Computer Assisted Methods in Engineering and Science, **24**: 157–180, 2017.

Development of simple effective cloud of nodes and triangular mesh generators for meshless and element-based analyses – implementation in Matlab

Sławomir Milewski
Cracow University of Technology
Faculty of Civil Engineering
Institute for Computational Civil Engineering
Warszawska 24, 31-155 Cracow, Poland
e-mail: s.milewski@L5.pk.edu.pl

This paper is devoted to the development of the Matlab software dedicated to the generation of 2D arbitrarily irregular clouds of nodes and triangular meshes. They may be applied in numerical analyses of boundary value problems, based on both meshless and finite element discretization techniques, especially in the case of numerical homogenization in which the domain partitioning into disjoint subdomains may be required. Several Matlab functions are extended on the basis of the simple computational geometry-based ideas and concepts of engineering nature. A set of Matlab functions, attached to this paper, is discussed in detail, and examined on selected boundary value problems.

Keywords: cloud of nodes, mesh generation, Delaunay triangularization, meshless methods, finite element method, implementation in Matlab.

1. Introduction

Nowadays, Matlab [5] constitutes a powerful tool for a variety of engineering purposes. One of its biggest advantages is that it enables effective implementation of computational algorithms by experienced programmers (IT specialists) and inexperienced users (students, beginner engineers). It is especially convenient in the analysis of more complex technical issues, which require substantial implementation time and calculation effort. Boundary value problems of mechanics and civil engineering [7, 21] are investigated in this paper. In some cases, existing commercial software may be insufficient for effective numerical analysis of such problems, due to its limitations in the selection of problem formulation, mesh generation, function approximation, numerical integration or final postprocessing of the results. Moreover, the user's task is to obtain control over every single step of calculation. Such approach is indispensable for the original, non-trivial problems which have not been thoroughly investigated in the past, and whose true nature is complex, unknown or uncertain.

Numerical modelling of the boundary value problems by means of the meshless methods (MM) [13, 18, 25], may be difficult when using the traditional codes and commercially available software. Most of them are based on the finite element (FE) [27] algorithms in which structural mesh is applied and function approximation is ascribed in terms of a FE (e.g., [19, 28, 33]). On the contrary, in case of MM, nodes and only nodes are applied for function approximation, without any imposed structure such as FE or regular mesh [2]. Examples of application of MM may be found in the moving boundary, crack propagation, concentrated loads, adaptation techniques etc. Therefore, development of one's own codes may be considered crucial, even though there exists a wide range of available software and FE codes.

The Matlab framework enables that, mainly due to the simplicity of the programming language, which is based on the matrix and vector variables. Most importantly, the Matlab user does not have to be an expert in information technology and computer programming. In most cases, the fundamentals of linear algebra as well as the basic knowledge of numerical modelling seem to be sufficient. The advantages of the Matlab software can be summarized in the following points:

- simple programming language based on the matrix notation and the features of linear algebra,
- existence of the colon (:) and dot (.) operations, which enable significant elimination of the time-consuming 'for' loops and reduction of the calculation time,
- effective visualization of the results (object-oriented graphics),
- simple construction of the graphical user interface (similar to Visual Basic techniques),
- existence of a variety of toolboxes, which contain thematically organized functions (e.g., symbolic operations).

On the other hand, Matlab may not be the fastest language when it comes to the real computational time. This is one of its major drawbacks; however, this should not discourage potential users from effective application of Matlab in the preliminary state of research. Usually, the final code for practical usage may be prepared by means of other faster languages, such as C++ or Java, on the basis of already generated Matlab algorithms, which are tested and verified.

This paper may be considered as a continuation of the paper [36] in which the meshless solution approach to 2D Poisson problem and its implementation in the Matlab are presented. In spite of complete meshless computational algorithms provided there, node generation was limited to very simple cases, such as rectangular domain or cloud of nodes generated *a priori* by the user and loaded from an external file. Therefore, various techniques of the arbitrarily irregular cloud of nodes and triangular mesh generation are investigated in this paper.

It is very important to distinguish between two notions namely *cloud of nodes* and *mesh*, as they assume different meanings in various computational approaches. In the finite element method (FEM) [27], *mesh* is understood as the grid consisting of both nodes and elements (simple geometrical figures). Thus, the approximation of the unknown function developed in terms of degrees of freedom assigned to the elements' nodes. Moreover, the approximation and integration meshes (for trial and test functions of the appropriate variational principle) are usually the same (Bubnov-Galerkin approach). It should be emphasised that in FEM it is impossible to develop function approximation without having any element structure.

In the meshless methods (MM) [13, 18, 25, 34], one deals with the *cloud of nodes*. Nodes may be arbitrarily irregularly distributed, without any *a priori* imposed structure, like a finite element (FEM) or mesh regularity (as in the case of finite difference method – FDM), or projection constraints. The approximation of the unknown function is ascribed in terms of nodes only. Therefore, generation and modification of the cloud of nodes may be performed more effective by than in the case of FEM. Nodes may be added, removed or shifted without any impact on mesh structure as it does not exist. In selected cases, additional integration mesh, totally independent of approximation nodes, may be required [13]. Moreover, it should be noted that this additional mesh cannot be applied to unknown function approximation. Otherwise, the method is not meshless anymore. Therefore, the notion *mesh*, in case of MM, should be understood as *topology* information rather than the imposed nodes' structure [2, 4]. If available, such topology may be convenient, e.g., for the optimal star (stencil) generation (set of nodes for the local function approximation, equivalent to the FE [18, 35]), numerical integration or visualization purposes.

In this paper, the cloud of nodes generation and triangular mesh generation are discussed separately, in order to provide clear distinction between the FE approach (which requires mesh) and meshless approach (which requires nodes only). Variety of common types of 2D domains may be taken into account, with straight and curved edge lines, convex and concave parts of the boundary,

multiples of adjacent regions etc. In the author's opinion, the most important novel contributions of this work are:

- possibility of simultaneous analysis of several domains without any common parts and points (e.g. for parallel calculations of the same problem),
- possibility of coupling subdomains with different node densities and non-conforming meshes, or with different approximation schemes (as in the coupled FEM/MFDM analysis, which is becoming more and more popular nowadays),
- independent modelling of the interface zone between two or more subdomains (potential application in contact mechanics),
- selected types of regular edge curves may be taken into account, with nodes located regularly along the curve,
- modelling of heterogeneous materials (holes, inclusions) with different material parameters (e.g., for numerical homogenization techniques),
- generation of both regular and irregular (randomly distorted) meshes of triangles,
- simple data structure, which may be prepared by non-expert users without any difficulties, allowing for the coupling of Matlab functions for nodes/mesh generation with other available functions for numerical solutions of boundary value problems (tested here on the basis of the already published *MFDMtool* toolbox).

The paper is organized as follows:

- Section 2 gives general information concerning the Matlab codes (script and functions), attached to this paper (*nodes_mesh_tool*).
- Section 3 presents the fundamentals of geometry modelling with the main concepts and ideas applied here, the set of required geometry and topology data, which has to be defined by the user for the proper running of all the discussed Matlab functions and description of the model benchmark problems.
- Section 4 (cloud of nodes generation) presents appropriate techniques for the generation of nodes at vertices, edge lines, and domain interior, according to the ascribed densities, as well as the elimination of all obsolete nodes (e.g., located outside the domain or inside the holes).
- Section 5 (mesh generation) presents appropriate techniques for the generation of mesh of triangular elements based on the already generated nodes (Sec. 2). The main concept is to apply the *delaunay* Matlab function and to improve its actions for the domains with arbitrary shape.
- Section 6 gives general information concerning possible mesh merging between adjacent subdomains.
- Section 7 presents numerical results of the Poisson problem solution, defined on the selected domains, obtained by using the Matlab package *MFDMtool*, being a software part of [36], available from the Numerical Algorithms webpage or from the NETLIB library as the na36 package.

In Conclusions, some general remarks are given as well as possible extensions of the discussed techniques and Matlab software development are mentioned.

2. Software information

A special set of m-files (*nodes_mesh_tool*) was prepared by the author of this paper. The software is available from the author's home page: *nodes_mesh_tool* software.

Installation of this software is not very complicated. The user is supplied with one zip file (*nodes_mesh_tool.zip*), which contains one directory with all files. This archive file should be uncompressed to the ascribed location. The only m-files from this directory that may be run directly are:

- *TEST.m*, script file for testing purposes of the cloud of nodes/mesh generation; its description is given in Sec. 3,

- *TEST_MFDM.m*, script file, which enables numerical solution of the Poisson problem using the same cloud of nodes/triangular mesh, generated by means of the previous *TEST.m* file (its description may be found in Sec. 7). Therefore, application of this file should follow the application of *TEST.m* (both files use global variables).

One may add the name of this directory to the Matlab search path in order to run this file from the *Matlab Command Window*. Otherwise, it should be opened in the *Matlab Editor* and run, although it is proceeded by a modification of the *Matlab Current Directory*.

The other files include:

- *generate_nodes.m*, primary function with procedures for cloud of nodes generation; its description is given in Sec. 4,

- *generate_mesh.m*, primary function with procedures for triangular mesh generation; its description may be found in Sec. 5,

- *internal_point.m*, secondary function that examines whether the point in consideration is located inside or outside the domain; its description is given in Sec. 4,

- *complete_mesh.m*, secondary function that examines the mesh and adds additional triangles wherever they are missing, due to the faulty application of the *delaunay* Matlab function; its description may be found in Sec. 5,

- *common_edges.m*, primary function that indicates which boundary edges are common for two adjacent subdomains; its description is given in Sec. 6,

- *plot_nodes.m*, primary function that plots cloud of nodes and real domain boundary,

- *plot_mesh.m*, primary function that plots triangular mesh.

Moreover, all files from the previously published *MFDMtool* toolbox are given in the separate directory, for the sake of convenience. Although all the codes were prepared and tested by means of the Matlab 8.1.0.604 (R2013a), they do not contain any functions or syntax, which might be aberrant for the most commonly applied Matlab versions. For effective modelling of the curved edge lines (given by explicit function formula), symbolic toolbox is required (it is not a default component in most of the Matlab releases). As long as it is unavailable, the domains with straight edge lines only may be considered. More details on the implementation techniques of the subsequent steps of the nodes/mesh generation algorithm applied can be found in the *README.txt* file as well as in the following chapters.

3. Geometry modelling and data information (*TEST.M*)

There are three main types of computational modelling of the 2D geometry, namely

1. Decomposition of the considered domain into a set of subdomains, being geometrical primitives (rectangle, triangle, circle) and definition of the appropriate Boolean operations on them (addition, subtraction, common part) [15, 17, 40]. Afterwards, generation of nodes for the subsequent subdomains is performed separately, with the emphasis on the appropriate continuity on the common edges (e.g., *pdetool* – convenient Matlab toolbox [5], though with many limitations).

2. Boundary representation, which assumes defining domain as the closed region by means of a set of straight edge lines. In such case, generation of the nodes is divided into two main steps: generation of the nodes on the boundary parts and inside the domain [6, 20, 23, 24, 26, 29, 31, 39].

3. Application of NURBS (Non-Uniform Rational B-Splines [8, 9]) in which all edge curves (usually given by parameterization $(x, y) = (x(t), y(t))$ in 2D) are defined on the basis of the control points. This technique is usually applied in FEM for the isogeometric finite elements (in which the unknown function and the element geometry are ascribed by means of the same shape functions, [30]).

In this paper, all three concepts are combined in one relatively fast, simple and effective technique. Firstly, the problem domain may be divided into several parts (subdomains with common parts, adjacent or completely separated from each other) with arbitrary shapes (Fig. 1a). Dealing with two or more separate subdomains (e.g., domains of the same shape and dimensions) allows for simultaneous (or parallelized) comparison of the results for different discretizations and/or approximations. In the case of adjacent subdomains (Fig. 1b), one may have different material parameters, loads, or approximation schemes (e.g., various combinations of FEM and meshless methods, [10, 14, 22, 37, 38, 41]) in subsequent subdomains. Contact mechanics problems can be modelled in such manner as well. Additionally, the modelling of holes assumes that one or more subdomains are located inside the main subdomain (Fig. 1c), which may be applied in the numerical homogenization problems (in that case, a hole may be treated as an inclusion and considered in the numerical

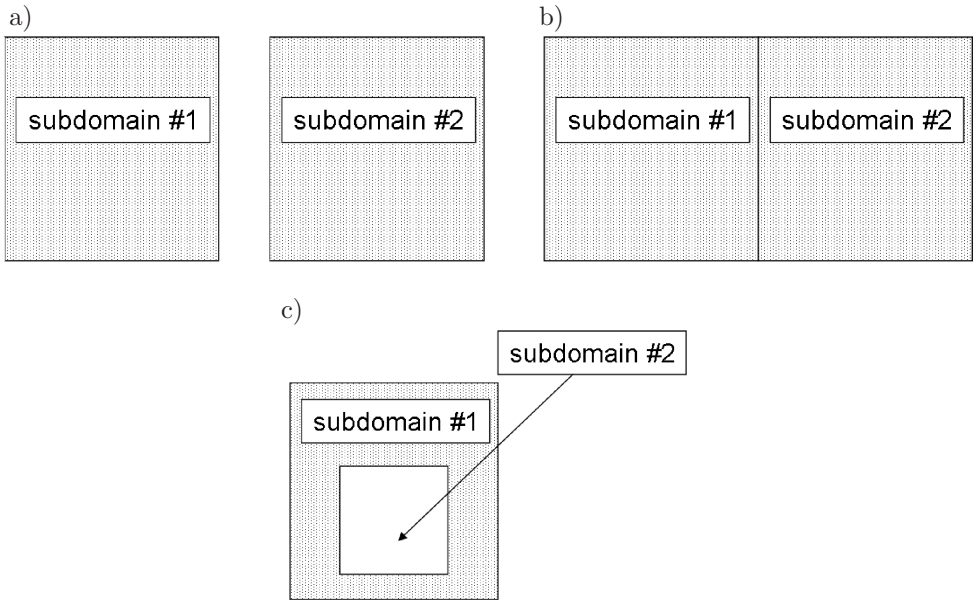

Fig. 1. Different types of 2D geometry modelling: a) two separate subdomains, b) two adjacent subdomains (one common side), c) domain with a hole.

analysis based upon the Representative Volume Element (RVE), [32]). Finally, subdomains may penetrate each other (a case not considered here).

All subdomains may have an independent boundary representation as a set of vertex points and edge lines connecting them. Moreover, selected parts of the boundary may be defined as non-linear functions (curves). No GUI (graphical user-interface) is provided. As a consequence, all necessary data has to be defined in the Matlab script file directly as the set of vector/matrix variables. The main Matlab m-file (script) *TEST.m* contains several examples of geometry modelling required for further proceedings. Moreover, it loads all the subsequent functions, leading to nodes and mesh generation. Parameters of several examples are given there (variable *data* = $1, 2, 3, ...$). For each example, the user has to define the following quantities (Matlab variables):

- *PARAMETERS* – numerical data matrix $[6 \times sdn]$ for subsequent subdomains (sdn – subdomains number), containing parameters:
  - first row: subdomain types (1-main contour, 2-hole),
  - second row: numbers of edges *noe* (including straight and curves edges),
  - third row: global nodes densities in x direction N_x,
  - fourth row: global nodes densities in y direction N_y,
  - fifth row: types of mesh (1-regular, 2-irregular),
  - sixth row: distortion amplitudes *amp* (for irregular meshes only).

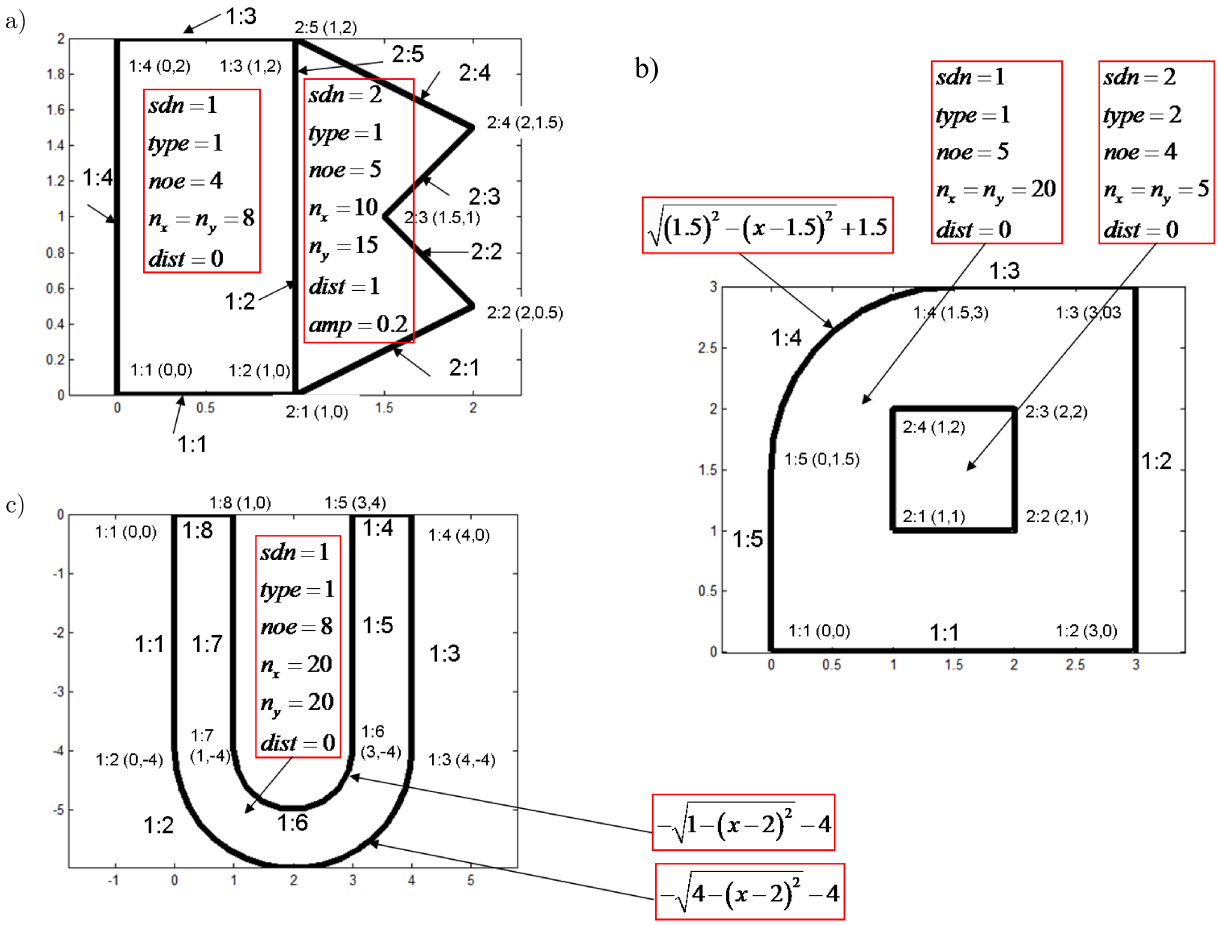

Fig. 2. Geometry data for three benchmark examples: a) example #1 (data = 1), b) example #2 (data = 2), c) example #3 (data = 3).

- *X_ver* $[noe \times sdn]$, *Y_ver* $[noe \times sdn]$ – numerical matrices of x and y coordinates of edge points (vertices) of all subdomains.
- *Edges* $[noe \times 3 \times sdn]$ – numerical matrix of edge topologies for subsequent subdomains (connections between vertices):
  - edge type (1-straight, 2-curve),
  - number of the first vertex point,
  - number of the last vertex point.
- *Edge_curves* $[noe \times sdn]$ – symbolic matrix, containing explicit mathematical formulas for subsequent edge curves (functions of x, where x is a pre-defined symbolic variable); if all edge parts are straight lines, it should be defined as an empty matrix [ ].

All numbers describing geometries of the selected three examples are presented in Fig. 2. The second and the third example ($data = 2$ and $data = 3$) require application of the symbolic Matlab toolbox [5]. Additional examples may be provided within the script file by the user. Moreover, several other pre-defined examples are available from help manuals of the subsequent functions, discussed in the following sections.

4. Generation of a cloud of nodes (*GENERATE_NODES.M*)

In this section, the generation of a cloud of arbitrarily distributed nodes (without any imposed element structure) is taken under consideration. The relevant algorithms, discussed in this paper, are implemented in the *generate_nodes.m*, which is a function m-file. Its arguments are the geometry and topology data presented in the previous section. On the other hand, this function yields another set of vector and matrix variables, containing cloud of nodes data (nodes coordinates *X_nodes*, *Y_nodes* and nodes' numbers *nodes_no* for all subdomains + boundary codes *Nodes_codes* as well as new edge lines topology information (*X_ver_real*, *Y_ver_real*, *Edges_real*, *edge_no_real*). This topology information is related to all edge lines, obtained after the numerical representation of curves, and reflects the real envelope line for the entire domain, taking into account all boundary nodes.

Node generation process is divided here into three steps, namely

(i) setting nodes at all vertices,

(ii) generating nodes at boundary parts (straight edges, curves),

(iii) generating nodes inside the domain.

Although locations of nodes from (i) are fixed, nodes from (ii) and (iii) may be relocated towards specified directions (e.g., along the boundary). In the *generate_nodes.m* file, the main loop *for* goes through all subdomains. For each subdomain, some reference names (variables nicknames) are provided. Then, all three steps (i)–(iii) are implemented in the subsequent subsections, indicated by appropriate comment lines. Generation techniques and general remarks are given and discussed below, whereas all generation steps are illustrated in the second example ($data = 2$) – see Fig. 3.

4.1. Generation of nodes at vertices

All vertices of the pre-defined subdomains become nodes (Fig. 3a). Their locations are fixed, and can not be modified by any random process. In matrix *Nodes_codes*, there are: code 1 (for all boundary nodes), the total number of edge lines/curves, on which the particular node is located (here: 2) as well as numbers (codes) of those lines/curves.

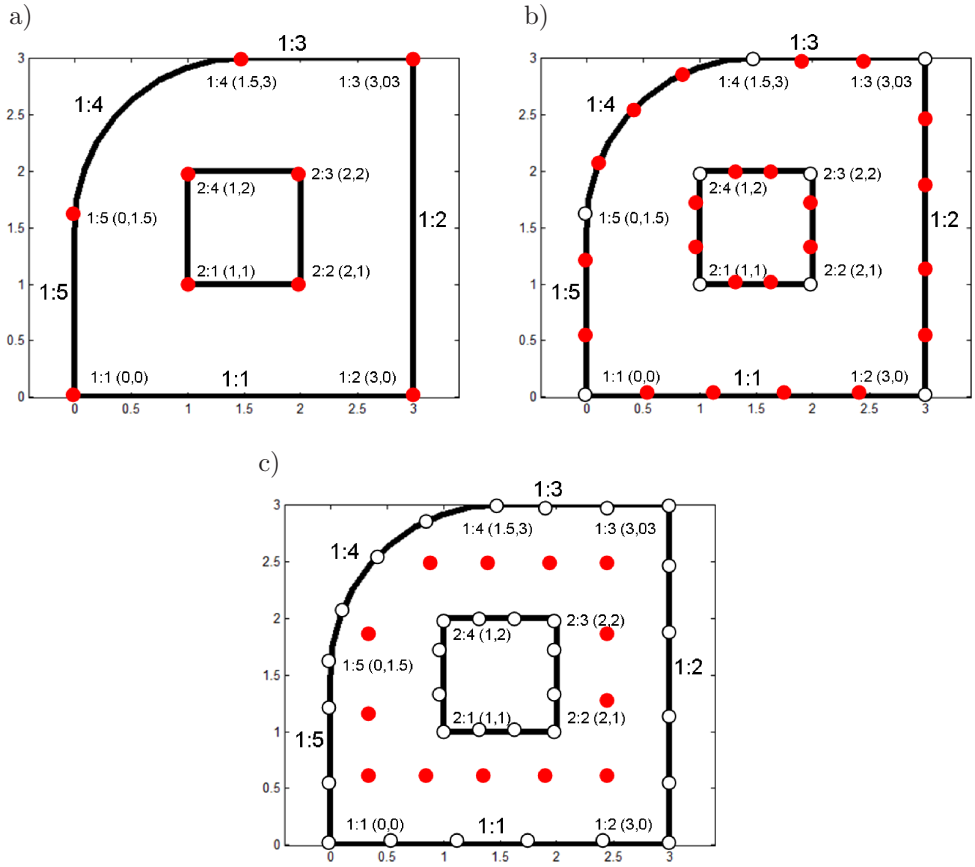

Fig. 3. Illustration of the three steps for cloud of nodes generation technique (black dots indicate new nodes for each step): a) example #2 – nodes at vertices, b) example #2 – nodes along the edge lines-curves, c) example #2 – internal nodes.

4.2. Generation of nodes at the boundary lines

Here, the main parameters are the global nodes densities N_x and N_y ascribed to each subdomain as well as the coordinates of the first $(x_{\text{start}}, y_{\text{start}})$ and the last point $(x_{\text{end}}, y_{\text{end}})$ of the boundary part. The selection of the appropriate procedure depends on the boundary part type (Fig. 3b). In the case of a straight line (Fig. 4), all node locations are found simultaneously, according to the following formulae:

$$\begin{cases} x_k = x_{\text{start}} + h \cdot c \cdot (k-1), \\ y_k = y_{\text{start}} + h \cdot s \cdot (k-1), \end{cases} \qquad k = 2, 3, ..., n-1 \tag{1}$$

($k = 1$ and $k = n$ are skipped since they are the already generated vertex nodes), where

$$\begin{aligned} s &= \frac{y_{\text{end}} - y_{\text{start}}}{L}, & c &= \frac{x_{\text{end}} - x_{\text{start}}}{L}, & h &= \frac{L}{n-1}, \\ L_x &= |x_{\text{end}} - x_{\text{start}}|, & L_y &= |y_{\text{end}} - y_{\text{start}}|, & L &= \sqrt{L_x^2 + L_y^2}, \\ n_x &= \left\lceil \frac{N_x \cdot L_x}{x_{\max} - x_{\min}} \right\rceil, & n_y &= \left\lceil \frac{N_y \cdot L_y}{y_{\max} - y_{\min}} \right\rceil, & n &= \max(n_x, n_y), \end{aligned} \tag{2}$$

in which s and c are the values of the sine $s = \sin(\alpha)$ and the cosine $c = \cos(\alpha)$ of the slope angle α of the edge line, L_x, L_x are the edge line semi-lengths for the x-axis and y-axis directions, L is the total length, and n_x, n_y are the local nodes' densities (integers), along the x and y axes, selected

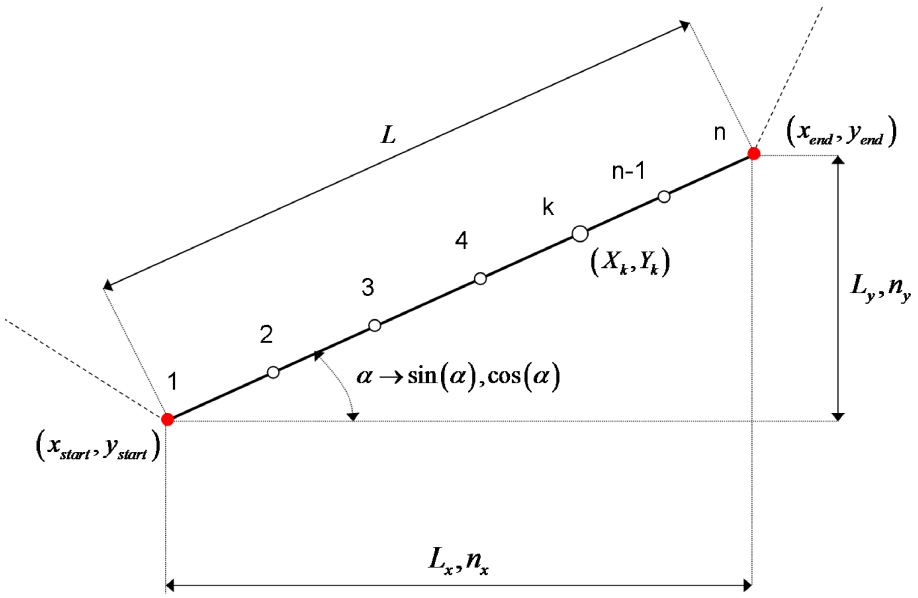

Fig. 4. Generation of nodes on the straight boundary line.

proportionally to the semi-lengths L_x and L_y and to the entire subdomain dimensions $x_{\max} - x_{\min}$ and $y_{\max} - y_{\min}$, whereas n is their representative value. Ceiling function $\lceil x \rceil$ yields the smallest integer, which is not less than x.

Boundary codes for all nodes (1) are set to 1. Moreover, all nodes are located on one (and only one) edge line, stored in the *Nodes_codes* matrix as well. Additionally, for every straight line, appropriate elements of "real" edge matrices are filled with the same quantities as for the user-defined edge matrices. In the case of straight edge lines, no boundary geometry changes occur.

Curved edge parts require special treatment (Fig. 5). The explicit formula for the curve is given by the function $y = f(x)$, in which $x \in (x_{\text{start}}, x_{\text{end}})$. Analytical approach for nodes generation on a curve assumes division of the total curve length

$$L_f = \int_{x_{\text{start}}}^{x_{\text{end}}} \sqrt{1 + (f'(x))^2}\, dx \tag{3}$$

into equal parts and generation of nodes along the curve with the same modulus. However, due to the potential complexity of the $f(x)$, both numerical and analytical (e.g. symbolical) direct calculations in Matlab of the above given integral may fail. Therefore, in order to generalise this appraoch, the total curve length T_f is calculated in a very simple numerical manner, whereas nodes on the curve are generated one by one (one node at once), by means of the appropriate iterative procedure [21].

The total curve length L_f is calculated by means of a division of $x \in (x_{\text{start}}, x_{\text{end}})$ interval into p_{opt} subintervals (x_i, x_{i+1}), $i = 1, 2, ..., p_{\text{opt}}$ of the same length each. It is assumed that on each small subinterval, the curve may be replaced by the straight line. Therefore, one obtains the approximate formula for L_f

$$L_f \approx \sum_{i=1}^{p_{\text{opt}}} \sqrt{(x_{i+1} - x_i)^2 + (f(x_{i+1}) - f(x_i))^2} \tag{4}$$

and the length of each curve segment (between neighboring nodes on the curve) $L_i = \dfrac{L_f}{n}$, in which n is obtained in the same way as for the straight boundary line (2).

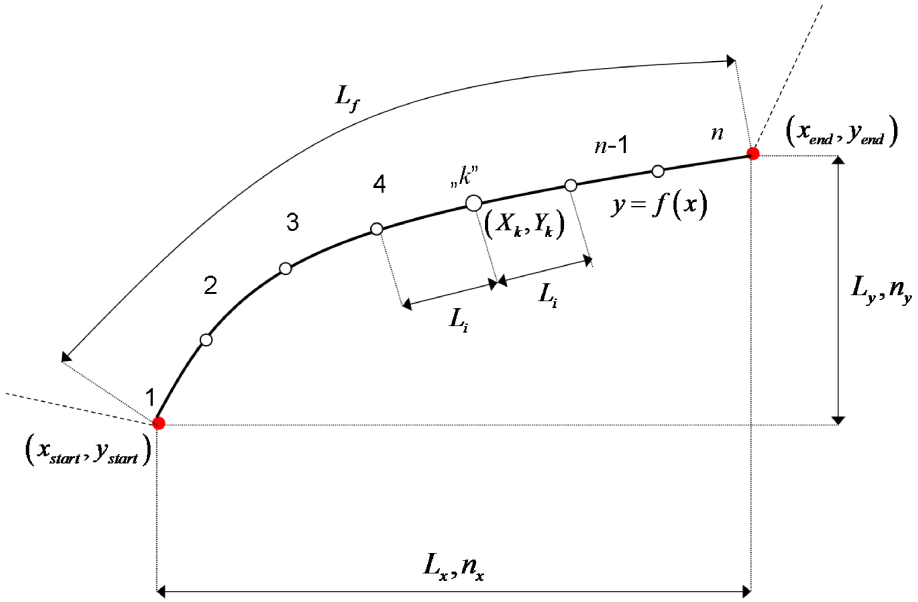

Fig. 5. Generation of nodes on the curved boundary line.

The optimal number of subintervals p_{opt} is selected in an adaptive strategy in which the above integral is evaluated with admissible relative error $\epsilon_{\text{adm}} = 10^{-6}$. Maximum number of subdivisions is assumed as $p_{\max} = 1000$.

Subsequent nodes are determined "node by node", by means of the standard bisection method, applied to nonlinear equation in the following form:

$$F(x) = \sqrt{(x_i - x)^2 + (f(x_i) - f(x))^2} - L_i. \tag{5}$$

Any other iterative procedure (Newton's method) may be applied as well, although the bisection method was selected due to its simplicity as well as convergence guarantee (as long as the equation root is located within the considered interval).

Equation (5) assumes that the curve length between x_i (previous node) and x (potential node location) is approximately equal to L_i. Therefore, determination of each node on the curve requires separate bisection approach. The middle point of the interval (a, b) is evaluated as $x_k = (a + b)/2$. The sign $(-, +)$ of the factor $F(x_k) \cdot F(a)$ determines whether the left subinterval (a, x_k) or the right subinterval (x_k, b) requires further division. The process is continued as long as the appropriate break-off conditions are satisfied. Here, it relies on two assumptions, namely: the admissible residual error $|F(x_k)| < \epsilon_{\text{adm}} = 10^{-6}$ and the maximum number of iterations = 100. Thus, the determined point $(x_k, f(x_k))$ becomes the following node on the curve.

For the first node, $a = x_{\text{start}}$ and $b = x_{\text{end}}$ are assumed. After determining the location of the first node x_2 on the curve, a changes into $a = x_{\text{start}} + x_2$, whereas b remains unchanged. The process is continued as long as locations of all $n-2$ nodes $(x_k, f(x))$ are found on the curve $(k = 2, 3, ..., n-1)$.

The curve number is stored in the *Nodes_codes* matrix. However, matrices ascribing real domain vertex points have to be modified with the determined locations of nodes on the curve. In fact, this angular line constitutes the real domain envelope line, which is required, e.g., for a proper mesh generation and graphical purposes.

4.3. Generation of the nodes inside the domain

After all boundary nodes are generated, one has to fill the domain interior with nodes, taking into account the selected generation criteria and limitations, such as admissible distance to boundary

nodes and/or boundary lines/curves or location inside the domain and/or outside the hole. Therefore, the generation of nodes inside the domain may be divided into three general steps, namely:

- Generation of nodes inside the rectangle built upon the extreme coordinates $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$ of the vertex points of the current subdomain (subdomain rectangular envelope). In such case, it is sufficient to apply the Matlab built-in function *meshgrid*, which produces regular mesh represented by two matrices, with x and y coordinates of all internal nodes, respectively. It is based upon the following mesh modules:

$$h_x = \frac{x_{\max} - x_{\min}}{N_x - 1}, \qquad h_y = \frac{y_{\max} - y_{\min}}{N_y - 1}, \tag{6}$$

  but with generation starting at the point $(x_{\min} + h_x,\ y_{\min} + h_y)$ and ending at the point $(x_{\max} - h_x,\ y_{\max} - h_y)$. Therefore, no boundary nodes will be generated multiple times.

  Although very simple in concept, selected nodes from this regular mesh have to be deleted (or shifted) due to numerous constraints.

- Elimination of the obsolete nodes located outside the domain requires an appropriate criterion of node location inside/outside the domain (polygon). Among many possible approaches, the ray-casting algorithm seems to be the optimal one.
  The ray-casting algorithm (also known as: crossing-number algorithm, even-odd rule algorithm or Jordan algorithm, [1, 12]) is a very simple method of finding whether the point is inside or outside a polygon. One needs to test how many times a ray, starting from the point (here: potential node) and going in any fixed direction, intersects with the edges of the polygon (Fig. 6). If the ray intersects the polygon edge an even number of times, the point is located outside the polygon. On the contrary, if the ray intersects the polygon edge an odd number of times, the point is located inside the polygon. This method is reported [12] not to work if the point is on the edge of the polygon, but this is not the case here, since only real internal nodes are generated in the first step. The results may be incorrect, if the point lies very close to that boundary, due to rounding errors. However, in such case, additional criterion needs to be applied, examining whether the node is not too close to the boundary lines/curves or to other boundary nodes.
  The approach is very simple in implementation. The appropriate Matlab code may be found in *internal_point* function m-file, which contains eight lines of code only. It requires one loop over all edges of a current subdomain. Function returns 1 if the point is inside the domain, or

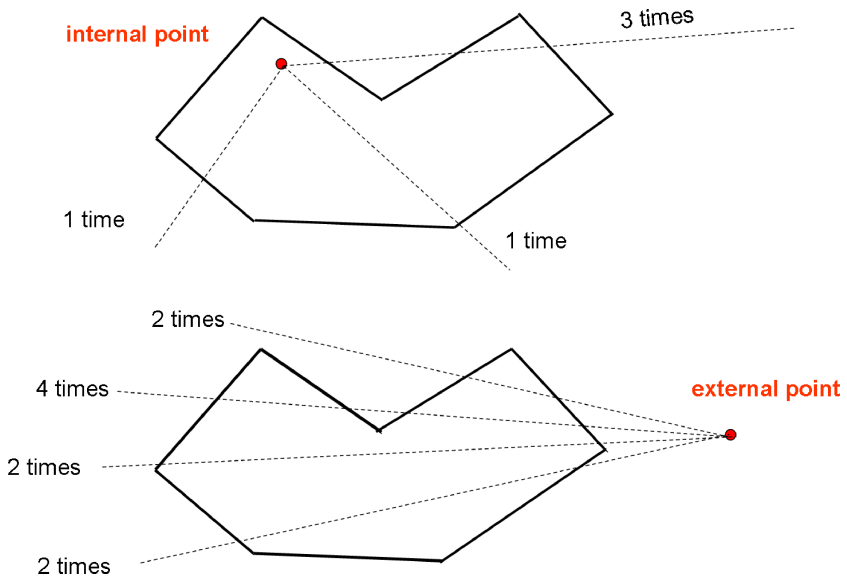

Fig. 6. Illustration for the ray-casting algorithm.

0 otherwise. In this stage of calculations, we examine whether the potential node is inside the current subdomain (value 1) and outside its all holes (value 0). If both criteria are fulfilled, it becomes a node.

- Elimination of nodes located "too close" to the boundary line and/or boundary nodes. In order to keep the cloud of nodes smooth in transitions between the boundary and the internal zones, all nodes which are located too close to the boundary have to be deleted. Firstly, all boundary lines and all internal nodes are taken into account. The distance between the i-th node and the k-th boundary line may be evaluated as

$$d_{k,i} = \frac{|A_k \cdot x_i + B_k \cdot y_i + C_k|}{\sqrt{A_k^2 + B_k^2}}, \tag{7}$$

in which $A_k = y_{\text{end},k} - y_{\text{start},k}$, $B_k = -\left(x_{\text{end},k} - x_{\text{start},k}\right)$ and $C_k = -A \cdot x_{\text{start},k} - B \cdot y_{\text{start},k}$ are the parameters of a k-th straight boundary line, given by means of a general equation $y_k(x) = A_k \cdot x + B_k \cdot y + C_k$. Node (x_i, y_i) is not deleted as long as the criterion

$$\left| d_{k,i} - \frac{1}{3} \cdot \max\left([h_x, h_y]\right) \right| < 10^{-12} \tag{8}$$

is not fulfilled. Factor $\frac{1}{3}$ is an arbitrary number, obtained after series of tests, reflecting the smoothness of the cloud of nodes near the boundary lines. The smaller this number is, the greater number of nodes remains unchanged. Number 10^{-12} is close to the working precision of Matlab and is introduced due to the truncation errors.

- Elimination of nodes located "too close" to the already generated boundary nodes. If the boundary line is long and/or the number of its boundary nodes is high, the above discussed criterion may be not sufficient. In order to improve the smoothness of the cloud of nodes, additional nodes are deleted according to the following condition:

$$\left| \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - \frac{1}{2} \cdot \max\left([h_x, h_y]\right) \right| < 10^{-12}. \tag{9}$$

Here, (x_i, y_i) are the coordinates of an examined (potential) internal node, whereas (x_j, y_j) are the coordinates of a current boundary node. Similarly to the previous criterion (8), factor $\frac{1}{2}$ was obtained after performing a variety of benchmark tests.

Function *generate_nodes.m* additionally produces a text report, displayed in the *Matlab Command Window*, showing all details concerning the numbers of generated and deleted nodes.

4.4. Randomization of nodes locations

Despite being based on regularity assumptions, locations of almost all nodes in the final cloud may be distorted by using random numbers. The only nodes with fixed locations are the vertex nodes. Nodes located on the boundary may be shifted along both the straight and curved edges, while nodes inside the domain may be shifted in all directions, and their new locations are limited by the same constraints as discussed in the previous subsection. Assuming that the randomization is active ($Amp \neq 0$), nodes' locations may be distorted by means of the *rand* Matlab function in the following general manner:

$$\begin{cases} x_k^{new} = x_k + h_x \cdot Amp \cdot (2 \cdot rand - 1), \\ y_k^{new} = y_k + h_y \cdot Amp \cdot (2 \cdot rand - 1), \end{cases} \qquad k = 1, 3, ..., n, \tag{10}$$

where $rand \in (0,1)$ is the pseudo-random number. Therefore, new randomly distorted coordinates of the k-th node are limited by

$$\begin{cases} x_k^{new} \in (x_k - h_x \cdot Amp, x_k + h_x \cdot Amp)\,, \\ y_k^{new} \in (y_k - h_y \cdot Amp, y_k + h_y \cdot Amp)\,. \end{cases} \tag{11}$$

4.5. Examples

Figure 7 presents selected examples of clouds of nodes, generated by means of the above discussed techniques. The first three examples were already introduced in this paper. The fourth example (bottom right corner) is domain bounded by a sine-shaped boundary part. All those results may be obtained by running the "TEST.m" file available in m-files attached to this paper. The user may modify parameters, by increasing or decreasing the nodes' density and switching from regular to irregular cloud of nodes (with ascribed irregularity parameter). Some other examples are available in the text help manual in Matlab (e.g., type *help generate_nodes* in the *Matlab Command Window*). Moreover, the user is encouraged to prepare her/his own examples within the *TEST.m* file.

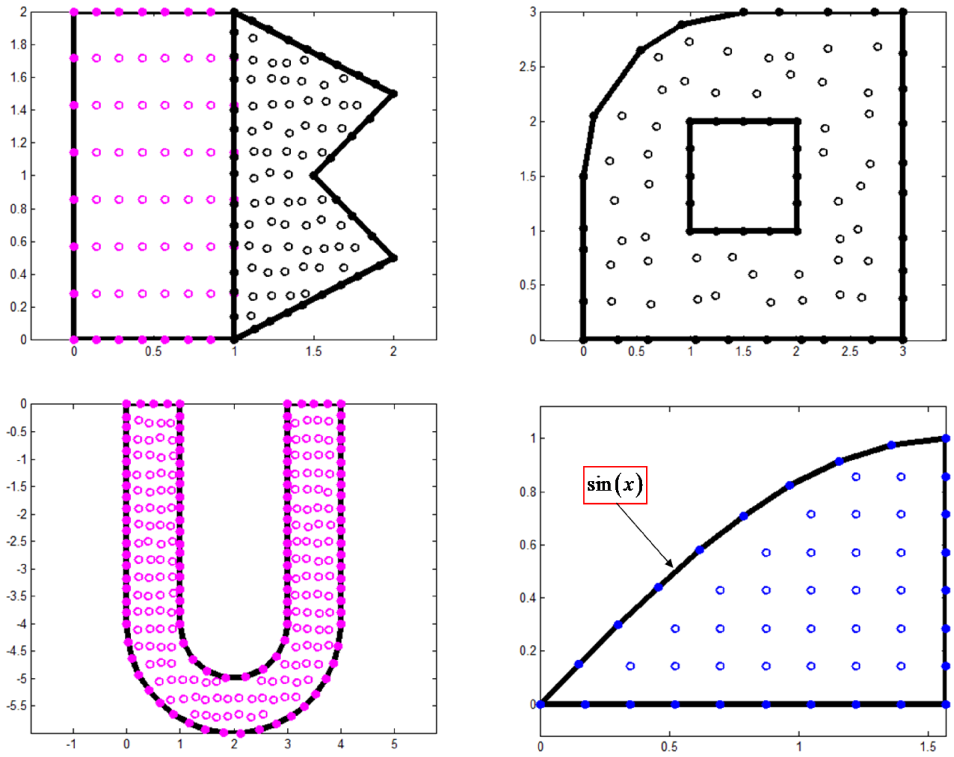

Fig. 7. Selected examples of cloud of nodes generation.

5. Generation of a triangular mesh of elements (*GENERATE_MESH.M*)

In some cases, regular mesh/irregular cloud of nodes may be applied directly to numerical analysis of the boundary value problems, posed in both local and variational (weak) formulations. However, its application is limited to MM only. In MM, unknown function approximation is ascribed by terms of nodes, therefore no mesh structure is required. However, additional background mesh (e.g., independent from nodes) may be helpful for numerical integration, graphical purposes (visualization of results) or general postprocessing (*a-posteriori* error estimation). With element-based methods

(such as FEM), additional mesh of simple geometrical figures (triangles [15], quadrangles) is required [2, 4].

In this Section, construction of a triangular mesh is presented. It is based on the Matlab function *delaunay*, which uses well-known Delaunay tessellation (partition of the entire domain into triangles, with nodes at their vertex points). Although very fast and effective, this function does not work properly with all types of domains or clouds of nodes. Its main drawback is that it does not comprise the domain boundary (Unconstrained Delaunay Triangulation, [15]). Therefore, in the case of non-convex domains or domains with holes, triangles are generated outside the domain or inside the holes. Moreover, the function is very sensitive to rounding errors, e.g., in the case of the skew boundary lines, triangles with almost zero surface area are generated. Usually, it is the user's task to improve its application. It may be stressed here that similar issues arise in the case of the *voronoi* Matlab function, which generates the set of Voronoi polygons, ascribed to each node. However, they are not investigated here. This problem is planned as the subject of next paper, with a particular focus on selected problems of the adaptive version of the meshless FDM.

One may distinguish four main steps of triangular mesh generation. They are described in the following four subsections.

5.1. The Delaunay tessalation

First, the Matlab function *delaunay* is applied. It requires a set of nodes only (x and y coordinates given by two vectors *X_nodes* and *Y_nodes*, respectively). It produces a set of integers, indicating the number of nodes ascribed to each triangle. Direct results of this function for selected domains are presented in Fig. 8.

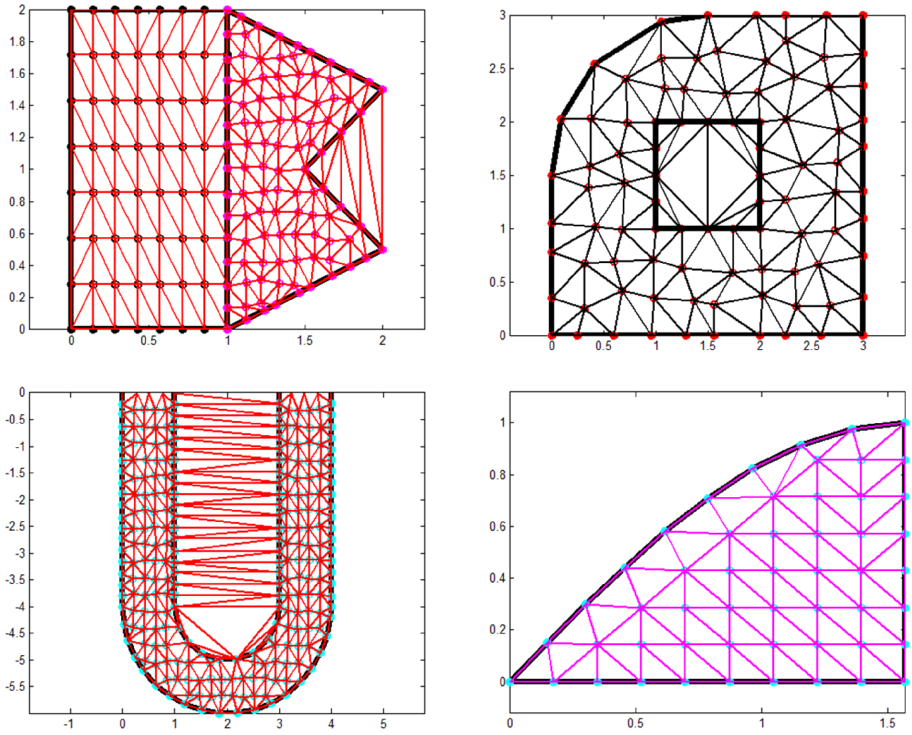

Fig. 8. Examples of Delaunay tessellation in Matlab – direct application of *delaunay* function.

Eventually, the only proper result may be observed for convex, simply connected domain (Fig. 8, left bottom graph). In the other cases, many bad triangles, located outside the domain, were generated. Therefore, application of the function needs to be improved.

5.2. Elimination of obsolete triangles located outside the domain

The author's original concept is based on the presumption that for each proper triangle, its center point is located inside the domain. Therefore, it is sufficient to examine the locations of the center points of each triangle:

$$x_s = \frac{x_1 + x_2 + x_3}{3}, \qquad y_s = \frac{y_1 + y_2 + y_3}{3} \tag{12}$$

and to determine whether they are located inside or outside the domain. The same ray-casting algorithm and the function *internal_node*, described in the previous section, may be applied here as well, for (x_s, y_s) coordinates. All triangles, whose center points do not fulfill that condition (i.e., result in 0 function value), have to be removed.

5.3. Elimination of obsolete triangles, located inside the holes

Similarly to the previous subsection, the same assumption may be extended to the triangles located inside the holes. If a proper triangle is located outside the hole (and inside the domain), its center point (12) has to be located outside this hole (inside the domain). Therefore, if the condition for location of a triangle's central point inside the hole is fulfilled (i.e., results in 1 function value), such triangle has to be removed.

5.4. Elimination of "too small" triangles

Due to truncation errors, the *delaunay* function may produce additional bad triangles for the nodes located on the straight skew line. If node coordinates are inaccurate, obsolete triangles with almost zero surface area are generated. Their elimination is crucial in order to keep the mesh free from singularities and ill-conditioning in further calculations.

Surface area P may be evaluated by using the simple Heron formula (Fig. 9)

$$P = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}, \tag{13}$$

where

$$\begin{aligned} p &= \frac{1}{2}(a+b+c), & a &= \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2} \\ b &= \sqrt{(x_3-x_2)^2 + (y_3-y_2)^2}, & c &= \sqrt{(x_3-x_1)^2 + (y_3-y_1)^2}. \end{aligned} \tag{14}$$

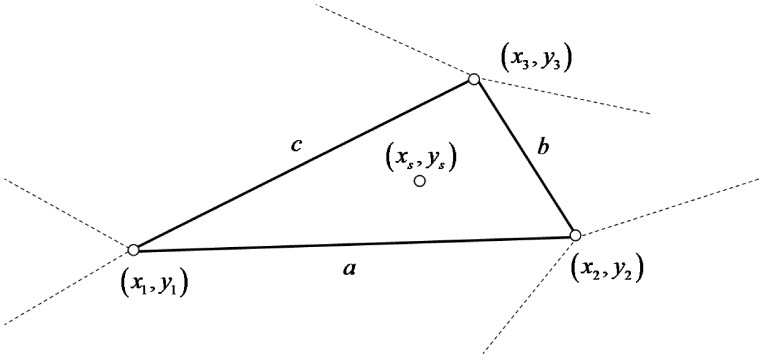

Fig. 9. Geometry of a Delaunay triangle.

Whenever the following criterion is fulfilled:

$$P < 10^{-12}, \tag{15}$$

such triangle has to be removed. It is worth stressing here that the resultant matrix *Triangles* stores surface area for each triangle in its fourth column. The first three columns contain the nodes' numbers of triangles vertices.

After applying additional elimination techniques, one obtains the quasi-final triangular meshes for each selected domain (Fig. 10). For instance, there were four "zero" triangles in the case of the first domain (right top graph). The appropriate text report is displayed in the *Matlab Command Window*.

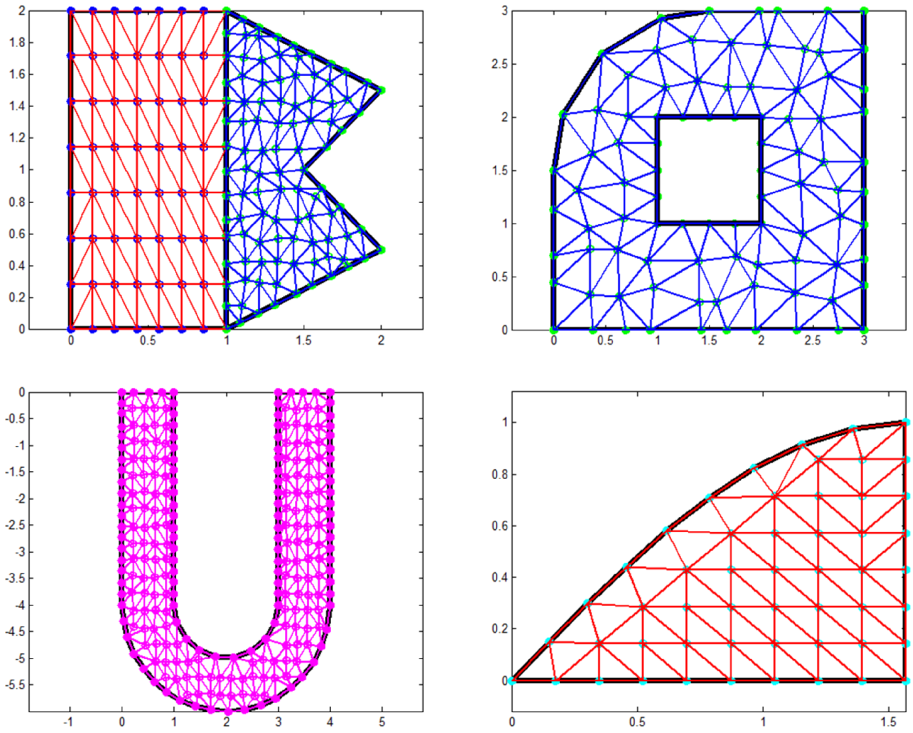

Fig. 10. Examples of proper triangular meshes.

5.5. Final mesh examination and addition of missing triangles

A series of executed tests indicated that the above described techniques may not be sufficient for the generation of a proper triangular mesh. In some cases, there are some triangles missing from the mesh, due to the defective application of the *delaunay* Matlab function near the domain boundary. Moreover, this problem is not caused by the elimination of previous triangles as may be observed for the domain and its mesh in Fig. 11. Figure 11a presents the domain geometry as well as the cloud of nodes, generated according to the ascribed densities. Figure 11b presents direct results of Delaunay tessellation, computed by means of the Matlab built-in function. It is clearly seen here that one triangle is missing from the mesh near the boundary zone of the non-convex part of the domain. Figure 11c shows the results of the elimination of all obsolete "external" triangles – the mesh is still incomplete. Finally, Fig. 11d presents the result of mesh completion (one triangle has been added).

The original algorithm applied here examines all the triangles whose vertex nodes are located on the boundary. Whenever there is no triangle connection between two neighboring boundary nodes, an additional new triangle is added to the mesh, with vertex points at those two nodes and one

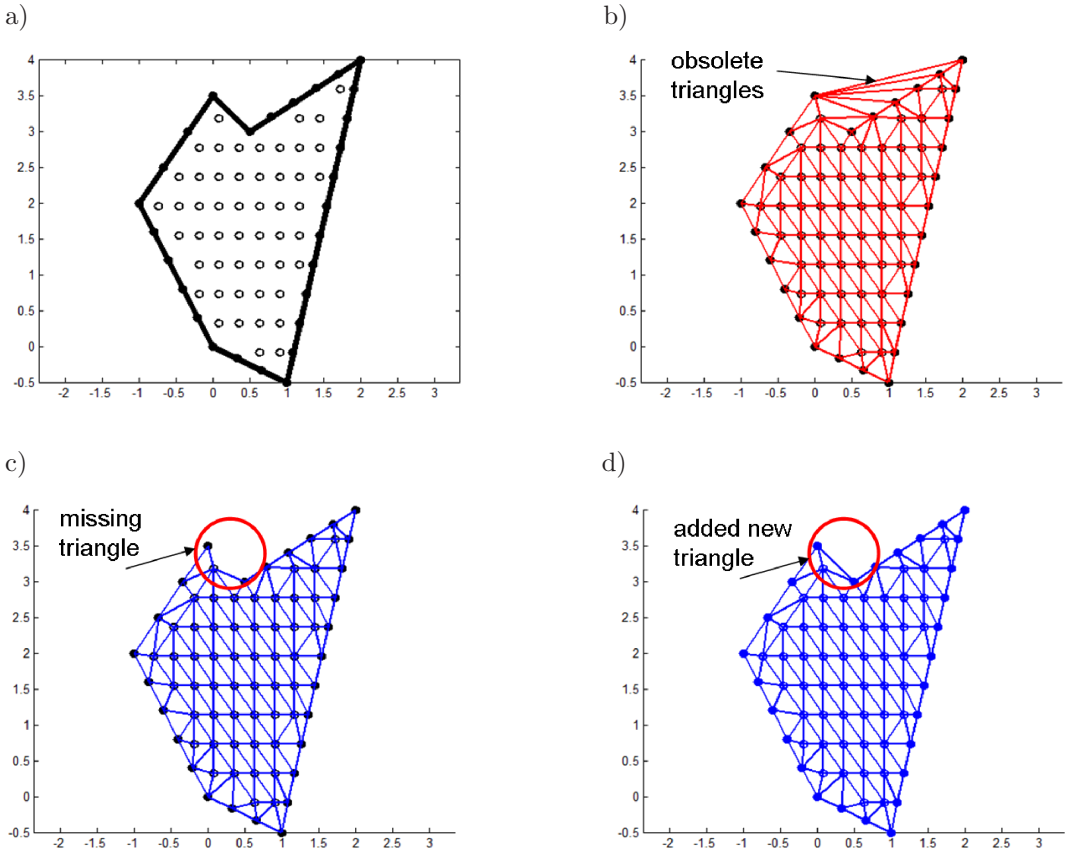

Fig. 11. Example of mesh completion with missing triangle: a) the domain and the cloud of nodes, b) direct result obtained from the "delaunay" function, c) result after the elimination of obsolete triangles, d) result after the mesh completion.

node from the inside of the domain, which is closest to the boundary line. This is the most complex and time-consuming algorithm applied here, since it needs to examine all pairs of boundary nodes, which, in general, do not have to be numbered subsequently. Therefore, the appropriate Matlab code was moved to another function file, named *complete_mesh*. It is called out from the *generate_mesh* function, if the appropriate flag (see *elim* binary vector) is up.

6. Merging of meshes

In this case, there are several subdomains considered (e.g., example #1), the non-conforming meshes are generated by means of the Matlab software, discussed in this paper. This means that on the common edges of two adjacent subdomains, nodes and triangles of those two subdomains may be completely independent from one another. Such approach is becoming increasingly popular nowadays, given that many researchers combine various approximation schemes or even various computational approaches (e.g., MFDM/FEM) in one domain simultaneously. It is then required for both subdomains to have non-conforming clouds of nodes and/or meshes. Afterwards, many various coupling techniques are applied to keep the approximation consistent on all common edges, such as special finite elements in the transition zone, hanging/fictitious nodes or additional integrals calculated along the common edges that join two approximations together with the ascribed accuracy.

Therefore, non-conforming clouds of nodes in different subdomains are left without any modifications although information concerning the common parts may be still obtained. The appropriate Matlab function *common_edges.m* is provided, in which all subdomains are examined and one addi-

tional topological matrix is determined. This matrix stores the information concerning all boundary parts, which are common for two or more adjacent subdomains.

The above discussion does not concern domains with holes. Although the main domain and its holes are defined as separate subdomains, all the nodes generated on the boundary of a hole are ascribed to the main domain and together with its nodes, they constitute one consistent cloud of nodes.

Selected results are presented in Fig. 12. Figure 12a shows two non-conforming meshes, related to two adjacent subdomains, whereas Fig. 12b shows one consistent mesh for the entire domain at once. In the first case, two subdomains (rectangle and triangle) have to be defined separately, whereas in the second case, only one polygon has to be defined.

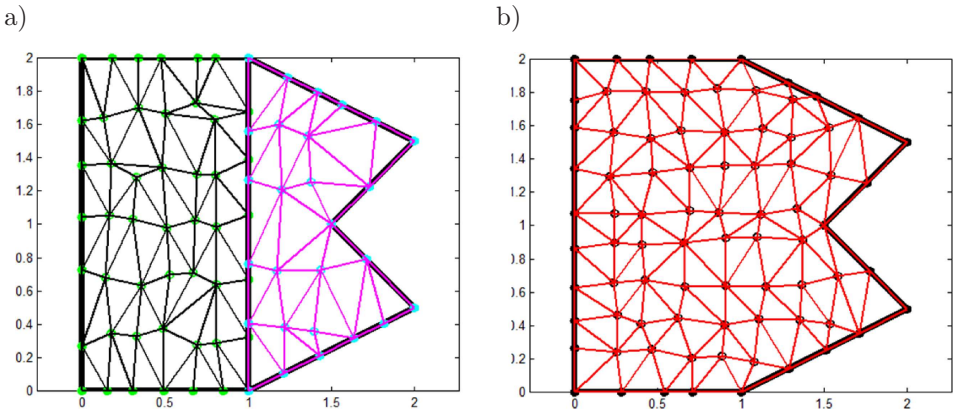

Fig. 12. Examples of non-conforming meshes and one consistent mesh: a) example #1 – two independent meshes, b) example #2 – one consistent mesh.

7. Numerical analysis of the boundary value problems

Software for cloud of nodes and triangular mesh generation, considered in this paper, is especially dedicated to numerical analysis of the boundary value problems of mechanics and civil engineering. It applies variety of computational methods and commercial packages. However, to keep all benchmark tests consistent, cloud of nodes and mesh generator were examined on the *MFDMtool* Matlab toolbox, which was designed and developed by the author of this paper. The *MFDMtool* toolbox is available on-line – the details are given in the introductory section (Sec. 1) of this paper.

The *MFDMtool* is a Matlab toolbox [36], based upon MFDM [18, 34] for the numerical determination of shear stresses

$$\tau = \sqrt{\tau_{zx}^2 + \tau_{zy}^2}, \qquad \tau_{zx} = \frac{\partial F}{\partial y}, \qquad \tau_{zy} = -\frac{\partial F}{\partial x} \tag{16}$$

in a prismatic bar of the specified cross-section subject to a torsional moment. Such problem may be posed in local formulation

$$\begin{cases} \nabla^2 F = -2G\theta & \text{in} \quad \Omega, \\ F = 0 & \text{on} \quad \partial\Omega, \end{cases} \tag{17}$$

as well as in the (weak) variational (Galerkin) one: Find such (trial) function $F \in H_0^1$ that for any (test) function $v \in H_0^1$ is satisfied

$$\int_\Omega \left(\frac{\partial F}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial F}{\partial y}\frac{\partial v}{\partial y} \right) \mathrm{d}\Omega = 2G\theta \int_\Omega v \mathrm{d}\Omega. \tag{18}$$

Here, $F = F(x, y)$ is a scalar Prandtl function (unknown primary solution), G is a Kirchhoff modulus (material parameter) and θ is a torsional angle (load). The local formulation (17) may be directly applied to MFDM (no mesh is required then). However, for FEM and variational MFDM (both using (18)) analyses, additional integration mesh/element mesh is required.

All details concerning the numerical aspects of the MFDM solution approach, along with the detailed description of all Matlab files included in the *MFDMtool* are given in [36] as well as in the instruction manual attached to it. The appropriate Matlab code, which is responsible for running the numerical calculations, is given in the *TEST_MFDM.m* script file, assuming that the cloud of nodes and mesh are already generated. Afterwards, numerical solution of the Prandtl problem is obtained and plotted, according to the desired formulation (*formulation* = 1 – local (17) and 2 – variational (18)). The following command lines are executed here in the same way, as prepared in the appropriate testing m-file, attached to the original *MFDMtool.m* toolbox. Moreover, all original functions of this toolbox (*star*, *mwls*, *localMFDM*, *varMFDM*) are provided here once again as well, for the sake of convenience for the potential users.

A number of selected results of the application of the considered nodes-mesh generator to the analysis of problems (17) and (18) are presented in Figs. 13–16. Calculations were performed for four examples, considered in the previous sections, i.e., example #1 (the second subdomain with straight skew edges, Fig. 13), example #2 (domain with a curved side and one hole, Fig. 14), example #3 (U-shaped domain, Fig. 15) as well as example #4 (domain with sine-like part of the boundary, Fig. 16), though with much denser discretizations (Fig. 17). Figures 13 and 15 present results for the local formulation (17), whereas Figs. 14 and 16 show results for the variational formulation (18). The MFD solution approach was applied in all the cases. In each figure, the following graphs are presented: primary solution $F(x, y)$ (Prandtl function, top left), shear stress components (16) (top right and bottom left) and the total stress (bottom right). The nodal solution was continued and smoothed by using the MWLS approximation [3, 11, 18]. For graphical purposes, mesh of triangles and Matlab function *patch* were successfully applied.

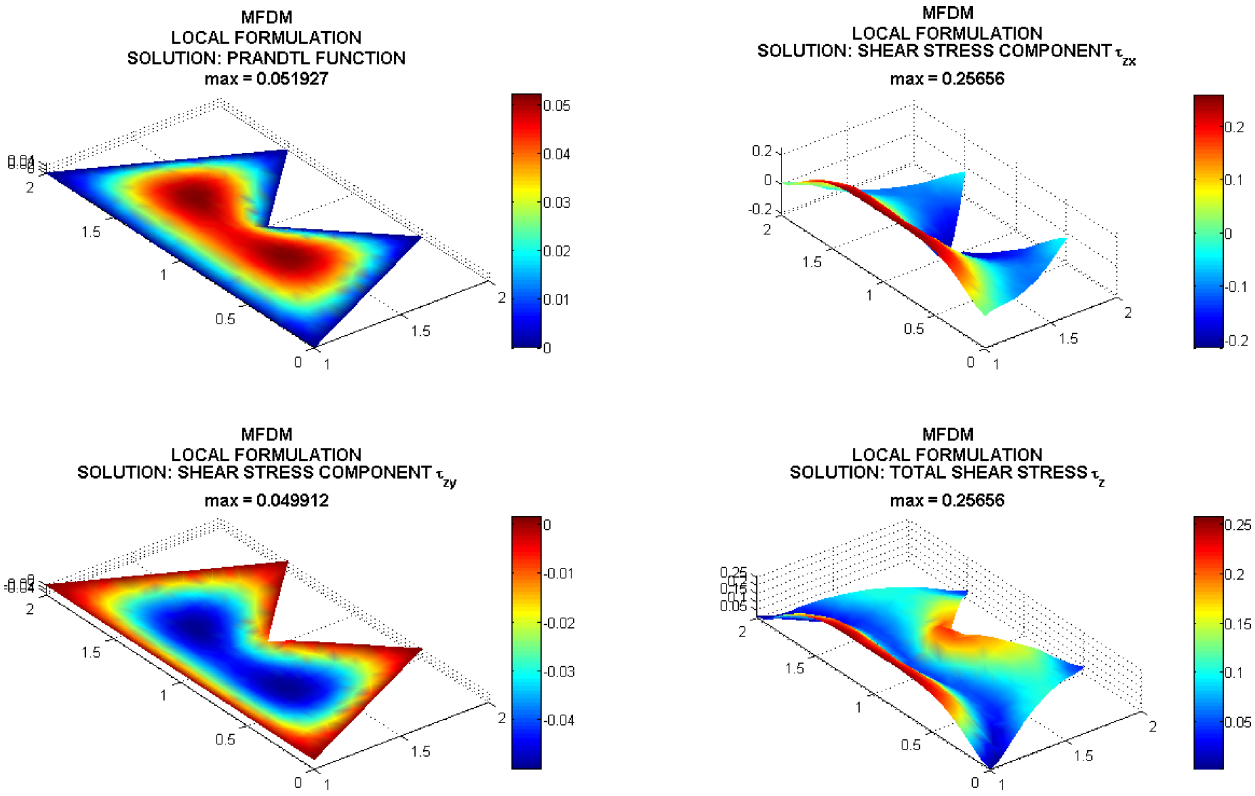

Fig. 13. Results of the Prandtl problem (local formulation) for example #1.

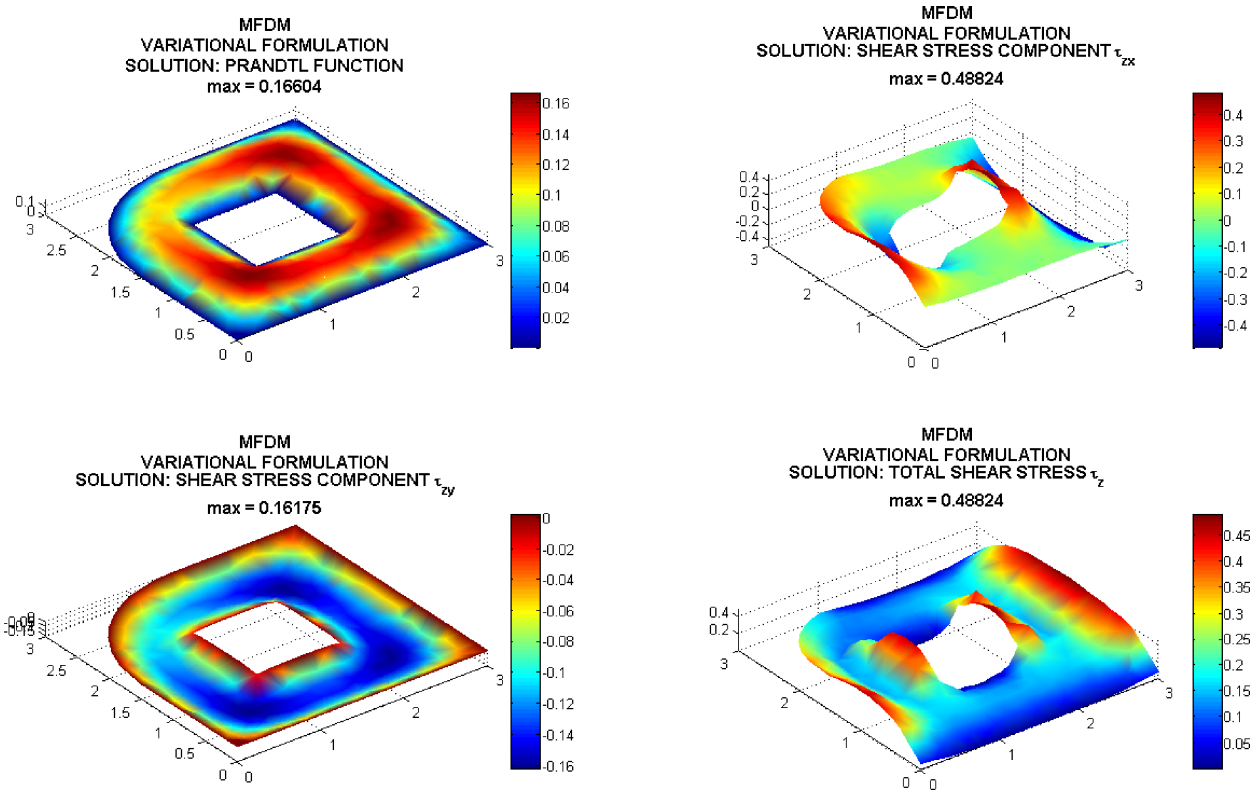

Fig. 14. Results of the Prandtl problem (variational formulation) for example #2.

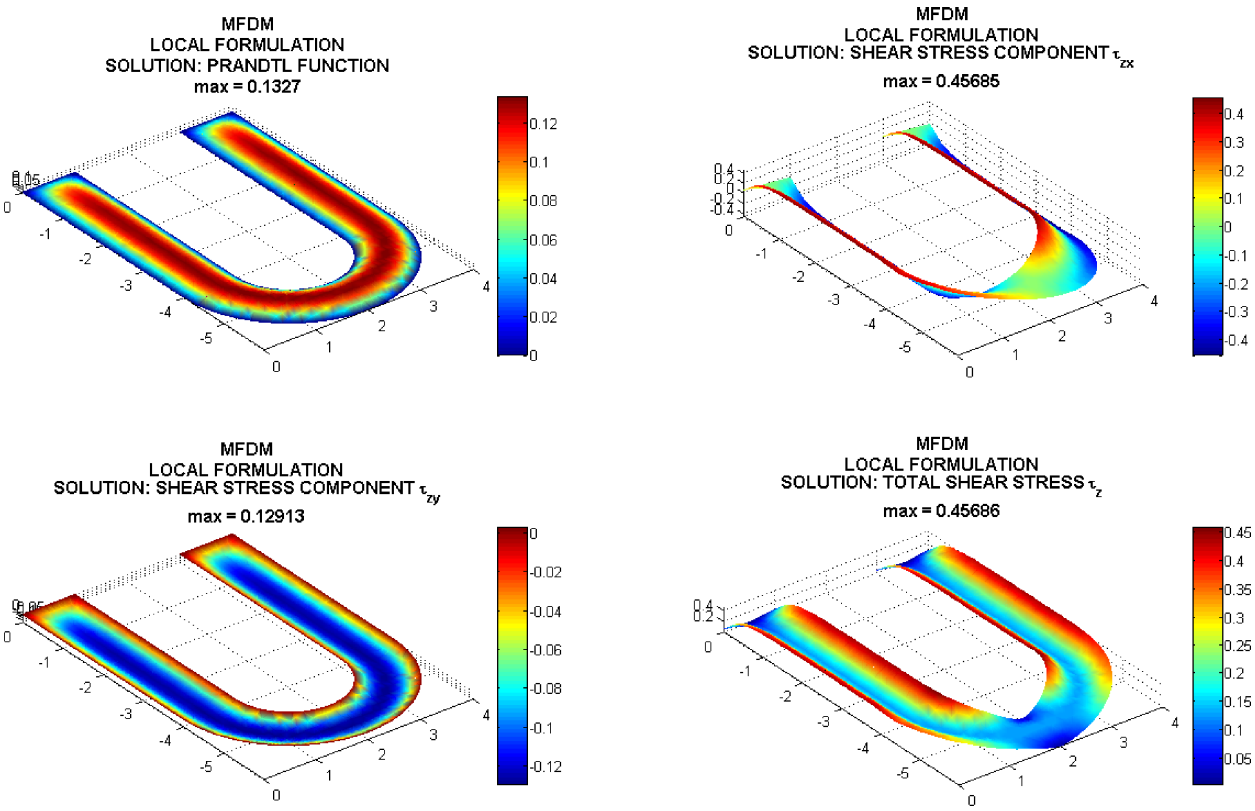

Fig. 15. Results of the Prandtl problem (local formulation) for example #3.

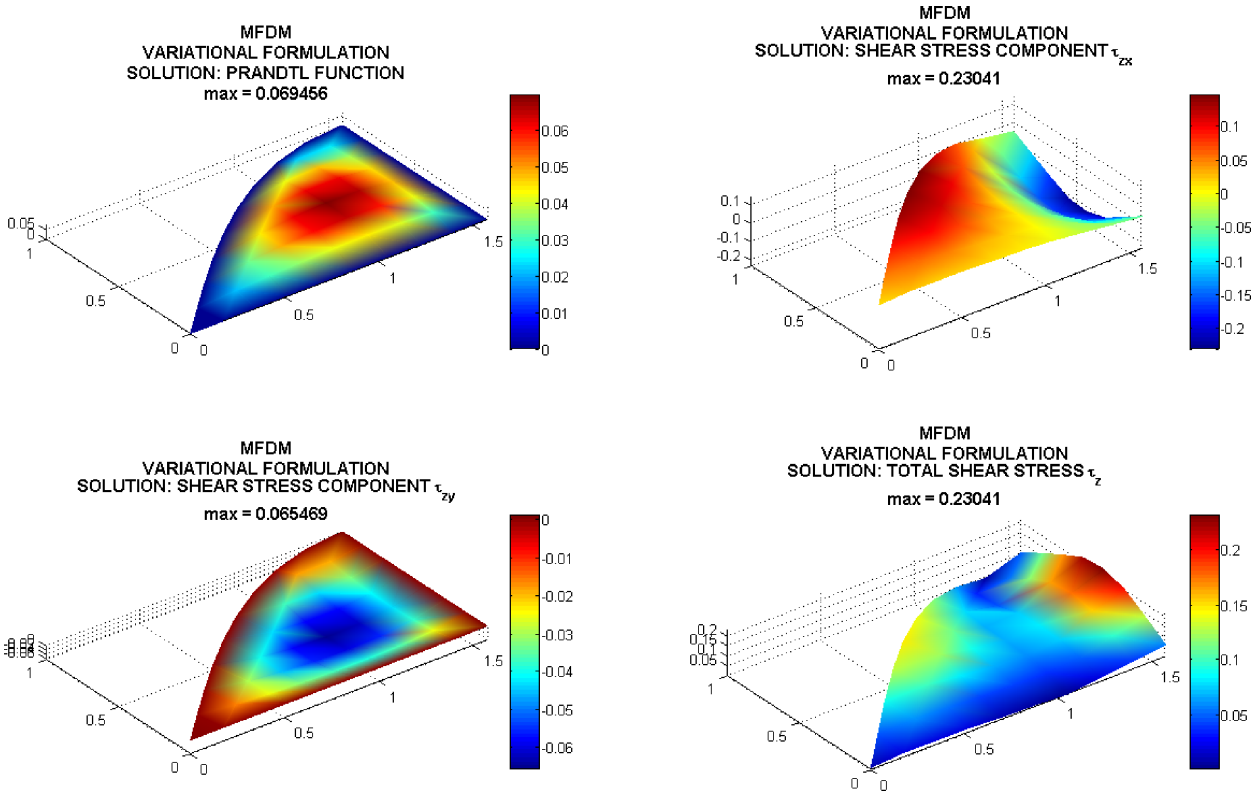

Fig. 16. Results of the Prandtl problem (variational formulation) for example #4.

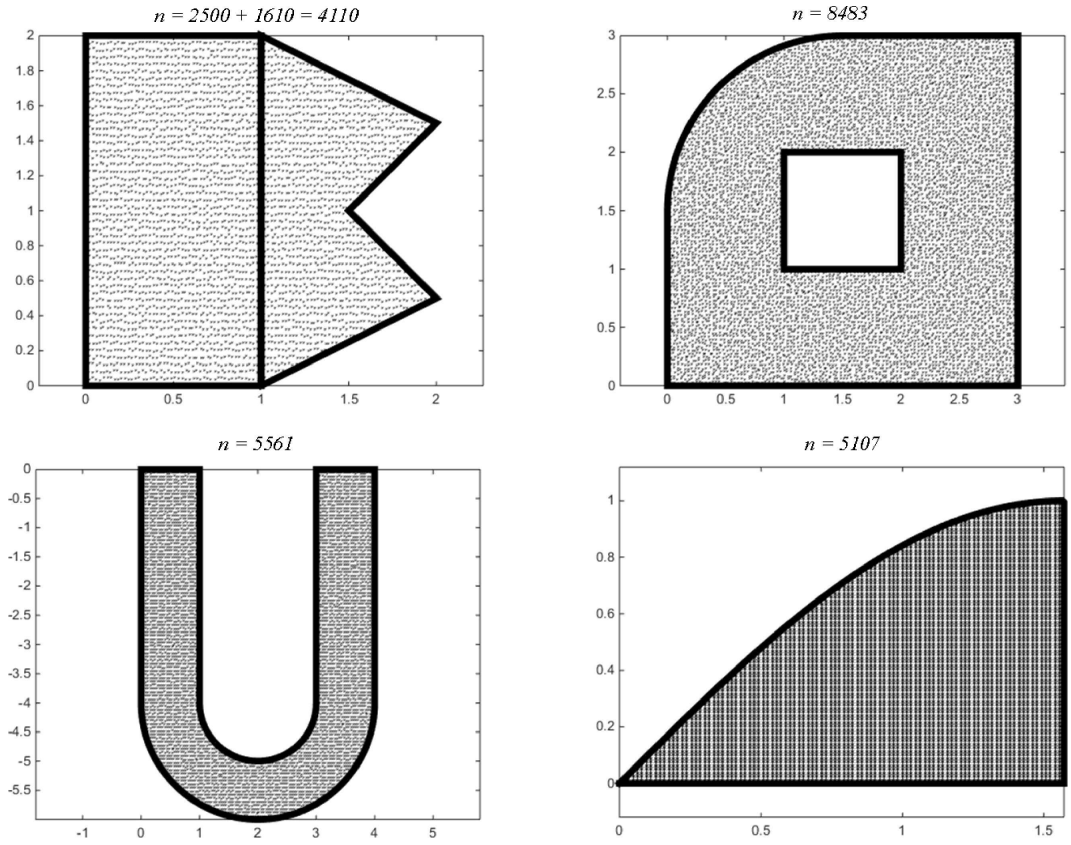

Fig. 17. Discretized domains applied to the Prandtl problem analysis.

8. Conclusions

In this paper, the Matlab toolbox *nodes_mesh_tool* was proposed and tested. It is designed for effective generation of both clouds of nodes (without any imposed structure) and triangular meshes for 2D cases. This toolbox may be applied in numerical analysis of the boundary value problems in mechanics and civil engineering. The most important features of the applied generation algorithms are:

- Parallel analysis of several disjoint or adjacent subdomains is possible.
- Boundary representation and topological information are required only.
- Generation of nodes is performed according to the ascribed densities.
- Regular nodes distributions and randomly disturbed clouds of nodes may be applied.
- Domain boundary may consist of straight lines and curved edges of any type. In the case of curved edges, nodes are determined with constant angle length.
- Domains with holes may be analyzed as well.
- All techniques are based on simple geometrical derivations.

The underlying assumption of all provided Matlab codes is that they are based on the well-known Matlab functions (*meshgrid, delaunay*) which are extended here to more general cases. All m-files may be applied to larger parts of codes without any difficulties. They produce vectors and matrices with clear geometrical interpretations. All procedures were examined on the selected examples. Both generation algorithms and numerical analysis of the 2D Poisson problem were taken into consideration. Future work may include

- Cloud of nodes/mesh generator for 3D domains, applying most techniques derived in this paper with improved implementation of curves and surfaces, given in more general parametric manner.
- Further examination and development of Matlab functions for geometrical applications (e.g., *voronoi* Matlab function).
- Development of the irregular clouds of nodes generator, with node distribution, controlled by the *a-posteriori* error estimation [16] of the solution of the boundary value problem. Such case assumes extension of the software discussed here to cloud of nodes/mesh refinement according to the desired accuracy related to the analyzed problem.

Aknowledgements

This research was supported by the National Science Centre, Poland, under the scientific project 2015/19/D/ST8/00816 ("Computational coupled FEM/meshless FDM analysis dedicated to engineering nonstationary thermo-elastic and thermo-plastic problems").

References

[1] M. Shimrat. Algorithm 112: position of point relative to polygon. *Communications of the ACM*, **5**(8), 434 pages, 1962.
[2] T. Liszka, J. Orkisz. The finite difference method at arbitrary irregular grids and its applications in applied mechanics. *Computers & Structures*, **11**: 83–95, 1980.
[3] P. Lancaster, K. Salkauskas. Surfaces generated by moving least squares method. *Mathematics of Computation*, **155**(37): 141–158, 1981.
[4] T. Liszka. An interpolation method for an irregular net of nodes. *Int. J. Num. Meth. Eng.*, **20**: 1599–1612, 1984.

[5] Matlab C Math Library. User's Guide. The MathWorks Inc., 1984-2015.
[6] F.P. Preparata, M.I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag Berlin- Heidelberg, 1985.
[7] K.E. Atkinson. *An Introduction to Numerical Analysis*. Wiley Ed., NewYork, 1988.
[8] B.A. Barsky, T.D. DeRose. Geometric continuity of parametric curves: three equivalent characterizations. *IEEE Comput. Graph. and Appl.*, **9**: 60–68, 1989.
[9] L. Piegl. Modifying the shape of rational B-splines. Part 1: curves. *Computer-Aided Design*, **21**(8): 509–518, 1989.
[10] J. Krok, J. Orkisz. A unified approach to the FE and generalized variational FD methods in nonlinear mechanics, concepts and numerical approach. *Discretization Methods in Structural Mechanics*, **1**: 353–362, 1990.
[11] P. Lancaster, K. Salkauskas. *Curve and Surface Fitting*. Academic Press Inc., 1990.
[12] K. Weiler. An incremental angle point in polygon test. [In:] Paul S.Heckbert [Ed.], *Graphics Gems IV*, pp. 16–23. Academic Press Professional Inc., San Diego, CA, USA, 1994.
[13] T. Belytchko, Meshless methods: an overview and recent developments. *Comp. Meth. Appl. Mech. Engng.*, **139**: 3–47, 1996.
[14] D. Hegen. Element-free Galerkin methods in combination with finite element approaches. *Comp. Meth. Appl. Mech. Engng.*, **135**: 143–166, 1996.
[15] J.R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. *Applied Computational Geometry Towards Geometric Engineering,* Springer, Berlin-Heidelberg, pp. 203–222, 1996.
[16] M. Ainsworth. J.T. Oden. A-posteriori error estimation in finite element analysis. *Comp. Meth Appl. Mech Engng.*, **142**: 1–88, 1997.
[17] F. Hecht. BAMG: bidimensional anisotropic mesh generator. INRIA Report, 1998.
[18] J. Orkisz. Finite Difference Method (Part III). [In:] M. Kleiber [Ed.], *Handbook of Computational Solid Mechanics*, pp. 336–431, Springer-Verlag, Berlin, 1998.
[19] J. Alberty, C. Carstensen, S.A. Funken. Remarks around 50 lines of Matlab: short finite element implementation. *Numerical Algorithms*, **20**: 117–137, 1999.
[20] N. Gershenfeld. *The Nature of Mathematical Modeling*. Cambridge University Press, ISBN 0-521-57095-6, 1999.
[21] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery. *Numerical Recipes in Fortran 90. The Art of Parallel Scientific Computing*. Cambridge Univ. Press, 1996.
[22] A. Huerta, S. Hernandez-Menez. Enrichment and coupling of the finite element and meshless methods. *Int. J. Numer. Methods Eng.*, **48**: 1615–1630, 2000.
[23] X.Y. Li, S.H. Teng, A. Üngör. *Point placement for meshless methods using sphere packing and advancing front methods*. [In:] Proceedings of ICCES'00 – International Conference on Computational Engineering Science, Los Angeles, 2000.
[24] S. Teng, X.Y. Li, A. Üngör. Generating a good quality point set for the meshless methods. *Comput. Model. Eng. Sci.*, **1**(1): 1017, 2000.
[25] T.P. Fries, H.G. Matthies. *Classification and Overview of Meshfree Methods*. Technische Universität Braunschweig, 2004.
[26] R. Löhner, E. Oñate. A general advancing front technique for filling space with arbitrary objects. *International Journal for Numerical Methods in Engineering*, **61**(12): 1977–1991, 2004.
[27] O.C. Zienkiewicz, R.L. Taylor. *Finite Element Method: Its Basis and Fundamentals*, Elsevier, 6th edition, 2005.
[28] C. Carstensen, S. Bartels, A. Hecht. P2Q2Iso2D= 2D Isoparametric FEM in Matlab. *J. Comput. Appl. Math.*, **192**: 219–250, 2006.
[29] C. Drumm, S. Tiwari, J. Kuhnert, H.-J. Bart. Finite pointset method for simulation of the liquid-liquid flow field in an extractor. *Computers and Chemical Engineering*, **32**(12): 2946–2957, 2008.
[30] J.A. Cottrel, T.J.R. Hughes, Y. Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. John Wiley and Sons, ISBN 978-0-470-74873-2, 2009.
[31] Y. Nie, W. Zhang, Y. Liu, L. Wang. A node placement method with high quality for mesh generation. *IOP Conference Series: Materials Science and Engineering*, Vol. 10, IOP Publishing, 2010.
[32] C. Pearce, L. Kaczmarczyk, J. Novak. Multiscale modelling strategies for heterogeneous materials. *Comuputational Technology Reviews*, **2**: 23–49, 2010.
[33] S. Funken, D. Praetorius, P. Wissgott. Efficient implementation of adaptive P1-FEM in MATLAB. *Computational Methods in Applied Mathematics*, **11**(4): 460–490, 2011.
[34] S. Milewski. Meshless finite difference method with higher order approximation-applications in mechanics. *Archives of Computational Methods in Engineering*, **19**(1): 1–49, 2012.
[35] I. Jaworska. On the ill-conditioning in the new higher order multipoint method. *Computers and Mathematics with Applications*, **66**(3): 238–249, 2013.
[36] S. Milewski. Selected computational aspects of the meshless finite difference method. *Numerical Algorithms*, **63**: 107–126, 2013.

[37] Z. Ullah, W. Coombs, C. Augarde. An adaptive finite element/meshless coupled method based on local maximum entropy shape functions for linear and nonlinear problems. *Computer Methods in Applied Mechanics and Engineering*, **267**: 111–132, 2013.
[38] S. Kumar, I. Singh, B. Mishra. A coupled finite element and element-free Galerkin approach for the simulation of stable crack growth in ductile materials. *Theoretical and Applied Fracture Mechanics*, **70**: 49–58, 2014.
[39] Y. Nie, W. Zhang, N. Qi, Y. Li. Parallel node placement method by bubble simulation. *Computer Physics Communications*, **185**(3): 798–808, 2014.
[40] P.O. Persson, G. Strang. A simple mesh generator in MATLAB. *SIAM Review*, **46**(2): 329–345, 2014.
[41] J. Jaśkowiec, S. Milewski. The effective interface approach for coupling of the FE and meshless FD methods and applying essential boundary conditions. *Computers and Mathematics with Applications*, **70**(5): 962–979, 2015.