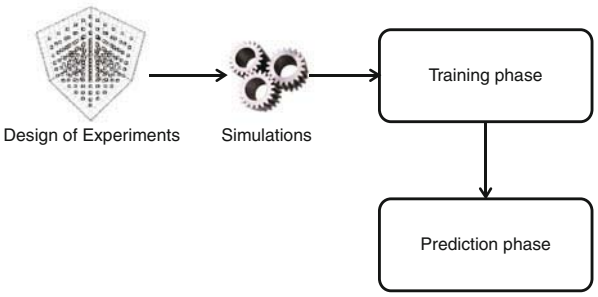

Fig. 4.2 Typical usage of a Response Surface Model

[1] Each architecture configuration is seen as a vector of configuration parameters. We use **bold** font to specify vector values.

[2] A *closure* is a first-class function with free variables that are bound in the lexical environment. Such a function is said to be "closed over" its free variables. (Source: Wikipedia).

The actual value of $\mathbf{w}_0$ is determined during the *training phase* by exploiting a set of observations $y(\mathbf{x})$ known as *training set*. The final value $\mathbf{w}_0$ is such that the estimation error:

$$\varepsilon = \Phi(\mathbf{x}, \mathbf{w}_0) - y(\mathbf{x}) \tag{4.3}$$

is minimized for both the known and future observations $y(\mathbf{x})$.

In more sophisticated cases, the structure of the function $\rho(\mathbf{x})$ is not defined a-priori but it is built by either using *neural*-like processing elements or composing elementary functions. In both cases, the training phase does not involve (only) the selection of parameters $\mathbf{w}_0$ but the navigation through a function-space to identify the optimal $\rho(\mathbf{x})$ that minimizes error ε. Of course, while prolonging the overall training process, these sophisticated model selection algorithms lend to better approximating functions $\rho(\mathbf{x})$.

4.2.1 RSM Categories

Response surface models are *surrogate models* which fit, within reasonable approximation limits, the response curve of a system with respect to the configuration $\mathbf{x}$. Being a *curve fitting* tool, RSMs can be categorized as follows:

- **Interpolation-based RSMs**. This category of curve fitting expressions is built with a constraint such that the curve $\rho(\mathbf{x})$ is equal to $f(\mathbf{x})$ for all the design points $\mathbf{x}$ belonging to the training set T:

$$\rho(\mathbf{x}) \equiv f(\mathbf{x}), \ \forall \mathbf{x} \in T \tag{4.4}$$

  while, for the remaining design points that still belong to the design space, $\mathbf{w}_0$ are calibrated such that the estimated error ε is minimal.
- **Regression-based RSMs**. This category of curve fitting expressions is such that the constraint in Eq. 4.4 does not hold; instead, the coefficients $\mathbf{w}_0$ are chosen such that a general measure of error on known training set T and the future observations is minimized.

Figure 4.3 shows a comparison between the two approaches when fitting a set of five observations of the Energy-Delay Product when varying the system cache size. As can be seen, the interpolation line (a *spline* function) passes through the observations while the regression curve (a second order polynomial) does not. Interpolation zeroes the error on the training observations but it may present an *over-fitting* effect that consists of a decreased prediction accuracy on the unknown observations. On the other hand, regression techniques, albeit with some errors on known observations, may present a greater generalization power. Nevertheless, in this book we will analyze both techniques in the domain of design space exploration.

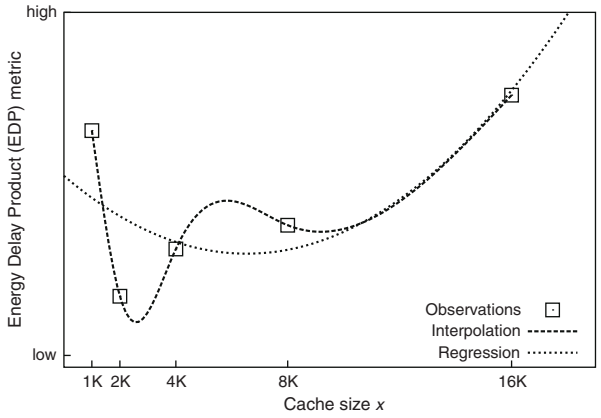

Fig. 4.3 Typical usage of a Response Surface Model

4.2.2 Design of Experiments

The training data used for identifying the parameters **w** is fundamental for creating reasonably accurate prediction models. Of course, the set should be limited given the simulation time needed to gather these data. The literature on RSM calls for a systematic Design of Experiments (DoE) [17] to identify the most suitable configurations with which the RSM can be trained. Design of experiments is a discipline that has had a very broad application across natural and social sciences and encompassed a set of techniques whose main goal is the screening and analysis of the system behavior with a small number of simulations. Each DoE plan differs in terms of the layout of the selected design points in the design space. Several design of experiments have been proposed in the literature so far. Among the most used DoEs for training RSMs we can find:

- *Random DoE*. In this DoE, design space configurations are picked up randomly by following a Probability Density Function (PDF).
- *Full Factorial DoE*. In statistics, a factorial experiment is an experiment whose design consists of two or more parameters, each with discrete possible values or "levels" and whose experimental units take on all possible combinations of these levels across all such parameters. Such an experiment allows studying the effects of each parameter on the response variable, as well as the effects of interactions between parameters on the response variable. The most important full-factorial DoE is called 2-level full factorial, where the only levels considered are the minimum and maximum for each parameter.
- *Central Composite DoE*. A Central Composite Design is an experimental design specifically targeted to the construction of response surfaces of the second order (quadratic) without requiring a three-level factorial.

It is important to note that, while factorial and central composite DoE layouts require a fixed number of points, the Random DoE can have a varying number of design points. In this book, we will leverage Random DoE for validating the proposed RSMs.

4.2.3 Over-Fitting

In statistics, over-fitting occurs when a statistical model captures systematically the random error (or *noise*) together with the underlying analytical relationship. Over-fitting generally occurs when the model complexity is excessive. This happens whenever the model has too many degrees of freedom (i.e., the size of vector **w**), in relation to the amount of data available. An over-fitting model, has poor predictive capabilities, as it can exaggerate minor sweeps in the data.

There are several methods to avoid the over-fitting risk; in the MULTICUBE project we employed techniques such as *model selection* to identify the *simplest* models (in terms of size of **w**) which can guarantee a reasonable error on the training set, and the *early stopping criterion*. The early stopping criterion consists of splitting the training data into two sets: a *training set* and a *validation set*. The samples in the training set are used to train the model, by decreasing both the error on the training data and on the validation data. The training algorithm stops as soon as the error on the validation set starts to increase.

4.3 How to Manage the Embedded Design Space

Problems emerging in the design of embedded computing systems present some characteristic features—such as the fact that all configuration parameters are discrete and possibly categorical—that deserves further discussion.

4.3.1 Discrete and Categorical Variables

SoC design problems are characterized by the fact that all configuration parameters are discrete and possibly categorical:

- Discrete variables quantify data with a finite number of values. There is a clear order relation between different values.
- Categorical (or nominal) variables classify data into categories. These are qualitative variables, in which the order of different values (categories) is totally irrelevant.

On the contrary, traditional RSM techniques usually deal with continuous design spaces. For this reasons, RSM algorithms simply ignore the discrete and/or categorical nature of variables, treating them as continuous ones.

In general this is not a pressing problem as regards discrete variables, given the order relation existing between different values. Even if the trained RSM would be able to predict the model “in the middle”, this unrequested generalization will never be implemented in practice, given the discrete nature of variables in evaluation points.

Concerning categorical variables, the matter is not so simple. As a justification for pragmatically treating these variables as continuous ones, there is the fact that in many applications it is a common practice to treat categorical parameters as simple discrete ones. So if discrete variables are treated as continuous ones, the same should apply for categorical ones. But in this case there is a non-negligible difference: there is no order relation between different variables values. For this reason, caution is requested: the training database should be examined, in order to determine, from case to case, if it is possible to treat categorical parameters as continuous ones. One criterion of decision could be to consider if different subsets corresponding to different categories present analogies (correlations) in response behavior. In case of positive answer, it probably makes sense to train the RSM profitably on the full database, taking advantage of the simple reduction to a continuous domain. If this is not the case, one possible solution could be to treat these different categories as separate sub-problems: different RSMs should be trained separately for each category.

As a final remark, in the design of embedded systems, many discrete input variables are power of two (e.g., memory size): $x = 2^m$. In these cases, it is convenient to perform a variable transformation, taking the exponent m as the actual input parameter, and considering x as a dependent (auxiliary) variable. In this way, the discrete parameter presents two characteristics that are desirable from the point of view of any RSM training algorithm: its values are equispaced and it is well scaled (as regards its range of variation).

4.3.2 Optimal DoE

The following considerations are usually found when dealing with Radial Basis Functions (where they have a straightforward formulation), but they can be generalized for all RSM algorithms. The generic application refers to multivariate scattered training data in a continuous design space.

A generic set of scattered training points $\{\mathbf{x}_i\,,\ i = 1, \ldots, n\}$ is characterized by two quantities: the *fill distance* h, and the *separation distance* q. These quantities, defined in the followings, are shown in Fig. 4.4.

The fill distance h is defined as the radius of the largest inner empty disk:

$$h = \max_{\mathbf{x}\in\Omega} \min_{1\le j\le n} \|\mathbf{x} - \mathbf{x}_j\|, \tag{4.5}$$

where $\Omega \subset \mathbb{R}^d$ is the domain of definition of $f(\mathbf{x})$. In order to achieve better approximation quality of the RSM one should minimize the fill distance: min$h$.

The separation distance q is defined as the minimum distance between two training points:

$$q = \min_{i\neq j} \left\{\|\mathbf{x}_i - \mathbf{x}_j\|\right\}. \tag{4.6}$$

In order to improve the numerical stability of the RSM training algorithm, one should maximize the separation distance: max$q$. Therefore, to improve both approximation

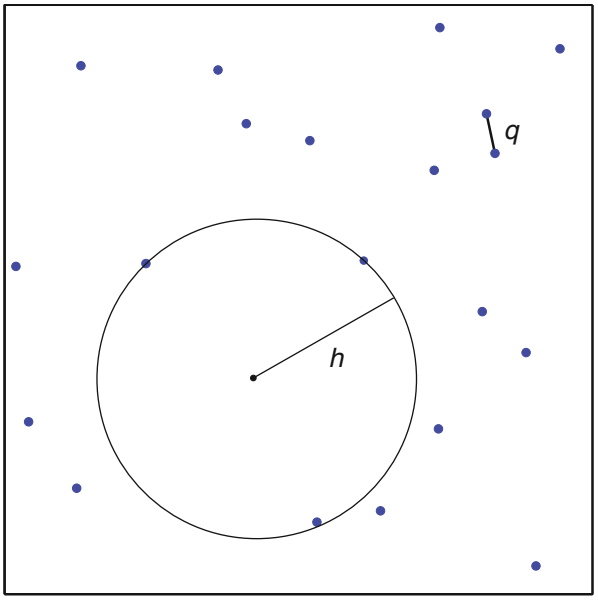

Fig. 4.4 Fill distance h and separation distance q

quality and numerical stability, one should maximize the ratio $\max(q/h)$. Clearly this objective is achieved for a well distributed, almost uniform, set of training points. But in general, for scattered data, one deals with $q \ll h$. In case of uniform distribution of data, there is no way for further improving both objectives: there is a trade-off situation between $\min h$ and $\max q$. This fact explains the choice of Sect. 4.2.2: *Random DoE* (i.e., a uniform Monte Carlo) is a good algorithm for generating points to be used as training database by RSM, since it maximizes the ratio $\max(q/h)$. In general any space filler DoE can be used.

Some enhanced techniques could be implemented in case a even more accurate uniformity is necessary. E.g., *Incremental Space Filler*, for augmenting an existing database in order to fill the space in a uniform way (filling the gaps), or *Uniform Latin Hypercube* that uses Latin Hypercube - an advanced Monte Carlo which maps better the marginal probability distribution of each single variable—for generating random numbers conforming to a uniform distribution.

In case of discrete variables, the separation distance q has a lower bound, strictly related to the variables resolution (number of levels). In general numerical stability is not a pressing problem. On the contrary, in order to achieve approximation quality, it is important to fill as uniformly as possible the gaps in the regular grid formed by the set of all the combinations of admissible discrete variables values. Usually space filling techniques, even though conceived for continuous design space, are able to manage correctly also discrete variables. So in general neither approximation quality is an insurmountable problem, given that the seek for uniformity in a discrete design space is properly considered.

4.3.3 Pre-Processing and Scaling Data

Data used to derive analytical models, also if originated from the same source/modeled architecture, due to different positions in the design space can have different values distribution and orders of magnitudes. Especially when the case is the latter, to create better prediction it is better to *pre-process* and/or *scale* data.

Data transformation is very important because in most of the cases the analytical models used to predict the data work better when data distribution follows some rules. As an example, if we consider an analytical model that uses the standard deviation of the training data to predict unknown data, this standard deviation values can be very high if the data distribution is skewed. In this case, it is highly recommended to first transform the data to approach a better symmetry and then to perform the model training and related prediction.

Box-Cox power transformation. A powerful transformation adopted in the above-mentioned cases is called Box-Cox power transformation [4]. The Box-Cox power transformation is a useful data pre-processing technique used to reduce data variation, make the data more normal distribution-like and improve the correlation between variables. The power transformation is defined as a continuously varying function, with respect to the power parameter λ:

$$y_k^{(\lambda)} = \begin{cases} (y_k^{\lambda} - 1)/\lambda, & \text{if } \lambda \neq 0 \\ \log y_k, & \text{if } \lambda = 0 \end{cases} \tag{4.7}$$

In the validation results of the models that we adopted in this book, we considered a family of transformations as potential candidates $\lambda\{1, 0.5, 0, -1\}$. All the Box-Cox power transformations are only defined with positive values. In case of negative values, a constant value has to be added in order to make them positive. To keep the prediction consistent with the actual objective functions of the target problem, an inverse Box-Cox transformation has been applied on the predicted data.

Some care has to be taken when performing the inverse Box-Cox transformation, since the condition $\lambda\, y_k^{(\lambda)} + 1 > 0$ (when $\lambda \neq 0$) has to be satisfied. Therefore possible unfeasible outcomes has to be taken into account.

Centering and scaling data. Another pre-processing step that is usually applied to the data after the data transformation is the centering and scaling step. The goal of this step is to remove the bias from the input data (mean equal to zero) and to standardize the variance (standard deviation equal to 1). This transformation is also called "Autoscaling". When the *autoscaling* transformation is applied to a set of data, from each value the mean value is removed and it is scaled by the standard deviation: $y_{\text{autoscaled}} = (y_{\text{original}} - \mu_y)/\sigma_y$. Another common alternative step is to normalize data in the unitary interval [0, 1]: $y_{\text{normalized}} = (y_{\text{original}} - \min y)/(\max y - \min y)$. Usually this normalization step is performed on input variables too: in this way data result to be well scaled, being perfectly comparable as regards range extensions. This solution prevent numeric issues that usually arise during RSM training in presence of different scaled variables.

4.4 Algorithm Descriptions

This section describes the fundamental concepts of the RSMs used in the Multicube design flow. All the presented RSMs have been integrated either in the Multicube Explorer open source tool and/or in the modeFRONTIER design tool. The set of RSMs consists of the following models:

- Linear regression (Regression).
- Splines (Regression).
- RBF (Interpolation).
- Neural Networks (Regression).
- Kriging (Interpolation/Regression).
- Evolutionary Design (Regression).

In the following paragraphs we will describe each model in detail, while some results of the application of RSM techniques to some industrial case studies will be reported in Chap. 8.

4.4.1 Linear Regression

Linear regression is a technique for building and tuning an analytic model $\rho(\mathbf{x})$ as a linear combination of $\mathbf{x}$'s parameters in order to minimize the prediction residual ε.

We apply regression by taking into account also the interaction between the parameters and the quadratic behavior with respect to a single parameter. We thus consider the following general model:

$$\Phi(\mathbf{x}, \mathbf{w} = [\mathbf{a}, \mathbf{b}, \mathbf{c}]) = a_0 + \sum_{j=1}^{n} a_j x_j^2 + \sum_{l=1}^{n} \sum_{j=1, j\neq l}^{n} b_{j,k} x_l x_j + \sum_{j=1}^{n} c_j x_j \qquad (4.8)$$

where x_j is the *level* (numerical representation) associated with the j-th parameter of the system configuration, while $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are a decomposition of the RSM family parameters $\mathbf{w}$ and n is the number of parameters of the design space.

Least squares analysis can be used to determine a suitable approximation of $\mathbf{w}$. The least squares technique determines the values of unknown quantities in a statistical model by minimizing the sum of the squared residuals (the difference between the approximated and observed values).

A measure of the *quality of fit* associated with the resulting model is called *coefficient of determination* and defined as follows:

$$R^2 = \frac{\sum_k (y_k - \bar{y})^2}{\sum_k (\rho_k - \bar{y})^2}. \qquad (4.9)$$

where y_k is the k-th observation, $\bar{y}$ is the average of the observations, and ρ_k is the prediction for the y_k observation. R^2 corresponds to the ratio of variability in a data

set that is accounted for by the statistical model. Usually, the higher R^2 the better is the quality of fit (with $0 \leq R^2 \leq 1$).

Although adding parameters to the model can improve the R^2, there is a risk of exceeding the actual information content of the data, leading to arbitrariness in the final model parameters (also called *over-fitting*). This reduces the capability of the model to generalize beyond the fitting data, while giving very good results on training-data. In particular, this phenomenon occurs when the model is excessively complex in relation to the amount of data available. A model which has been over-fit, generally has poor predictive performance, as it can exaggerate minor fluctuations in the training data.

To this purpose, we introduce an 'adjusted' definition of the R^2. This terms adjusts for the number of explanatory terms in a model; it increases only if the terms of the model improve it more than expected by chance and will always be less than or equal to R^2; it is defined as:

$$1 - (1 - R^2)\frac{N-1}{N-p} \tag{4.10}$$

where p is the total number of terms in the linear model (i.e., the set of coefficients a, b, c), while N is sample set size. Adjusted R^2 is particularly useful in the *model selection* stage of model building.

Linear regression model selection. In order to understand the optimal number of terms of the linear model and the corresponding model order, we analyzed the behavior of the RSM cross-validation error and adjusted R^2 by varying the number of random training samples (derived from the simulations of the target applications). Equation 4.10 represents an improved measure of the overall *quality of fit* of the linear regression: it is inversely proportional to the model's *degrees of freedom* (i.e., $N - p$) which, in turn, depend on the order of the chosen polynomial $\rho(\mathbf{x})$. As a matter of fact, higher degrees of freedom increase the chance of reduced variance of the model coefficients thus improving model stability while avoiding over-fitting [9, 10]. In this book, our heuristic model selection tries to maximize the number of degrees of freedom and, at the same time, to minimize (in the order of 200) the number of simulations needed to build the model. Thus, as a "rule of thumb", we set a maximum number of terms (to increase the chance of good quality of fit) and eventually, we limit the set of considered models to the following configurations:

1. First order model, without any interaction between parameters.
2. First order model, with interaction between parameters.
3. Second order model, without any interaction between parameters.

4.4.2 Radial Basis Functions

Radial basis functions (RBF) represent a widely used interpolation/approximation model [13]. The interpolation function is built on a set of training configurations $\mathbf{x}_k$

as follows:

$$\Phi(\mathbf{x}, \mathbf{w}) = \sum_{k=1}^{N} w_k \, \gamma(\|\mathbf{x} - \mathbf{x}_k\|) \tag{4.11}$$

where γ is a scalar *distance* function, w_k are the weights of the RBF and N is the number of samples in the training set. In this paper, we consider the following definitions for γ

$$\gamma(z) = \begin{cases} z & \text{linear} \\ z^2 \log z & \text{thin plate spline} \\ (1+z^2)^{1/2} & \text{multiquadric} \\ (1+z^2)^{-1/2} & \text{inverse multiquadric} \\ e^{-z^2} & \text{gaussian} \end{cases} \tag{4.12}$$

The weights w_k are the solution of a matrix equation which is determined by the training set of configurations $\mathbf{x}_k$ and the associated observations y_k:

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ A_{21} & A_{22} & \dots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \dots & A_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{4.13}$$

where:

$$A_{jk} = \gamma(\|\mathbf{x}_j - \mathbf{x}_k\|), \qquad j, k = 1, 2, \dots, N, \tag{4.14}$$

4.4.3 Splines

Spline-based regression has been recently proposed by Lee and Brooks [7] as a powerful method for the prediction of power consumption and performance metrics. A spline RSM is composed of a number of piecewise polynomials σ_L associated with each of the parameters x_j of the architecture:

$$\Phi(\mathbf{x}, \mathbf{w} = [\mathbf{a}, \mathbf{b}, \mathbf{k}]) = a_0 + \sum_{j=1}^{N} a_j \sigma_L(x_j, \mathbf{b}, \mathbf{k}) \tag{4.15}$$

The piece-wise polynomial σ_L is divided into L intervals defining multiple different continuous polynomials with endpoints called *knots*. The number of knots L can vary depending on the amount of available data for fitting the function and the number of levels associated with parameter x_j, but more knots generally lead to better fits. In fact, relatively simple linear splines may be inadequate for complex, highly

non-linear relationships. In the context of the project, we exploited *restricted* cubic splines (RCS) where the expressions associated with σ are third order polynomials. Restricted cubic splines are such that the first and second order derivative at the knots is the same for *adjacent* polynomials while they present a linear behavior on the *tails*.

As an example, a closed form expression of a RCS with three knots ($L = 3$) is the following:

$$\sigma_3(x_j, \mathbf{b}, \mathbf{k}) = b_{0,j} + b_{1,j}x_j + b_{2,j}x_j^2 + b_{3,j}x_j^3 + b_{4,j}(x_j - k_1)^3 + b_{5,j}(x_j - k_2)^3 + b_{6,j}(x_j - k_3)^3$$

where k_j are the knot points in the function domain.

To determine the number of knots, we started by considering that five knots or fewer are generally sufficient for restricted cubic splines. While fewer knots may be required for small data sets, with a large number of knots increases the risk of over-fitting the data. In particular, we adopted the following policies for the selection of the number of knots depending on the number of levels of the j-th parameter x_j:

- If the number of levels is greater than 5, $L = 5$.
- If the number of levels is smaller than 3, $L = 0$ (the spline is a linear function of the parameter).
- Otherwise, the number of knots is equal to the number of levels.

4.4.4 Neural Networks

For function approximation purposes, Feed-forward Neural Networks (also known as Multilayer Perceptrons) are a very efficient and powerful tool. Feed-forward networks are organized in successive layers of neurons. The data flow is unidirectional: the data pass from the first *input* layer to the last *output* layer, and are elaborated incrementally by the intermediate *hidden* layers.

The model of each single neuron is straightforward:

$$u = \sum_{i=1}^{n} w_i\, x_i + b \qquad y = f(u) = f\left(\sum_{i=1}^{n} w_i\, x_i + b\right) \tag{4.16}$$

The *net input* u is a linear combination of the input values x_i: the *weights* w_i and the *bias* b (or *firing threshold*) represent the free parameters of the model. The net input is transformed by means of a *transfer function* f (or *activation function*)—that in general is non linear—giving the neuron's output y. The behavior and the complexity of the network are defined by the way its neurons are connected: so in general the model $\Phi(\mathbf{x})$ is a complex recursive function that is usually regarded as a black-box, with no given explicit analytical expression.

It has been shown in [3] that Neural Networks (NN) with one single non-linear hidden layer and a linear output layer are sufficient for representing any arbitrary (but

sufficiently regular) function. The necessary condition is a sufficiently high number of neurons in the hidden layer.

NNs learn by example: given a training data set of inputs and a set of targets as outputs, the network's weights and biases are adjusted in order to minimize errors in its predictions on the training data. The best known training algorithm is back-propagation: a very effective approach consists of the Levenberg–Marquardt algorithm, as outlined in [2].

The initialization of the weights of NN for training is usually set at random small values. However the Nguyen-Widrow initialization technique greatly reduces the training time (see [11]).

In this book, we use classical Feed-forward NN with one hidden layer: this layer has a sigmoid transfer function, while the output layer has a linear transfer function. Back-propagation training algorithm is used, in particular the Levenberg–Marquardt algorithm is implemented, with a fixed number of iterations. The Nguyen-Widrow initialization technique is also implemented. The training data are internally normalized, both in input and in output, in order to exploit the natural range of definition of transfer functions.

In this context, there is only one free parameter to be set: the number of neurons in the hidden layer. Automatic network sizing has been implemented, using the value proposed in [16].

4.4.5 Kriging

Kriging is a very popular regression methodology based on Gaussian Processes [14]. This RSM algorithm can be interpolating or approximating, according if a noise parameter is set to zero or to nonzero values.

Kriging is a Bayesian methodology (named after professor Daniel Krige), used as a main tool for making previsions employed in geostatistics, e.g., for soil permeability, oil and other minerals extraction, etc. (originally it was developed for predicting gold concentration at extraction sites). The formalization and dissemination of this methodology is due to Professor Georges Matheron [8], who indicated the Krige's regression technique as *krigeage*.

The Kriging estimator is a linear estimator, i.e., the estimated value is expressed as a linear combination of the training values, in other words:

$$\Phi(\mathbf{x}) = \sum_{k=1}^{N} \lambda_k(\mathbf{x})\ y_k \tag{4.17}$$

where the weights $\lambda_1, \ldots, \lambda_N$, are obviously point-dependent.

Kriging can also produce an estimate of the error, i.e., a prediction value and an expected deviation from the prediction.

The Kriging behavior (smoothness of the model) is controlled by a covariance function, called the variogram function, which rules how varies the correlation between the response values in function of the distance between different points. A function can be rougher or smoother, can exhibit large or small ranges of variation, can be affected by a certain amount of noise, and all these features can be resumed in a variogram model.

More precisely, the covariance function $\mathrm{Cov}(\mathbf{x}_1, \mathbf{x}_2)$ only depends on the distance between two points:

$$\mathrm{Cov}(\mathbf{x}_1, \mathbf{x}_2) = \sigma - \gamma(\|\mathbf{x}_1 - \mathbf{x}_2\|) \tag{4.18}$$

where $\gamma(h)$ is the variogram function, and σ is the sill, i.e., the asymptotic value of γ.

There are several variogram types that can be employed: Gaussian, Exponential, Matèrn, Rational Quadratic. Usually Gaussian is the first (default) choice: the generated metamodel is infinitely differentiable.

Each variogram function is characterized by three different parameters: range, sill, and noise.

The variogram range of the covariance function corresponds to a characteristic scale of the problem. If the distance between two points is larger than the range the corresponding outcomes should not influence each other (completely uncorrelated). Range is inversely related to the number of oscillations of the function. Small ranges mean sudden variations, while large ranges mean very regular trends, with very few oscillations.

The variogram sill correspond to the overall variability of the function. The gap between the values of very distant points should be of the same scale of magnitude of the sill.

Variogram noise can also be tuned to fit the expected standard error in the observations. Larger amount of noise will result in smoother responses, while zero noise means exact interpolation.

Parameters determination can be based on previous knowledge on similar problems, or may be guessed by following two automatic fitting strategies: maximizing the Likelihood of the model given the training dataset or maximizing the Leave-One-Out (LOO) Predictive Probability. The Likelihood of the variogram is the probability that a statistical distribution associated to the variogram parameters could generate the given dataset. Likelihood is larger for good fitting models, but penalizes unnecessarily complex models. Maximum likelihood models are the smoothest models with best agreement to the dataset. The Leave-One-Out Predictive Probability gives a measure of the goodness of the model also removing one point at a time in the dataset and estimating the value at the removed site on the basis of the remaining designs. Models predicting the smallest errors at "difficult" points are rewarded with high LOO Predictive Probability. Maximum LOO Predictive Probability privilege good fitting models which do not loose prediction performance by removing some design. However, computation of the LOO Predictive Probability can be quite intensive, more than the computation of the Likelihood.

4.4.6 Evolutionary Design

Evolutionary Design (ED) [1] is an effective implementation of Genetic Programming (GP) methodology [6] for symbolic regression.

In general the goal of the regression task is to discover the relationship between a set of inputs, or independent variables **x** given an observable output, or dependent variable y. In standard regression techniques the model functional form $\Phi(\mathbf{x}, \mathbf{w})$ is known beforehand. The only unknown values are some coefficients **w**, i.e., the free parameters of the model.

ED uses low-level primitive functions. These functions can be combined to specify the full function. Given a set of functions, the overall functional form induced by genetic programming can take a variety of forms. The primitive functions are usually standard arithmetical functions such as addition, subtraction, multiplication and division but could also include trigonometric and transcendental functions. Any legal combination of functions and variables can be obtained.

Each individual in GP corresponds to a given mathematical expression, and it is represented by means of a parse tree. For example, the expression

$$f(x, y) = 2xy - (x + 1)^2 \tag{4.19}$$

is represented by the parse tree depicted in Fig. 4.5.

Symbolic regression is then a composition of input variables, coefficients and primitive functions such that the error of the function with respect to the expected output is minimized. Shape and size of the solution is not specified before the regression. Number of coefficients and their values is another issue that is determined in the search process itself. By the use of such primitive functions, genetic programming is in principle capable of expressing any functional form that use the primitive functions provided by the user. Unlike the traditional methods, the Evolutionary Design process automatically optimizes both the functional form and the coefficient values.

ED is capable of providing answers in the symbolic language of mathematics, while others methods only provide answers in the form of sets of numbers, weights, valid in the context of a model defined beforehand. So, after the training, the explicit formula of the regression model is promptly available.

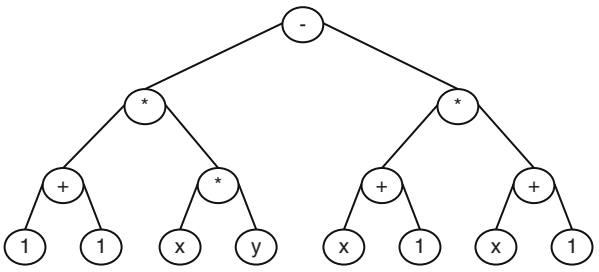

Fig. 4.5 Parse tree representing the mathematical expression $f(x, y) = 2xy - (x + 1)^2$

4.5 General Validation Flow of RSMs

To verify the quality of the predictions generated by a RSM we will compare the predicted metric values with the actually observed ones to obtain then a quality index for the model. To do so, we introduce the *average normalized error*:

$$\eta = \frac{\sum_{i \in \Theta} \left(\frac{\sum_{\mathbf{x} \in \Delta} \left| \frac{\rho(\mathbf{x})_i - y(\mathbf{x})_i}{y(\mathbf{x})_i} \right|}{|\Delta|} \right)}{|\Theta|} \tag{4.20}$$

where:

- Θ is the set of system metrics,
- Δ is the set containing all the design space points,
- $y(\mathbf{x})_i$ is the actual value of the metric $i \in \Theta$ for the design space point $\mathbf{x} \in \Delta$, and
- $\rho(\mathbf{x})_i$ is the estimated value of metric $i \in \Theta$ for the design space point $\mathbf{x} \in \Delta$.

An appropriate RSM should present a behavior where the greater is the training set size, the better is the average normalized error. As a matter of fact, we will verify that the error decreases growing the training set size, and this is done re-running the model construction with a greater training sets and calculating the average normalized error.

If random processes are involved during the selection of the design space points used as training set, the prediction results can present variability (especially for training sets with small size with respect to the design space). To characterize this effect, the validation methodology will be repeated to identify a set of stable statistical properties.

4.6 Conclusions

In this chapter, we have introduced a set of statistical and machine learning techniques that can be used to improve the performance and/or accuracy of design space exploration techniques by predicting system level metrics without resorting to long simulations. The presented techniques (called Response Surface Models) leverage a set of closed-form analytical expressions to infer an approximation of the actual system response by either exploiting regression or interpolation modeling.

References

1. Fillon, C.: New strategies for efficient and practical genetic programming. Ph.D. thesis, Università degli Studi di Trieste (2008)

2. Hagan, M.T., Menhaj, M.B.: Training feedforward networks with the marquardt algorithm. IEEE Trans. on Neural Networks **5**(6) (1994)
3. Irie, B., Miyake, S.: Capabilities of three-layered perceptrons. In: Proceedings of the IEEE International Conference on Neural Networks, pp. I–641 (1998)
4. Joseph, P., Vaswani, K., Thazhuthaveetil, M.: Construction and use of linear regression models for processor performance analysis. High-Performance Computer Architecture, 2006. The Twelfth International Symposium on pp. 99– 108 (2006)
5. Keutzer, K., Malik, S., Newton, A.R., Rabaey, J., Sangiovanni-Vincentelli, A.: System level design: Orthogonolization of concerns and platform-based design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **19**(12), 1523–1543 (2000)
6. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
7. Lee, B.C., Brooks, D.M.: Accurate and efficient regression modeling for microarchitectural performance and power prediction. Proceedings of the 12th international conference on Architectural support for programming languages and operating systems **40**(5), 185–194 (2006). DOI http://doi.acm.org/10.1145/1168917.1168881
8. Matheron, G.: Les variables régionalisées et leur estimation: une application de la théorie des fonctions aléatoires aux sciences de la nature. Masson, Paris (1965)
9. Montgomery, D.C., Runger, G.C.: Applied Statistics and Probability for Engineers. Wiley (2006)
10. Montgomery, D.C.: Design and Analysis of Experiments. John Wiley and Sons (2005)
11. Nguyen, D., Widrow, B.: Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In: Proceedings of IJCNN, vol. 3, pp. 21–96 (1990)
12. Perelman, E., Hamerly, G., Biesbrouck, M.V., Sherwood, T., Calder, B.: Using simpoint for accurate and efficient simulation. In: ACM SIGMETRICS Performance Evaluation Review, pp. 318–319 (2003)
13. Powell, M.J.D.: The theory of radial basis functions. In: Advances in Numerical Analysis II: Wavelets, Subdivision, and Radial Basis Functions, W. Light (ed, pp. 105–210. University Press (1992)
14. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)
15. Renau, J., Fraguela, B., Tuck, J., Liu, W., Prvulovic, M., Ceze, L., Sarangi, S., Sack, P., Strauss, K., Montesinos, P.: SESC simulator (2005). Http://sesc.sourceforge.net
16. Rigoni, E., Lovison, A.: Automatic sizing of neural networks for function approximation. In: SMC2007, pp. 2005–2010 (2007)
17. Santner, T.J., B., W., W., N.: The Design and Analysis of Computer Experiments. Springer-Verlag (2003)
18. Woo, S., Ohara, M., Torrie, E., Singh, J., Gupta, A.: Splash-2 programs: characterization and methodological considerations. Proceedings of the 22th International Symposium on Computer Architecture p. 2436 (1995)