FAST GENERATION OF VARIABLE DENSITY NODE DISTRIBUTIONS FOR MESH-FREE METHODS

JURE SLAK & GREGOR KOSEC
Parallel and Distributed Systems Laboratory, "Jožef Stefan" Institute, Slovenia

ABSTRACT

Problems with pronounced differences in intensity, such as those that appear in contact mechanics, where locally concentrated high stresses are present, are usually attacked with a spatially variable nodal distribution. In a meshless context, such distribution has to be quasi-smooth with minimal spacing requirements to produce satisfactory results. To that end, development of fast discretization procedures, which distribute the nodes according to given (non-constant) spacing function, has become of interest. We improve a recently published algorithm for fast 2D meshless discretizations, by lowering its time complexity from $O(NS)$ to $O(N \log S)$ resulting in an algorithm that generates a million nodes per second. The proposed algorithm is independent of the dimensionality of the space, does not rely on a coordinate system and has provable minimal node spacing requirements. The original and new algorithms are compared in terms of node quality and execution time. The usability and robustness of the new algorithm is presented by solving PDE examples on irregular 3D domains with the RBF-FD method and by using it as a node generation algorithm in fully the automatic adaptive solver for linear elasticity.

Keywords: node generation, adaptive discretizations, mesh-free methods, RBF-FD, PDEs.

1 INTRODUCTION

One of the most attractive features of mesh-free methods is their ability to work on unstructured sets of nodes without any connectivity relations. This does not imply that any distribution of nodes will produce satisfying results, as especially many strong form methods have minimal spacing requirements that impact stability [2] and need locally regular distributions for good performance. An example of such a method is the Radial Basis Functions Finite Differences (RBF-FD) method [1]. Nonetheless, node positioning is an easier problem than mesh generation and specialized algorithms for fast generation of nodes on irregular domains have been developed [3], [4]. Often the nodes are required to be approximately distributed according to a spacing function, allowing for variable density node distributions to be generated. The need for such distributions arose from problems with pronounced differences in intensity, such as contact problems in linear elasticity. Furthermore, the high-intensity regions might not be known in advance and adaptive iteration is needed to solve the problem. In such cases, it is essential for the node positioning algorithm to be fast and robust, to be able to generate quality distributions as required by the error indicator automatically.

In this work, we improve a recently published algorithm by Fornberg and Flyer for 2D meshless discretizations [3] by improving its time complexity and generalize it to higher dimensions. We propose and alternative algorithm in Section 3 and compare the algorithms in terms of execution time and node quality in Section 4. In Section 5, the practical application of the new algorithm is demonstrated on irregular 3D domains and in the fully automatic adaptive setting.

2 IMPROVEMENTS OF 2D ALGORITHM BY FORNBERG AND FLYER

The algorithm by Fornberg and Flyer [3] fills a rectangle with nodes spaced according to the spacing function δr. The algorithm works as an advancing front algorithm that begins by

WIT Transactions on Engineering Sciences, Vol 122, © 2019 WIT Press
www.witpress.com, ISSN 1743-3533 (on-line)
doi:10.2495/BE410151

filling the bottom line of the rectangle with candidates for node positions. On every iteration, a point p with minimal y-coordinate is chosen and is added to the solution set. The point is then removed from the candidate list as well as other points being too close to it. A small constant number, e.g. 5, of new candidates are then computed, by spacing them uniformly by angle on an arc of radius $\delta r(p)$ spanning between the closest remaining left and right candidates, and are inserted into the list. The procedure finishes when the minimal y-coordinate is above the upper rectangle boundary.

A naïve implementation of the above algorithm using arrays has time complexity $O(NS)$, where N is the number of generated nodes and S is the maximal number of points in the candidate list array. For uniform nodal spacing h, it holds that $N = O(1/h^2)$ and $S = O(1/h)$, bringing the total time complexity to $O(1/h^3)$.

Additionally, if an irregular 2D domain is treated, the algorithm first fills its bounding box, discards all the nodes outside the domain and nodes too close to the boundary. The boundary discretization is then superimposed on the remaining nodes, and the distribution is then further regularized using a few iterations of a repel-type algorithm.

2.1 Time complexity improvements

The naïve implementation of the above algorithm can be improved by taking note of the required operations in each iteration. First, a point p with minimal y-coordinate is found, then all candidates in radius $\delta r(p)$ are removed from the candidate list and some new candidates are inserted instead. The insertions and removals can be sped up using a linked list instead of an array, performing them in constant time from a known position. The complexity of the search operation is not changed and as the candidates are ordered according to the x-coordinate, only a few candidates left and right from the known position must be checked. The only operation left is the minimum extraction, which can be done fast, if additionally, a data structure that allows fast minimum extraction (e.g. a heap) is used in the combination with a linked list. The heap contains references to the linked list elements and extracting the minimum is now an $O(\log S)$ operation. Introducing the heap requires some more bookkeeping at insertions and removals. Upon insertion of a candidate in the linked list, a reference to it must also be inserted in the heap. Upon removal of the candidate from the linked list, it is lazily marked as removed and al the removed elements are ignored at the next minimum extraction. This brings the complexity of one iteration down to a few constant time updates of the linked list and $O(\log S)$ time updates of the heap, making the overall time complexity $O(N \log S)$. For uniform nodal spacing this results in a time complexity $O(1/h^2 \log 1/h)$, improving the $O(1/h^3)$ complexity of the naïve algorithm.

Both proposed implementations were compared in practice using uniform nodal spacing h. They were implemented in C++11, compiled using g++v7 on Linux and run on a laptop computer with 4 Intel® Core™ i7-7700HQ CPU @ 2.80 GHz cores and 16 GB DDR4 RAM. The execution times are shown in Fig. 1.

As expected, the naïve implementation is faster for small N up to around $N = 10\,000$ due to less bookkeeping overhead. For large N the improved implementation with better asymptotic behaviour is faster and generates a million nodes under one second.

2.2 Generalization to higher dimensions

The algorithm and the efficient implementation can be extended to higher dimensions $d \geq 2$. Instead of a line, the advancing front forms a $d - 1$ dimensional plane. In each iteration, the

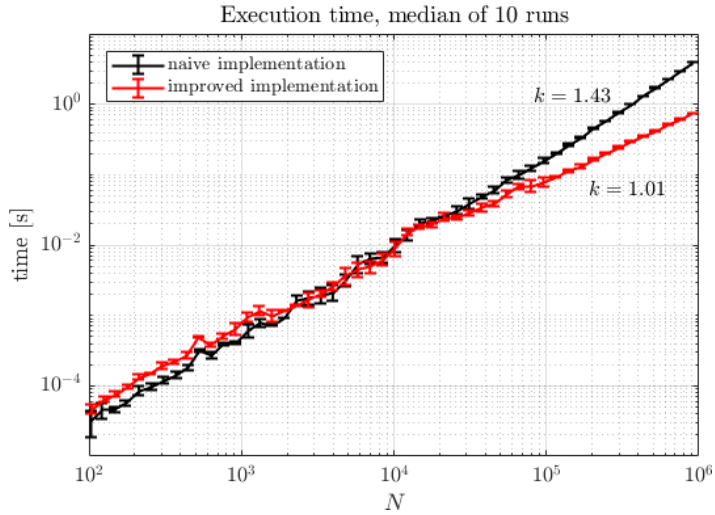

Figure 1: Execution time of naive and improved implementation. The error bars indicate standard deviation over 10 runs.

candidate p with the minimal d-th coordinate is extracted and its neighbourhood in radius $\delta r(p)$ is removed. New candidates are then inserted instead. It is not entirely obvious how to generalize the definition of new candidates in higher dimensions. One direct generalization would be to find the two closest remaining points to p along each of the $d - 1$ dimensions, one in each direction. The angles between those points would then be split into uniform parts, thus defining the angle of the new candidate along each dimension. The scale is given by $\delta r(p)$ as in 1D case. In other words, angles of the new candidates form a tensor product mesh and the points are scaled to have radius equal to $\delta r(p)$.

Efficient implementation can also be generalized to higher dimensions: instead of a sorted doubly linked list with a heap, a spatial search structure, such a k-d tree must be used in combination with a heap to achieve similar time complexity. This makes the implementation somewhat more complex and out of the scope of this work.

2.3 Minimal spacing guarantees

It is often desirable for node generation algorithms to satisfy minimal node spacing guarantees, as small distances between nodes can cause ill conditioning in numerical methods [2]. If the nodes were generated using a constant density function $\delta r(p) = \delta r_0$, it is reasonable to expect that $\|p_i - p_j\| \geq \delta r_0$ holds for $i \neq j$. For variable density functions, some constraints were introduced by [5], were assumptions on the Lipschitz constant of the function must be made. Even if no such assumptions can be made (and none are needed by any of the algorithms in this work), a weak requirement $\|p_i - p_j\| \geq \min \delta r$, for $i \neq j$ can still be made. However, the algorithm by Fornberg and Flyer does not satisfy even that, as shown in Fig. 2.

A unit square was filled with uniform density $h = 0.01$ and marked points violate the proximity criterion. Most of the violations appear on the boundary, with some also appearing sporadically in the domain interior.

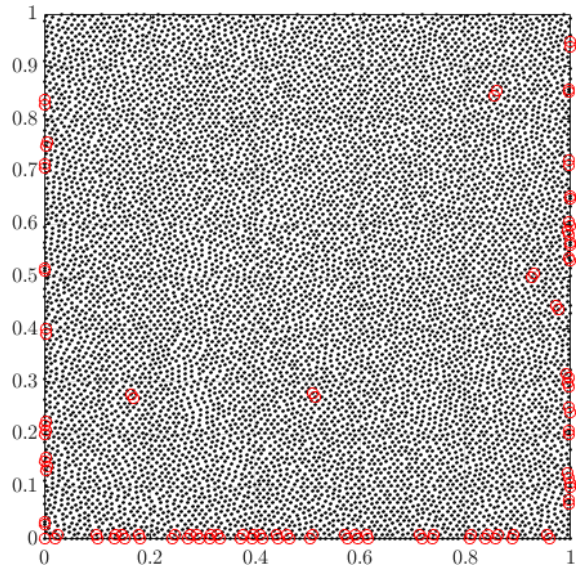

Figure 2: A sample run with marked proximity violations.

3 POISSON DISK SAMPLING BASED ALGORITHM

The algorithm proposed by Fornberg and Flyer has a few undesired properties. First, it has no minimal spacing guarantees, it is directionally dependent, and it does not work for arbitrary shapes without additional algorithms. Therefore, we propose an alternative algorithm for node generation, which is not directional, it is dimension independent, works with irregular domains, and satisfies minimal spacing criteria.

The algorithm starts with a given boundary discretization and works inwards. All boundary nodes and optionally some other starting user-supplied nodes are placed in a queue of *non-expanded* nodes. In each iteration, a new node p is popped from the queue and new candidates arising from it are considered. The candidates are considered one by one. If a candidate is outside of the domain, it is removed from consideration immediately. Otherwise, if it is not too close to any existing nodes, it is accepted and added to the queue and the result set. This iteration continues until the queue is empty.

The new candidates arising from each point can be generated in a few different ways: either chosen uniformly at random from a sphere with radius $\delta r(p)$ or generated in a deterministic way. One good way is such that hex packing is reproduced in 2D and 3D when using uniform spacing.

3.1 Time complexity

Each iteration consists of considering one node and adding a few new nodes in the queue. The most time-consuming operation is to check that new candidates are not too close to the existing nodes. There are different approaches to answering these types of queries, using e.g. background cell meshes or spatial trees. If k-d trees are used, then insertions and queries can be performed in $O(n \log n)$, where n is the number of points in the tree. As each candidate is inserted in the tree at most once and queries at most once, the overall time complexity is $O(N \log N)$.

4 COMPARISON OF ALGORITHMS

4.1 Node quality

Both algorithms are first compared in terms of node quality. A sample function

$$\delta r(x,y) = \frac{1}{8.5979}\left(3(x-1)^2 e^{-x^2-(y+1)^2} - 5e^{-x^2-y^2} + 4.2166\right)\cdot(H-h) + h \quad (1)$$

was used to compare and asses the quality of both algorithms. The function is such that H is the largest and h is the smallest node spacing used. Values $H = 0.1$ and $h = 0.01$ were used in our examples. The generated nodes are shown in Fig. 3.

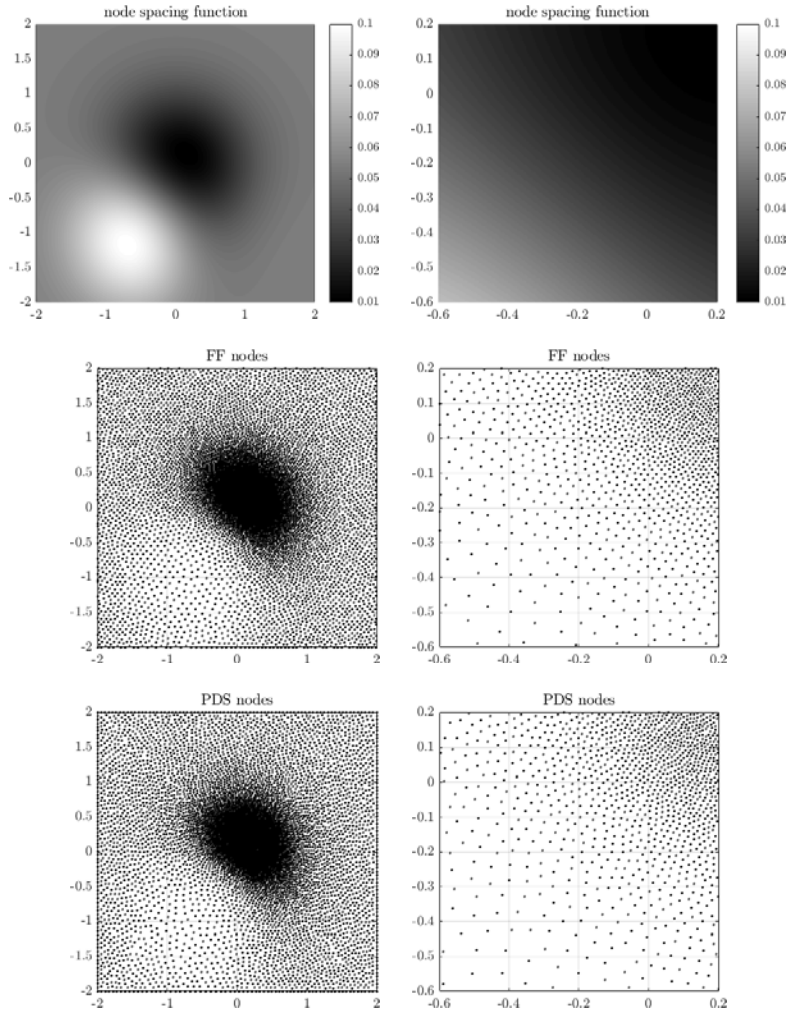

Figure 3: The left column represents the density function, the nodes generated by the algorithm published by Fornberg and Flyer (FF) and the nodes generated by the Poisson Disk Sampling (PDS) based algorithm, respectively. The right column shows zoomed in versions of the left column to better asses the node quality.

The node distributions produced by both algorithms do clearly conform to the specified nodal spacing function. Additionally, both distributions are locally regular as shown in the zoomed in plots. Further analysis of node quality can be done by comparing generated node distributions against the theoretical distribution, shown in Fig. 4.

Both algorithms produce a matching distribution of nodal spacing, with 9,277 and 8,693 nodes, respectively.

4.2 Execution time

Execution times of both algorithms are compared. The same density function as before (eqn (1)) was used. The procedure was run using $H = 10h$ with h varying in such a way that the number of generated nodes ranged approximately from 100 to 1,000,000. The domain of choice was an annulus, centred at $(0, 0)$ with inner radius 0.25 and outer radius 2. The proposed Poisson Disk Sampling based algorithm was applied directly, while the algorithm by Fornberg and Flyer was applied to the bounding box $[-2, 2] \times [-2, 2]$ and generated nodes were later filtered to keep only the nodes inside the domain and not too close to the domain boundary. The results are shown in Fig. 5.

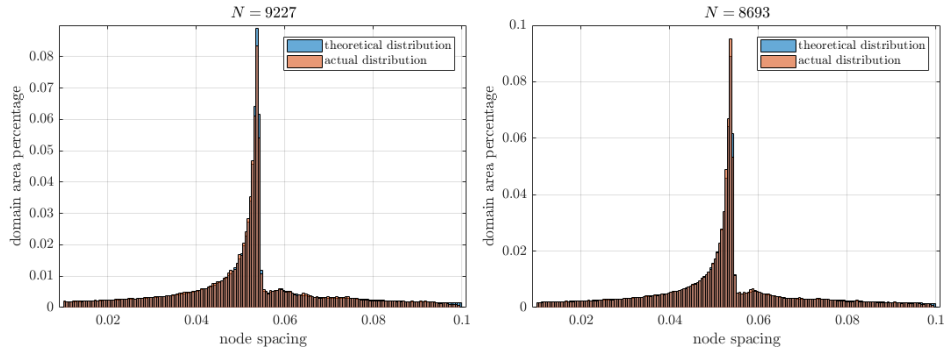

Figure 4: Spacing distribution produced by FF (left) and PDS algorithms (right).

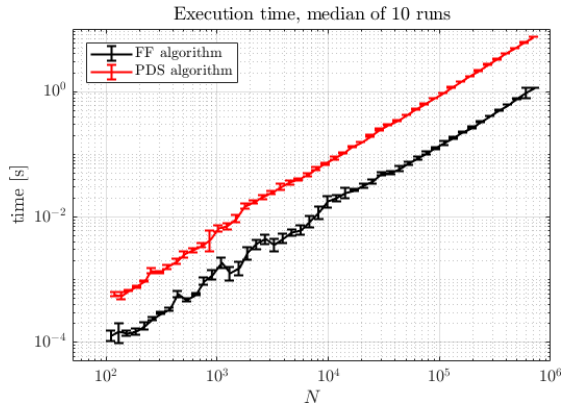

Figure 5: Comparison of execution time of both algorithms on an annulus domain. Shown execution time is the median of 10 runs, with error bars representing standard deviation.

An irregular domain was used on purpose to include a moderate overhead that the first algorithm has due to generating nodes within the whole bounding box. Nonetheless, it outperforms the Poisson Disk Sampling based algorithm for all tested N and exhibits very similar time complexity.

5 NUMERICAL EXAMPLES

The numerical examples in this work serve to demonstrate that nodes generated by the Poisson Disk Sampling based algorithm described in Section 3, can be used to solve PDEs using the RBF-FD method.

5.1 Electrostatics in 2D

Let us begin with a simple two-dimensional example originating from electrostatics. The electric field $\vec{E}$ satisfies $\nabla \cdot \vec{E} = \rho/\varepsilon$ and $\nabla \times \vec{E} = 0$. Introducing the electric potential ϕ, such that $\vec{E} = -\nabla\phi$, we get the Poisson's equation

$$\nabla^2\phi = -\rho/\varepsilon. \tag{2}$$

In the absence of charge, this becomes the Laplace's equation.

The first case will computing the electric potential around 4 uniformly charged circles in a box. The Medusa framework [7] implementation of RBF-FD was used in the solution procedure, specifically; a neighbourhood of 9 support nodes and a basis consisting of 9 Gaussian functions with shape parameter $\sigma = 30$. The obtained solution is shown in Fig. 6.

The plotted values are values of potential ϕ obtained in computational nodes, which are distributed densely enough to create an illusion of a full plot.

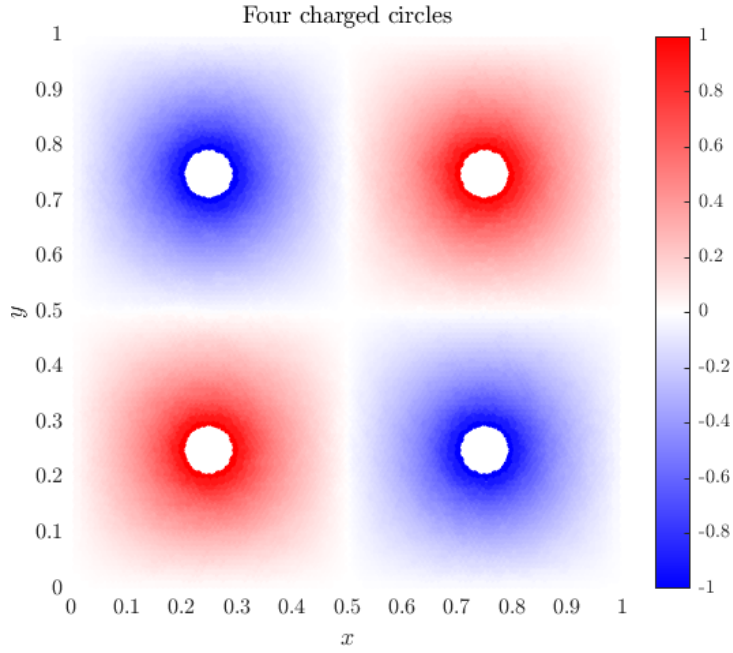

Figure 6: Electric potential around four uniformly charged circles in a box.

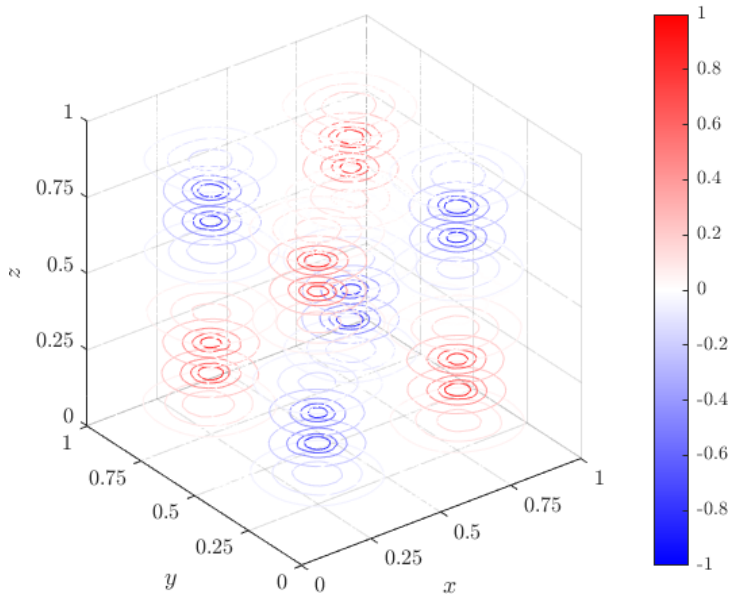

Figure 7: Electric potential around eight uniformly charged spheres in a box.

5.2 Electrostatics in 3D

The second case is a generalization of the first case to three dimensions. Eight uniformly charges spheres are put in a box, and the electric potential ϕ is computed using RBF-FD as before. A neighbourhood of 7 support nodes and a basis consisting of 7 Gaussian functions with shape parameter $\sigma = 30$ were used to solve the problem. The z cross-sections are shown in Fig. 7.

5.3 Adaptivity in linear elasticity

The final example deals with a linear elasticity problem, specifically fretting fatigue contact on the case described by Pereira et al. [6]. Two cylindrical pads act on a specimen of width W, length L and thickness t, creating a contact area, that is split into the stick and slip zones. Numerically, the problem is simplified to two dimensions and solved on the domain $\Omega = [-L/2, L/2] \times [-W/2, 0]$, with known top and right side tractions, symmetry boundary conditions on the bottom boundary and no displacement boundary conditions on the left. For further explanation and specific parameter values, the reader is referred to [9]. An important feature of this problem is that stick and slip contact zones cause high stress peaks on top boundary, which cannot be captured with uniform discretizations on current day computers. Refinement is thus needed and its behaviour in RBF-FD context has been investigated recently by Slak and Kosec [8]. Here, a fully automatic adaptive solution of the problem will be shown, with the purpose of demonstrating the capabilities of the node generation algorithm, as described in Section 3.

Fig. 8 shows top traction of the RBF-FD solution compared to a solution obtained using well-established FreeFem++ framework [10]. The solution is gradually improved over the course of 4 iterations, demonstrating that the node generator was generating distributions,

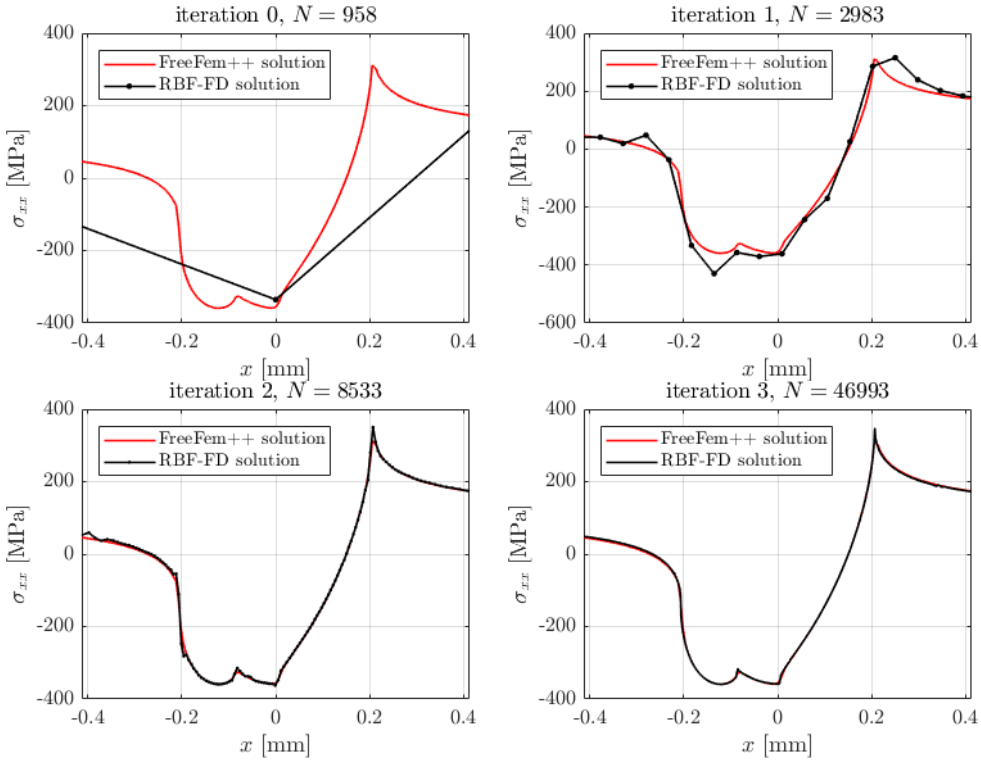

Figure 8: Adaptive solution of fretting fatigue contact problem. Top traction σ_{xx} is shown.

which were suitable for RBF-FD method. The generated distributions under the contact area are shown in Fig. 9, coloured by von Mises stress. In the first iteration, only three nodes on the top of the domain are present. In the next iteration, the node distribution becomes denser, capturing the contact phenomenon well enough to separate high and low stress areas. Consequently, the third iteration produces a variable density distribution, and the final distribution is already well adapted to the solution behaviour.

6 CONCLUSIONS

A recently published algorithm for node positioning in 2D was improved, by lowering its theoretical time complexity and demonstrating the decrease in execution time in practice, generating 1 000 000 nodes per second. A new, slower, but dimension and domain shape independent algorithm with provable minimal spacing requirements and same time complexity is proposed. The algorithm is shown to generate node distributions conforming to an arbitrary nodal spacing function, and the produced nodal distributions can be used to solve problems with the RBF-FD method on irregular domains in two and three dimensions. Furthermore, it can also be used in the fully automatic adaptive setting.

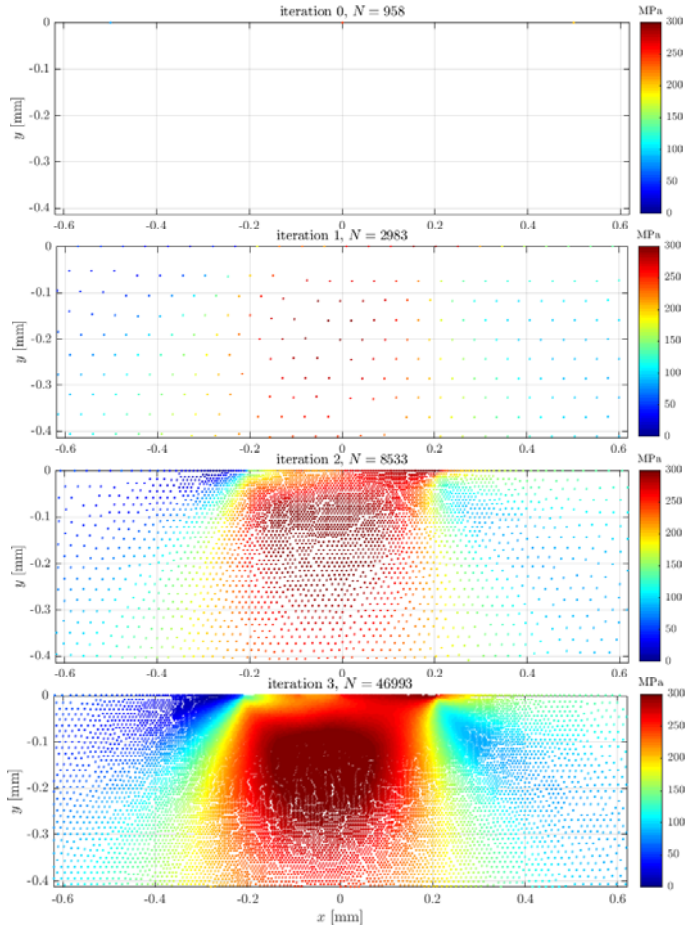

Figure 9: Nodes under contact in final iteration coloured by von Mises stress.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the financial support of the Research Foundation Flanders (FWO), the Luxembourg National Research Fund (FNR) and Slovenian Research Agency (ARRS) in the framework of the FWO Lead Agency project: G018916N Multi-analysis of fretting fatigue using physical and virtual experiments, and the ARRS research core funding No. P2-0095.

REFERENCES

[1] Tolstykh, A.I. & Shirobokov, D.A., On using radial basis functions in a “finite difference mode” with applications to elasticity problems. *Computational Mechanics*, **33**(1), pp. 68–79, 2003. DOI: 10.1007/s00466-003-0501-9.

[2] Liu, G.R., *Mesh Free Methods: Moving Beyond the Finite Element Method*, CRC Press, 2002.

[3] Fornberg, B. & Flyer, N., Fast generation of 2-D node distributions for mesh-free PDE discretizations. *Computers and Mathematics with Applications*, **69**(7), pp. 531–544, 2015. DOI: 10.1016/j.camwa.2015.01.009.
[4] Shankar, V., Kirby, R.M. & Fogelson, A.L., Robust node generation for mesh-free discretizations on irregular domains and surfaces. *SIAM Journal on Scientific Computing*, **40**(4), pp. A2584–A2608, 2018. DOI: 10.1137/17m114090x.
[5] Mitchell, S.A., Rand, A., Ebeida, M.S. & Bajaj, C., Variable radii Poisson-disk sampling, extended version. *Proceedings of the 24th Canadian Conference on Computational Geometry*, **5**, 2012.
[6] Pereira, K. et al., On the convergence of stresses in fretting fatigue. *Materials*, **9**(8), p. 639, 2016.
[7] Medusa: coordinate free meshless method implementation. http://e6.ijs.si/medusa/. Accessed on: 28 Aug. 2018.
[8] Slak, J. & Kosec, G., Refined Meshless Local Strong Form solution of Cauchy–Navier equation on an irregular domain. *Engineering Analysis with Boundary Elements*, 2018. DOI: 10.1016/j.enganabound.2018.01.001.
[9] Trobec, R. & Kosec, G., Parallel scientific computing: theory, algorithms, and applications of mesh based and meshless methods. *Springer Briefs in Computer Science*, Springer, 2015.
[10] Hecht, F., New development in FreeFem++. *Journal of Numerical Mathematics*, **20**(3–4), pp. 251–266, 2012. DOI: 10.1515/jnum-2012-0013.