Statistical Emulation Applied to a Very Large Data Set Generated by an Agent-based Model

Wim De Mulder
and Geert Molenberghs
and Geert Verbeke

Leuven Biostatistics and
Statistical Bioinformatics Centre
Leuven, Belgium
Email: wim.demulder@cs.kuleuven.be

Bernhard Rengs
and Thomas Fent

Wittgenstein Centre (IIASA, VID/ÖAW, WU)
VID/ÖAW
Vienna, Austria

***Abstract*—This paper presents a method to apply statistical emulation on very large data sets, making use of cluster analysis. It is shown how integrating cluster analysis with the interpolation method called inverse distance weighting, naturally generalizes the basic emulation framework where a single Gaussian distribution is used, to a framework where a mixture of Gaussians is employed. Our results indicate that it might be advantageous to choose a different number of Gaussians in the mixture depending on the goal of the subsequent analysis. For example, is a certain estimation needed or is a confidence interval required? Some interesting future research questions are discussed. Our work is truly interdisciplinary, covering the fields of statistical emulation, demography, agent-based models, genetic algorithms and cluster analysis.**

***Keywords*–Statistical emulation; Agent-based models; Cluster analysis; Genetic algorithms.**

I. INTRODUCTION

Statistical emulation, more extensively discussed in Section III-A, is a fairly recent methodology to approximate deterministic computer models. The use of emulation has several advantages, such as the fact that the emulator (the approximation to the computer model) generates its output for a given input almost instantaneously [1], an especially desired property if the considered computer model is computationally expensive. Furthermore, emulation offers a framework that is well suited to perform a sensitivity or uncertainty analysis of the computer model under consideration, see e.g. [2] and [3].

However, the standard emulation framework is not suitable for very large data sets or when stationarity of the data cannot be assumed, as discussed in [4]. The authors of the cited work applied a treed partition model, meaning that they divide up the input space by making binary splits on the value of a single variable so that partition boundaries are parallel to coordinate axes. Partitioning is hierarchical, in the sense that each new partition is a sub-partition of a previous one.

We elaborate on the idea of constructing different emulators for different parts of the input space, but instead of using hierarchical partitioning we apply a non-hierarchical clustering algorithm, namely k-means (see Section III-B). The work that is discussed here is mainly of an exploratory nature, where we experiment with six different implementations. First, we compare results in terms of two alternative definitions of the distance of a non training data point to a given cluster, where we consider the possibilities of defining this distance as the minimum over all distances from the given point to the training data points belonging to this cluster versus adopting the commonly used distance concept in cluster analysis where the distance from a point to a cluster is taken as the distance to the center of this cluster. Secondly, results are compared in terms of the number of clusters, or equivalently the number of emulators, that are used in obtaining an output for a non training data point. That is, we consider the possibility of assigning a non training data point to more than one cluster, quite similar to fuzzy clustering [5].

The paper is organized as follows. In Section II we describe our agent-based model for which we have constructed an emulator. Section III discusses the three methodologies that play a role in this construction: statistical emulation, cluster analysis and genetic algorithms. Several choices on implementation details have to be made. This results in six different implementation methods, described in Section IV. Section V gives a description of the measures that are used to evaluate the results of these implementation methods. Results are then analyzed in Section VI.

II. APPLICATION TO DATA GENERATED BY AN AGENT-BASED MODEL

A. Short introduction to agent-based models

An agent-based model (ABM) is a computer model that simulates the behavior and interactions of autonomous agents [6]. A key feature is that *population* level phenomena are studied by modeling the interactions of *individuals* that are members of the population of interest. The macro level behavior that emerges from such individual, i.e. lower level, interactions is typically complex, more often than not to the extent that the behavior on the macro scale cannot be deduced by knowledge of the system's parts alone. Thus, for example, an ABM is very well suited to study traffic jams, since a traffic jam results from the behavior of and interactions between individual vehicle drivers, but it may be moving in the direction opposite that of the cars that cause it [7].

B. Description of our agent-based model

We developed an agent-based model to analyze the effectiveness of family policies under different assumptions regarding the social structure of a society. The agents represent the female partner in a household. They are heterogeneous with respect to age, household budget, parity, and intended fertility. A network of mutual links connects the agents to a small subset of the population to exchange fertility preferences. The agents are endowed with a certain amount of time and money which they allocate to satisfy their own and their children's needs. We consider two components of family policies: 1. the policy maker provides a fixed amount of money or monetary equivalent per child to each household and 2. a monetary or nonmonetary benefit proportional to the household income is received by the household. The output on the aggregate level that is produced by the ABM consists of the cohort fertility, the intended fertility and the fertility gap. The inputs that are provided to the ABM include the level of fixed and income dependent family allowances, represented by the parameters b^f and b^v, and parameters that determine the social structure of a society, such as a measure for the agents' level of homophily α, and the strength of positive and negative social influence, which are denoted as pr_3 and pr_4 resp. Analysis of the model outcomes indicates that both fixed and income dependent child supports have a positive and significant impact on fertility, but that the effectiveness of such family policies strongly depends on the social structure of the considered country. This has been described extensively in previous work [8].

C. Data set

The input variables of our ABM are given equidistant values from the input domain and the ABM is applied to generate the corresponding outputs. A total of 741,312 data points in input space were considered in [8]. However, since it is not our purpose here to use emulation to enhance the analysis of the ABM outcomes, but rather to explore the application and behavior of emulation when the training data set has been partitioned into clusters, we limit the size of the data set as follows. First, on the output side we consider only cohort fertility. Secondly, on the input side we consider only those variables that were found to have the largest influence on the outcomes, namely b^f, b^v, α, pr_3 and p_4. This reduces the data set to 11,232 data points. From this data set we selected randomly 500 points to be used as test points, leaving 10,732 data points as training data set. This is still a very large data set which suffices for our purposes. In this way, we limit computation time to some extent without affecting our objectives.

III. Methodology

A. Statistical emulation

Statistical emulation provides an approximation to a deterministic computer model that maps real vectors to real vectors. Such a computer model can be seen as a function ν that generates output $y = \nu(\mathbf{x})$ for any given vector $\mathbf{x}$ that belongs to the input domain. In our case study, y is a real number.

The approximation to ν, i.e. the emulator, is determined via the Bayesian formalism as follows. In the first step, it is assumed that nothing is known about ν. The value $\nu(\mathbf{x})$ for any $\mathbf{x}$ is then modeled as a Gaussian distribution with mean $m(\mathbf{x}) = \sum_{i=1}^{q} \beta_i \, h_i(\mathbf{x})$, where β_i are unknown coefficients and where h_i represent regression functions chosen by the user. As is custom in statistical emulation, we choose h_i linear. The covariance between $\nu(\mathbf{x})$ and $\nu(\mathbf{x}')$ is modeled as $\mathrm{Cov}\Big(\nu(\mathbf{x}), \nu(\mathbf{x}') \,|\, \sigma^2\Big) = \sigma^2 \, c(\mathbf{x}, \mathbf{x}')$, where σ^2 denotes a constant variance parameter and where $c(\mathbf{x}, \mathbf{x}')$ denotes a function that models the correlation between $\nu(\mathbf{x})$ and $\nu(\mathbf{x}')$. We adopt the most common choice for c:

$$c(\mathbf{x}, \mathbf{x}') \quad = \quad \exp\Big[-\sum_i \Big((x_i - x_i')/\delta_i\Big)^2\Big]$$

with x_i and x_i' the ith component of $\mathbf{x}$ and $\mathbf{x}'$ resp., and where the δ_i represent parameters, commonly called the correlation lengths. One way to optimize δ_i is via maximum likelihood [9]. In the second step, training data $(\mathbf{x}_1, \nu(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, \nu(\mathbf{x}_n))$ are obtained and, via appropriate applications of Bayes' rule, used to determine the parameters involved and to update the Gaussian distributions to Student's t-distributions. However, it is custom to approximate each t-distribution with a Gaussian distribution. The mean $m^\star(\mathbf{x})$ and variance $v^\star(\mathbf{x})$ of this distribution, given an input point $\mathbf{x}$, are produced by the trained emulator. The value $m^\star(\mathbf{x})$ is then considered as the approximation for $\nu(\mathbf{x})$ and we determine a 95% confidence interval $CI(\mathbf{x})$ as $[m^\star(\mathbf{x}) - 2\sqrt{v^\star(\mathbf{x})}, m^\star(\mathbf{x}) + 2\sqrt{v^\star(\mathbf{x})}]$. We refer to [10] for a more detailed account on emulation. We adopt the widespread practice of considering the distributions that are produced by the trained emulator as Gaussian. This is legitimate, given the sufficiently large number of training data points we use for training.

The correlations between each pair of training data points are encapsulated in a correlation matrix A. The inverse of A is used in the analytical expression for the optimal values of the β_i and of σ^2. However, when the size of the training data is large it is typically found that A is computationally non-invertible. A simple and widely applied solution is then to replace A by the matrix $A + \alpha I$, where I is the identity matrix and where $\alpha > 0$ is a parameter called the nugget [11].

B. Cluster analysis

Cluster analysis is the unsupervised partitioning of a data set into groups, also called clusters, such that data elements that are member of the same group have a higher similarity than data elements that are member of different groups. Similarity is typically expressed in terms of a user-defined distance measure, such as the commonly used Euclidean distance which we employ here. The best known clustering algorithm is k-means, an iterative algorithm to compute the centers of a predefined number of clusters such that each given data point is assigned to the cluster with the nearest center [5]. We use k-means to subdivide the very large training data set into clusters and then construct an emulator for each of these clusters. Indeed, it is practically impossible to apply emulation directly to the full training data set, as we found that it is even impossible to store in memory the correlation matrix, which contains a number of elements that is of quadratic order in the training data size. Even on much more powerful computers, it is not advisable to train one emulator for the complete training data set, due to computational instabilities in inverting the correlation matrix (see Section III-A). Furthermore, constructing different emulators for different subdomains is a more flexible approach compared to the standard emulation framework, since it allows the parameters β_i and σ^2 to vary from subdomain to subdomain.

C. Genetic algorithms

Determination of the parameters β_i and σ^2 is simple, as analytical expressions exist for their optimal values (see, e.g., [10]). However, such expressions do not exist for the correlation lengths. These are typically obtained by applying the maximum likelihood principe, as described in [9]. The optimization of their density function is a nontrivial task here as this function is a $\mathbb{R}^5 \rightarrow \mathbb{R}$ mapping (there are 5 correlation lengths, one for each of the input variables b^f, b^v, α, pr_3 and pr_4), which may have many local optima. We use genetic algorithms for this optimization task.

Genetic algorithms are a type of heuristic optimization method that mimics some aspects of the process of natural selection, in that a population of candidate solutions to an optimization problem is evolved toward better solutions. Many possible implementations have been developed. The variant we use is essentially the same as described in [12], Section 2.3, and may be summarized as follows:

1) A fitness function is chosen by the user. This fitness function measures the quality of a candidate solution. We simply choose the fitness of a vector of correlation lengths as the value of the density function in this vector. The larger the fitness value, the higher the quality of the candidate solution.
2) An initial population of candidate solutions is chosen, randomly selected from the input space. The population size is chosen as 100.
3) Select the worst half of the population, i.e. the 50 candidate solutions with the lowest fitness value. This population is transformed in the following way. Select repeatedly pairs of different candidate solutions from this subpopulation. Each such pair, say $\mathbf{v}$ and $\mathbf{w}$, representing vectors of correlation lengths, is changed into a new pair as follows. Generate a random, uniformly distributed $0 < \beta < 1$ and a random number $l \in \{1, \dots, 5\}$ where each of numbers 1, …, 5 have equal probability to be selected. Then change the value of the lth component v_l and w_l of $\mathbf{v}$ and $\mathbf{w}$ resp. into new values v_l' and w_l' via the following rule:

$$\begin{aligned} v_l' &= (1-\beta)v_l + \beta w_l \\ w_l' &= (1-\beta)w_l + \beta v_l \end{aligned}$$

   This process is called crossover and amounts to exploring the search space. The crossover process is applied to 25 such pairs.
4) Combine the transformed worst half of the population with the unchanged best half into a new population.
5) Each of the candidate solutions in the new population possibly undergoes a mutation. That is, each candidate solution is considered in turn and with a certain fixed probability, called the mutation rate, its value (i.e. each of the five components) is changed to a random new value. The mutation rate is typically chosen small; we choose it as 0.05. The purpose of mutation is to maintain diversity in the population and thus to avoid that the algorithm converges to a purely local optimum.
6) Go back to the third step unless a predefined maximum number of iterations has been reached. We choose this maximum number as 100. If the maximum number of iterations is reached, the candidate solution that has the largest fitness value of all candidate solutions, over all 100 iterations, is selected as the optimal vector of correlation lengths.

Key advantages of genetic algorithms are that they only employ function evaluations (and thus not, e.g., information about derivatives as is required by many other optimization methods, such as, for example, gradient descent) and that they are well suited to avoid getting stuck in local optima. Both characteristics make them particularly useful to optimize the density function of the correlation lengths.

IV. Description of the six implementation methods

Given a set of emulators, each trained on a different part of the input domain, it has to be decided how this set is used to determine an approximate value $\hat{\nu}(\mathbf{x})$ to $\nu(\mathbf{x})$ given a non training data input point $\mathbf{x}$. First, we have to choose whether we use just one emulator or we combine the outputs of several emulators into one suitable approximate value $\hat{\nu}(\mathbf{x})$. Secondly, in order to choose which specific emulator or emulators to employ we have to make a choice on the definition of the distance from $\mathbf{x}$ to any given cluster, since it is intuitive to use the emulator(s) that correspond to the cluster(s) to which $\mathbf{x}$ is closest.

A. One emulator versus several emulators

Given a single emulator that is to be used to determine an approximate output value in $\mathbf{x}$, we simply take the posterior mean $m^\star(\mathbf{x})$ of this emulator as $\hat{\nu}(\mathbf{x})$ and we use the corresponding confidence interval $CI(\mathbf{x})$ produced by this emulator to model the uncertainty in using $\hat{\nu}(\mathbf{x}) = m^\star(\mathbf{x})$ as the true output.

Alternatively, we could use more than one emulator to determine $\hat{\nu}(\mathbf{x})$. This idea is inspired by the use of ensemble methods in machine learning, a technique by which a new data point is classified by taking a weighted vote of the predictions of several classifiers. Such ensembles often perform better than any single classifier [13].
Given an input point $\mathbf{x}$, the question is then how to determine $\hat{\nu}(\mathbf{x})$ on the one hand and how to obtain an appropriate confidence interval on the other hand. To be more concrete, consider the k emulators that correspond to the k closest clusters, where distance is measured according to either the minimum or the average distance, as explained in the next section. The distances to these clusters are denoted as $d_1, \dots, d_k$, and the posterior means of the corresponding emulators as $m_1^\star(\mathbf{x}), \dots, m_k^\star(\mathbf{x})$. We apply an intuitive method to obtain $\hat{\nu}(\mathbf{x})$, called inverse distance weighting, which approximates an unknown value in a given input point by a weighted average of known values in training data input points where the weights are inversely related to the distance from the considered input point to each of the training data points [14]. Using this technique, we determine $\hat{\nu}(\mathbf{x})$ as

$$\hat{\nu}(\mathbf{x}) = \sum_{i=1}^{m} \frac{\frac{1}{d_i} m_i^\star(\mathbf{x})}{\sum_{j=1}^{m} \frac{1}{d_j}} \tag{1}$$

To determine an appropriate confidence interval, we notice that (1) in fact expresses the mean of a mixture of Gaussian distributions. Thus it is intuitive to obtain the confidence interval from the cumulative distribution function of this mixture. This can be done using the regula falsi method [15].

We have thus shown that combining an idea from machine learning, namely ensemble methods, with an idea from the domain of interpolation, namely inverse distance weighting, leads to a natural extension of the basic emulation framework where instead of using a single Gaussian distribution to model an output a mixture of Gaussian distributions is employed.

B. Center distance versus minimum distance

The k emulators that are used to determine $\hat{\nu}(\mathbf{x})$ and to which we referred in the previous section, are chosen as the emulators that are trained with the k clusters of the training data set that are closest to $\mathbf{x}$. This requires to define the distance from an input point $\mathbf{x}$ to a cluster C, denoted as $d(\mathbf{x}, C)$. We consider two possibilities, to which we refer as the center distance and the minimum distance resp.

First, $d(\mathbf{x}, C)$ can be defined as the distance from $\mathbf{x}$ to the center of C. Intuitively speaking, the center of a cluster is the data point (not necessarily belonging to the training data set) that is most representative of this cluster. The centers are given as part of the output of the k-means algorithm. This definition of distance from a point to a cluster is commonly used in cluster analysis.

Secondly, $d(\mathbf{x}, C)$ can be defined as the minimum distance from $\mathbf{x}$ to each of the training data points belonging to C. Such a definition conforms the definition of distance from a point to a set in a metric space.

Both definitions rely on the distance between two points, which we take here as the commonly used Euclidean distance.

C. The resulting implementation methods

We obtained results, described below, for $k = 1, 2, 3$. Combining this with the two possibilities for the definition of distance, this gives six alternative methods to determine $\hat{\nu}(\mathbf{x})$ in each of the test points $\mathbf{x}$. We refer to the method using $k \in \{1, 2, 3\}$ emulators and using the minimum distance as E_{k_min} and to the method using $k \in \{1, 2, 3\}$ emulators and using the center distance as E_{k_cen}. For example, the method where $\hat{\nu}(\mathbf{x})$ is determined using the two emulators that correspond to the two closest clusters, where distance is measured according to the center distance, is denoted as E_{2_cen}.

V. Description of the evaluation measures

Each of the implementation methods described in the previous section determine a value $\hat{\nu}(\mathbf{x})$ as approximation for $\nu(\mathbf{x})$ and a confidence interval $[l(\mathbf{x}), u(\mathbf{x})]$ around $\hat{\nu}(\mathbf{x})$. The 500 test points are used to evaluate each of the implementation methods. We use two measures to perform this evaluation, one to evaluate the produced approximation to $\nu(\mathbf{x})$, called the average relative difference, and one to evaluate the produced confidence interval, called the average interval score.

A. Average relative difference

Given a test input point $\mathbf{x}$ with corresponding true output $\nu(\mathbf{x})$ and an approximation $\hat{\nu}(\mathbf{x})$ produced by one of the six methods, we can evaluate the quality of the approximation as the relative difference between $\nu(\mathbf{x})$ and $\hat{\nu}(\mathbf{x})$ as follows:

$$RD(\mathbf{x}) \quad = \quad \left| \frac{\hat{\nu}(\mathbf{x}) - \nu(\mathbf{x})}{1/2(\hat{\nu}(\mathbf{x}) + \nu(\mathbf{x}))} \right|$$

The average relative difference for a particular method, denoted ARD, is then the average of $RD(\mathbf{x})$ over all test points $\mathbf{x}$.

B. Average interval score

The quality of a confidence interval $[l(\mathbf{x}), u(\mathbf{x})]$ around $\hat{\nu}(\mathbf{x})$ is evaluated using the interval score described in [16]. Given an $(1-\alpha)\%$ confidence interval $[l(\mathbf{x}), u(\mathbf{x})]$, with $\alpha = 0.05$ chosen in this paper, the interval score is defined as

$$\begin{aligned} IS(\mathbf{x}) \quad = \quad & \Big(u(\mathbf{x}) - l(\mathbf{x})\Big) + \frac{2}{\alpha}\Big(l(\mathbf{x}) - \nu(\mathbf{x})\Big)\mathbf{1}_{\{\nu(\mathbf{x})<l(\mathbf{x})\}} + \\ & \frac{2}{\alpha}\Big(\nu(\mathbf{x}) - u(\mathbf{x})\Big)\mathbf{1}_{\{\nu(\mathbf{x})>u(\mathbf{x})\}} \end{aligned}$$

where $\mathbf{1}_{\{expr\}}$ refers to the indicator function, being 1 if expression $expr$ holds and 0 otherwise. This scoring rule rewards narrow intervals, while penalizing lack of coverage. The lower its value, the higher the quality of the confidence interval is considered. The average interval score, denoted AIS, is simply the average of $IS(\mathbf{x})$ over all test points $\mathbf{x}$.

VI. Results

A. Clustering the training data set

In clustering the training data set containing 10,732 data points we have to find a compromise between the desire to obtain clusters with a large enough number of data points to ensure that enough information is available to determine suitable parameter values for the emulator, and the desire to have clusters with a small enough number of data points to avoid computational instabilities (see also Section III-A). Although a nugget can help to avoid computational instabilities (see again Section III-A), it should be used with caution, since the nugget introduces additional uncertainty in an ad hoc way which might not be justified. In any case, the nugget should be smaller than 1, so that the introduced uncertainty is smaller than the process uncertainty [17]. As a rule of thumb, we do not want the nugget to be larger than 0.25 for our emulators. Intuitively speaking we expect that the larger a cluster the larger the nugget has to be to avoid computational instabilities. We determine an appropriate maximum cluster size by randomly selecting subsets of the training data set of varying sizes, and playing with several values for the nugget. This trial and error exercise indicated that for a size of 500 data points a nugget of 0.25 is sufficient to avoid any computational instability. Significantly larger sizes require a larger nugget than 0.25, while a size of 500 with significantly smaller nuggets results in computational instabilities. To illustrate this last point, by choosing the nugget 0.2 we found that there are subsets of the training data set of size 500 for which our software written in R outputs an infinite likelihood, for some values of the δ_i parameters.

A maximum cluster size cannot be given as input to the k-means algorithm, thus we find appropriate clusters by trial and error. Different numbers of clusters are given as input to the k-means algorithm and the resulting number of data points in each cluster is inspected. We find that choosing 32 clusters comes close to the requirement of a maximum cluster size of 500: all but 2 clusters contain at most 500 data points. The 2 largest clusters, having sizes 648 and 652, are split into 2 smaller clusters by applying k-means to each of these clusters and requiring 2 clusters as output for each of them. We then end up with 34 clusters where the largest cluster has size 500.

B. Obtaining an emulator for each cluster

For each cluster an emulator is trained. Although each emulator is trained on a much smaller data set than the full

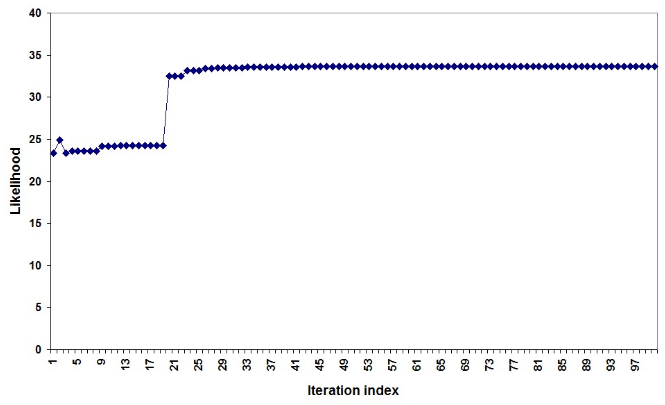

Figure 1. Application of genetic algorithm to find the correlation lengths (first illustration)

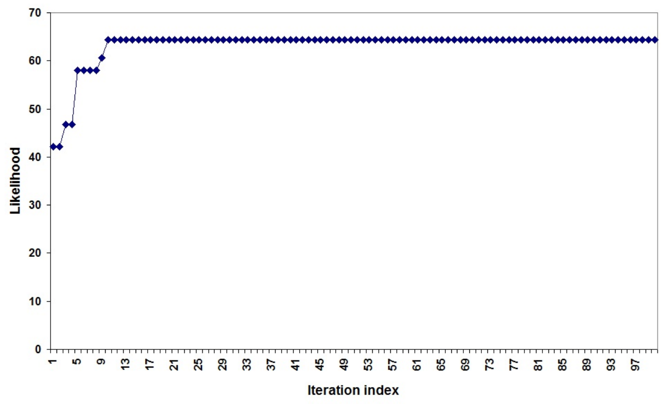

Figure 2. Application of genetic algorithm to find the correlation lengths (second illustration)

training data set, a nugget is still required. The nugget for each of the emulators is chosen heuristically, taking into account the observation that we found subsets of the training data set of size 500 for which a nugget of 0.25 is sufficient to avoid computational instabilities while a significantly smaller nugget does not work (see previous section). If the cluster size is smaller than 150, the nugget is chosen as 0.1, for a cluster size between 150 and 300 the nugget is taken as 0.15, for a cluster size between 300 and 400 as 0.2, and for cluster sizes larger than 400 as 0.25. Computational instabilities are not encountered for any emulator.

As described in Section III-C, a genetic algorithm is used to determine optimal values for the correlation lengths. Figures 1 and 2 show the evolution of the largest fitness value in the population over the 100 iterations for two of the emulators. The fast convergence is also observed for the other emulators.

C. Evaluation of the six implementation methods

The six implementation methods, described in Section IV, are evaluated as explained in Section V. The results are shown in Table I.

Considering ARD, i.e. the evaluation of how well $\hat{\nu}(\mathbf{x})$ approximates $\nu(\mathbf{x})$, we see that E_{1_min} is the best method, with a second place for E_{2_min}. It is also seen that using the minimum distance gives better results than using the center distance.

TABLE I. EVALUATION OF THE SIX IMPLEMENTATION METHODS

Implementation method	*ARD*	*AIS*
E_{1_min}	0.14	3.27
E_{1_cen}	0.17	3.95
E_{2_min}	0.21	2.06
E_{2_cen}	0.23	3.21
E_{3_min}	0.24	2.53
E_{3_cen}	0.26	3.52

Analysis of AIS, i.e. the evaluation of the quality of the produced confidence intervals, shows another picture. For this measure E_{2_min} is by far the best method, while the second best is E_{3_min}. Again, better results are obtained with the minimum distance than with the center distance.

Thus, our experiment points to the following observations:

- The minimum distance gives for both evaluation measures better results. This is counterintuitive to the widespread use of the center distance in cluster analysis.
- For ARD it holds that the *less* emulators are used to determine an approximate value $\hat{\nu}(\mathbf{x})$ the *better*.
- For AIS it holds that using *several* emulators to determine a confidence interval around $\hat{\nu}(\mathbf{x})$ is *better* than using one emulator.

VII. CONCLUSION AND FURTHER RESEARCH

In this paper we have shown how to apply statistical emulation to a very large data set, generated by an agent-based model, using cluster analysis.
In terms of methodology, the main contribution of our work is in demonstrating how the concept of ensemble methods in machine learning can be combined with the inverse distance weighting interpolation method to generalize the standard emulation framework where a single Gaussian distribution models the output of a deterministic computer model, to a framework where a mixture of Gaussian distributions is used for this modeling process.
In terms of experimental results, the presented work shows that the choice of definition of distance from a point to a cluster is an important one, and that one cannot take the center distance, widely used in cluster analysis, for granted. Furthermore, the results demonstrate that it can be advantageous to consider a different number of distributions in a mixture of Gaussians depending on whether an approximation to the deterministic computer model output in a given input is to be determined or a confidence interval is to be sought. For our experiment we found that the less emulators are used to determine an approximation to the output the better, i.e. using one emulator is preferred, while determining a confidence interval is best done with several emulators. Of course, this is a very preliminary conclusion and is by no means to be considered as a general rule. Further research is definitely needed to shed more light on this.

Other further research includes:

- In our work, we allowed a non training data point to belong to several clusters, in contrast to the training data points which were assigned to a single cluster using k-means. Instead of using k-means to cluster the training data set, one can think of applying fuzzy clustering. However, this will result in the loss of a basic property of emulation, namely that in the training data points it holds that $m^\star(\mathbf{x}) = \nu(\mathbf{x})$. This property will be lost when using fuzzy clustering, since in this case a training data point will belong to several clusters. On the other hand, fuzzy emulation, as we might call it, has the advantage that it can preserve continuity of $\hat{\nu}(\mathbf{x})$ by allowing the membership degrees of an input point $\mathbf{x}$ in each of the clusters to change gradually as $\mathbf{x}$ itself changes gradually. This does not hold for our k-means emulation framework, since points $\mathbf{x}$ and $\mathbf{x}'$ that are close together can belong to two different clusters and thus having outputs determined by different emulators.

- Comparison with treed partitioning could provide additional insights into our method as well as into treed partitioning itself. Treed partitioning has the disadvantage of producing an approximation $\hat{\nu}$ to ν that is not continuous, while fuzzy emulation does not have this deficiency. Possibly, ideas from both methods can be integrated in a fruitful way.
- Methodological further research includes developing methods to find an appropriate number of clusters for the given training data set in a systematic fashion, and developing methods to determine the coefficients in the mixture of Gaussians according to some optimality criterion. After all, the inverse distance weighting method determines the coefficients in a purely heuristic way, without taking into account any evaluation measure.

REFERENCES

[1] S. Conti and A. O'Hagan, "Bayesian emulation of complex multi-output and dynamic computer models," Journal of Statistical Planning and Inference, vol. 140, 2010, pp. 640–651.

[2] A. Sarri, S. Guillas, and F. Dias, "Statistical emulation of a tsunami model for sensitivity analysis and uncertainty quantification," Natural Hazards and Earth System Sciences, vol. 12, 2012, pp. 2003–2018.

[3] E. Chang, M. Strong, and R. Clayton, "Bayesian sensitivity analysis of a cardiac cell model using a Gaussian process emulator," PLoS One, vol. 10, 2015.

[4] R. Gramacy and H. Lee, "Bayesian treed Gaussian process models with an application to computer modeling," Journal of the American Statistical Association, vol. 103, 2008, pp. 1119–1130.

[5] A. Jain and R. Dubes, Eds., Algorithms for clustering data. Prentice Hall College Div, 1988.

[6] N. Gilbert, Ed., Agent-based models: quantitative applications in the social sciences. SAGE Publications, Inc, 2007.

[7] A. Bazghandi, "Techniques, advantages and problems of agent based modeling for traffic simulation ," IJCSI International Journal of Computer Science Issues, vol. 9, 2012.

[8] T. Fent, B. Aparicio Diaz, and A. Prskawetz, "Family policies in the context of low fertility and social structure," Demographic Research, vol. 29, 2013, pp. 963–998.

[9] I. Andrianakis and P. G. Challenor, "The effect of the nugget on Gaussian process emulators of computer models," Computational Statistics and Data Analysis, vol. 56, 2012, pp. 4215–4228.

[10] J. Oakley and A. O'Hagan, "Bayesian inference for the uncertainty distribution of computer model outputs," Biometrika, vol. 89, 2002, pp. 769–784.

[11] R. Gramacy and H. Lee, "Cases for the nugget in modeling computer experiments," Statistics and Computing, vol. 22, 2012, pp. 713–722.

[12] J. Carr, "An introduction to genetic algorithms," url: https://karczmarczuk.users.greyc.fr/TEACH/IAD/GenDoc/carrGenet.pdf, 2014.

[13] T. Dietterich, "Ensemble methods in machine learning," in Proceedings of the First International Workshop on Multiple Classifier Systems. Springer-Verlag, 2000.

[14] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in Proceedings of the 1968 ACM National Conference.

[15] J. Sloan and S. Sinha, "Bayesian predictive intervals for a mixture of exponential failure-time distributions with censored samples," Statistics & Probability Letters, vol. 11, 1991, pp. 537–545.

[16] T. Gneiting and A. Raftery, "Strictly proper scoring rules, prediction, and estimation," Journal of the American Statistical Association, vol. 102, 2007, pp. 359–378.

[17] P. Ranjan, R. Haynes, and R. Karsten, "Stable approach to Gaussian process interpolation of deterministic computer simulation data," Technometrics, vol. 53, 2011, pp. 366–378.