Generation of Error Indicators for Partial Differential Equations by Machine Learning Methods

Alexey Muzalevskiy, Pekka Neittaanmäki, and Sergey Repin

Abstract Computer simulation methods for models based on partial differential equations usually apply adaptive strategies that generate sequences of approximations for consequently refined meshes. In this process, error indicators play a crucial role because a new (refined) mesh is created by analysis of an approximate solution computed for the previous (coarser) mesh. Different error indicators exploit various analytical and heuristic arguments. The main goal of this paper is to show that effective indicators of approximation errors can be created by machine learning methods and presented by relatively simple networks. We use the "supervised learning" conception where sequences of teaching examples are constructed due to earlier developed tools of a posteriori error analysis known as "functional type error majorants". Insensitivity to specific features of approximations is an important property of error majorants, which allows us to generate arbitrarily long series of diverse training examples without restrictions on the type of approximate solutions. These new (network) error indicators are compared with known indicators. The results show that after a proper machine learning procedure we obtain a network with the same (or even better) quality of error indication level as the most efficient indicators used in classical computer simulation methods. The final trained network is approximately as effective as the gradient averaging error indicator, but has an important advantage because it is valid for a much wider set of approximate solutions.

Key words: mesh adaptive methods, machine learning, network error indicator

Alexey Muzalevskiy
Peter the Great St. Petersburg Polytechnic University, 195251, St. Petersburg, Polytechnicheskaya, 29, Russia

Pekka Neittaanmäki · Sergey Repin (✉)
University of Jyväskylä, Faculty of Information Technology, P.O. Box 35, FI-40014 University of Jyväskylä, Finland, e-mail: pekka.neittaanmaki@jyu.fi, sergey.repin@jyu.fi

1 Introduction

Machine learning methods offer new possibilities for analysis of various mathematical and technical models. In particular, they are used in scientific computing and mathematical modelling as solvers (or parts of solvers) trained to solve particular classes of problems (e.g., see Sirignano and Spiliopoulos [25], Tsui et al. [26], Wang et al. [30] and references cited in these publications). In Iqbal and Carey [14], Manevitz et al. [18], these methods were applied for a closely related problem: generation of meshes for approximate solutions.

This paper is concerned with another important problem in numerical analysis of PDEs that arises in the process of mesh adaptive computations. It is the *error indication problem*. Known approaches to error indication are briefly discussed in Sect. 2, where we present the main known groups of indicators and introduce two characteristics of their accuracy (strong and weak). They are later used to compare the performance of different error indicators. The main goal of the paper is to introduce and study error indicators of a new type obtained by methods of supervised machine learning. We call them *Network Error Indicators*.

At first glance, the problem of creating a networkable to make successful error indication in a real life computational problem seems to be extremely difficult because the set of geometrical and numerical parameters (which are associated with a mesh and numerical solution) may be very large. Moreover, in different examples this set may be quite different, so that a network well trained for one particular problem could be useless for others. The idea by which we manage to overcome these difficulties and make the training geometry-agnostic is presented in Sect. 3. Theoretically, it is based on the decomposition principle combined with special analytical tools (a posteriori estimates of the functional type) that provide guaranteed error bounds for a wide set of approximations. These estimates have the form of integral functionals, which admit decomposition into local quantities associated with subdomains. The network is trained for a small mesh fragment (patch) and, therefore, the corresponding set of input data have a relatively small dimension. The output set consists of certain parameters to be defined (in the considered examples, the parameters are defined as normal components of fluxes on the element boundaries). If these parameters are properly defined, then the error majorant is close to the true error and, moreover, it generates a very good indicator of local errors (a systematic consideration of these questions is presented in Repin [23], Mali et al. [17]). In general, the parameters are unknown and finding them may requirer essential computational efforts. However, the range of additional efforts essentially depends on the goal of error estimation. It may be high if we wish to find guaranteed and very sharp error bounds for an an arbitrary approximation (which does not possess special properties such as Galerkin orthogonality or local conservation). In this case, finding exact parameters may be computationally expensive. But for making error indication with the quality sufficient for practical applications, the exact values of parameters are not very important. In numerous experiments (see Mali et al. [17]), it was confirmed that the method provides good error indication if computed parameters are only "close" to the exact ones. This fact suggest the idea to use specially trained networks as fast solvers that provide suitable values of the parameters. In this case, such a network can be viewed as certain *reduced order model* formed by the machine learning technology. Considering the mesh cells (patches) iteratively we improve the global set of output parameters and finally obtain the parameters that imply an effective indicator of local errors and also a guaranteed bound of the overall error (see Sect. 3.1).

To be computationally successful an error indicator must possess a collection of features. It must be fast and sufficiently accurate. At the same time, it must be insensitive (robust) with respect to properties of meshes and approximations. Network error indicators discussed in Sect. 3.1 satisfy these requirements because they are developed for small (basic) patches. Therefore, they are rather simple and easily adaptable to changing geometry of mesh cells. They are robust because originate from a posteriori estimates of the functional type, which are valid for any conforming approximation. Numerical examples presented in Sect. 4 show that network error indicators are indeed effective. They have the same quality as the most efficient indicators typically used in computer simulation methods (in many cases network indicators perform even better). For Galerkin approximations of sufficiently regular exact solutions they provide almost the same indication as gradient averaging indicators. But network indicators has an important advantage. The use of them is not burdened by specific conditions (e.g., Galerkin orthogonality, regularity of meshes and exact solutions). Therefore, network indicators are method-robust and can be successfully used for analysis of approximations generated by different methods. Moreover, the authors believe that their effectivity (with respect to computational time and quality of error indication) can be multiply enlarged (notice that the method is perfectly adapted for parallelisation).

Training procedure is another essential component of the suggested computational technology. To make it fast and stable we use networks special structure, which separates the geometrical data and the parameters associated with numerical approximations (see Fig. 4). This structure allows us to determine characteristics (weights) of the network efficiently using teaching problems with minimal dimensionality. Experiments presented in Sect. 4 are performed using MATLAB computations with video-cards (GPU support). Certainly network indicators can be realised using other software (e.g., Python). We plan to present the corresponding results in subsequent publications.

2 Indicators of Computational Errors

2.1 Adaptive Numerical Methods and Error Indicators

In the vast majority of cases, modern computer simulation methods for models based on partial differential equations (PDEs) use mesh-adaptation and a posteriori error indicators (e.g., see Ainsworth and Oden [2], Babuška and Rheinboldt [3], Bangerth and Rannacher [6], Bank and Weiser [7], Eriksson et al. [12], Johnson and Szepessy [16], Johnson and Hansbo [15], Mali et al. [17], Verfürth [27]). The general scheme

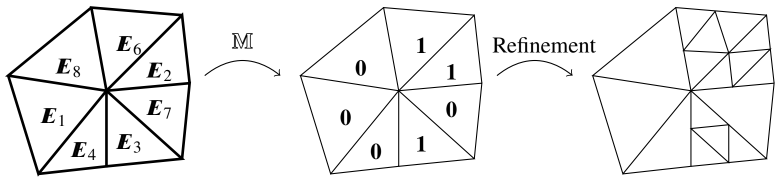

Fig. 1 Error indicator, marker, and mesh adaptation

is as follows. A continuous (differential) problem

$$Au = F, \quad u \in V, \ F \in V^*, \tag{1}$$

is replaced by a finite dimensional problem

$$A_h u_h = F_h, \quad u_h \in V_h \subset V, \tag{2}$$

where $A : V \longrightarrow V^*$ is a bounded operator, V is a Banach space, V^* is the space dual to V, F is a given function, V_h is a finite dimensional space ($\dim V_h = N(h) < +\infty$) used instead of V, and A_h and F_h are discrete counterparts of A and F, respectively. Proper selection of the subspace V_h is an important and difficult task. Even if types of approximations and mesh cells are defined, it may be difficult to detect a priori the best subspace V_h (in the sense of error minimization) among many others having the same dimensionality $N(h)$. Certainly, the choice of V_h also depends on the selected measure of the difference between u and u_h. Typically, it is the measure is generated by energy norm associated with the equation studied, but other (e.g., local, goal-oriented) measures can be used as well. Exact evaluation of local errors may be too complicated or even impossible. Therefore, many computer simulation codes (e.g., the PDE Toolbox of MATLAB) use special tools (called Error Indicators $\mathbb{E}$) that furnish information on the error e and control the generation of new (adapted) meshes.

In the mesh-adaptive computational strategies, the problem is firstly solved on the coarsest mesh (associated with the space V_1). The corresponding approximate solution u_1 is examined by an error indicator, which detects zones with exceptionally large errors. A new (refined) space V_2 has more degrees of freedom (nodes) in these regions and, therefore, produces a more accurate solution u_2. The error indicator it applied to u_2 and the next iteration is done with the space V_3. Schematically the process is depicted as follows:

$$V_1 \xrightarrow{\mathbb{E}(u_1)} V_2 \xrightarrow{\mathbb{E}(u_2)} \quad \ldots \quad V_k \xrightarrow{\mathbb{E}(u_k)} V_{k+1}. \tag{3}$$

An error indicator produces local quantities $\mathbb{E}_k$ associated with subdomains (mesh cells). A practically useful indicator must satisfy certain requirements, such as *computability* (it should be easily computable), *efficiency* (indicator correctly reproduces the distribution of local errors), and *robustness* (indicator is applicable to a wide set of approximations). Certainly some of these requirements may contradict others, e.g., in real life problems it is unlikely to have a computationally inexpensive error indicator that will be very efficient and robust simultaneously.

First error indication methods were suggested at the end of 19 century by C. Runge (a discussion of the heuristic Runge's rule can be found in Repin [23]). At present, the amount of publications related to this question is huge (see, e.g., Ainsworth and Oden [2], Babuška and Strouboulis [5], Bangerth and Rannacher [6], Repin [23], Wahlbin [28] and many references cited therein). The most known classes of error indicators widely used in adaptive computations are as follows:[1]

A. Residual method (Ainsworth and Oden [2], Babuška and Strouboulis [5], Babuška and Rheinboldt [3], Verfürth [27]),
B. Averaging (post-processing) methods (Babuška and Rheinboldt [3], Babuška and Rodriguez [4], Carstensen and Bartels [9], Ewing et al. [13], Wahlbin [28], Wang [29], Zienkiewicz and Zhu [32], Zio [33]),
C. Functional type error estimates and indicators (Repin [22, 23], Mali et al. [17]),
D. Hierarchical indicators (Agouzal [1], Bank and Weiser [7], Duran and Rodriguez [11]),
E. Dual-weighted residual method (Bangerth and Rannacher [6], Johnson and Szepessy [16], Rannacher [21]),
F. Goal oriented indicators (Bangerth and Rannacher [6], Eriksson et al. [12], Johnson and Hansbo [15]).

First residual type error indicators were suggested in Babuška and Rheinboldt [3] and some other publications of the authors. After that, this class of indicators was under a very intensive investigation. Indicators of this type exploit the Galerkin orthogonality property (hence they are applicable to Galerkin approximations only) and special type interpolation estimates associated with local patches (unions of neighbouring elements).

Another widely used group of error indicators exploits various post-processing procedures to recover computed gradients (or functions). One of the first publications in this direction is Zienkiewicz and Zhu [32]. It generated a large interest to error indication methods based on post-processing of numerical solutions. Nowadays indicators of this type are widely used. Very often they produce quite good indication of approximation errors for successful mesh adaptive procedures (e.g., see a systematic analysis in Carstensen and Bartels [9] and numerous subsequent publications of same authors). Mathematical justifications of the most used variant of this approach (gradient recovery methods) follow from the so-called *superconvergence* phenomenon (see, e.g., Ewing et al. [13], rížek and Neittaanmäki [24], Oganesyan and Rukhovets [20], Wahlbin [28]). However, in general, these error indication methods are also applicable to Galerkin approximations (or to approximations very close to them). Indicators of the third group follow from a posteriori estimates of the functional type (see Repin [22], Neittaanmäki and Repin [19], Repin [23], Mali et al. [17] and many publications cited therein). We discuss them later in Sect. 2.3.

[1] There exist many publications associated with each of the items (A)–(F). In brackets we mention several well-known representatives. Using them, the reader may find many others related to the same class.

Hierarchically based methods generate error indicators with the help of auxiliary problems constructed for enriched finite dimensional subspaces (local or global). These methods can be viewed as further development of the idea suggested by C. Runge. Among recent publications related to this method we mention Buffa and Garau [8], Carstensen et al. [10], Yu et al. [31]. In these publications, the reader will find many other references related to the method (see also Agouzal [1], Bank and Weiser [7], Duran and Rodriguez [11] and the references therein). Error indicators using adjoint problems are often applied if error control is performed it terms of specially constructed (goal oriented) quantities. These methods are very popular in engineering applications.

Dual-weighted residual (DWR) and goal-oriented methods use the *adjoint* boundary value problem. This method is free from the difficulties related to interpolation constants (that arise in the explicit residual type method). Another attractive feature is that it offers a way to obtain computable error indicators for specially constructed error functionals focused on most interesting (important) features of a numerical solution (see a systematic exposition in Bangerth and Rannacher [6], Johnson and Szepessy [16] and many other publications cited therein).

2.2 Accuracy of Error Indicators

We begin with a concise overview of the basic terminology and notion important for subsequent sections. Here we follow the lines of the book Mali et al. [17, Chapter 2], where the reader can find a systematic consideration of the error indication theory for approximations of differential equations.

There are many different error indicators that estimate the computational error

$$e_h(x) := u(x) - u_h(x).$$

They analyse not only $e_h(x)$ but also other quantities formed by e_h (e.g., various norms of this function), which we denote by one common symbol $\mathcal{E}(e_h)$. (They could be the L^2 norm of derivatives $\|\nabla e_h(x)\|_{2,\Omega}$ or L^q norm of the error $\|e_h(x)\|_{q.\Omega}$, mean values $\{\mid e_h \mid\}_\omega := \frac{1}{|\omega|}\int_\omega e_h dx$, maximal values $\max_{x\in\Omega} e_h(x)$, etc.) Depending on the goal of computer simulation, a particular quantity $\mathcal{E}(e_h)$ is selected to measure the quality of approximations and a series of consequent mesh adaptations are performed to minimize it.

Error Indicator $\mathbb{E}(x)$ is aimed to adequately represent the error quantity. This function must be computable and satisfy the relation

$$\mathbb{E}(x) \sim \mathcal{E}(e_h). \tag{4}$$

Different error indicators use different meanings of the equivalence symbol $\sim$. However, almost all of them can be collected into two main groups treating this symbol in a weak and strong sense, respectively.

To explain the difference, we need to define one other notion always used in mesh adaptive technologies. Let $B(\mathcal{T}_h)$ denote the set of boolean functions defined on partition (mesh) $\mathcal{T}_h$. We assume that $\mathcal{T}_h$ consists of polygonal cells T_h (e.g., simplexes), whose diameters are bounded from above and below by h with some multipliers independent of h and $\text{Card}(\mathcal{T}_h) = n$.

A marking operator (*marker*) M maps real valued functions to boolean valued ones, i.e., $M : \mathbb{E} \to B(\mathcal{T}_h)$. For any $\mathfrak{b} \in B(\mathcal{T}_h)$, $\mathfrak{b} = (\mathfrak{b}_1, \mathfrak{b}_2, ..., \mathfrak{b}_n)$, the norm is defined by the relation $\|\mathfrak{b}\|_B := \sum_i \mathfrak{b}_i$.

Definition 1 $\mathbb{E}$ is a weak (boolean) ϵ-approximation of the quantity $\mathcal{E}(e_h)$ if

$$\frac{1}{n}\|M(\mathbb{E}) \iff M(\mathcal{E}(e_h))\|_B \leq \epsilon. \tag{5}$$

Here $\iff$ denotes the logical equality, ϵ is a small positive number (accuracy of error indication), and M is the marker used in the process of transferring real valued data to the boolean valued data.

In this approximation concept, it is not required that the indicator $\mathbb{E}$ reproduces true values of local errors. The goal of this *weak* error indicator is to correctly detect the subdomains, where the errors are sufficiently larger than on others. In principle, an indicator may provide a weak approximation with one marker and do not provide it with another one. Usually the indicators are used with the so-called "greedy marking algorithm", which we also use in the examples. Typically mesh adaptation procedures used in various codes are based on error indicators, which are correct in a weak sense only. They do not provide a reliable information on the real quality of a numerical solution.

The *strong* error indication method treats the symbol $\sim$ in a different sense.

Definition 2 $\mathbb{E}$ is a strong ϵ-approximation of $\mathcal{E}$ if

$$\int_\Omega |\mathbb{E} - \mathcal{E}(e_h)| dx \;\leq\; \epsilon. \tag{6}$$

If $\mathbb{E}$ is correct in this (strong) sense, then

$$\int_\Omega \mathcal{E}(e_h) dx \quad \text{is close to} \quad \int_\Omega \mathbb{E}\, dx,$$

i.e., $\int_\Omega \mathbb{E}\, dx$ provides a correct quantitative estimation of the overall error. Moreover, if ϵ is small, then the Lebesgue measure of the set

$$\omega_\epsilon := \{x \in \Omega \mid |\mathbb{E}(x) - \mathcal{E}(e_h)(x)| \geq \epsilon\}$$

is small. This means, that the indicator $\mathbb{E}$ and the real error $\mathcal{E}(e_h)$ may be essentially different only in some small parts of the domain and, therefore, the computed indicator $\mathbb{E}$ provides a correct view on the distribution of local errors. It is clear that any indicator providing ϵ-approximation in the strong sense will be also coorect in the weak sense (this fact directly follows from (5) and (6)). However, the opposite statement is not true. Getting error indicators of this (strong) type may be a difficult task (especially for nonlinear problems and problems with complicated geometry). Therefore, computer codes using adaptivity are mainly supplied with indicators that satisfy Definition 1, but may not satisfy Definition 2.

2.3 A Class of Boundary Value Problems

To discuss error indication methods we consider a diffusion type boundary value problem, which often serves as the first test model to be studied if different error indication methods are compared with each other. The problem is to find u such that

$$\operatorname{div} A\nabla u + f = 0 \quad \text{in } \Omega, \qquad u = u_0 \quad \text{on } \Gamma, \tag{7}$$

where $\Omega \in \mathbb{R}^d$ is a bounded connected domain with Lipschitz continuous boundary Γ, $u_0 \in H^1(\Omega)$, $f \in L^2(\Omega)$, and $A = \{a_{ij}\}$ is a symmetric matrix with bounded coefficients, which satisfies the condition

$$c_1^2|\xi|^2 \leq A\xi \cdot \xi \leq c_2^2|\xi|^2, \quad c_2 \geq c_1 > 0, \quad \forall \xi \in \mathbb{R}^d.$$

Let v be a conforming approximation of u. Henceforth, we focus attention on the following three error indicators, which are often used in computer simulation and mesh adaptive numerical methods for partial differential equations.

Patch Averaging of Fluxes
In this method, the numerical flux $p_h := A\nabla u_h$ is computed for each element $T_h \in \mathcal{T}_h$. The averaging operator G_h averages these values on each patch of elements having a common node. Then, the corresponding nodal values are extended to Ω using standard (e.g., piecewise affine) approximations. There are many other variants of this method, which in the literature is often called *gradient averaging*. The function $\mathbb{E}_{\mathrm{GA}} = p_h - G_h p_h$ computed on each element T_h generates an error indicator, which often serves as an efficient representer of the actual error (see the literature cited in the item B).

Residual Based Error Indicator
Indicators of this type use specially weighted sums of equation residuals (i.e., $\| \operatorname{div} A\nabla u_h + f\|_{T_h}$) defined on elements of $\mathcal{T}_h$ and jumps of normal fluxes computed on interelement boundaries. A systematic exposition of this approach can be found in the monographs Ainsworth and Oden [2], Babuška and Strouboulis [5], Verfürth [27]. Henceforth, this indicator is denoted $\mathbb{E}_{\mathrm{Resid}}$.

Error Indicator Generated by A Posteriori Estimates of the Functional Type
It is known (see Repin [22, 23] and references to some other publications cited therein) that

$$\|\nabla e\|_A^2 = \inf_{y \in H(\Omega, \operatorname{div})} \mathbb{M}_{\oplus}(v, y), \tag{8}$$

where

$$e := u - v, \quad \|\nabla e\|_A^2 := \int_\Omega A\nabla e \cdot \nabla e \, dx,$$

and the functional $\mathbb{M}_\oplus$ (error majorant) is defined by the relation

$$\mathbb{M}_\oplus(v, y) := \int_\Omega \Big(A\nabla v \cdot \nabla v + A^{-1} y \cdot y - 2y \cdot \nabla v \Big) \, dx + C_{F\Omega}^2 \|f + \operatorname{div} y\|^2. \quad (9)$$

Here $H(\Omega, \operatorname{div})$ is the Hilbert space of vector valued functions in $L^2(\Omega, \mathbb{R}^d)$ with the divergence in $L^2(\Omega)$, $\|\cdot\|$ stands for the norm of $L^2(\Omega)$, and $C_{F\Omega}$ is a constant in the Friedrichs type inequality for the functions vanishing on $\partial\Omega$.

It is easy to see that infimum in (8) is attained if $y = p := A\nabla u$. In this case, the second term of the majorant vanishes and the integrand of the first one coincides with the exact squared error:

$$e^2(x) := A\nabla e \cdot \nabla e(x).$$

This means that a *vector-valued function* y_τ found by minimisation of the error majorant $\mathbb{M}_\oplus(v, y)$ on a certain finite-dimensional space Y_τ implies an *efficient error indicator*

$$\boldsymbol{E}_{\mathrm{Maj}}(v, y_\tau) := A\nabla v \cdot \nabla v + A^{-1} y_\tau \cdot y_\tau - 2y_\tau \cdot \nabla v. \quad (10)$$

Since $e^2 - \boldsymbol{E}_{\mathrm{Maj}}(v, y_\tau) = A^{-1} p \cdot p - A^{-1} y_\tau \cdot y_\tau + 2(y_\tau - p) \cdot \nabla v$, it is easy to see that the indicator $\boldsymbol{E}_{\mathrm{Maj}}$ is indeed accurate (in the sense of Definition 2) provided that y_τ is close to p. This property holds for any conforming approximation v and, therefore, the indicator $\boldsymbol{E}_{\mathrm{Maj}}$ is robust with respect to the approximation type.

Error indicators of this type were tested for various elliptic and parabolic problems and have confirmed high robustness and efficiency (see Mali et al. [17]). At the same time, these indicators are more expensive then indicators of the group (B). Below we use $\boldsymbol{E}_{\mathrm{Maj}}$ for two purposes. The first (and the main) application is to create an *arbitrary large amount of diverse teaching examples* for learning a network indicator $\boldsymbol{E}_{\mathrm{ML}}$.

3 Network Error Indicator

Now we discuss a way to generate an error indicator $\boldsymbol{E}_{\mathrm{ML}}$ by the machine learning technology. It is worth noting, that the idea to use ML methods for error indication problems is quite natural. This technology is known to be very successful in analysis of images and error indication can be treated as a special kind of the Image Recognition (IR) problem. Indeed, the problem consists of getting an adequate image of the error distribution by analysing the geometrical and computational data encompassed in the approximate solution u_h, i.e.,

We wish to create a network able to analyse u_h and generate a set of boolean data that show elements to be refined. Moreover, such a network should be able to make

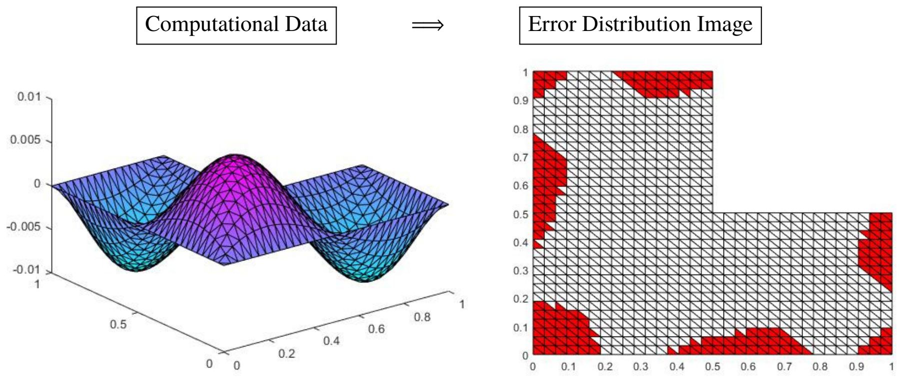

Fig. 2 Approximate solution (left) and the corresponding image $B(\mathcal{T}_h)$ generated by an error indicator and a marker (right)

a conclusion on the quality of a numerical solution and give an adequate picture of the error distribution.

Let $D(u_h)$ denote the set of data related to u_h, $\mathcal{E}$ be the set of local errors defined on the elements of $\mathcal{T}_h$, and $B(\mathcal{T}_h)$ be the boolean set that defines marking of elements. An error indicator $\mathbb{E}$ is a part of the estimation-marking process

$$\boxed{D(u_h)} \to \mathbb{E} \to \boxed{\mathcal{E}} \longrightarrow \boxed{B(\mathcal{T}_h)}$$

Typically, the sets $D(u_h)$, $\mathcal{E}$, and $B(\mathcal{T}_h)$ are very large (they may contain millions of elements). It is practically impossible to generate an efficient network indicator $\mathbb{E}_{\mathrm{ML}}$ directly on the global level because the corresponding network would be too large and too complicated. Besides, such a network would be strongly connected with a particular mesh. To overcome these difficulties caused by high dimensionality, we use the *decomposition principle*, by which a complicated highly dimensional problem is reduced to a sequence of simple (low-dimensional) sub-problems. For this purpose, we use the principal identity (8). In accordance with (9), $\mathbb{M}_{\oplus}$ is presented by an integral. Therefore, the problem of finding y_τ can be localised and reduced to problems for small subdomains (patches). Networks presented in the next section are able to solve these local subproblems very fast. Applying them iteratively we obtain a suitable y_τ and the corresponding error indicator $\mathbb{E}_{\mathrm{ML}}$.

3.1 Local Networks

In what follows, we consider problem (7) with the unit matrix A and assume that Ω is a polygonal set covered by a simplicial partition $\mathcal{T}_h$. The set $D(u_h)$ contains the data of a numerical solution u_h (nodal values, geometrical parameters). The sets $D^k(u_h)$ are associated with the mesh fragments of two types. The type 1 is presented

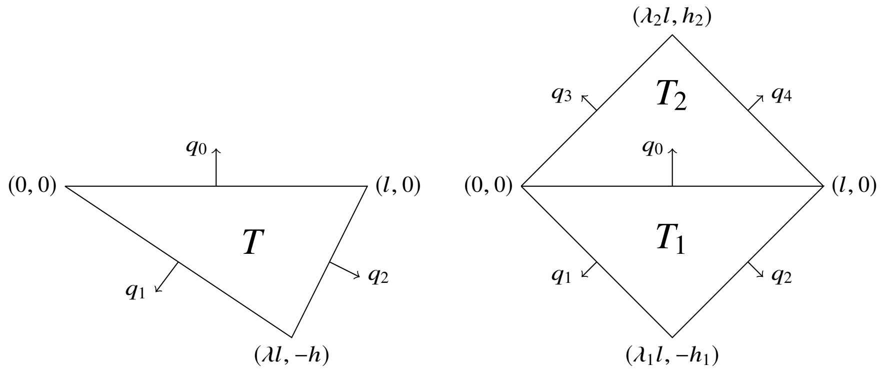

Fig. 3 Exterior simplex and two neighbouring interior simplexes.

by two neighbouring simplexes (it is used for the interior of Ω). The type 2 is used for elements having common edge with the boundary $\partial\Omega$ and uses only one simplex. Since u_h is known, we can find the corresponding flux q_h and define its normal components on edges of all simplexes. Our goal is to construct networks that make post-processing of q_h and to use it for getting an error indicator.

Consider first a mesh fragment of the second type. Without a loss of generality, we present it as the simplex with vertexes $(0,0)$, $(l,0)$, $(\lambda l,-h)$ (see Fig. 3, left). Notice, that by scaling and rotation any simplex can be represented in such a form.

Assume that the values of q_1 and q_2 are given. We find q_0 in the following form:

$$q_0 = \sum_{i=1}^{2} a_i q_i + a_3\{f\}_T + a_4 \frac{\partial u_h}{\partial x_1} + a_5 \frac{\partial u_h}{\partial x_2}, \tag{11}$$

where a_i are some coefficients defined by the networks. This representation is natural because the optimal value of q_0 (i.e., such that it minimises the corresponding contribution in the error majorant) should clearly depend on two other components q_1 and q_2, mean value of f, and components of ∇u_h. Coefficients a_i depend on geometrical parameters only. Certainly (11) is the simplest (linear) representation and one can also use more complicated forms with nonlinear terms. However, for our problem this simple representation provides sufficiently accurate results provided that the networks computing a_i are well taught. It is convenient to organise the process of machine learning by setting only one of the "main" parameters $q_1, q_2, \{f\}_T, \frac{\partial u_h}{\partial x_1}, \frac{\partial u_h}{\partial x_2}$ equal to 1 and setting all others zero. Then, the corresponding a_i is generated by a network, which input is geometrical data ζ_i (in our case they are presented by the parameters l, λ, h). Thus, the process follows the principle of *separate learning*, where the networks for a_i are learned independently, but within the framework of one and the same generator of teaching examples.

It is worth noting that neural networks are not adapted to the multiplication of input data. Therefore, a proper selection of input parameters is an essential question. In our case, the parameters ζ_i were selected as follows:

$$\zeta_1 = \frac{h}{l}, \quad \zeta_2 = \frac{l}{h}, \quad \zeta_3 = \frac{\lambda l}{h}, \quad \zeta_4 = \frac{\lambda^2 l}{h}, \quad \zeta_5 = \frac{C_F^2}{|T|}.$$

We generate a teaching sequence of any desired length by means of the majorant (9), where the vector valued function y is defined by means of the Raviart-Thomas elements. We use the lowest order elements RT^0, for which the boundary normal fluxes q_0, q_1, and q_2 are uniquely define y inside T. The constant C_F entering the majorant is also considered as one of the input parameters. The majorant (9) yields the following local subproblem for the definition of q_0:

$$\min_{q_0,\, \beta>0} G(\mathbf{q}, \boldsymbol{\zeta}, \beta), \tag{12}$$

where $\mathbf{q} = \{q_1, q_2, q_3\}^T$ and $\boldsymbol{\zeta}$ is the above defined vector of geometrical parameters. The function G has the form

$$G(\mathbf{q}, \boldsymbol{\zeta}, \beta) := (1+\beta)\left[\frac{1}{2}\left(M + \frac{C_F^2}{\beta}K\right)\mathbf{q}\cdot\mathbf{q} + L_1\cdot\mathbf{q} + \frac{C_F^2}{\beta}L_2\cdot\mathbf{q}\right], \tag{13}$$

where

$$M := \begin{pmatrix} \frac{(1-3\lambda+3\lambda^2)l}{6h} + \frac{h}{2l} & \frac{(1-\lambda-\lambda^2)l}{6h} - \frac{h}{6l} & \frac{(-1+3\lambda-\lambda^2)l}{6h} - \frac{h}{6l} \\ \frac{(-1+3\lambda-\lambda^2)l}{6h} - \frac{h}{6l} & \frac{(1-3\lambda+3\lambda^2)l}{h} + \frac{h}{2l} & \frac{(1-\lambda-\lambda^2)l}{h} - \frac{h}{6l} \\ \frac{(1-\lambda-\lambda^2)l}{6h} - \frac{h}{6l} & \frac{(-1+3\lambda-\lambda^2)l}{h} - \frac{h}{6l} & \frac{(1-3\lambda+3\lambda^2)l}{h} + \frac{h}{2l} \end{pmatrix},$$

and $K \in \mathbb{M}^{3\times3}$ has the coefficients equal to $\frac{2}{|T|}$. The vectors L_1 and L_2 are defined by the relations

$$L_1 := \frac{l(1-2\lambda)}{3}\frac{\partial u_h}{\partial x_1}\begin{pmatrix}1\\1\\1\end{pmatrix} - \frac{2h}{3}\frac{\partial u_h}{\partial x_2}\begin{pmatrix}1\\1\\1\end{pmatrix} \quad \text{and} \quad L_2 := 2\{f\}_T\begin{pmatrix}1\\1\\1\end{pmatrix}.$$

For an interior edge of $\mathcal{T}_h$ (which length is l), we need to consider two neighbouring elements T_1 and T_2 (see Fig. 3, right). Their geometrical properties are defined by the parameters h_1, h_2, λ_1, and λ_2. By scaling and rotation any pair of nonintersecting neighbouring simplexes can be represented in such a form. In this case, we define q_0 by the relation analogous to (11)

$$q_0 = \sum_{i=1}^{4} a_i q_i + a_5\{f\}_{T_1} + a_6\varkappa_{11} + a_7\varkappa_{12} + a_8\{f\}_{T_2} + a_9\varkappa_{21} + a_{10}\varkappa_{22}, \tag{14}$$

where

$$\varkappa_{11} = \frac{\partial u_h}{\partial x_1}|_{T_1}, \quad \varkappa_{12} = \frac{\partial u_h}{\partial x_2}|_{T_1}, \quad \varkappa_{21} = \frac{\partial u_h}{\partial x_1}|_{T_2}, \quad \varkappa_{22} = \frac{\partial u_h}{\partial x_2}|_{T_2}.$$

Networks for the coefficients a_i are constructed by the same method using a generator of teaching examples, which gives the correct q_0 for any given q_1, q_2, q_3, q_4, geometrical parameters l, λ_1, h_1, T_1, λ_2, h_2, T_2, mean values on elements $\{f\}_{T_1}$, $\{f\}_{T_2}$, and values of the derivatives of u_h on triangles T_1 and T_2.

Analysing the majorant, we obtain the following subproblem to be used for generating correct values of q_0 (which are used in the teaching process). Now the main parameters vector is $\mathbf{q} = \{q_0, q_1, q_2, q_3, q_4\}^T$. Then the corresponding function G has a form similar to (13), where

$$M := \begin{pmatrix} m_1 + m_2 & m_3 & m_4 & -m_5 & -m_6 \\ m_4 & m_1 & m_3 & 0 & 0 \\ m_3 & m_4 & m_1 & 0 & 0 \\ 0 & 0 & -m_6 & -m_2 & -m_5 \\ 0 & 0 & -m_5 & -m_6 & -m_2 \end{pmatrix}, \quad K := \begin{pmatrix} k_1 + k_2 & k_1 & k_1 & -k_2 & -k_2 \\ k_1 & k_1 & k_1 & 0 & 0 \\ k_1 & k_1 & k_1 & 0 & 0 \\ 0 & 0 & -k_2 & -k_2 & -k_2 \\ 0 & 0 & -k_2 & -k_2 & -k_2 \end{pmatrix},$$

and the linear terms are defined by the relations

$$L_1 := \frac{l\varkappa_{11}}{3}\begin{pmatrix} -2\lambda_1 \\ 1 - 2\lambda_1 \\ 1 - 2\lambda_1 \\ 0 \\ 0 \end{pmatrix} - \frac{2h_1\varkappa_{12}}{3}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \frac{l\varkappa_{21}}{3}\begin{pmatrix} 2\lambda_2 \\ 0 \\ 0 \\ 2\lambda_2 - 1 \\ 2\lambda_2 - 1 \end{pmatrix} - \frac{2h_2\varkappa_{22}}{3}\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix},$$

and

$$L_2 := 2\{f\}_{T_1}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} - 2\{f\}_{T_2}\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

Here $k_1 = \frac{2}{|T_1|}$, $k_2 = \frac{2}{|T_2|}$, and

$$m_1 = \frac{(1 - 3\lambda_1 + 3\lambda_1^2)l}{6h_1} + \frac{h_1}{2l}, \qquad m_2 = \frac{(1 - 3\lambda_2 + 3\lambda_2^2)l}{6h_2} + \frac{h_2}{2l},$$

$$m_3 = \frac{(1 - \lambda_1 - \lambda_1^2)l}{6h_1} - \frac{h_1}{6l}, \qquad m_4 = \frac{(-1 + 3\lambda_1 - \lambda_1^2)l}{6h_1} - \frac{h_1}{6l},$$

$$m_5 = \frac{(1 - \lambda_2 - \lambda_2^2)l}{6h_2} - \frac{h_2}{6l}, \qquad m_6 = \frac{(-1 + 3\lambda_2 - \lambda_2^2)l}{6h_2} - \frac{h_2}{6l}.$$

We see that as in the previous case, the set of external data can be split into two principally different subsets: the set of main parameters and the set of geometrical data. In Fig. 4, we depict the principal architecture of the network, which reflects this decomposition. Here $\zeta_1, \ldots, \zeta_k$ are the geometrical data related to the selected simplex (or two neighbouring simplexes) and $g_1, \ldots, g_m$ are the data associated with the differential equation and numerical solution u_h.

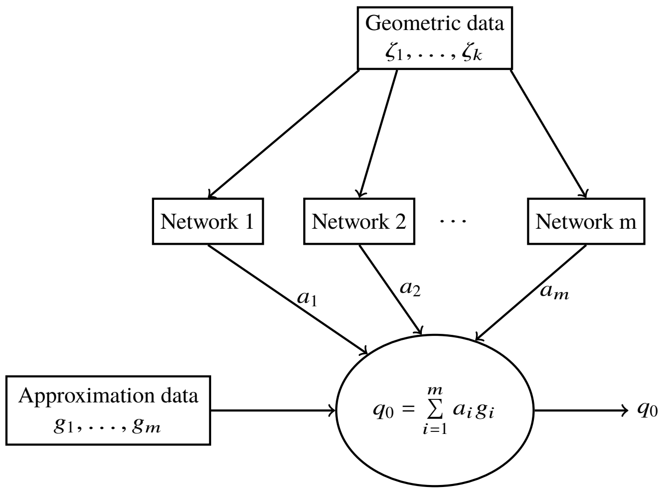

Fig. 4 Principal architecture of the global net. The data g_i is formed by the equation and approximate solution.

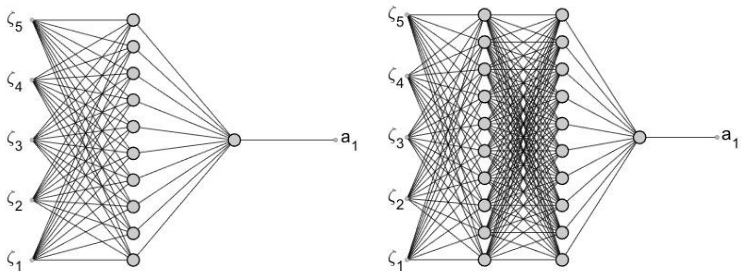

Fig. 5 Networks ML10 and ML10+10

Local networks constructed for a_i may have different levels of complexity[2]. Figures 5 and 6 show three different networks used in different tests for the coefficient a_1. Networks for other coefficients look analogously.

After constructing a network we verify its quality by a series of control tests of the size N (typically $N = 100000$) and define the quantity (accuracy index)

$$I_{\text{net}}(a_i) := \frac{1}{N} \sum_{k=1}^{N} \frac{|a_i^{(k),\text{net}} - a_i^{(k)}|}{|a_i^{(k)}|},$$

where $a_i^{(k)}$ is the exact value of the parameter a_i in the test i and $a_i^{(k),\text{net}}$ is the value generated by the network. The results generated by a network are considered as "good" if I_{net} is close to zero.

[2] Certainly complicated networks require larger amount of teaching examples, but in general provide better indication of errors.

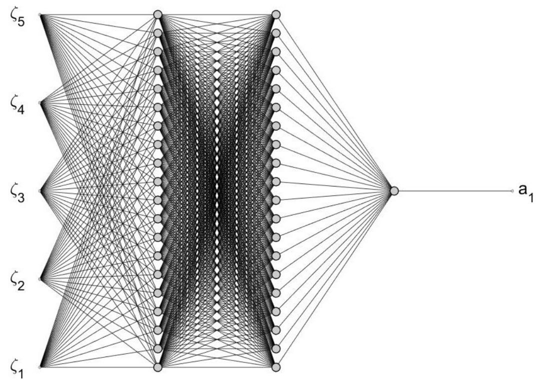

Fig. 6 Network ML20+20

Table 1 Index $I_{\rm net}$ for the networks calculating a_1, a_2, a_3, a_4

Net	$I_{\rm net}(a_1)$	$I_{\rm net}(a_2)$	$I_{\rm net}(a_3)$	$I_{\rm net}(a_4)$
8	2.1e-2	1.6e-2	2.0e-2	2.1e-2
10	4.1e-3	2.9e-3	7.5e-3	5.2e-3
20	1.7e-3	1.5e-3	8.3e-4	9.5e-4
10+10	2.1e-5	1.8e-5	2.1e-5	2.2e-5
20+20	2.1e-6	3.5e-6	3.1e-6	2.8e-6

Table 1 and Fig. 7 present these quantities for the networks created for the coefficients a_1, a_2, a_3, a_4 (numbers in the left column show the structure of the network, e.g., 8 is related to the simplest network consisting of only one layer with 8 neurones and 20 + 20 is the most complicated network with two layers having 20 neurones each). Since the coefficients represent similar mathematical structures, it is not surprising that the corresponding networks (constructed by different codes[3]) have similar accuracy. Table 1 and the histogram in Fig. 7 show that the networks become more efficient if the amount of neurones enlarges and one-layer networks are replaced by two-layer ones.

4 Examples

Here we present some of the results obtained in the process of verification of $\mathbb{E}_{\rm ML}$ and comparison with other indicators. In the tests, we computed and analysed the following indicators of elementwise errors:

1. indicator based on the true error distribution;
2. the gradient averaging indicator;

[3] Here we have used inbuilt MATLAB functions trainlm and trainbr.

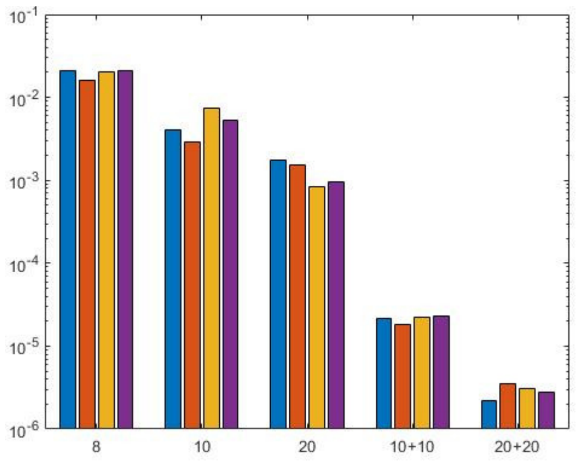

Fig. 7 Index I_{net} for the networks calculating a_1, a_2, a_3, a_4 on patches formed by two neighbouring simplexes; teaching examples are generated by the majorant.

3. by the residual indicator used in MATLAB;
4. indicator computed by minimisation of the majorant (9);
5. indicator computed by the network 8.
6. indicator computed by the network 10.
7. indicator computed by the network 10+10.
8. indicator computed by the network 20+20.

The results are presented in different forms. First, we depict them with the help of histograms (as, e.g., Fig. 9). The histogram uses special numeration of elements (from 1 to n), where the elements are numbered in accordance with values of the corresponding errors (from the smallest to the largest). Hence for the true error the very last peak represents the largest local error related to a certain element (which is now numbered n). The exact error indicator produces the picture in the left upper corner of Fig. 9. If the indicator is "good", then the picture should be similar (may be with some defects as on the histograms GA or Maj which, however, do not change the picture essentially). For a "bad" indicator, the two pictures are very different and the monotonicity property is lost completely. Using the histograms, we can see how accurate is an indicator in the strong sense (see Definition 2). Also, we need a quantitative qualification of the accuracy of error indicators. This is done by "efficiency indexes" presented in Table 3.

Another part of the results is related to marking. By the "greedy marker" M we select the elements to be refined (e.g., see Fig. 10 where these elements are coloured red). We compare the results produced by different indicators with the correct marking done by the same marker M applied to the exact error function. These pictures give a presentation on how accurate are the indicators in the weak sense (see Definition 1).

In the examples, we analyse approximations of the equation $\Delta u + f = 0$ with homogeneous Dirichlet boundary conditions in different domains Ω and for different source terms f.

Example 1 Let $\Omega = (0,1)^2 \setminus [1/2,1]^2$, and

$$f = (-12x_1 + 6)x_2(1 - x_2)(2x_2 - 1) + (-12x_2 + 6)x_1(1 - x_1)(2x_1 - 1).$$

The exact solution is $u = x_1(1 - x_1)(2x_1 - 1)x_2(1 - x_2)(2x_2 - 1)$ and u_h is computed on different finite element meshes with the amount of elements ($N_t = 384, 670, 1237, 2397$, see Table 4). Hence we know the exact error $e := u - u_h$ between the exact solution and Galerkin approximations. We compare it with the results produced by the indicators (2)–(8). Figure 9 shows that the indicator based on the error majorant (9) is the most accurate. Error indicator based on gradient averaging works with practically the same accuracy and the indicator ML20+20 is very close to these two. The indicators ML10 and ML10+10 produce more coarse results, which however can be viewed as "acceptable". The residual based indicator of MATLAB does not generate acceptable in the sense of Definition 2. Table 4 shows the results of mesh adaption based on different error indicators. The first line corresponds to the adaptation process using exact error. The corresponding finite element meshes consist of 384, 670, 1237, and 2397 elements and on the finest mesh the energy error norm is equal to 3.045×10^{-3}. If the adaptation process uses gradient averaging indicator and the indicator generated by the majorant then the results are very close.

All network indicators work with the same efficiency as the exact one and produce meshes of approximately same size with very close values of the error. Residual based indicator of MATLAB is much lesser efficient. It generates the final mesh with twice larger amount of elements and practically the same accuracy as other indicators have achieved using much smaller meshes. Figure 8 depicts the situation graphically. We see that in this (rather simple) example all network indicators work with the same efficiency as the exact one. Figure 10 presents the corresponding markings of elements that was done by means of the "greedy marking" algorithm with the selection of elements forming 50 % of the total error. Comparison of different indicators in other test examples lead to similar conclusions.

When discussing the efficiency of error indicators we should take into account the computational cost of an indicator measured in terms of the computation (CPU) time. Table 2, compares different indicators from this position. Evidently the CPU time may be essentially different depending on a particular computer so that making measurements in terms of real time units does not have sense. Therefore, we accept the time used by the GA indicator as the time unit and expressed working times of other indicators in these units.

We see that the proportion does not depend on the amount of elements. Network indicators are as fast as the GA indicator (which is known to be one of the most easily computable). It is not surprising that the computation of Maj indicator is more expensive. This indicator is applicable to all conforming approximations and is robust with respect to violations of Galerkin orthogonality and other additional

Table 2 CPU time used by different indicators

N_t	GA	ML10	ML10+10	ML20+20	Maj
384	1	1.1	1.1	1.1	2.8
1536	1	1.1	1.1	1.1	3.1
6144	1	1.1	1.1	1.1	2.9

Table 3 Example 1: Global efficiency indexes for Galerkin and not Galerkin approximations for various error indicators

Method	Maj	GA	Res	ML8	ML10	ML10+10	ML20+20
Non-Galerkin	1.46	4.10	21.50	2.35	1.89	1.69	1.70
Galerkin	2.09	7.20	49.63	2.74	2.71	2.76	2.70

conditions (e.g., regularity of meshes and exact solutions). Moreover, it generates guaranteed error bounds. Certainly, all these additional properties are based on more sophisticated analysis, which requires more time. Table 2 is related to the first example. The results of time measurements for other examples are analogous.

Also we studied not Galerkin approximations. It is worth noting that in real life computations approximate solutions often do not satisfy the Galerkin condition and, therefore, robustness of an error indicator with respect to violations of this condition is practically important. In our tests, we used certain disturbances of u_h and defined the approximation as $v = u_h + 0.005\pi_h(\sin(2\pi x_1)\sin(2\pi x_2))$. For a finite element mesh with $N_t = 384$ similar simplexes, we estimate the errors (Fig. 11) and the marking (Fig. 12).

It is not surprising that the gradient averaging method produces rather coarse estimation (it is valid only for approximations of regular exact solutions close to Galerkin ones). The same is true for the residual error indicator. Network error indicators work quite successfully. This fact is seen on the pictures. Also, it is confirmed by the data presented in Tables 3 and 4.

It is commonly accepted to compare the efficiency of an error estimator by means of the so-called *efficiency index*

$$I_{\text{eff}} := \frac{\text{Error estimate}}{\text{Error}}.$$

In Table 3, we present the values of I_{eff} for different error indicators applied to the above mentioned approximation v. Error estimator for the residual method was computed by the inbuilt procedure of MATLAB. Table 3 shows the evolution of mesh size in the process of adaptation for different error indicators. The first line is related to the case, where the exact error distribution was used for making and mesh refinement. These results could serve as an "etalon". We see that on the last step this (optimal) indicator has generated a mesh with 2397 nodes and the corresponding approximate solution has the error (in the energy norm) 3.045×10^{-3}. Similar accuracy was achieved by all other indicators, but with rather different meshes. If

Table 4 Example 1: Mesh adaptation with different indicators and respective errors. N_t is the amount of elements.

	step 0	step 1	step 2	step 3
N_t	384	670	1237	2397
$\|\|\nabla e\|\|^{(\text{Exact})}$	8.090e-03	5.804e-03	4.133e-03	3.045e-03
N_t	384	704	1358	2544
$\|\|\nabla e\|\|^{(\text{GA})}$	8.090e-03	5.690e-03	3.976e-03	2.905e-03
N_t	384	880	2074	4388
$\|\|\nabla e\|\|^{(\text{Res})}$	8.090e-03	5.412e-03	3.370e-03	2.331e-03
N_t	384	685	1355	2505
$\|\|\nabla e\|\|^{(\text{Maj})}$	8.090e-03	5.722e-03	4.012e-03	3.045e-03
N_t	384	714	1420	2618
$\|\|\nabla e\|\|^{(\text{ML8})}$	8.090e-03	5.521e-03	3.857e-03	2.885e-03
N_t	384	724	1439	2397
$\|\|\nabla e\|\|^{(\text{ML10})}$	8.090e-03	5.501e-03	3.882e-03	2.881e-03
N_t	384	719	1426	2622
$\|\|\nabla e\|\|^{(\text{ML10+10})}$	8.090e-03	5.519e-03	3.871e-03	3.045e-03
N_t	384	719	1440	2622
$\|\|\nabla e\|\|^{(\text{ML20+20})}$	8.090e-03	5.519e-03	3.853e-03	2.882e-03

we compute the coefficient $\kappa = \frac{\text{error}}{N_t}$ (it shows how efficiently we use degrees of freedom), then the best result is achieved for ML20+20: $\kappa \approx 0.11 \times 10^{-5}$, all others give $\kappa \approx 0.12 \times 10^{-5}$ except the residual indicator with $\kappa \approx 0.5 \times 10^{-6}$.

Example 2 In this example, we consider a Π-shaped domain

$$\Omega = (-1,1) \times (0,1) \setminus ([-0.5,0.5] \times [0,0.5])$$

and $f = -(12x_1^2 - 2.5)(x_2^3 - 1.5x_2^2 + 0.5x_2) - (x_1^4 - 1.25x_1^2 + 0.25)(6x_2 - 3)$. The exact solution is $u = (x_1 - 1)(x_1 + 1)(x_1 - 0.5)(x_1 + 0.5)x_2(x_2 - 1)(x_2 - 0.5)$.

The function compared with u is the Galerkin approximation u_h computed for the space of piecewise affine functions defined on a simplicial mesh. The corresponding results are exposed in Table 5. As in the previous example, the histogram in Fig. 14 gives a presentation on how accurate the indicators are in the strong sense and Fig. 15 depicts results of element marking done by the "greedy marking" algorithm with the selection of 50% of elements.

Again all network indicators produce markings almost the same as the marking produced by the exact error indicator. Figure 13 shows the meshes obtained on the last step of adaptation by three different error indicators. Two of them (produced by the GA indicator and the network indicator ML10) are quite similar. The mesh generated by the MATLAB residual indicator is quite different and contains much larger amount of elements.

Table 5 Example 2: Mesh adaptation with different indicators and respective errors. N_t is the amount of elements.

	step 0	step 1	step 2	step 3
N_t	768	1298	2416	4587
$\|\|\nabla e\|\|^{(\mathrm{Exact})}$	8.542e-03	6.211e-03	4.507e-03	3.310e-03
N_t	768	1396	2673	4916
$\|\|\nabla e\|\|^{(\mathrm{GA})}$	8.542e-03	5.977e-03	4.236e-03	3.105e-03
N_t	768	1686	3986	8538
$\|\|\nabla e\|\|^{(\mathrm{Res})}$	8.542e-03	5.757e-03	3.577e-03	2.511e-03
N_t	768	1287	2481	4634
$\|\|\nabla e\|\|^{(\mathrm{Maj})}$	8.542e-03	6.264e-03	4.493e-03	3.307e-03
N_t	768	1388	2731	5064
$\|\|\nabla e\|\|^{(\mathrm{ML8})}$	8.542e-03	5.956e-03	4.198e-03	3.1248e-03
N_t	768	1412	2807	5198
$\|\|\nabla e\|\|^{(\mathrm{ML10})}$	8.542e-03	5.921e-03	4.110e-03	3.076e-03
N_t	768	1385	2714	5064
$\|\|\nabla e\|\|^{(\mathrm{ML10+10})}$	8.542e-03	5.964e-03	4.159e-03	3.083e-03
N_t	768	1384	2733	5088
$\|\|\nabla e\|\|^{(\mathrm{ML20+20})}$	8.542e-03	5.955e-03	4.167e-03	3.082e-03

Example 3 Here we consider the unit square $\Omega = (0,1)^2$ and jumping right-hand side

$$f = \begin{cases} 1 & \text{if } x_1 < 0.5, \\ 2 & \text{overwise.} \end{cases}$$

Since the exact solution is unknown, we used instead the so-called "reference" solution computed on a very fine mesh with 299313 nodes.

Table 6 exposes the results and Fig. 16 depicts the meshes generated by different error indicators. Here the results are not so good, but nevertheless networks ML10+10 and ML20+20 produce final meshes with 921 and 931 elements and the error 1.4×10^{-2} what is comparable with the result obtained by the exact error indicator (646 elements and the error 1.8×10^{-2}). Figure 16 present the meshes generated by different indicators. Here again we used the "greedy marking" algorithm with the selection of 50% of elements. For Galerkin solutions network indicators generate results close to those obtained by the GA and Maj error indicators. They are essentially better than results produced by the indicator used in MATLAB.

Table 6 Example 3: Mesh adaptation with different indicators and respective errors. N_t ia the amount of elements.

	0	1	2	3
N_t	128	182	346	646
$\|\nabla e\|^{(\text{Exact})}$	4.651e-02	3.365e-02	2.366e-02	1.796e-02
N_t	128	222	449	894
$\|\nabla e\|^{(\text{GA})}$	4.651e-02	3.085e-02	2.084e-02	1.481e-02
N_t	128	320	726	1659
$\|\nabla e\|^{(\text{Res})}$	4.651e-02	2.724e-02	1.706e-02	1.132e-02
N_t	128	213	432	833
$\|\nabla e\|^{(\text{Maj})}$	4.651e-02	3.158e-02	2.231e-02	1.622e-02
N_t	128	241	497	929
$\|\nabla e\|^{(\text{ML8})}$	4.651e-02	2.783e-02	1.948e-02	1.389e-02
N_t	128	241	505	949
$\|\nabla e\|^{(\text{ML10})}$	4.651e-02	2.783e-02	1.935e-02	1.384e-02
N_t	128	241	497	921
$\|\nabla e\|^{(\text{ML10+10})}$	4.651e-02	2.783e-02	1.948e-02	1.403e-02
N_t	128	241	504	931
$\|\nabla e\|^{(\text{ML20+20})}$	4.651e-02	2.783e-02	1.934e-02	1.391e-02

5 Conclusions

We present a network type indicator of computational errors by combining a posteriori error estimates of the functional type with the machine learning technology. The indicator $\mathbb{E}_{\text{ML}}$ is compared with the main classes of error indicators used standard computer simulation methods for PDEs. It was found that after a suitable teaching procedure, the indicators $\mathbb{E}_{\text{ML}}$ are

- *robust* (i.e., they are universal and applicable for any approximation v);
- *correct in the weak sense* (i.e., provide correct marking for mesh adaptation procedures);
- *correct in the strong sense* (i.e., provide a good presentation on quantitative values of local errors).

In many cases, $\mathbb{E}_{\text{ML}}$ produces better results than other indicators (in particular, it is better than the residual type indicator inbuilt in MATLAB). We believe that similar indicators can be obtained for other partial differential equations, provided that the process of machine learning is correctly organised and based on a sufficiently large collection of teaching examples. These first examples show high potential of artificial intelligence and machine learning methods for fast analysis of computational errors.

Appendix

For convenience of the reader we collect here graphical materials illustrating the results exposed in Sect. 4.

References

[1] A. Agouzal. On the saturation assumption and hierarchical a posteriori error estimator. *Comput. Methods Appl. Math.*, 2(2):125–131, 2002.

[2] M. Ainsworth and J. T. Oden. *A Posteriori Error Estimation in Finite Element Analysis*. Wiley and Sons, New York, 2000.

[3] I. Babuška and W. C. Rheinboldt. Error estimates for adaptive finite element computations. *SIAM J. Numer. Anal.*, 15(4):736–754, 1978.

[4] I. Babuška and R. Rodriguez. The problem of the selection of an a posteriori error indicator based on smoothening techniques. *Internat. J. Numer. Methods Engrg.*, 36(4):539–567, 1993.

[5] I. Babuška and T. Strouboulis. *The Finite Element Method and Its Reliability*. Oxford University Press, New York, 2001.

[6] W. Bangerth and R. Rannacher. *Adaptive Finite Element Methods for Differential Equations*. Birkhäuser, Basel, 2003.

[7] R. E. Bank and A. Weiser. Some a posteriori error estimators for elliptic partial differential equations. *Math. Comp.*, 44(170):283–301, 1985.

[8] A. Buffa and E. M. Garau. A posteriori error estimators for hierarchical B-spline discretizations. *Math. Models Methods Appl. Sci.*, 28(8):1453–1480, 2018.

[9] C. Carstensen and S. Bartels. Each averaging technique yields reliable a posteriori error control in FEM on unstructured grids. I. Low order conforming, nonconforming, and mixed FEM. *Math. Comp.*, 71(239):945–969, 2002.

[10] C. Carstensen, D. Gallistl, and Y. Huang. Saturation and reliable hierarchical a posteriori Morley finite element error control. *J. Comput. Math.*, 36(6): 833–844, 2018.

[11] R. Duran and R. Rodriguez. On the asymptotic exactness of Bank-Weiser's estimator. *Numer. Math.*, 62(3):297–303, 1992.

[12] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. Introduction to adaptive methods for differential equations. *Acta Numer.*, pages 105–158, 1995.

[13] R. E. Ewing, R. D. Lazarov, and J. Wang. Superconvergence of the velocity along the Gauss lines in mixed finite element methods. *SIAM J. Numer. Anal.*, 28(4):1015–1029, 1991.

[14] S. Iqbal and G. F. Carey. Neural nets for mesh assessment. Technical report, Defense Technical Information Center, Fort Belvoir, VA, 2005.

[15] C. Johnson and P. Hansbo. Adaptive finite elements in computational mechanics. *Comput. Methods Appl. Mech. Engrg.*, 101(1–3):143–181, 1992.

[16] C. Johnson and A. Szepessy. Adaptive finite element methods for conservation laws based on a posteriori error estimates. *Comm. Pure Appl. Math.*, 48(3): 199–234, 1995.

[17] O. Mali, P. Neittaanmäki, and S. Repin. *Accuracy Verification Methods: Theory and Algorithms*. Springer, Berlin, 2014.

[18] L. Manevitz, M. Yousef, and D. Givoli. Finite-element mesh generation using self-organizing neural networks. *Comput.-Aided Civil Infrastruct. Engrg.*, 12 (4):233–250, 2002.

[19] P. Neittaanmäki and S. Repin. *Reliable Methods for Computer Simulation: Error Control and A Posteriori Estimates*. Elsevier, Amsterdam, 2004.

[20] L. A. Oganesyan and L. A. Rukhovets. Study of the rate of convergence of variational difference schemes for second-order elliptic equations in a two-dimensional field with a smooth boundary. *USSR Comput. Math. Math. Phys.*, 9(5):158–183, 1969.

[21] R. Rannacher. The dual-weighted-residual method for error control and mesh adaptation in finite element methods. In J. Whiteman, editor, *The Mathematics of Finite Elements and Applications, X, MAFELAP 1999 (Uxbridge)*, pages 97–116, Oxford, 2000. Elsevier.

[22] S. Repin. A posteriori error estimation for variational problems with uniformly convex functionals. *Math. Comp.*, 69(230):481–500, 2000.

[23] S. Repin. *A Posteriori Estimates for Partial Differential Equations*. Walter de Gruyter, Berlin, 2008.

[24] M. Křížek and P. Neittaanmäki. On superconvergence techniques. *Acta Appl. Math.*, 9(3):175–198, 1987.

[25] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.

[26] W. Tsui, M. Slim Masmoudi, F. Karray, I. Song, and M. Masmoudi. Soft-computing-based embedded design of an intelligent wall/lane-following vehicle. *IEEE/ASME Trans. Mechatron.*, 13(1):125–135, 2008.

[27] R. Verfürth. *A Review of A Posteriori Error Estimation and Adaptive Mesh-refinement Techniques*. Wiley-Teubner, New York, 1996.

[28] L. B. Wahlbin. *Superconvergence in Galerkin Finite Element Methods*, volume 1605 of *Lecture Notes in Mathematics*. Springer, Berlin, 1995.

[29] J. Wang. Superconvergence analysis of finite element solutions by the least-squares surface fitting on irregular meshes for smooth problems. *J. Math. Study*, 33(3):229–243, 2000.

[30] M. Wang, S. W. Cheung, W. T. Leung, E. T. Chung, Y. Efendiev, and M. Wheeler. Reduced-order deep learning for flow dynamics. The interplay between deep learning and model reduction. *J. Comput. Phys.*, 401:108939, 20 pp., 2020.

[31] P. Yu, C. Anitescu, S. Tomar, S. P. A. Bordas, and P. Kerfriden. Adaptive isogeometric analysis for plate vibrations: An efficient approach of local refinement based on hierarchical a posteriori error estimation. *Comput. Methods Appl. Mech. Engrg.*, 342:251–286, 2018.

[32] O. C. Zienkiewicz and J. Z. Zhu. A simple error estimator and adaptive procedure for practical engineering analysis. *Internat. J. Numer. Methods Engrg.*, 24(2):337–357, 1987.

[33] E. Zio. Reliability engineering: Old problems and new challenges. *Reliab. Engrg. System Safety*, 94(2):125–141, 2009.

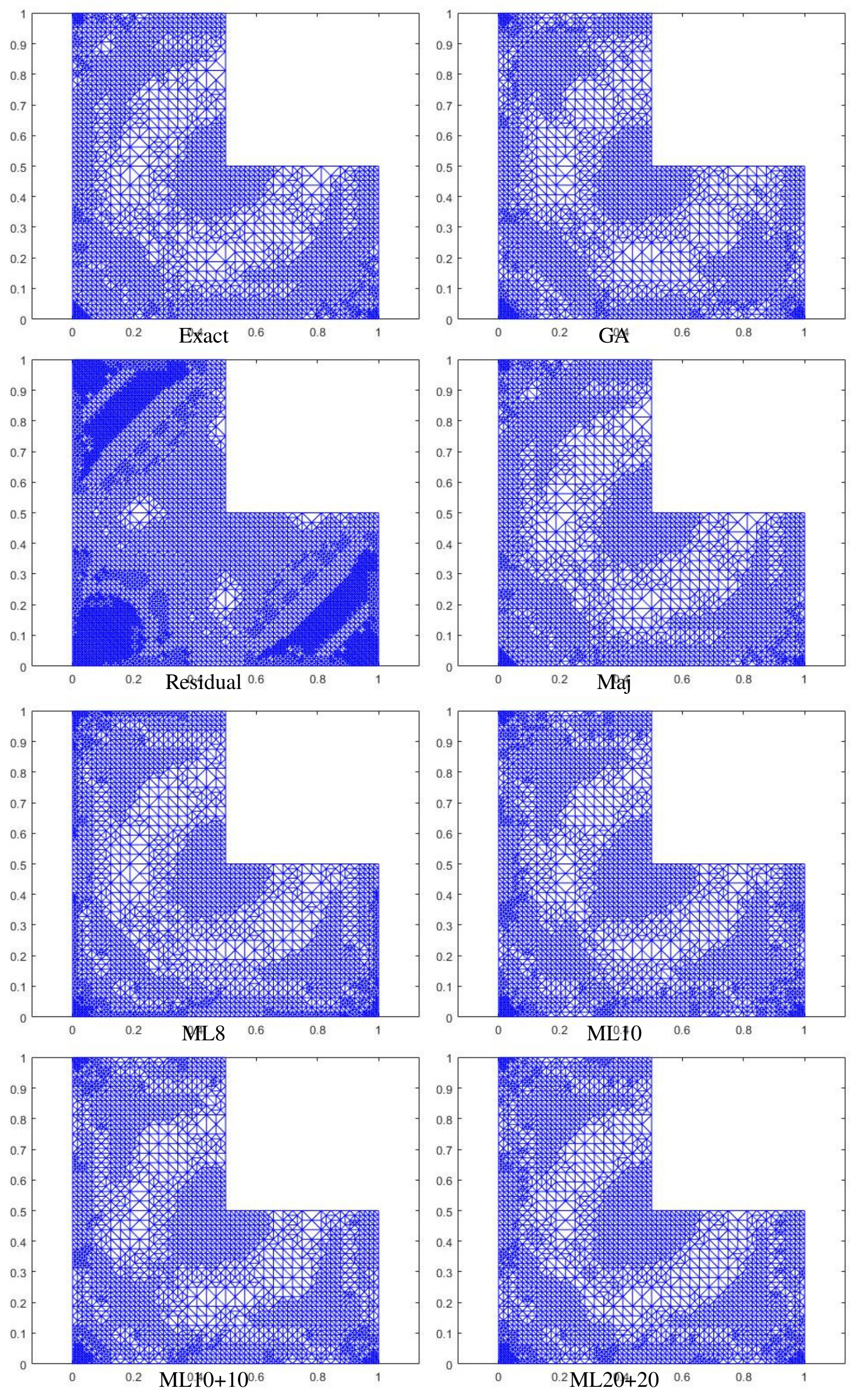

Fig. 8 Example 1: Meshes $\mathcal{T}_h$ generated by different error indicators; see column 5 of Table 4

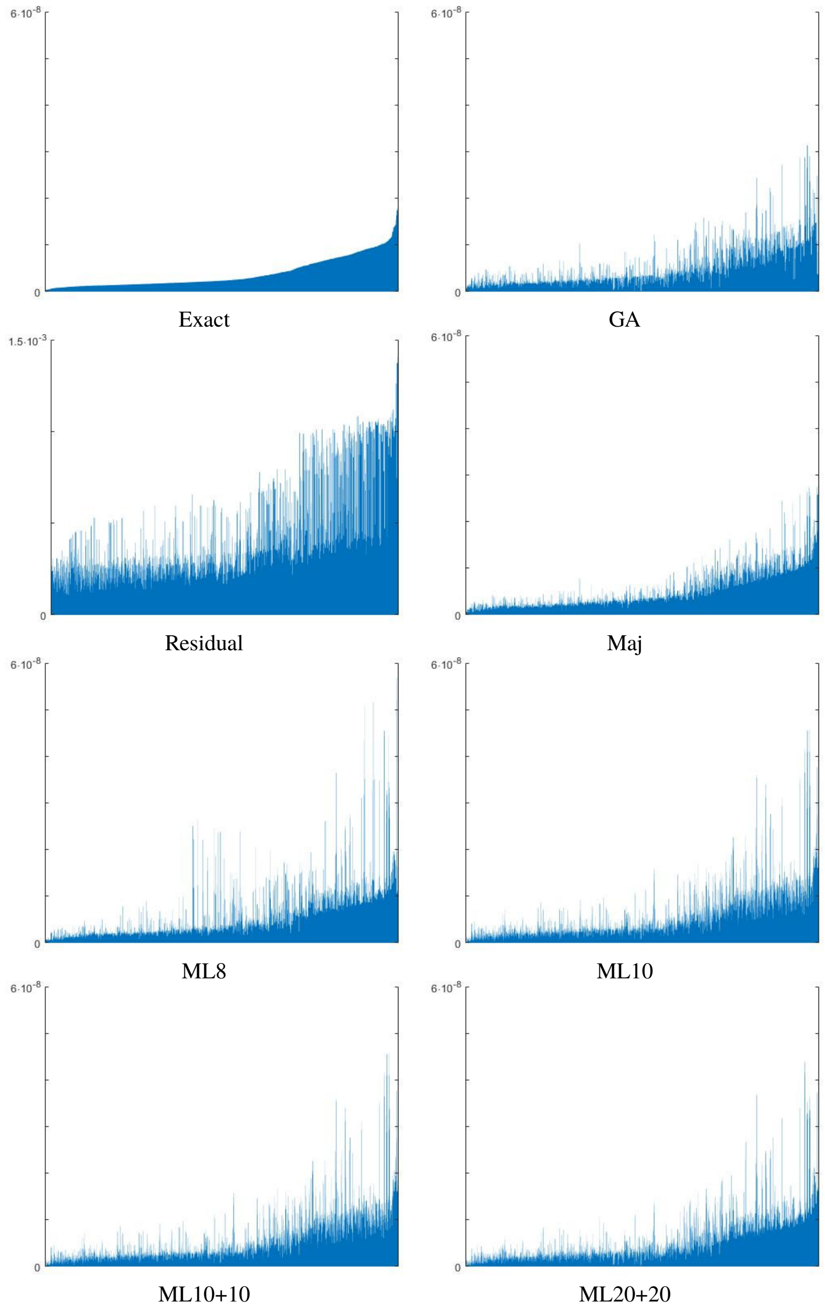

Fig. 9 Example 1: Histograms of local errors; column 5 of Table 4

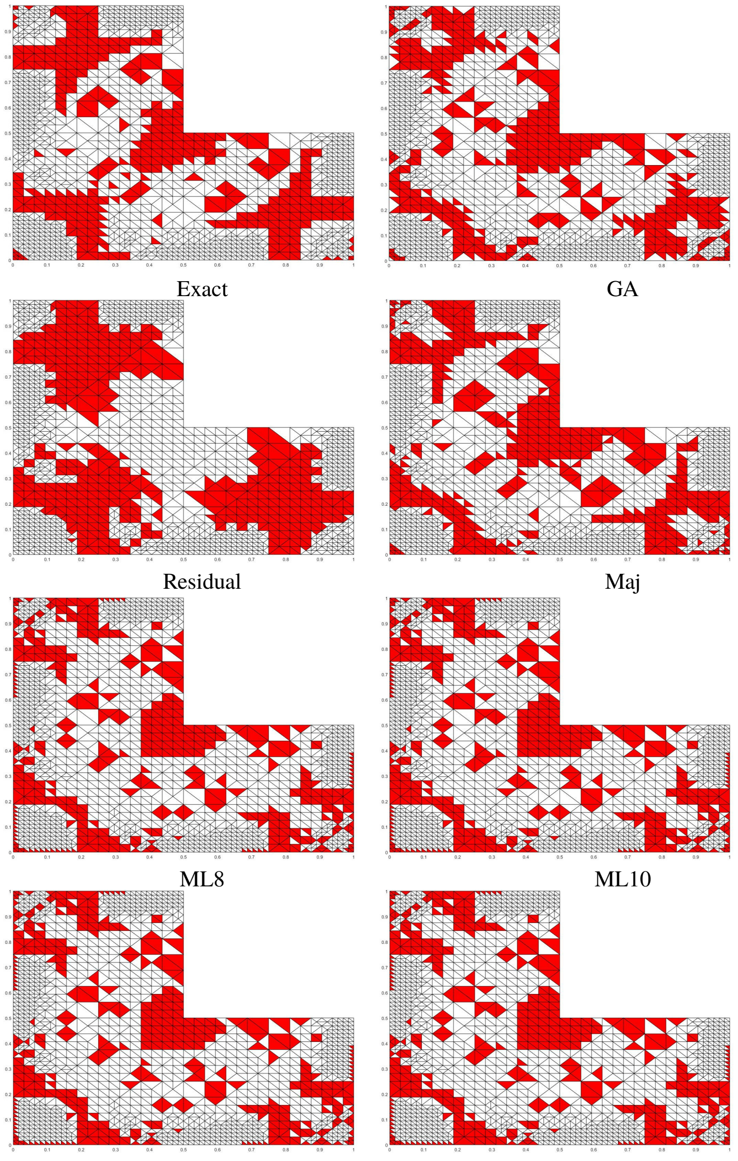

Fig. 10 Example 1: Errors marking $M(\boldsymbol{E})$; column 4 of Table 4

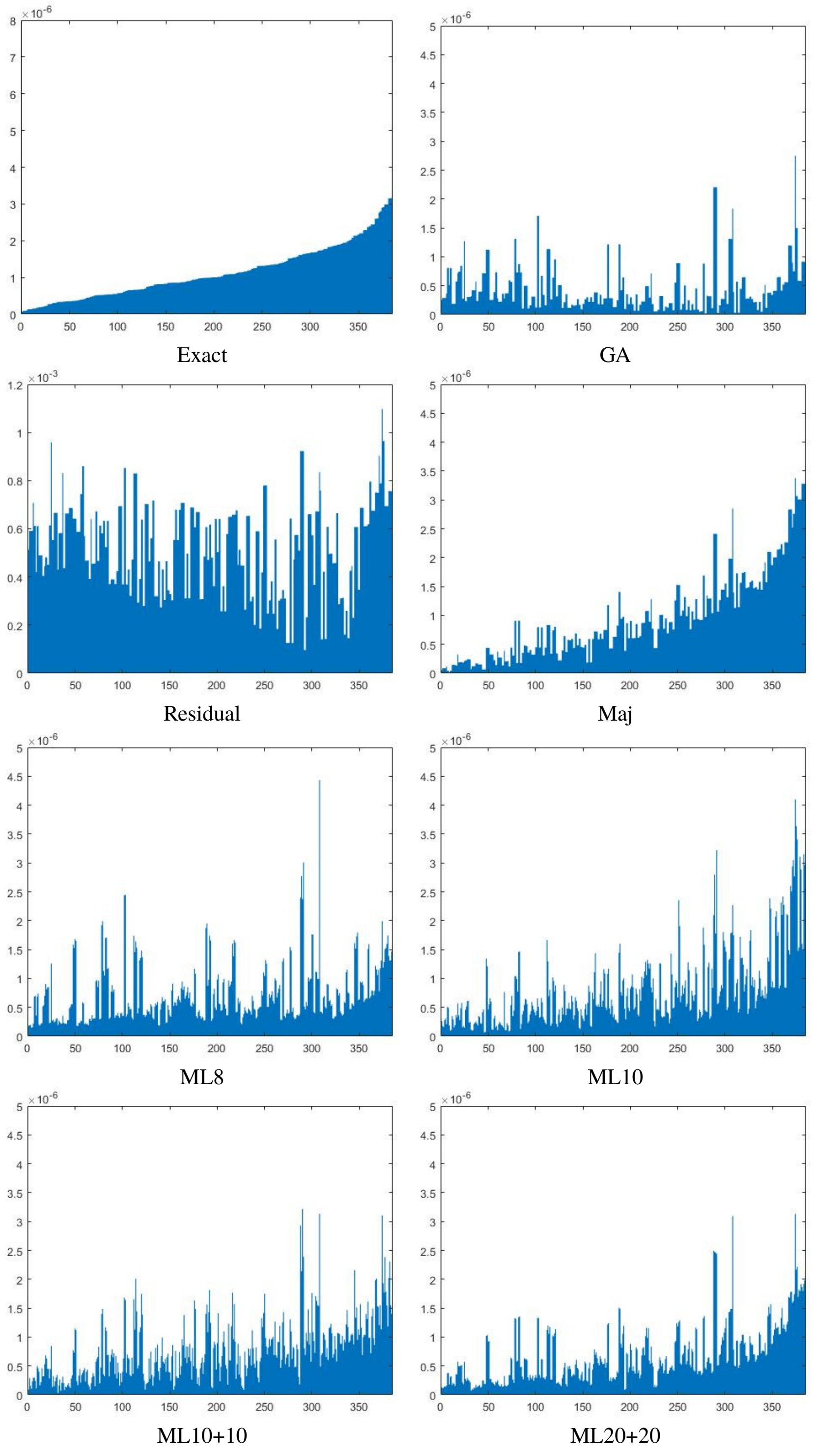

Fig. 11 Example 1: Histograms of errors generated by different indicators. Non-Galerkin approximation. $N_t = 384$.

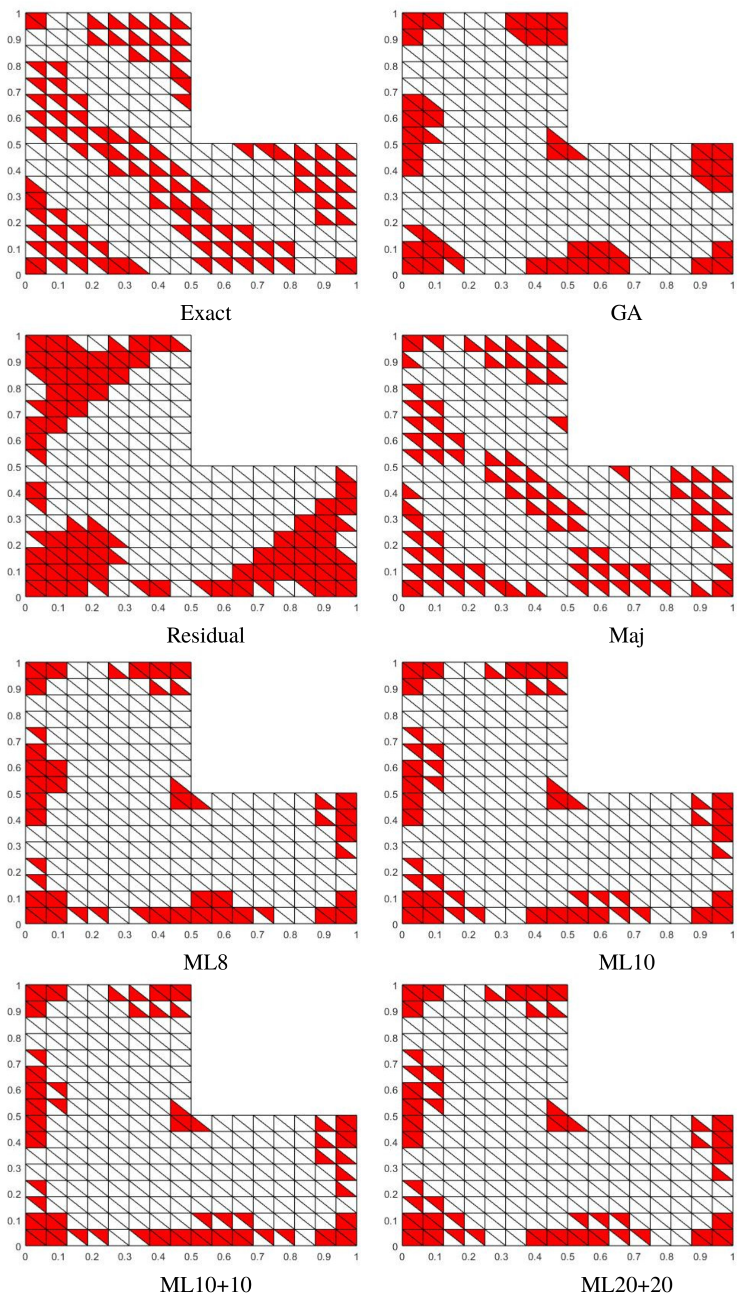

Fig. 12 Example 1: Errors marking $M(\boldsymbol{E})$. Non-Galerkin approximation. $N_t = 384$.

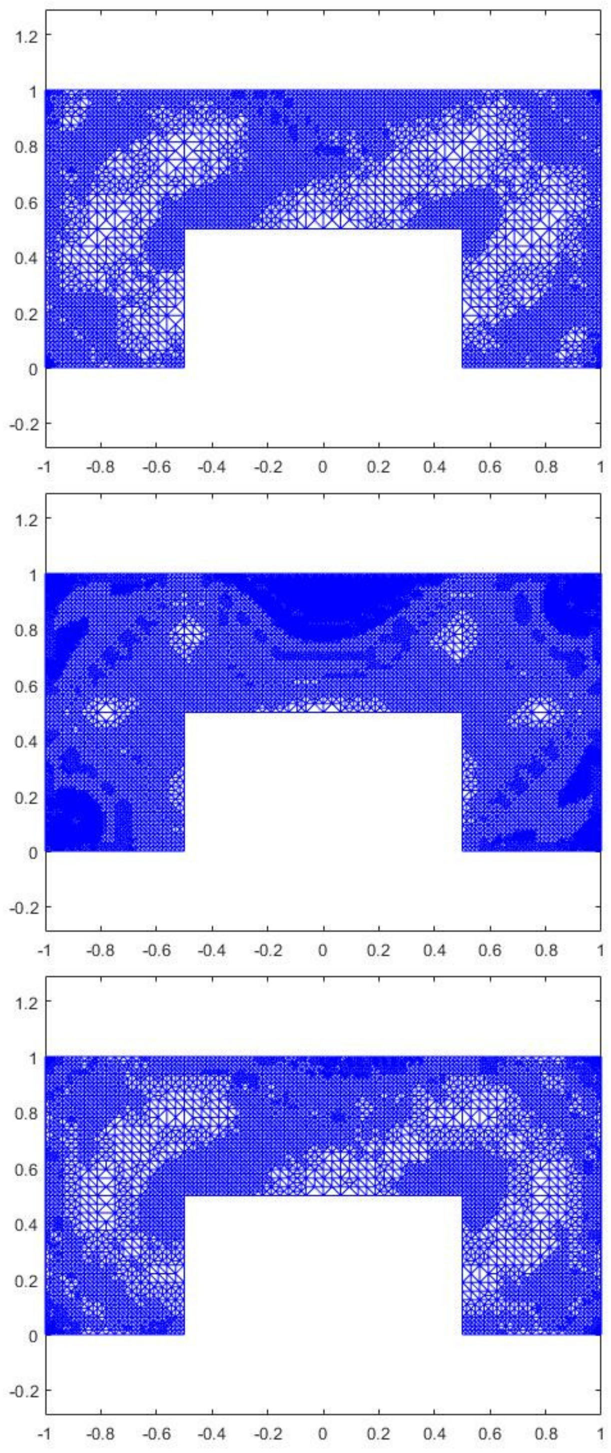

Fig. 13 Example 2: Structure of the meshes $\mathcal{T}_h$ obtained on the last step of adaptation: exact error indicator (top), residual indicator (middle), indicator ML10 (bottom)

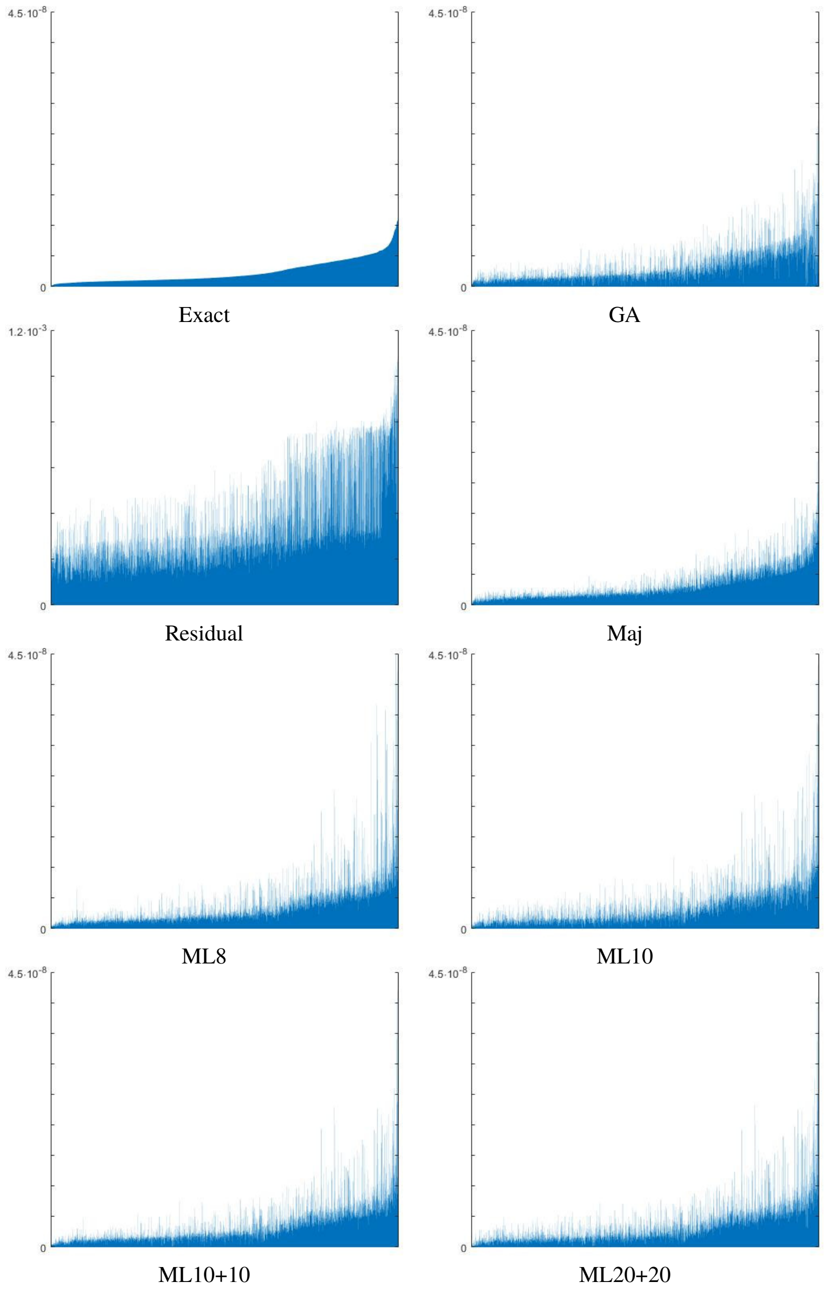

Fig. 14 Example 2: Histograms of errors; the last step of mesh adaptation, $N_t = 4587$.

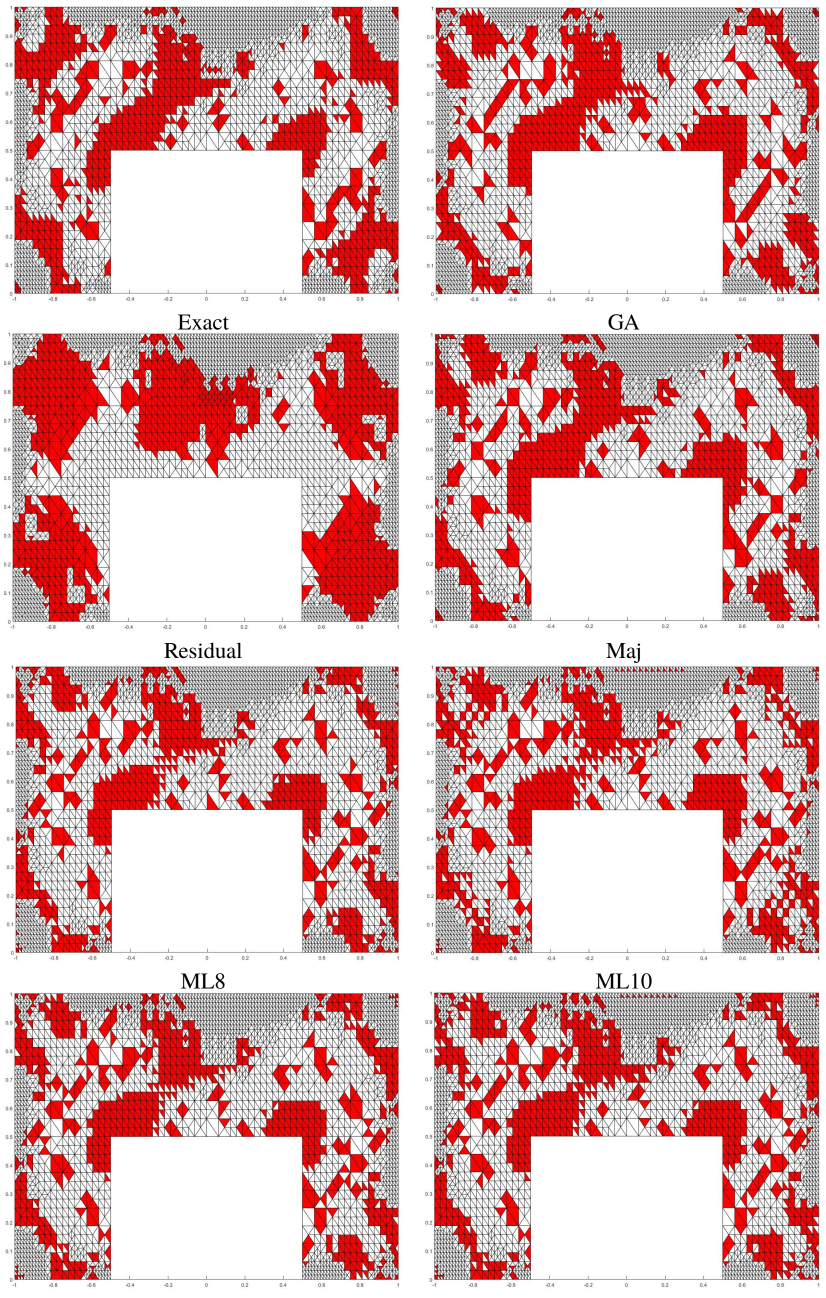

Fig. 15 Example 2: Errors marking $M(\boldsymbol{E})$, $N_t = 2416$

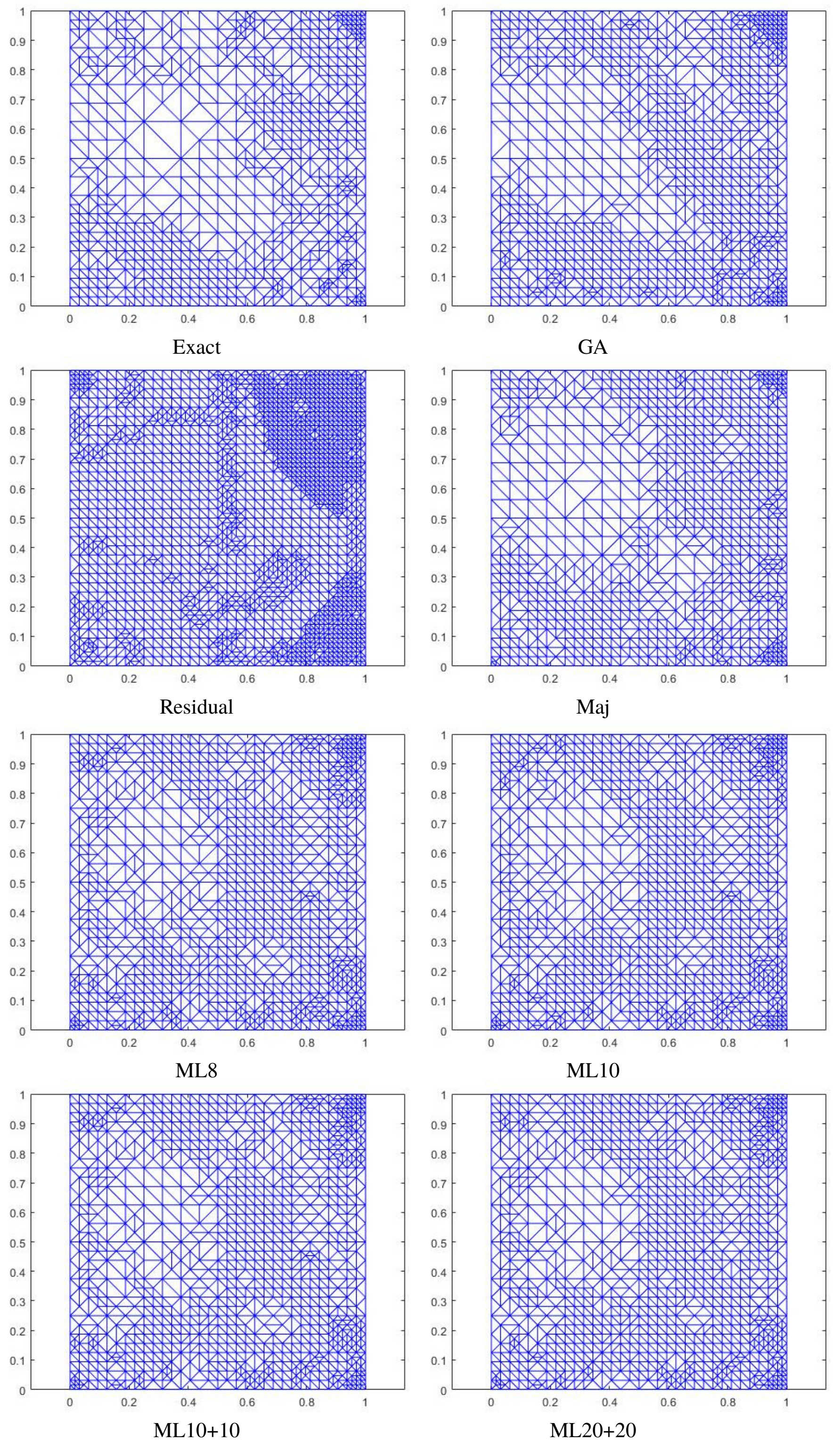

Fig. 16 Example 3: Meshes $\mathcal{T}_h$ generated by different error indicators; see the last column of Table 6

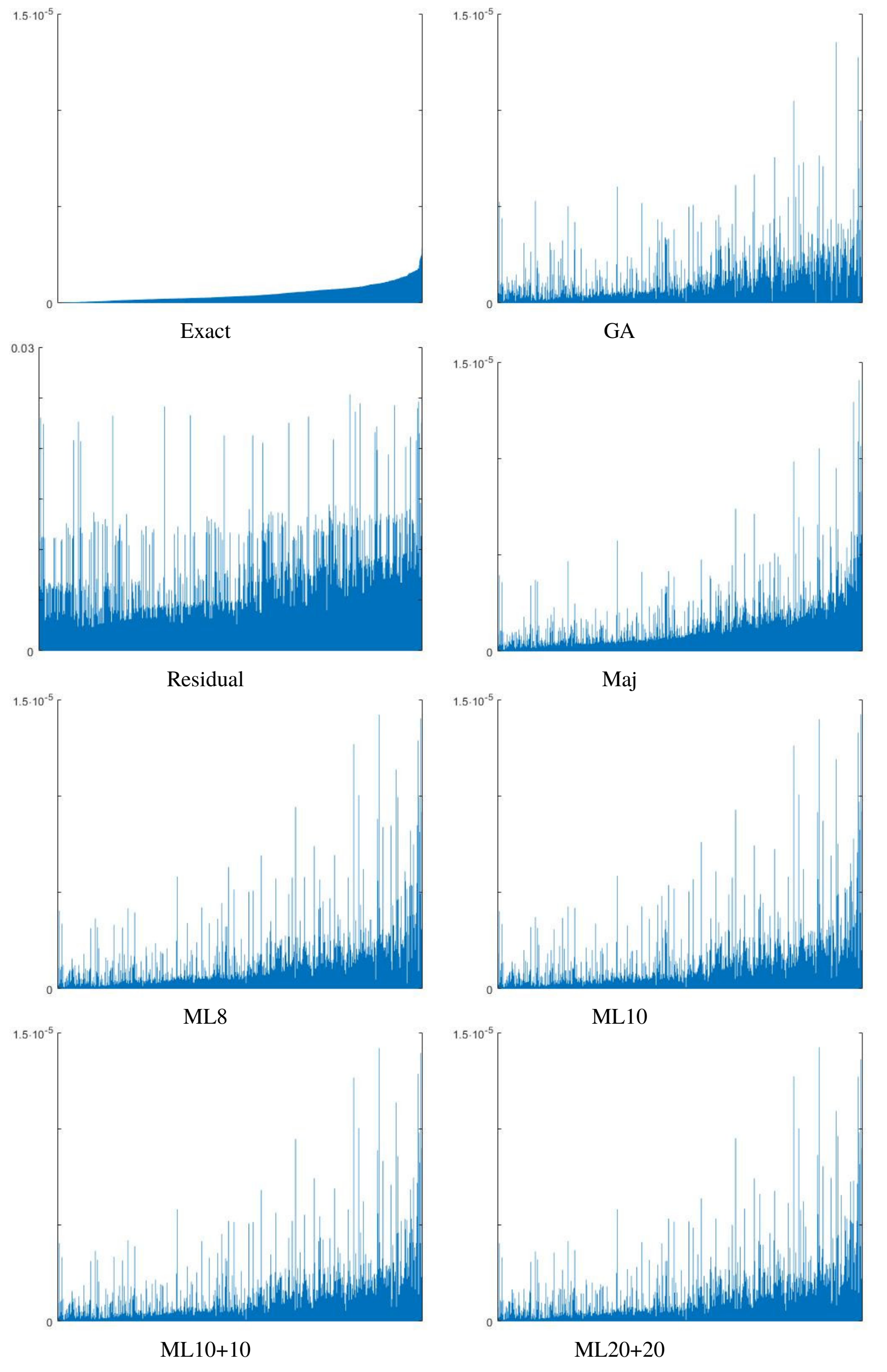

Fig. 17 Example 3: Histograms of local errors, $N_t = 646$