Groundwater Flow Meta-model for Multilevel Monte Carlo Methods

Martin Špetlík, Jan Březina

Technical University of Liberec, Faculty of Mechatronics, Informatics and Interdisciplinary Studies,
Institute of New Technologies and Applied Informatics
Studentská 1402/2, 461 17 Liberec 1, Czech Republic
martin.spetlik@tul.cz, jan.brezina@tul.cz

Abstract: This paper presents a meta-modeling-based technique to reduce the computational cost of Monte Carlo methods. A stochastic simulation of groundwater flow is substituted with a graph neural network meta-model. This type of neural network can deal with a non-euclidean structure of the input data, which in our case is a graph representing a random field on an unstructured mesh. In order to find the most suitable meta-model, a comparison of the standard support vector regression with spectral and spatial graph convolutional neural networks is provided. Both the Monte Carlo method and the multilevel Monte Carlo method are extended by the meta-model level. In both cases, up to 50% savings in computational cost are achieved while maintaining the accuracy of Monte Carlo estimates.

1 Introduction

Groundwater flow in the vicinity of a future nuclear waste repository is the major motivation for our research. In the first stage, our quantity of interest (QoI) is the total flow through the observed area. However, since not all indispensable properties of the rock environment are known, it is not feasible to determine the desired total flow. For that reason, and given the nature of the rock, the missing properties are modeled as random fields (RFs). The probability density function (PDF) of our QoI is under scrutiny. To approximate the PDF, the maximum entropy method is adopted. The method utilizes so-called generalized statistical moments. In order to estimate these moments, the Monte Carlo methods are employed. These methods consist in repeating a random experiment. Depending on the required accuracy, it might result in thousands of groundwater flow simulations, which can significantly affect the total computational cost.

This fact was behind the idea to substitute simulations with an approximation model, often called a meta-model. Furthermore, we use the multilevel Monte Carlo method (MLMC) instead of the standard Monte Carlo method to reduce the variance of estimates more effectively. Since the MLMC uses simulations of different accuracy, we aim to substitute those with the lowest accuracy, performed in the largest number. Their approximation shouldn't be excessively challenging for meta-modeling. We presume that this approach could reduce the computational cost of MLMC estimates. It is also necessary to maintain the accuracy of the estimates to achieve a good PDF approximation. Graph convolutional neural networks and the support vector regression [2] are used as meta-models.

First, the groundwater flow problem definition and the maximum entropy method brief introduction are provided in section 2. Then, Monte Carlo methods are presented in section 3. Spektral and spatial graph convolutional neural networks used as meta-models are delineated in section 5. Section 6 provides the link between the MLMC and a meta-model, including the related work. Finally, section 7 is devoted to the results and discussion.

2 Problem Definition

A 2D benchmark problem of groundwater flow through a porous medium is used to test our proposed methods. The same problem can be found in the paper by Blaheta et al. [5]. The groundwater flow is described by the boundary value problem on the unit square:

$$\begin{aligned} -div(K(x)\nabla p) &= 0 \\ -K\nabla p \cdot \vec{n} &= 0, \end{aligned} \tag{1}$$

K is the hydraulic conductivity, p is the pore water pressure, $-K\nabla p$ is Darcy's velocity, $\vec{n}$ is the unit normal vector. We are interested in the total flow Y through the specified area

$$Y = \int_0^1 [-K\nabla p \cdot \vec{n}](1,y)dy. \tag{2}$$

The problem is prescribed on unstructured meshes and is solved by the finite element method using the Flow123d software [6]. Since the proposed approach shall be adopted for more complicated problems with complex irregular geometry, unstructured meshes are preferred over structured ones. The hydraulic conductivity is considered as a random field (RF) with the exponential covariance function $C(r) = \sigma^2 exp(-\frac{r}{\lambda})$, where λ is a correlation length and σ is a standard deviation. The GSTools software library [23] is utilized to generate RF; see Hesse et al. [16] for theoretical details.

2.1 Maximum Entropy Method

When considering the hydraulic conductivity as a random field, the total flow Y is a real random variable. The aim is

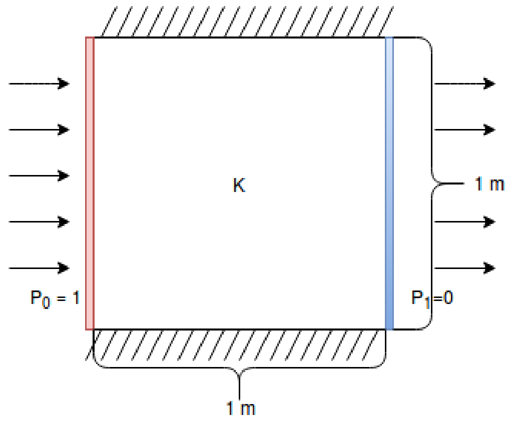

Figure 1: An illustration of the porous media flow problem

to determine its probability density function $\rho(Y)$, which is nonnegative on a bounded domain Ω. The maximum entropy method [17] (MEM) is employed for this purpose. Its basic idea lies in approximating a PDF from generalized statistical moments and functions that calculate them:

$$\begin{gathered}\int_\Omega \rho(y)dy = 1,\\ \int_\Omega \phi_r(y)\rho(y)dy = \mu_r, \quad r = 1,\dots,R\end{gathered} \tag{3}$$

where $\rho(Y)$ is an unknown, $\{\mu_r\}_{r=1}^R$ are values of generalized moments, $\{\phi_r\}_{r=1}^R$ are linearly independent functions (see [3, p. 1350]) used for calculating generalized moments and satisfying $\phi_1 = 1$, $\phi_r \in C^R(\Omega)$, $r = 2,\dots,R$. In our case, $\{\phi_r\}_{r=1}^R$ are the Legendre polynomials, $R \in \mathbb{N}$ is the number of moments.

Provided that the system of equations 3 has a solution, then it has an infinite number of them. Therefore, the most apposite solution is the one with the maximum Shannon entropy (defined in [27]) according to E. Jaynese [17, p. 623]. Determining $\rho(Y)$ with the maximum entropy takes the form of finding the global maximum of the functional

$$H(\rho) = -\int_\Omega \rho(y)\ln(\rho(y))dy \tag{4}$$

under the constraints prescribed in equations 3.

A numerical solution and general limitations of the MEM are beyond the scope of this paper. A more detailed insight is provided in [3, 4].

3 Monte Carlo Methods

Since the MEM utilizes moment values $\{\mu_r\}_{r=1}^R$ that are also random variables derived from some random input X (which is the hydraulic conductivity K in this study). It is necessary to estimate their expected values. Monte Carlo methods are used for this.

The standard Monte Carlo method (MC) is an approach for estimating expected values of some stochastic simulation variable. The basic idea comes from the law of strong numbers. Therefore, in order to obtain an unbiased estimate of the expected value $Y = \mathbb{E}[P(x)]$, the MC consists in the arithmetic mean of N independent samples

$$\hat{Y} = \frac{1}{N}\sum_{n=1}^{N} P(x_n), \tag{5}$$

where P is a random variable depending on X. According to the central limit theorem:

$$\hat{Y} \sim \mathcal{N}\left(\mu, \frac{\sigma^2}{N}\right). \tag{6}$$

Consequently, the computational cost of reducing the variance (increasing N) of $\hat{Y}$ can be very high. To overcome this drawback (for details, see [11, p. 4]), the multilevel Monte Carlo method (MLMC) was formulated.

3.1 Multilevel Monte Carlo Method

In the case of the MLMC [11], the expected value of a random variable P is estimated based on the sequence of its approximations P_1, ..., P_L:

$$\mathbb{E}[P] = \mathbb{E}[P_1] + \sum_{l=2}^{L}\mathbb{E}[P_l - P_{l-1}], \tag{7}$$

where $P_{l-1} \approx P_l$. The estimate of the expected value of P improves from P_1 to P_L. The unbiased estimate of $\mathbb{E}[P]$:

$$\hat{P} = \frac{1}{N_1}\sum_{n=1}^{N_1} P_1(x_n^1) + \sum_{l=2}^{L}\left\{\frac{1}{N_l}\sum_{n=1}^{N_l}\left(P_l(x_n^l) - P_{l-1}(x_n^l)\right)\right\}, \tag{8}$$

where L is the total number of levels, N_l is a number of simulation samples at level l. Input random data x_n^l are dependent for each simulation pair n at level l. But they are independent across levels. Since $P_l - P_{l-1}$ are also independent across levels, the total estimated variance $\hat{V}$ of $\hat{P}$ has the following form:

$$\hat{V} = \sum_{l=1}^{L}\frac{\hat{V}_l}{N_l}, \tag{9}$$

where $\hat{V}_1$ is an estimate of P_1 variance and $\hat{V}_l$ for $l > 1$ is an estimate of $P_l - P_{l-1}$ variance. The MLMC computational cost:

$$C = \sum_{l=1}^{L} N_l C_l, \tag{10}$$

where C_l is a cost of a single simulation sample at level l measured in, for instance, the number of computational operations, execution time, CPU time, etc. Let C_1 denote a cost of P_1, C_l is a cost of $P_l - P_{l-1}$ for $l > 1$.

Given the MLMC theoretical properties stated in [11, theorem 1], variance V_l should decrease from $l = 1$ to $l = L$, while computational cost C_l should increase from $l = 1$ to $l = L$.

3.2 Number of Simulation Samples

The way of determining N_l is a crucial part of the MLMC. There are two main approaches:

- minimizing the total variance V with respect to the prescribed computational cost
- minimizing the total computational cost C with respect to the prescribed target variance V_t.

Our attention is paid to the latter approach, which has the form of finding the minimum of equation 10 under the constraint

$$V_t = \sum_{l=1}^{L} \frac{\hat{V}_l}{N_l}. \tag{11}$$

After some calculus and concerning the use of the MEM, N_l are determined with respect to R moment values

$$N_l^r = \sqrt{\frac{\hat{V}_l^r}{C_l} \frac{\sum_{i=1}^{L} \sqrt{\hat{V}_i^r C_i}}{V_t}}, \; r = 1, \dots, R, \tag{12}$$

where $\hat{V}_l^r$ is an estimated variance of $\phi_l^r(x) - \phi_{l-1}^r(x)$ for r-th moment at level l. Finally, $N_l = \max\limits_{r=1,\dots,R} N_l^r$.

The mlmc software library [7] is employed to schedule samples and post-process results, including our MEM implementation.

4 Meta-modeling

A meta-model is a simpler and explicit mathematical function that approximates complicated functions that can be both implicit and evaluated by a simulation model, measurements data, or experiments. Alternative names can be found in the literature, such as surrogate model, surface response model, emulator, etc. Meta-modeling is the process of designing meta-models. The common meta-models used in the finite element analysis include support vector regression (SVR), artificial neural network (ANN), Kriging, also known as Gaussian process [19], radial basis function (RBF) [35], etc.

Artificial neural networks have become very popular for meta-model design. A regression problem is solved by an ANN trained by supervised learning. The aim is to obtain an unknown mapping of input neuron activities to an output neuron activity. With regard to the nature of our input data, which is a 2D correlated random field, it would be suitable to use a convolutional neural network (CNN) meta-model. Nevertheless, CNNs cannot be applied directly to data on unstructured meshes (see [31]).

There are few ways how to overcome this difficulty. The basic solution is to directly apply a 1D CNN to nodal values of an unstructured mesh considered as a vector. However, it has limited success [15]. The other option is interpolating data from an unstructured mesh to a structured mesh and then using a CNN. This approach leads to an additional error caused by interpolation. It requires a larger number of grid points to capture features from original unstructured data, according to [22]. Wang et al. [29] proposed a CNN on unstructured meshes. This novel approach is limited to random fields described by the spectral representation method or the Karhunen-Loève expansion. Another popular approach is the representation of unstructured meshes as undirected graphs, which allows us to use graph convolutional neural networks (GCNs) [31]. In this study, we use GCN-based meta-models and support vector regression meta-models.

4.1 Graph Representing a Random Field

A graph $G = (V, E)$ is an unweighted undirected graph describing the mesh structure on which a random field is generated. V is a set of vertices representing mesh elements, $V = \{v_1, v_2, \dots, v_S\}$, S is the number of vertices (=mesh elements). The neighborhood of a vertex v is defined as $N(v) = \{u \in V | (v, u) \in E\}$. Each vertex has a feature representing a random field value at the corresponding mesh element. E is a set of edges. An edge connects adjacent mesh elements. An adjacency matrix A is used to represent G on a computer.

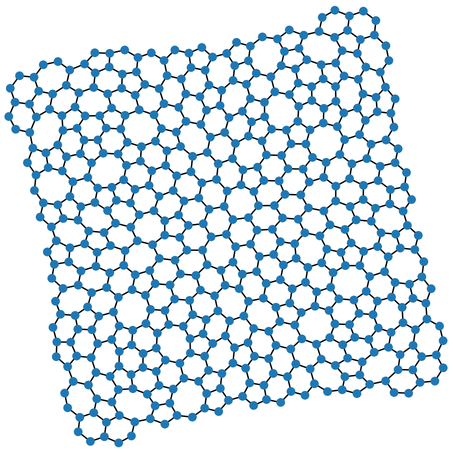
Figure 2: An example of a graph on 546 mesh elements

5 Graph Neural Networks

A variety of real-world problems can be represented as graphs. Imagine social networks, molecules, or in our case, random fields on unstructured meshes.

Graph neural networks (GNNs) are deep learning-based models that operate on the graph domain [34]. They have a wide range of applications in classification, relation extraction, molecular fingerprints prediction, and so on. A comprehensive survey on graph neural networks can be found in [30]. In brief, GNNs are categorized into several groups, such as graph convolutional networks [33], graph attention networks, graph recurrent networks, etc. The graph convolutional networks (GCNs) are the most important ones because they are the fundamental of other graph neural network models (see [21]). GCNs can be divided into spectral GCNs and spatial GCNs.

5.1 Spectral Graph Convolutional Neural Networks

Spectral GCNs are based on knowledge from the field of graph signal processing. The well-known convolution has the following form:

$$(f * g)(x) = \int_{R^k} f(t) g(x-t) dt, \tag{13}$$

but it is unclear how to interpret the translation $g(x-t)$ for a graph signal. Thus the convolution operation is not defined on structures like graphs. W. Hamilton [13] or K. Otness [24] provides a detailed explanation of the graph convolution. The basic idea is to take advantage of the fact that convolution in the spatial domain corresponds to the point multiplication in the spectral domain.

A graph signal $x \in R^S$ (vector of all G vertex values) is transformed by a graph Fourier transform from the spatial domain to the spectral domain. Loosely speaking, the standard Fourier transform is connected to the eigendecomposition of the Laplace operator. In the case of graphs, the Laplace operator is represented by the Laplacian matrix [33, p. 4]. The normalized Laplacian matrix L can also be used as a Laplace operator:

$$L = I - D^{-1/2} A D^{-1/2}, \tag{14}$$

where A is a graph adjacency matrix, and D is a diagonal matrix of vertex degrees. L is a symmetric real-valued positive semi-definite matrix that can be factorized $L = U \Lambda U^T$, where U is a matrix of eigenvectors and Λ is a diagonal matrix of eigenvalues. Then the graph Fourier transform of x is [34, p. 60]:

$$F(x) = U^T x \tag{15}$$

and its inverse:

$$F^{-1}(\hat{x}) = U \hat{x}. \tag{16}$$

Finally, the graph convolution of x and a filter $g \in R^k$:

$$x *_G g = F^{-1}(F(x) \odot F(g)), \tag{17}$$

where $*_G$ is the convolutional operator on a graph, and $\odot$ represents the element-wise Hadamard product. The signal x can be filtered in the spectral domain by a filter g_θ in this way (for details, see [13, p. 83]):

$$x *_G g_\theta = U g_\theta(\Lambda) U^T x, \tag{18}$$

where $g_\theta(\Lambda)$ is a polynomial of the eigenvalues of the Laplacian. The filter represents learnable weights.

ChebNet GCN with a Chebyshev convolutional layer was first introduced by Defferrard et al. [10] in 2016. The filter g is approximated by a truncated expansion of Chebyshev polynomials $T_k(x)$ up to the K^{th} order:

$$g_\theta = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}), \tag{19}$$

where $\theta_k \in R^k$ is a vector of Chebyshev coefficients, $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I$, λ_{max} is the maximum eigenvalue from Λ. The Chebyshev polynomials are defined recursively by $T_{k \geq 2}(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1$, $T_1(x) = x$. Then x filtered by g:

$$x *_G g = U(\sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})) U^T x. \tag{20}$$

The eigendecomposition can be avoided [30, p. 10]:

$$x *_G g = (\sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})) x, \tag{21}$$

where $\tilde{L} = 2L/\lambda_{max} - I$. $\tilde{L}$ is called the rescaled graph Laplacian, the eigenvalues are mapped from $[0, \lambda_{max}]$ to $[-1, 1]$, the Chebyshev polynomials form an orthogonal basis.

Several K settings were tested. Since $K > 1$ did not bring improvement for our problem, $K = 1$ is used in our study. For this setting, the ChebNet is very similar to the GCN proposed by Kipf and Welling [18]. Filters are exactly K-localized. It means the filter modifies information at a particular vertex based on the information from vertices in its K neighborhood. It essentially connects spectral-based methods with spatial-based ones.

5.2 Spatial Graph Convolutional Neural Networks

As for spatial GCNs [9], the convolution is performed in the graph domain by propagating information along edges between adjacent vertices. The concept is based on message passing neural networks [12]. The spatial graph convolution is defined as follows [30, p. 12]:

$$h_v^{(k)} = U_k(h_v^{(k-1)}, \sum_{u \in N(v)} M_k(h_v^{(k-1)}, h_u^{(k-1)}, x_{vu}^e)), \tag{22}$$

where $h_v^{(k)}$ represents features of vertex v in a hidden layer k, x_{vu}^e is an optional feature vector of an edge (v,u). U_k and M_k are the update and message functions with learnable parameters.

GraphSage is an aggregation-based learning model proposed by Hamilton et al. [14]. It performs the convolution as follows:

$$h_v^{(k)} = \sigma(W^{(k)} \cdot f_k(h_v^{(k-1)}, \{h_u^{(k-1)} \forall u \in S_{N(v)}\})), \tag{23}$$

where h_v^0 is a representation vector of vertex v features, σ represents a nonlinear activation function, $W^{(k)}$ is a weight matrix in layer k, f_k represents an aggregation function, $S_{N(v)} \subseteq N(v)$. Thus GraphSage enables the use of huge graphs by selecting a subset from each vertex neighbourhood instead of using all neighbours. There are different operations used as f_k, such as average, sum, max, min, etc. In this study, we use GraphSage with a single layer, f is the summation.

A comparison between spectral GCNs and spatial GCNs is summarized by Wu et al. [30, p. 13]. They state that spatial GCNs are usually preferred over spectral GCNs. However, in our problem 2, we aim to approximate the solution of the elliptic partial differential equation. To do that numerically, it is possible to use spectral methods [8]. This entitles us to assume that spectral GCNs will be more appropriate.

6 Monte Carlo Methods with Meta-model

There are several approaches how to use meta-models to reduce the computational cost of Monte Carlo estimates. The relatively frequent approach (e.g., [1, 20, 28]) is to construct a meta-model of a simulation, then the MC is conducted with the meta-model instead of the simulation. Other approaches consist in replacing the whole Monte Carlo estimate by a meta-model. For instance, Safta et al. [26] use polynomial chaos expansion meta-model, which requires a significantly lower number of samples than the MC. Rosenbaum and Staum [25] construct a stochastic simulation meta-model by way of the MLMC.

We propose a different approach based on adding a new coarse meta-model-based level to the original Monte Carlo method (MC or MLMC). The MLMC estimate from equation 8 has now the following form:

$$\hat{P}_{meta} = \frac{1}{N_1}\sum_{n=1}^{N_1}\tilde{P}_1(x_n^1) + \frac{1}{N_2}\sum_{n=1}^{N_2}\left(P_1(x_n^2) - \tilde{P}_1(x_n^2)\right) + \sum_{l=3}^{L}\left\{\frac{1}{N_l}\sum_{n=1}^{N_l}\left(P_l(x_n^l) - P_{l-1}(x_n^l)\right)\right\}, \tag{24}$$

where $\tilde{P}_1$ denotes a meta-model approximation of P_1. Since the MLMC assumes $P_{l-1} \approx P_l$, it is completely valid to employ a meta-model as the coarsest level (which we denote as the meta-level) and the difference between P_1 and $\tilde{P}_1$ as the subsequent first level. Let MLMC$_{\text{meta}}$ denote the multilevel Monte Carlo method with the meta-level.

In order to meet basic theoretical properties of the MLMC (mentioned in section 3.1) the meta-level computational cost C_1 should be lower than C_2, and meta-level samples variance V_1 should be greater than V_2.

6.1 Computational cost

The total MLMC cost (see equation 10) depends on C_l, in our case measured in execution time. As a meta-model brings additional computational costs, parts of the learning process, including their costs, are introduced.

First, the number of training samples N_{tr} is determined, and numerical simulations are executed. Then, for each training sample, a stored random field is pre-processed to become a meta-model input, let C_{pr} denote the cost of this operation. The pre-processing can slightly differ for different meta-models. The SVR input is a vector of random field elements, while the GCN input is their graph. Afterward, a meta-model is trained, and the cost of this operation is C_{ml}. Once the meta-model is prepared, it is possible to make predictions at the cost of C_{pred}. In the case of the meta-level, there is no stored random field because there is no simulation executed. Thus a random field is generated and pre-processed at the cost of C_{rf}.

Let C_2 denote the computational cost of a sample at the first MLMC$_{\text{meta}}$ level:

$$C_2 = C_1^* + (C_2^{tr} + C_2^{pr}(N_2 - N_{tr}))/N_2, \tag{25}$$

where C_1^* is the cost of a simulation sample. C_2^{tr} is the cost of a meta-model training procedure:

$$C_2^{tr} = C_{pr}N_{tr} + C_{ml} + C_{pred}N_{tr}, \tag{26}$$

C_2^{pr} represents the cost of a first level meta-model prediction sample:

$$C_2^{pr} = C_{pr} + C_{pred}. \tag{27}$$

Let C_1 denote the cost of a meta-level sample that utilizes meta-model trained at the first level:

$$C_1 = C_{rf} + C_{pred}. \tag{28}$$

Sample costs C_l for $l > 2$ are not affected by the meta-modeling. Meta-models training is performed on the cluster; 16 CPUs (Intel Xeon Silver 4114 CPU (2.2GHz)) and 16 GB RAM (DDR4 2400 ECC Reg dual rank) are assigned for that task. Also, MLMC simulations are executed in parallel. Therefore, C measured in execution time is not equal to the real elapsed time. However, the savings in C can significantly affect the total elapsed time, especially for a small V_t, which is necessary to obtain a neat PDF by the MEM.

7 Results

The proposed meta-modeling techniques are investigated in this section. The most suitable one is used by Monte Carlo methods. The models are compared for different mesh sizes and random field parameters.

The cross-validation-like procedure is implemented to compare models. Initially, N = 50000 simulation samples are generated. The number of training samples N_{tr} is empirically determined based on the properties of the MLMC. Since we are interested in $V_t \leq$ 1e−5 (for the sake of the MEM capability to approximate a PDF neatly), the number of samples at the coarsest level is at least 2000 for our problem. Thus, the procedure is as follows: 2000 training samples ($N_{tr} = 2000$) out of N are chosen for meta-model training. The rest is considered to be test data. Validation data accounts for 20% of N_{tr}. This procedure is repeated 25 times with independent training sets. Given that we use a random field with the exponential correlation function, the logarithm of the RF values is used as meta-models input to facilitate their training. In all of the following cases, the final N_l is determined based on the geometric sequence of the initial number of samples decreasing across levels from $N_1^0 = 2000$ to $N_L^0 = 100$. The number of moments $R = 25$ is utilized.

7.1 Meta-models Setting

We have tried dozens of GCN topologies. The most promising ones consist of one ChebNet/GraphSage layer with 8/60 output channels and the ReLU activation function, following by a global summation pool and the output layer with one neuron with the identity activation function. Table 1 contains main hyperparameters that enable to obtain an accurate and computationally efficient meta-model. Another important hyperparameter is the number of output

Table 1: GCN hyperparameters common to all cases

optimizer	Adam
hidden activation	ReLU
output activation	identity
learning rate	0.001
regularization	None
loss	MSE
max epochs	2000
patience	150

channels. The optimal number differs across types of GCN layers. It forms a vector of hidden representations corresponding to each vertex. It is possible to think of each channel as responding to some different set of features, so different channels could become specialized to recognize different objects as described by Zhang et al. [32].

Regarding the SVR, the Gaussian radial basis function (RBF) kernel and the regularization parameter $W = 0.06$ are used. An explanation of the role of SVR parameters is provided in [2].

7.2 Meta-models on Different Meshes

Three meta-model techniques are compared: the SVR, the ChebNet GCN, and the GraphSage GCN. Recall that low accurate simulations are supposed to be substituted with meta-models. Hence the emphasis is placed on meta-models trained on small meshes. In particular, meshes of 6, 48, and 546 elements are compared. Random field parameters are by default $\lambda = 0.1$ and $\sigma = 1$. For a given mesh size, the models are trained and tested on the same data.

Table 2 and Table 3 show the models final accuracy on training data and test data, respectively. The arithmetic mean and the standard error of 25 calculations of the mean squared error (MSE): $\frac{1}{D}\sum_{i=1}^{D}(y^i - y^i_{meta})^2$ are provided, where y^i is a correct value and y^i_{meta} is a predicted value, D is a number of samples. The relative squared error (RSE) is used to compare the models across cases (see Table 4).

Table 2: Meta-models train MSE

train MSE	number of mesh elements		
	6	48	546
arithmetic mean			
SVR	0.01711	0.007185	0.004657
ChebNet GCN	0.03476	0.009441	0.004633
GraphSage GCN	0.02368	0.01045	0.006848
standard error			
SVR	4.1e−4	6.5e−5	3.6e−5
ChebNet GCN	7.0e−3	8.1e−5	2.2e−4
GraphSage GCN	4.3e−4	9.9e−5	1.8e−4

Table 3: Meta-models test MSE

test MSE	number of mesh elements		
	6	48	546
arithmetic mean			
SVR	0.03694	0.008968	0.006696
ChebNet GCN	0.04331	0.008392	0.004595
GraphSage GCN	0.02284	0.009200	0.006839
standard error			
SVR	6.0e−4	5.4e−5	6.1e−5
ChebNet GCN	2.6e−3	7.9e−5	1.9e−4
GraphSage GCN	1.9e−4	1.2e−4	1.9e−4

Table 4: Meta-models test RSE

test RSE	number of mesh elements		
	6	48	546
SVR	0.1263	0.1551	0.1335
ChebNet GCN	0.1481	0.1451	0.1064
GraphSage GCN	0.1162	0.1591	0.1364

Data in the tables show that all models provide similar results in terms of the train MSE and the test MSE, and also, the RSE values are of the same order of magnitude. Importantly, presented data show no significant outliers that would be highly undesirable for our purposes. Although the results are very similar, it can be seen that the optimal meta-model varies depending on the mesh size. While the GraphSage GCN or the SVR is more advantageous for very small meshes, the use of the ChebNet GCN is most suitable in the case of larger meshes.

Nevertheless, additional experiments show that the meta-model approximation is not sufficiently accurate for meshes of thousands of elements. We face the so-called curse of dimensionality due to the limited number of training samples. An increasing number of samples can overcome this difficulty, but many simulations need to be performed in such a case, and learning cost increases. Thus, to keep $N_{tr} = 2000$, we limited ourselves to meta-models based on simulations on meshes with a maximum of ca. 1000 elements.

In practice, we see that the ChebNet GCN is not only better for larger meshes (from ca. 500 up to ca. 1000 elements) but also provides a more stable learning process in terms of a smooth decrease of a validation loss than the GraphSage GCN. Therefore the ChebNet GCN is preferred over the GraphSage GCN for our task.

7.3 Role of Random Field Parameters

Random field parameters affect the meta-model learning ability. Changing the standard deviation σ only scales features of graph vertices, which our learning procedure can handle. The correlation length λ plays a more significant role. As λ decreases, the correlation between the features of vertices decreases as well. For small correlation lengths, e.g., $\lambda = 0.001$, all features are almost uncorrelated, which has a similar effect as increasing the number of vertices with the original correlation length $\lambda = 0.1$. Thus, increasing the number of training samples is necessary to obtain the same results for $\lambda = 0.001$.

This reveals one of the SVR drawbacks, which is the requirement of a lot of training samples [2, p. 76]. While the number of training samples 5000 is enough for the ChebNet GCN on meshes of 546 elements and $\lambda = 0.001$. In the case of the SVR, even 15000 training samples is not enough to get similar results for $\lambda = 0.001$ as for $\lambda = 0.1$. For this reason, the ChebNet GCN is preferred for further analysis. It is also worth noting that naturally, fewer training samples are enough for $\lambda > 0.1$.

7.4 MC extended by the meta-level

Let use the trained meta-models with the Monte Carlo method described in section 6. To illustrate our approach, we use three different MC with simulations on meshes of 6, 48, and 546 elements, each extended to two-level MLMC$_{\text{meta}}$. Table 5 shows the ratio between the total cost C_{meta} of our MLMC$_{\text{meta}}$ and the total cost C of the original MC, $V_t = 1\text{e}{-5}$. It is important to note that the ratio C_{meta}/C is a bit smaller for $V_t << 1\text{e}{-5}$, where the meta-model learning cost is negligible due to the larger N_2. It emerges that the computational cost savings of at least 50% are achieved for $R = 2$.

Table 5: Computational cost of MLMC$_{\text{meta}}$ to MC

C_{meta}/C	number of mesh elements		
	6	48	546
SVR	0.2077	0.4947	0.4666
ChebNet GCN	0.3648	0.4371	0.4395
GraphSage GCN	0.2812	0.4485	0.4494

In the rest of this section, MLMC$_{\text{meta}}$ employs the ChebNet GCN on 546 mesh elements. Since the MEM utilizes moment values μ, it is important to verify if their estimates by the MLMC$_{\text{meta}}$ are the same as estimates by the original MC. We construct a reference MC (MC$_{\text{ref}}$) of 50000 samples on meshes of 217208 elements, moments estimates are denoted as $\hat{\mu}^i_{ref}$. Figure 3 shows the MSE: $\frac{1}{25}\sum_{i=1}^{25}(\hat{\mu}^i_{ref} - \hat{\mu}^i)^2$ for estimated moment values by the MC, and the MSE: $\frac{1}{25}\sum_{i=1}^{25}(\hat{\mu}^i_{ref} - \hat{\mu}^i_{meta})^2$ for moments estimated by the MLMC$_{\text{meta}}$.

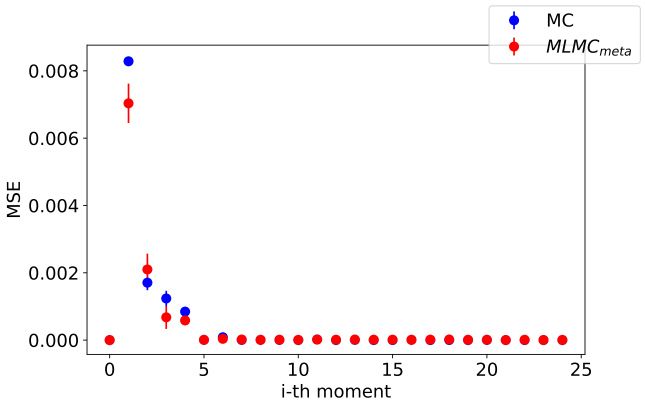

Figure 3: Comparison of the MSE between reference moments and moments estimated by MC and MLMC$_{\text{meta}}$, $V_t = 1\text{e}{-5}$

Since the error of moments estimates is similar for both the MC and the MLMC$_{\text{meta}}$, it is a good prerequisite for a fine PDF approximation. Figure 4 shows an example of a PDF approximated by the MEM based on moments from the MC and the MLMC$_{\text{meta}}$. Given the 25 repetitions, the average KL divergence $\overline{KL}(\rho_{ref}||\rho_{MC}) = 0.034$ and $\overline{KL}(\rho_{ref}||\rho_{MLMC_{meta}}) = 0.031$, for $R = 25$, $V_t = 1\text{e}{-5}$. Thus, it is possible to get comparable results by both approaches.

7.5 MLMC extended by the meta-level

Since we cannot effectively train the meta-models on meshes with more than ca. 1000 elements, it is advisable to use initially the MLMC that generally reduces the computational cost compared to the MC. Applying formula 24, a meta-model is trained on the coarsest level, where a

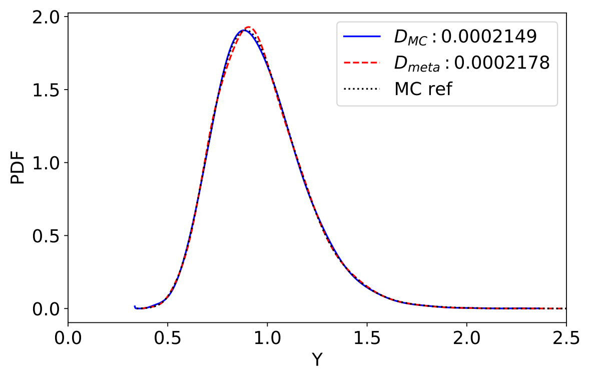

Figure 4: Comparison of PDFs approximated by moments from the MC (blue) and the MLMC$_{\text{meta}}$ (red dotted), $D_{MC} = KL(\rho_{ref}||\rho_{MC})$ is the Kullback-Leibler (KL) divergence between PDF ρ_{MC} from MC data and the reference PDF ρ_{ref}, $D_{meta} = KL(\rho_{ref}||\rho_{MLMC_{meta}})$, where $\rho_{MLMC_{meta}}$ is the density from MLMC$_{\text{meta}}$ data, V_t = 1e−6, $R = 25$

mesh should have a small number of elements. Thus, meta-model learning is feasible. To illustrate the MLMC extended by the meta-level, let assume 3 level MLMC with simulations on meshes of 546, 6772 and 87794 elements, and the ChebNet GCN meta-model. RF parameters: $\lambda = 0.1$, $\sigma = 1$.

Figure 5 shows the variance decrease from the coarsest level to the finest level. Levels are here defined by a simulation step h. The smaller the h, the finer the simulation. It can be observed that in our current MLMC implementation, the decrease is steeper for lower moments. In general, the more the variance across levels decreases, the more effective the use of the MLMC. Thus, in our case, a higher R results in a less effective MLMC compared to a lower R. The number of moments affects the final N_l and consequently the total computational cost C.

Table 6 provides the ratio between the MLMC$_{\text{meta}}$ total cost C_{meta} and the MLMC total cost C. To measure a quality of a PDF approximation, KL divergence $KL(\rho^{25}_{ref}||\rho^{R}_{MLMC_{meta}})$ is added to the table, where ρ^{25}_{ref} denotes a PDF based on 25 moments estimated by the MC$_{\text{ref}}$, $\rho^{R}_{MLMC_{meta}}$ is a PDF approximated from moments estimated by the MLMC$_{\text{meta}}$. It can be seen that the computational cost savings are greatest for the smallest number of moments. However, $R = 2$ is insufficient to obtain a decent PDF by the MEM. In our case, we need at least $R = 5$. On the other hand, with V_t >= 1e−5 and $R > 30$, the moments estimation error causes the PDF to contain obvious ripples. Using $5 \leq R \leq 30$, we still achieve savings in the computational cost of at least 10%. For you to get an idea, for $R = 25$, V_t = 1e−6, the absolute computational costs in seconds are as follows: $C_{meta} \approx 73266$ and $C \approx 84603$.

The effect of R is also manifested in the case of the MC extended by the meta-level. In both cases, it can be noted that cost savings are almost constant for $R \geq 15$.

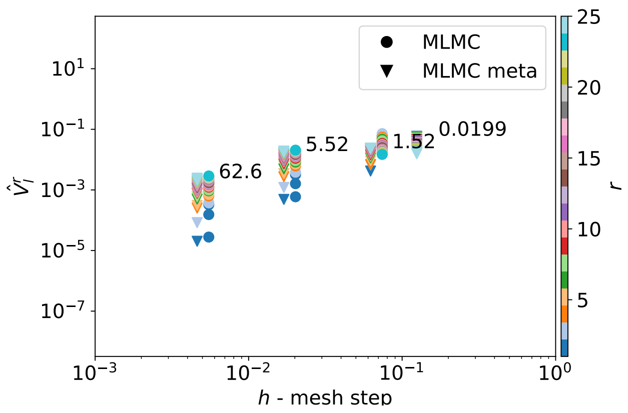

Figure 5: Moment variances across MLMC levels. The level is identified by its simulation mesh step h, added number represents the MLMC$_{\text{meta}}$ C_l

Table 6: Computational cost of MLMC$_{\text{meta}}$ to MLMC

R	C_{meta}/C	$KL(\rho^{25}_{ref}\|\|\rho^{R}_{MLMC_{meta}})$
2	0.41	0.522
5	0,64	0.00227
15	0,83	0.00338
25	0,87	0.00503
50	0,87	0.0112
75	0.85	0.0176

8 Conclusions

In this paper, we dealt with the meta-model design for the groundwater flow problem. The motivation was to incorporate a meta-model into the multilevel Monte Carlo method and achieve additional savings in the MLMC computational cost. After comparing the meta-models based on the ChebNet GCN, the GraphSage GCN, and the support vector regression, the ChebNet GCN was selected as the most suitable for our task. The proposed meta-modeling technique is effective for simulations on unstructured meshes with a maximum of ca. 1000 elements. Computational cost savings of up to 50% were achieved for both the MC and the MLMC extended by the meta-level. Due to the observed uneven decrease in variances of MLMC estimates across levels, the amount of computational cost savings depends on the number R of generalized statistical moments. In order to obtain a good PDF approximation by the MEM, we required $R \geq 5$. In these cases, we were still able to achieve at least 10% savings in computational costs.

Although the obtained results are auspicious, to provide more general conclusions, it is necessary to try our approach with a more complex simulation, which will be more challenging for the meta-model design.

Acknowledgement

This work was partly supported by the Student Grant Scheme at the Technical University of Liberec through project nr. SGS-2020-3068.
The project leading to this work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 847593.

References

[1] Fatma Abid, Khalil Dammak, Abdelkhalak El Hami, Tarek Merzouki, Hassen Trabelsi, Lassaad Walha, and Mohamed Haddar. Surrogate models for uncertainty analysis of micro-actuator. *Microsystem Technologies*, 26(8):2589–2600, 2020.

[2] Mariette Awad and Rahul Khanna. Support vector regression. In *Efficient Learning Machines*, pages 67–80. Apress, Berkeley, CA, 2015-04-27.

[3] Andrew R. Barron and Chyong-Hwa Sheu. Approximation of Density Functions by Sequences of Exponential Families. *The Annals of Statistics*, 19(3):1347–1369, September 1991.

[4] Claudio Bierig and Alexey Chernov. Approximation of probability density functions by the Multilevel Monte Carlo Maximum Entropy method. *Journal of Computational Physics*, 314:661–681, 2016.

[5] R Blaheta, M Béreš, and S Domesová. A study of stochastic FEM method for porous media flow problem. In *Applied Mathematics in Engineering and Reliability*, pages 281–289. CRC Press, 2016-04-13.

[6] Jan Březina, Jan Stebel, Pavel Exner, and Jan Hybš. Flow123d. `http://flow123d.github.com`, repository: `http://github.com/flow123d/flow123d`, 2011–2021.

[7] Jan Březina and Martin Špetlík. MLMC Python library. `http://github.com/GeoMop/MLMC`, 2021.

[8] C. Canuto. *Spectral methods in fluid dynamics*. Springer-Verlag, New York, 1988.

[9] Tomasz Danel, Przemysław Spurek, Jacek Tabor, Marek Śmieja, Łukasz Struski, Agnieszka Słowik, and Łukasz Maziarka. Spatial graph convolutional networks. In *Neural Information Processing*, pages 668–675, Cham, 2020. Springer International Publishing.

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[11] Michael B. Giles. Multilevel Monte Carlo methods. *Acta Numerica*, 24:259–328, May 2015.

[12] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[13] William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[15] C. Heaney, Yuling Li, O. Matar, and C. Pain. Applying convolutional neural networks to data on unstructured meshes with space-filling curves. *ArXiv*, abs/2011.14820, 2020.

[16] Falk Heße, Vladyslav Prykhodko, Steffen Schlüter, and Sabine Attinger. Generating random fields with a truncated power-law variogram: A comparison of several numerical methods. *Environmental Modelling & Software*, 55:32–48, May 2014.

[17] E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4):620–630, 1957.

[18] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[19] J. Kleijnen. Kriging: Methods and applications. *ERN: Computational Techniques (Topic)*, 2017.

[20] Shaoning Li and Luca Caracoglia. Surrogate model monte carlo simulation for stochastic flutter analysis of wind turbine blades. *Journal of Wind Engineering and Industrial Aerodynamics*, 188:43–60, 2019.

[21] Yi Ma, Jianye Hao, Yaodong Yang, Han Li, Junqi Jin, and Guangyong Chen. Spectral-based graph convolutional network for directed graphs, 2019.

[22] Julian Mack, Rossella Arcucci, Miguel Molina-Solana, and Yi-Ke Guo. Attention-based convolutional autoencoders for 3d-variational data assimilation. *Computer Methods in Applied Mechanics and Engineering*, 372, 2020.

[23] Sebastian Müller and Lennart Schüler. GSTools. `https://github.com/GeoStat-Framework/GSTools`, 2019.

[24] Karl T. Otness. Graph convolutions and machine learning. Thesis or dissertation, Harvard University, 2018.

[25] Imry Rosenbaum and Jeremy Staum. Multilevel Monte Carlo metamodeling. *Operations Research*, 65(4):1062–1077, 2017.

[26] Cosmin Safta, Richard L.-Y. Chen, Habib N. Najm, Ali Pinar, and Jean-Paul Watson. Toward using surrogates to accelerate solution of stochastic electricity grid operations problems. In *2014 North American Power Symposium (NAPS)*, pages 1–6, 2014.

[27] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[28] R.E. Stern, J. Song, and D.B. Work. Accelerated monte carlo system reliability analysis through machine-learning-based surrogate models of network connectivity. *Reliability Engineering System Safety*, 164:1–9, 2017.

[29] Ze Zhou Wang, Changlin Xiao, Siang Huat Goh, and Min-Xuan Deng. Metamodel-based reliability analysis in spatially variable soils using convolutional neural networks. *Journal of Geotechnical and Geoenvironmental Engineering*, 147(3), 2021.

[30] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[31] Mengfei Xu, S. Song, Xuxiang Sun, and Weiwei Zhang. Ucnn: A convolutional strategy on unstructured mesh. *ArXiv*, abs/2101.05207, 2021.

[32] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2020. https://d2l.ai.

[33] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks. *Computational Social Networks*, 6(1), 2019.

[34] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[35] L. Zhou, G. Yan, and J. Ou. Response surface method based on radial basis functions for modeling large-scale structures in model updating. *Comput. Aided Civ. Infrastructure Eng.*, 28:210–226, 2013.