

Automatic Mesh Generation (for Finite Element Method) Using Self-Organizing Neural Networks*

Larry Manevitz¹, Malik Yousef¹, Dan Givoli²

¹Department of Mathematics and Computer Science
University of Haifa

²Faculty of Aerospace Engineering
Technion

Haifa, Israel

Abstract

In the paper, we present a method to automatically fit a topological grid (mesh) to the geometry of a given domain in such a way as to place the density of nodes in close correspondence with a given density across the domain. This is important in, for example, the preprocessing of the finite element method; where one wants to get the best possible approximation to a solution of a partial differential equation for a given computational resource. Here the density function corresponds to the "areas of interest" in the domain. Our method uses the notion of a self-organizing neural network (due to Kohonen and others) for the basic organization with some additional adaptations appropriate for our problem. We have also implemented an improvement on the algorithm suggested by Tzvi and Iaakov.

Background

The finite element method (FEM) is a well-known albeit computationally intensive numerical method for solving partial differential equations [2]. Most commonly it is applied to boundary value problems given in two or three

*Partially Supported by Joint Technion-University of Haifa Research Grant

spatial dimensions. To use the method however, one has to divide up the available computational resources in an appropriate manner, the different stages of which typically require expertise.

In more detail, when applying the FEM to a given domain, one has to divide the domain into a finite number of non-overlapping subdomains (elements). (In two dimensions, the elements are usually triangles or quadrilaterals.) One also has to define a finite number of nodes, which are the vertices of the elements, and possibly other points as well. The collections of elements and nodes (and the connections among them) constitute the finite element mesh, which is an essential input for all finite element codes. See, e.g. [2] for details.

To accomplish this, one has to decide on the appropriate size and topology of the mesh and decide how it should be placed on the domain. Afterwards one has to make decisions regarding the organization of the data on the mesh which has an affect on the ease of computation. Each of these areas requires expertise.

In this paper, we work on the second of these problems: i.e. given a specific mesh (i.e. given the number of nodes and its topology), deciding how to place it on the domain in such a way as to optimize the productivity of the finite element method. (For work concerning the third point, i.e. efficient numbering of the nodes, see [5].)

The density of the mesh affects the accuracy of the finite element results. A finer mesh would give more accurate solutions, but would also necessitate a larger computational effort. Thus, the actual density of the mesh used in a certain computation is a compromise between accuracy and cost. The main parameter that controls the density of the mesh is called the "mesh parameter;" this is roughly the size of the largest element in the mesh. Of course, the density of the mesh should not necessarily be uniform. The mesh may be finer in some regions and coarser in others.

The problem of generating a mesh, say in two dimensions, is not merely a problem of dividing a given area into non-overlapping triangles and/or quadrilaterals of a given maximum size. This is because finite element meshes must have certain properties in order to be acceptable for computation. The following guidelines are considered standard. In stating them, we refer to the two-dimensional case for simplicity.

1. The mesh should be finer in regions where the solution is believed to be changing rapidly or to have large gradients. Thus, smaller elements should be used near singularity points such as re-entrant corners or

cracks, near holes, near small features of the boundary, near the location of rapidly-changing boundary data, at and near inhomogeneities, etc.

2. All elements should be well proportioned. The aspect-ratio of the element (namely the ratio between its largest and smallest dimensions) should be close to unity. Square elements are the best quadrilaterals, but even an aspect ratio of 1.5 or 2 is acceptable.
3. All interior angles of the element must be significantly smaller than 180 degrees. For example, a quadrilateral with three of its vertices lying on a nearly straight line is usually unacceptable.
4. Transition from large elements to small elements must be made gradually. The ratio between the sizes of two neighboring elements may be 1.5 or 2, but not much greater than this.

In the early days of the FEM (namely, in the sixties and early seventies), finite element meshes were produced manually. This was a tedious task, and also easily admitted errors in the data description. As the method was applied to successively larger problems, time for mesh preparation also became prohibitive. These difficulties have been alleviated by the development of automatic mesh generation algorithms.

There are several methods for automatic mesh generation. One major class of schemes is based on conformal mapping. See e.g. [7, 10]. Here a regular mesh in a simple domain (e.g. a rectangle) is mapped into the actual domain under consideration, using numerical conformal mapping. This procedure produces high quality meshes, but is in some ways limited and is sometimes expensive. Other two-dimensional schemes are triangulation schemes, which produce, by some construction, meshes made of triangles. See e.g. [12, 13, 11, 1]. Although it is well known that quadrilaterals usually perform better than triangles [6], it is much easier in general to generate an acceptable triangulation than an acceptable mesh composed of quadrilaterals. Three-dimensional mesh generation is yet much more complicated.

A review and discussion of various mesh generation methods can be found in the two recent books [4, 8].

Methods of this paper

Our approach has been to split this rather complicated global optimization problem into several parts. On the one hand, there is the decision of the

size of the mesh and the appropriate densities in different regions of the domain; while on the other hand, one has to realize the mesh by specific assignments of geometric coordinates to the nodes. We anticipate the first part being accomplished by an expert system which will, based on geometric and physical considerations, decide on the regions of interest and desired density distribution of the elements; the second part (which is the work presented here) is realized by a self-organizing neural network [9].

A self-organizing neural network, as described, e.g. by Kohonen, is a system of neurons linked by a topology. Such a network, can then learn to adjust its weight parameters based on the input, in such a way as to automatically create a map of responsive neurons that topologically resembles the input data. The optimization of this map should in principle automatically favor most of the heuristic rules stated above, and so the map can be taken as the placement of the mesh.

The methods presented here are independent of the specific topology chosen for the mesh. However, the meshes we have used in our experiments have been chosen to consist of quadrilaterals. (While it is known that quadrilateral meshes generally give better results than triangular ones, the accepted methods of placing the mesh are more developed in the case of triangles.) However, in principle, our methods will work for any mesh, even mixed triangular and quadrilateral ones. To evaluate our methods, we compared our results with one of the most developed automatic mesh generator (for triangles) PLTMG [1] by comparing the results in solving a battery of partial differential equations with boundary conditions over a two-dimensional domain.

The appropriate placement of the mesh has as a heuristic component the choice of the "density function" which expresses which part of the domain should be approximated more closely than others (typically the density is higher near corners, or other interesting geometric phenomena where the linear approximation inside an element is intrinsically worse). Note that the best density choice may actually depend on the solution to the differential equation. Nonetheless, partial information is typically available prior to the solution of the equation, (e.g. from the boundary conditions) so it is not unreasonable for a system to generate an appropriate density function based on the statement of the p.d.e. problem and boundary conditions. (In our examples, we chose the density function by hand, with the knowledge of the available exact solutions.)

Mesh quality can be judged by eye in the two-dimensional case. However, we also found it necessary to define an analytic measure of the quality.

Below, we describe this measure of mesh quality; essentially it is a mathematical realization of the above heuristic rules on the way a mesh should vary to obtain good numerical results. (Currently, we used visual quality to decide when to cease improving the mesh, but this can be replaced by examining the changes in mesh quality.)

As stated, the essence of our implementation is the self-organizing neural network algorithm of Kohonen [9]. This algorithm allows a network to choose its weights in such a way as to fix its topological elements in "weight-space" in such a way as to mimic as closely as possible the arrangement of sample input data. In other words, the neural network becomes a representative map of the sample data information. This is exploited by us, in order to arrange for the placement of the finite element mesh, by identifying the mesh nodes with neural nodes, and identifying the weight space with the physical space of the domain thereby causing the network to be an approximation of the density function. This happens by randomly choosing sample points to input to the self organizing neural network in direct correspondance to the density function.

Thus the "coordinization" of the mesh is carried out automatically by the Kohonen algorithm, with the only input necessary being sample points of the domain chosen randomly to reflect the desired density function. The mesh then "self-organizes" to make the best possible representation of the domain by the mesh elements.

It turns out that this procedure has some difficulties in our context; i.e. a finite element mesh has to fit exactly inside the domain and reach the boundaries; in addition computational requirements are somewhat high. The algorithm we present here works well for convex, or close to convex domains, but there are difficulties which require some additional techniques for strongly non-convex domains. We have sped up the algorithm somewhat by using the improvements suggested by the paper of Tzvi an Iaakov [14] (presented at the BISFAI-93 meeting). (This results in an increase in speed of around 75% without degradation of performance.)

Discussion of Quality of Results and Future Work

We point out that the work presented here should be thought of as preliminary in the sense that much work needs to be done if it is to compete directly with the highly tuned professional mesh generators such as is used in PLTMG. Nonetheless, in our experiments, even this current version is always within a few percentage of PLTMG and on occasion superior to it

(for roughly the same size meshes). This is very encouraging because our method (as opposed to PLTMG) does not change the topology dynamically, which is a substantial handicap especially near the boundary and when the density is not uniform. (This additional change may be added in the future, perhaps following the ideas of [3].) We expect further improvements by tweaking¹ somewhat our output (currently it is not strong enough with regard to point 3 (non 180 degrees) above) and by some changes in the way the mesh is mapped onto the boundary.

One important advantage for this method in the future is that the use of the self-organizing neural networks is *NOT* sensitive, in essence, to the dimension of the domain. That is, in principle, the same method should work to fit a three dimensional mesh inside a three dimensional domain. The main work needed for this case has to do with making the appropriate projections onto the boundaries.

On the other hand, the method as presented here is not competent with non-convex domains; and additional methods (e.g. conformal mappings and their inverses) may need to be invoked to handle these cases. Along this line, we mention that an alternative design we considered for the use of the Kohonen algorithm which would in fact handle non-convex domains. This method had the topology of the mesh delineated as to which were the boundary nodes and which were the interior nodes. Then one would apply the Kohonen algorithm in 1-dimensional space to the boundary. Once this is fixed, one applies the Kohonen algorithm to the 2-dimensional network constrained by the boundary. (This method has the advantage that it is directly generalizable to 3-dimensions.)

However, various technical difficulties in the implementation led us to use current somewhat simpler method, wherein the entire 2-dimensional network is handled ab initio and the boundary is reached because of the numerical approximation. It is unclear which method is preferable for a 3-dimensional mesh.

Descriptions and Tables of Tests

We used the following as our measure of the "quality" of a mesh:

Here for a given element b^e refers to the largest side of the quadrilateral, a^e refers to the smallest. $E_1^e = 1 - b^e/a^e$, giving a measure of the aspect ratio, $E_2^e = \max_{i=1}^4 |1 - \frac{2}{\pi} angle_i|$ measuring how close all the quadrilateral angles

¹This can be done by running a heuristic critic based on the 4 criteria (listed above) over the mapped mesh between iterations, and modifying the worst offender(s).

are to 90° s, $E_3^e = \max_{neighbors} \left| 1 - \min_{n=1}^{N^e} \{a^e/a_n^e, a_n^e/a^e\} \right|$, measuring how similar an element is to its neighboring ones.

Then $Quality(Mesh) = \sum_{elements} E_1^e + E_2^e + E_3^e$.

This function allows one to measure how various changes in the algorithm result in improvements in the mesh. However, in order to compare with PLTMG, one can not directly compare the two methods, because one uses triangles and one uses quadrilaterals. Accordingly, we compared the quality of solutions of p.d.e.s with boundary conditions as solved using the different meshes. That is, we took the following:

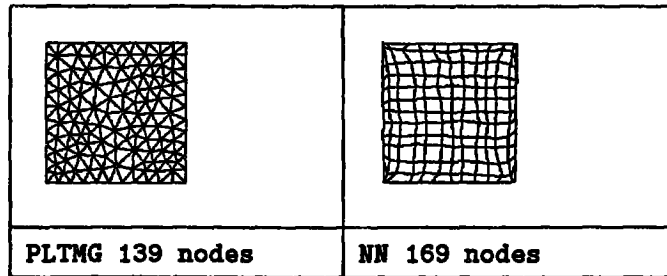
- a set of domains, all either convex or “near-convex”.
- a suite of p.d.e. problems with boundary conditions on these domains; exact solutions are known for these problems.
- different densities; appropriate for the different problems.
- different size meshes for each problem; we allowed both our program and PLTMG to produce meshes of approximate equivalent size.

For each generated mesh and problem we input it into a FEM solver, and then computed the quality of the solution using the following formulas: (Here u gives the exact value, u_h gives the computed value.)

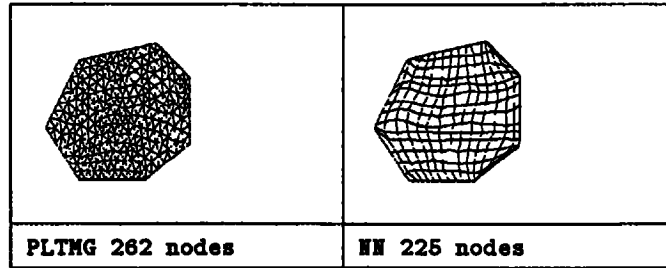
$Error/Node = \sum_{nodes} |u(node) - u_h(node)| / |\#(nodes)|$,
 $Error/Value = \sum_{nodes} |u(node) - u_h(node)| / \sum_{nodes} |u(node)|$

We solved problems of the form $u_x x + u_y y + f(x, y) = 0$. In each case we chose $u(x, y)$ so that an exact solution was known.

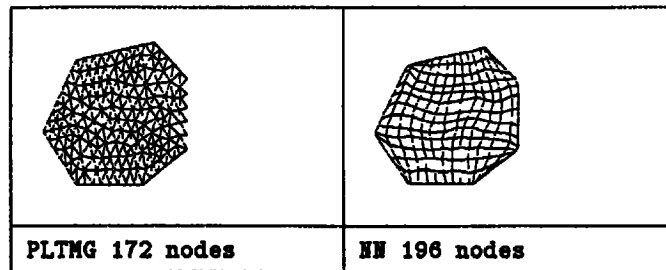
Here we present some representative sample results:



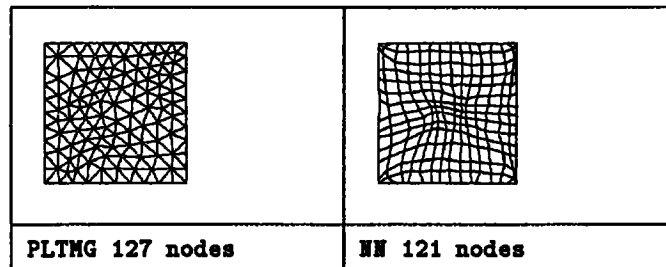
Rectangular Domain - PLTMG (139 nodes, 232 elements) NN (169 nodes, 144 elements)					
$u(x, y)$	$f(x, y)$	Error/Node		Error/Value	
		PLTMG	NN	PLTMG	NN
$x^3 + y^3$	$-6x - 6y$	6.427036E-03	5.480817E-03	2.082413E-04	1.798470E-04
$\sin x + \sin y$	$\sin x + \sin y$	3.085165E-04	2.832740E-04	2.68856E-04	2.442777E-04



7 Sided Domain - PLTMG (262 nodes, 465 elements) NN (225 nodes, 196 elements)					
$u(x, y)$	$f(x, y)$	Error/Node		Error/Value	
		PLTMG	NN	PLTMG	NN
$x^3 + y^3$	$-6x - 6y$	3.627359E-03	1.108236E-02	1.338744E-04	3.947753E-04
$\sin x + \sin y$	$\sin x + \sin y$	2.491710E-04	1.137806E-01	1.899917E-04	8.978176E-02



7 Sided Domain - PLTMG (172 nodes, 296 elements) NN (196 nodes, 169 elements)					
$u(x, y)$	$f(x, y)$	Error/Node		Error/Value	
		PLTMG	NN	PLTMG	NN
$x^3 + y^3$	$-6x - 6y$	6.638000E-03	7.970939E-03	2.408920E-04	2.923190E-04
$\sin x + \sin y$	$\sin x + \sin y$	3.316738E-04	5.992429E-04	2.566378E-04	4.650650E-04



Rectangular Domain - PLTMG (127 nodes, 212 elements) NN (121 nodes, 100 elements)					
$u(x, y)$	$f(x, y)$	Error/Node		Error/Value	
		PLTMG	NN	PLTMG	NN
$e^{-(x-2)^2} e^{-(y-2)^2}$	$-u_{xx} - u_{yy}$	6.244993E-02	6.43370E-02	1.226566E-01	1.210840E-01

Summary

Looking over the above table and the meshes themselves, we see that the NN approach produces roughly the same quality solutions for the chosen partial differential equations. The variance seems to be related to the “shape” of the domain, for rectangles, our method seems superior, whereas for the septagon (many angles) marginally worse. Our method seems to have advantages when an interior density point is needed; the density option is somewhat less efficacious for “hot-points” near the boundary.

Overall, it seems that the weaknesses arise from our keeping the topology fixed. This can result in contortions, and thereby poor individual elements in certain densities or geometries. We expect to improve the method by (1) adding a “critical tweaker” which will look over the mesh and move some nodes of the worst elements (2) adding the ability to change the mesh topologically as suggested in [3].

References

- [1] R.E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations*. SIAM publications, Philadelphia, 1994.
- [2] G.F. Carey and J.T. Oden. *Finite Elements, Vol. III: Computational Aspects*. Prentice-Hall, Englewood Cliffs, NJ, 1984.
- [3] B. Fritzsche. Growing cell structures - a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7:1441–1460, 1994.
- [4] P.L. George. *Automatic Mesh Generation*. Wiley, Chichester, UK, 1991.
- [5] L. Manevitz D. Givoli and M. Margi. Heuristic finite element node numbering. *Computing Systems in Engineering*, 4:159–168, 1993.
- [6] T.J.R. Hughes. *The Finite Element Method*. Prentice-Hall, New York, 1987.
- [7] R.E. Jones. *A Self-Organizing Mesh Generation Program*. ASME Publication 74-PVP-13, 1974.
- [8] P. Knupp and S. Steinberg. *Fundamentals of Grid Generation*. CRC Press, Boca Raton, FL, 1993.

- [9] T. Kohonen. *Self-Organization and Associative Memory, Second Edition*. Springer-Verlag, Berlin, 1988.
- [10] F.C. Thames J.F. Thompson C.W. Mastin and R.L. Walker. Numerical solutions for viscous and potential flow about arbitrary two-dimensional bodies using body-fitted coordinate systems. *J. Comput. Phys.*, 24:245–273, 1977.
- [11] R. Renka. *Triangulation and Bivariate Interpolation for Irregularly Distributed Data Points*. PhD thesis, University of Texas at Austin, 1981.
- [12] D. Rhynsburger. Analytic delineation of thiessen polygons. *Geogr. Anal.*, 5:133–144, 1973.
- [13] F. Thomasset. *Appendix to Navier-Stokes Problems*. North Holland, 1977.
- [14] T. Tzvi and E. Iaakov. Towards real-time self-organizing network with parallel and noisy inputs. Preprint, Hebrew University, 1994.