# Automating the Finite Element Method: A Test-Bed for Soft Computing*

Larry Manevitz[1] and Dan Givoli[2]

[1]Department of Computer Science
University of Haifa

[2]Faculty of Aerospace Engineering
Technion - Israel Institute of Technology

Haifa, Israel

**Abstract**

This paper is an interim report on a programme to automate the finite element method. The overall programme is discussed, and the implementations of three specific sub-problems (node numbering, mesh placement, and adaptive meshing) are described. It is also argued that the overall architecture of an "intelligent finite element package" can serve as a "test-bed" for many soft-computing techniques.

## Scientific Background

The finite element method (FEM) is a computationally intensive method for the numerical solution of partial differential equations. It is a widely used tool and in many cases is the method of choice. This is especially (but not exclusively) true in structural engineering and solid continuum mechanics. The finite element procedure has been proved most effective when applied to linear boundary value problems in spatial domains which are complicated geometrically. (See e.g. [16])

Basically (and oversimplifying) the FEM works, by deciding a priori on a certain kind of simple approximation to the solution, and by dividing up the region of solution into small "elements", and allowing the parameters of the simple approximations to vary from element to element. The requirement that the individual local solutions remain consistent together with e.g. boundary conditions, results in linear constraints on the parameters. These are then solved by standard linear algebra techniques.

However, in practice one can not use the FEM as a "black-box" solver; i.e. it is not sufficient to know the governing equations, the geometry, and the boundary and initial conditions in order to obtain high-quality numerical results. It is well appreciated among finite element users in industrial and scientific communities that the successful application of this technique requires substantial amount of experience and expertise in order to make the computation feasible and the results accurate at the same time. This is true with regard to any of the large commercial finite element packages available currently, although various codes may have a different amount of flexibility.

This is because there is a very large number of parameters that need to be chosen; computational limitations make the choice of these parameters crucial for successful use. These are knowledge based

*Partially supported by joint U.Haifa-Technion Research Grant

requirements; and current usage requires much human expertise. There are no known effective algorithms that can replace a human user in all cases; and many of the problems are known to be computationally intractable in the general case [11].

On the other hand, the complexity of some of the problems are beyond the realm of efficient manual control (e.g. in more than two dimensions or in quite complicated situations like parallel implementations which are not directly amenable to human intuitions).

In recent years there has been a large development of tools for artificial intelligence, neural networks, fuzzy logic and related disciplines, (called by some "Soft Computing" [34], [33] [32]), which our work and analysis indicate are applicable to the problems under discussion here. Each problem can in fact be attacked by a number of methods, but *usually* a certain approach suggests itself as the most promising one in each case. Thus expert system technology [6] seems most appropriate to replace human numberers, self-organizing neural networks [19] and fuzzy logic "critics" [32] seem appropriate for mesh placement, neural network non-linear predictors are appropriate for dynamic mesh placement for solutions to time dependent PDEs [31] , distributed artificial intelligence [27] and genetic algorithms [14] can be appropriate for load balancing in parallel computation, and so on . The specific problems and tools we have in mind are described in figure 2.

Our main goal in working on the finite element method is, of course, the automation of the method; and any efficient means towards that is welcome. In this paper, we report on our results so far.

In addition, we have come to realize [35] [13] that the FEM serves as a rich test-bed appropriate for the serious use of these techniques in real world (i.e. non-"toy") problem settings. Evaluating these techniques in realistic settings is in itself an important research goal. In fact, our initial work has already resulted in advancing somewhat [21] one of the most classical neural network algorithms as a result of evaluating the needs in the realistic setting.

We point out that since one can compare results with analytic ones in certain settings, it allows for quantitative evaluation of the effectivity of the techniques. In principle, this allows to compare and evaluate competing techniques and directions both between themselves and with current commercial packages.

By a test-bed we have in mind the a system which allows one to implement a variety of techniques on various problems, run it easily on "real-world" applications, and be able to do some sort of evaluation/comparison studies with other techniques.

Intelligent automation of the FEM can be appropriate for this because

- There are a wide variety of optimization/satisfaction sub-problems to be solved. Various techniques are applicable to the different problems.
- The architecture of the automation is modular. This means that each of the sub-problems can be solved more or less independently. ("More or less" means that when all the solution steps are in place, one has to consider another optimization problem concerning trade-offs. However, this affects only the potential optimality of the FEM solution, and not the effectivity of the individual steps.) Each of the steps can be evaluated individually or how it affects the global quality of the numerical solutions.
- The nature of the FEM, a numerical solver for partial differential equations, makes it easy to generate test problems. In fact any partial differential equation with boundary conditions defines a test case. This means it is almost as easy to work on real world problems as on artifical simple or "toy" ones. One can also provide test cases with analytic solutions to provide a natural "gold standard" for evaluations.
- The importance of the FEM means that there are commercial codes available for comparison.

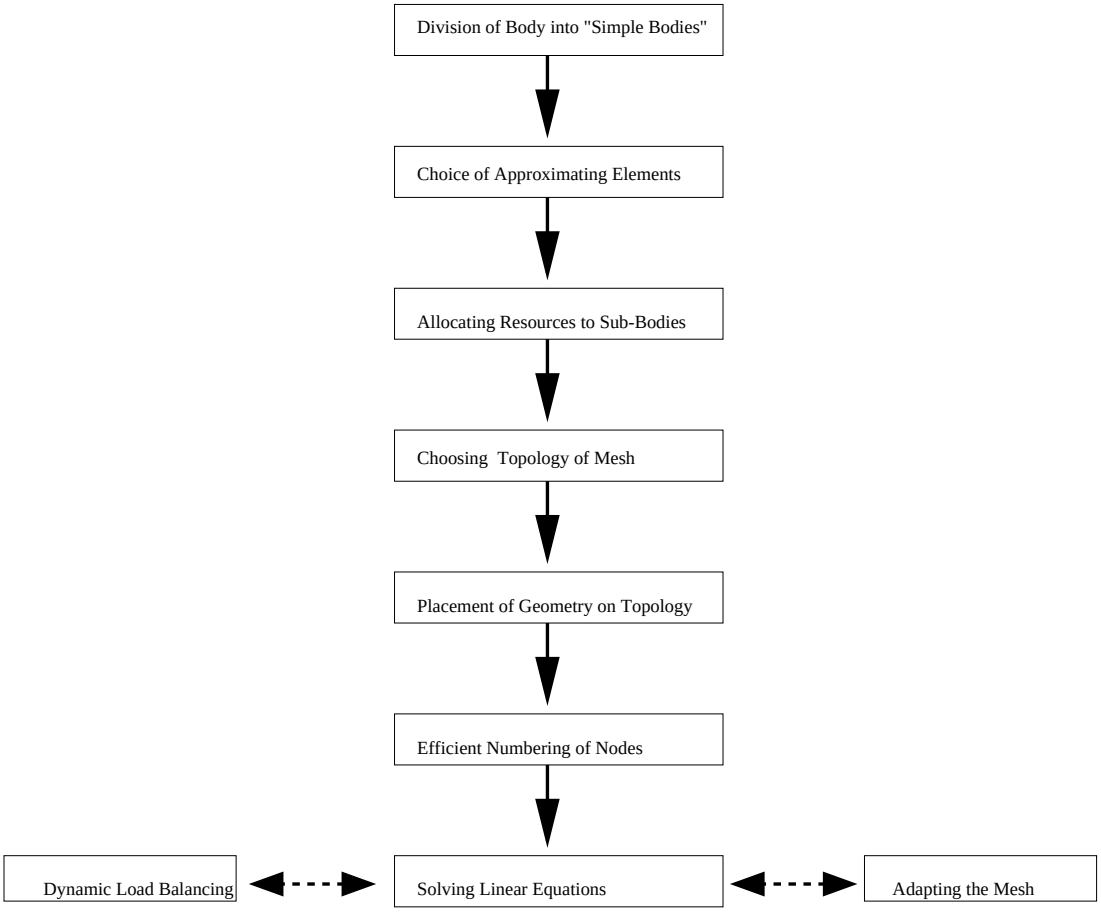



Figure 1: **Some general tasks in the Finite Element Method.**

## Some Sub-problems for the Finite Element Method

In Figure 1 , we list some of the human-based tasks that must be performed in using the FEM.

Figure 2 lists some corresponding possible techniques.

So far, together with our students Miha Margi and Malik Yousef, we have implemented two of these solutions: (automated mesh numbering via an expert system and mesh placement via a self-organizing neural network). In addition a third sub-problem (mesh adaptivity) is under current development using feed-forward neural network run in a temporal predictive mode. In the following sections, we sketch the results of these sub-problems (full details appear or will appear elsewhere [13], [35]).

## Heuristically Numbering the Nodes

When applied to linear boundary value problems, the finite element discretization of the governing partial differential equations leads finally to a linear system of algebraic equations (see [16]),

$$\mathbf{Kd} = \mathbf{F}$$

.

Here $\mathbf{K}$ is the so-called *stiffness matrix*, $\mathbf{F}$ is the load vector, and $\mathbf{d}$ is the vector of unknowns. The dimension of the system (1) in typical industrial applications is of order 100 or 1000 or even 10000. The computational effort involved in solving such a large system is reduced considerably because most entries $\mathbf{K_{ij}}$ are zero; i.e. it is a *sparse* matrix.

The degree of sparseness of $\mathbf{K}$ depends on the specific finite element discretization of the spatial domain, or more explicitly on the *finite element mesh*. The spatial domain is divided into *elements*,

| Problem | Approach |
|---|---|
| Partitioning given "body" | Computational Geometry |
| Choice of approximating Elements | Expert System |
| Allocating resources to sub-bodies | Expert System; NNs; GAs (?) |
| Choosing Topology of Mesh | Expert System |
| Assigning Geometry to Topology | NN(Kohonen-style); fuzzy critic |
| Efficient Numbering of Nodes | Expert System |
| Adaptive Solving | |
| Prediction of Error Indicator | NN (learning network) |
| Load Balancing | Distributed AI; Automatic Negotiation; GA |

Figure 2: **Table of Problems and Soft Computing Solution Approaches**

which are connected to each other by *nodes*. In two dimensions, the elements may be quadrilateral or triangular, and the nodes are their vertices. Points on the edges and in the interior of the elements may also be defined as nodes.

Two nodes are said to *interact* if they both belong to a common element in the mesh. If there is a total number of $N$ nodes in the mesh and they are numbered from 1 to $N$, then $\mathbf{K_{ij}} \neq \mathbf{0}$ only if nodes $i$ and $j$ interact. Thus the structure of the stiffness matrix is determined by the node interaction. Since the mesh is typically imposed on a physical domain and thus can be thought of as a planar or spatial graph, most $\mathbf{K_{ij}}$ are zero, i.e. $\mathbf{K}$ is sparse.

The discussion above implies that the order in which the nodes are numbered is crucial, since this numbering determines how sparse $\mathbf{K}$ is. Note first that optimal numbering of a given mesh is an NP complete problem ([11]). Thus for a mesh containing a large number of nodes and elements it is of course impractical to find an exact solution for the optimization problem by performing a full enumeration. However, there are so-called "algorithmic" solutions which have been incorporated in commercial software. Anecdotal evidence about this software reports that: (i) a human expert can often outperform it. (ii) The best such algorithms typically result in rather complicated numberings involving substantial "windings" and for some applications there is an advantage in the simplicity of the numbering. (These algorithms include the methods described by Cuthill and McKee [4], King [18], Collins [3], Akin and Pardue [1] Gibbs [12] Razzaque [26] [20], Pina [24] Sloan and Randolph [28], Fenves and Law [8], and Sloan [29].)

According, we decided to test the feasibility of developing an expert system to try and mimic the heuristic mechanisms of a proficient human numberer. Actually only some of the possible heuristics were implemented, but the system proved to compare favorably with the human user on a variety of test examples. (Full details appear in [13] .)

In order to state the optimization problem mathematically we introduce some notation. Let the dimension of the matrix $\mathbf{K}$ be $N \times N$. Let $b_i$ be defined as $b_i = i + 1 - \max_{K_{ji} \neq 0} j$ . In other words, $b_i$ is the height of column $i$ starting from the diagonal and up to the skyline. Now, let the *average half bandwidth* AHB and the *root-mean-square bandwidth* RMSB be defined as AHB $= 1/N \sum_{i=1}^{N} b_i$, RMSB $= \left(1/N \sum_{i=1}^{N} {b_i}^2\right)^{1/2}$. It is apparent that the bandwidth depends on the nodal numbering system. This dependence becomes strong for large meshes. The optimization problem under consideration can thus be stated as follows: Find a numbering system for the nodes (from 1

to $N$) such that AHB (or RMSB) will be *minimal*.

### A brief description of the expert system characteristics

One of the hardest steps in producing an expert system is to construct the heuristics according to which the expert system will perform, and which will mimic the considerations of a human expert. To this end, we note the following functions typically performed when numbering the nodes of a finite element mesh manually and briefly comment on our system's approach to these tasks.

0. Preparation of numbering strategies for "paradigm simple blocks" with different geometries and topologies. Varying parameters of freedom are associated with each strategy. These strategies are achieved by experience, by trial and error, and sometimes by full analysis. This step is not "performed" for each mesh separately, but is rather a data base of knowledge an expert has accumulated. Examples of simple blocks are rectangles, annuli, discs.

1. Subdivision of the mesh into a disjoint union of simple blocks. Our system does not perform this, but receives it as given.

2. Choosing the order in which these blocks are picked for numbering.

In this step the order of the simple blocks is determined. To optimize the numbering a user tries to keep the node numbers as continuous as possible. However when passing from block to block this is often impossible. Thus the goal is to keep the "jumps" across block interfaces as small as possible. In the discussion that follows, a *component* is a topological component, i.e. a maximal connected sub-body.

In the implemented system the procedure for block ordering is as follows:

• Choose the first block to be numbered as the largest simple block, based on the number of nodes.

• Remove this simple block from the mesh. (This may cause the remaining mesh to have several disconnected components.)

• Place all resulting separate components in increasing order in a stack.

• Until the stack is empty do the following: • Remove the smallest component from the stack.
• From the chosen component, choose the next simple block to be numbered in the following way:

• Consider all simple blocks in the component with an interface with the previously numbered block (there are always such).

• Choose the one with the largest interface.

• Remove the chosen block from the component. (This may divide the remaining blocks into several connected components.)

• Place the new components on the stack in increasing order.

See figure 3 for an example of how the blocks are numbered.

3. For each block, choosing a numbering strategy, depending on the topology and the geometry of the block. That is, choosing the best match to a paradigm simple block. Essentially this has two parts: (1) solution of a pattern recognition problem; i.e. given a simple block find which of the paradigm simple blocks it is closest to; (2) after identificat ion choosing a numbering method for the block form amongst the possible methods mentioned in step 0.

In the implemented version, all the blocks are assumed to be pseudo-rectangles and there is only one numbering strategy for numbering rectangles, so this step is to a large extent vacuous.

Notwithstanding, it remains for the system to perform the following analysis: (a) to decide on the orientation of the block, regarded as having a rectangular shape; (b) to be able to ignore small perturbations. The first is done using an algorithm involving the convex hull of the block and regression ; the second is accomplished using a heuristic (also involving the convex hull) that defines a "grain size".

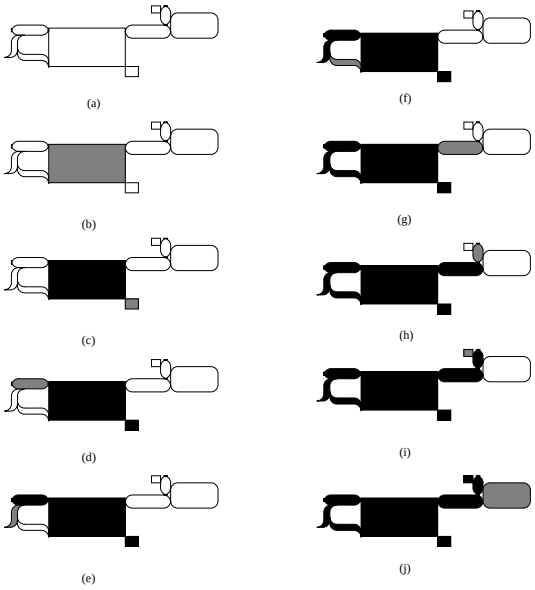



Figure 3: Ordering the simple blocks

4. For each block, determining the free parameters associated with the strategy chosen for that block, such as the node from which to start the numbering. For the rectangular strategy which is handled in the implemented version, the only free parameter is the node from which to start the numbering, and there are four possibilities, corresponding to the four corners of the pseudo-rectangle. This is selected by a heuristic that is based on both topological and geometrical considerations.

It is interesting and crucial to note that while the mesh is a purely topological object as far as node numbering is concerned, the user decides on the division based not only on topological criteria, but also on *geometric* criteria. This is an obvious logical fallacy and it is easy to devise artificial mesh examples where the results are extremely bad. Nonetheless for real meshes the results seem acceptable.

The reason why human users can be so successful in numbering meshes although basing their decisions on geometric considerations is that they intuitively rely on the fact that *actual meshes are designed with implicit geometric constraints*. For example, a good practice in mesh design is to use elements which have aspect-ratios close to unity, namely quadrilaterals and triangles which are nearly equilateral. Another good practice is to pass from a crude region (large elements) to a refined region (small elements) in a gradual manner. (See e.g. Irons and Ahmad [17].)

Our expert system tries to mimic a human user; therefore it uses geometric considerations as well as topological ones. This is in contrast to the "algorithmic approach", where only the topological properties of the mesh play a part in the node reordering procedure.

The system was tested on a number of meshes taken from the literature; here we mention the largest mesh tested to date similar to one that appears in [36] which has 359 nodes and 559 triangular elements. Here the average half band width of the system was 17.6; that of the human expert was 15.9 while for comparison a random numbering resulted in a value of 210.2.

Some other meshes were taken from applications in large deformation continuum mechanics discussed in [15] and solved by the finite element code NIKE2D. The results were are quite comparable to that of the human expert. (In one case the expert system actually outperformed the human expert.)

The results show that although the human expert performs better than the expert system, the differences between the two AHBs are reasonably small. Other examples have been tested as well and the results compared favorably with those of a human expert.

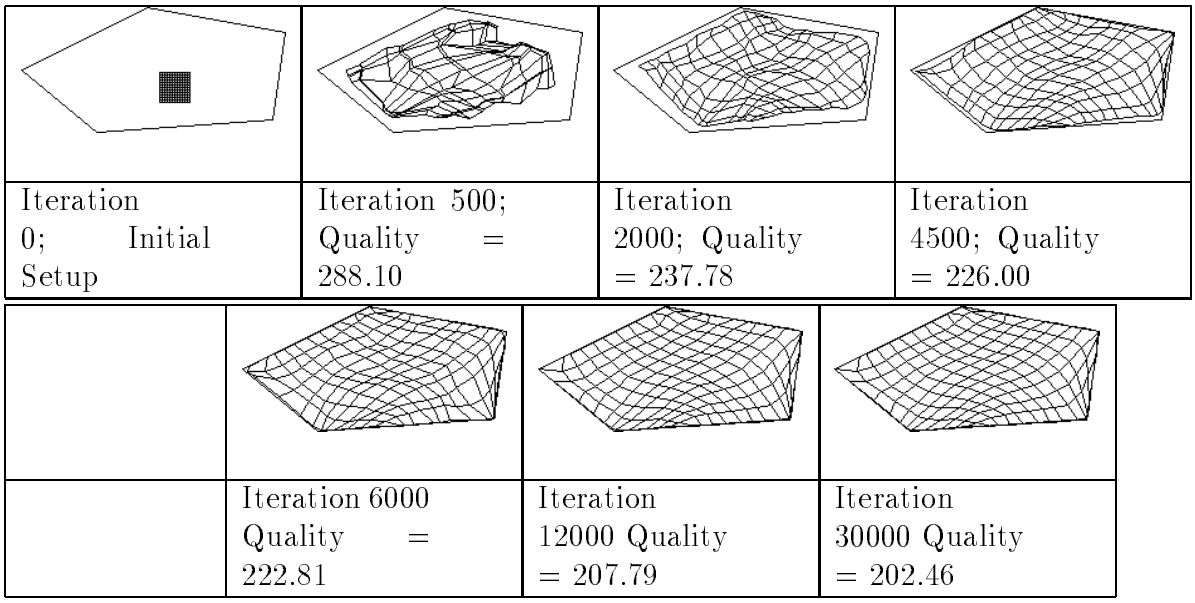



Figure 4: **A sequence of snapshots of a mesh being placed via a NN algorithm**

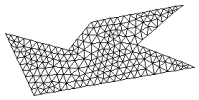
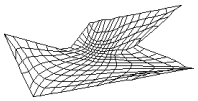
## Mesh Placement via Self-Organizing Neural Networks

Once a topological mesh has been defined (a problem we have suggested is appropriate for an expert system), the mesh has to be given its geometry; i.e. it has to be placed appropriately on the body. Moreover, there are points and regions which one wants to cover with a finer mesh than other areas. These are regions or points of "interest" where the approximation used in the finite element method is intrinsically worse. Essentially it is an optimization problem; given a fixed amount of computation that one wants to expend, which translates into a fixed amount of elements; how should one best distribute these elements so as to obtain the best approximation using the finite element method. A better mesh results in a better approximation.

There are several requirements for the quality of a mesh. For example, one wants the proportions of the elements to be as close as possible to those with good aspect ratios (the aspect ratio is the ratio of the radii of circumscribed circle to that of the inscribed circle; "good" means here close to one). For a quadrilateral then, for the best results from the finite element method one wants quadrilaterals close to squares and triangles close to equilaterals. In addition, one wants the change in size between elements determined by the mesh to be gradual. Quantitatively, one wants to keep the ratio of radii of circumscribed circles of adjacent elements to be close to one; globally that means that the maximum and minimum of such ratios of all pairs of adjacent elements should be as close to one as possible.

Moreover, in a typical finite element mesh, the density of the nodes is taken to vary from being very high near certain critical regions or points to more sparse where simpler approximations will suffice.

This can be handled using a modification of Kohonen's self organizing the neural network approach ([35], [22]) by setting the probability density function of the input to correspond to the desired density of the network. The "self-organizing" feature of the Kohonen map will then place the highest density of nodes according to the sampling of data according to this density function. In this implementation, one identifies the weight of the neurons with the geometric coodinates of the body. The Kohonen map results in an equiprobably response map of the neural net (constrained by its given topology) subject to the sample data. Actually, the algorithm we use is a substantial adaptation of the basic Kohonen map since we require that the network map boundaries to boundaries. This is accomplished by using a weaving of one dimensional and two dimensional Kohonen maps. This is discussed further in [21]. Figure 4 gives a view of the algorithm in action.

|  |  |
|---|---|
| PLTMG 249 nodes | NN 225 nodes; Quality = 278.713715; ‘‘Hot-spot" (2,2) near center; 46650 Iterations |

| 7-Sided Non-Convex Domain PLTMG (249 nodes, 437 elements) N N (225 nodes, 196 elements) | | | | | |
|---|---|---|---|---|---|
| | | Error/Node | | Error/Value | |
| $u(x,y)$ | $f(x,y)$ | PLTMG | NN | PLTMG | NN |
| $e^{-(x-2)^2}e^{-(y-2)^2}$ | $-u_{xx}-u_{yy}$ | 2.412143E-02 | 7.530449E-03 | 4.515054E-02 | 9.097765E-03 |

Figure 5: **A typical comparison between the NN and a commercial package**

In a series of experiments this algorithm was compared with one of the most popular finite element packages over a series of problems, and was found to be generally superior. Figure 5 gives a sample comparison with the PLTMG [2] package. Full details of this experiment can be found in [35].

### Dynamic Mesh Generation for Temporal PDEs

Time is often treated distinctly from spatial dimensions in the solution phase of PDEs . That is, the typical method of choice for solution of such equations is *not* to treat time as simply another dimension, but to "simulate time"; i.e. to repeatedly solve the equations for different times; using the previous solution as the starting conditions for the next one.

However, in a dynamic system, this implies that one should not use the same mesh at different times since the "areas of interest" are, of course, changing with time.

For example, when the solution of a hyperbolic problem involves a shock wave which propagates through the mesh, the location of the "area of interest," namely the shock vicinity, keeps changing in time. Another example is a problem of fluid flow in a cavity, where flow cells are generated and undergo continuous changes in their shapes and size as time proceeds [9] .

Thus the mesh choice should be dynamic; varying with time. In current usage, the method is to use the solution at time $t_n$ to indicate where the mesh should be modified (where it should be refined and where it can be made coarser) at time $t_{n+1}$. The work [9] used this mechanism.

However, this suffers from the obvious defect that one is always one step behind. If the area of interest is propagating (a common phenomenon) then one may be always refining directly behind the most interesting phenomenon. This is also assuming that one does not miss the "action" altogether. One can look at this as a special instance of a control problem.

It would be preferable to "predict" the area of interest at time $t_{n+1}$ based on the solution at $t_k$ for $k \leq n$. There is what we propose to try using one of several neural network approaches. Since a NN is a universal approximator [5, 10, 25, 20] the intention is to use as input the areas of "interest" at recent times and predict the areas of interest at the next time stage.

The simplest such mechanism is to use a feed-forward neural network trained with a delay to do the prediction. This is what we propose to try using a neural network plant predictor, which has been developed in the context of control theory and signal analysis [31, 30] . The basic idea can be seen in Figure 6. Various parameters; (e.g. function values at the node at several prior timesteps, values at neighboring nodes, distance from "anomalies" in the body) will be entered to the network;

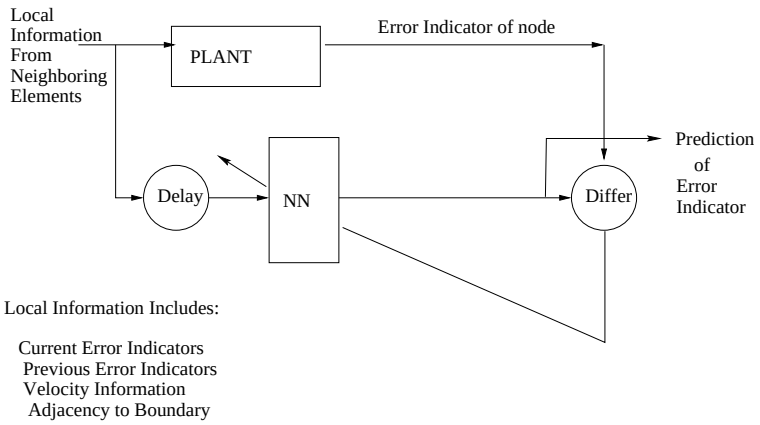



Figure 6: **A neural network designed to predict**

the output will be an encoding on the need for refinement following a delay.

The problem in such an approach, is (1) to decide on the appropriate information to train the network with, and (2) to decide on the appropriate breakdown into sub-problems. That is, although it is known that simple NNs are in principle universal approximators; in practice, one should not attempt to make one network do too many things, and the best results in the literature are obtained when the NNs are applied to specific sub-problems.

Such a mechanism was used to good affect in the recent work "GloveTalk" [7] where a neural network was taught to translate hand-language to phonemes by using varied information; including rate of change of hand position. We mention this work, because we think its use of velocity; computed by including the values of a function over several time steps (and acceleration; by including the velocity over several time steps) as input to a NN is something that will be useful for our implementation. We anticipate including the solution over several previous time steps in order to predict where the solution will have a large change in the *next* time step. Substantial experimentation will be needed to decide on the correct encoding of information for this prediction.

Another breakdown we anticipate is the use of different such neural networks for different parts of the mesh. That is, we expect that training such a network will be more effective by dividing nodes in the mesh into different classes and using separate nets for the different classes. The underlying philosophy here is that the more similar the elements of a class are; the simpler the representation of the predictor function should be; and the easier and faster the training. For example, one would use a different network for nodes near boundaries than ones in the interior of a body. On the other hand, one does not want to divide the body into too many subclasses. (In the limit, this would lead to having a different network for *every* node.) A predictor should work quickly, and thus use the "experience" gained at some nodes to predict density needs at others. A priori, it is our estimate that such a method will work well on propagation solutions; such as occur in models measuring, e.g. the heat equation, or shock [23].

# References

[1] J.E. Akin and R.M. Pardue, *Element resequencing for frontal solutions*, Mathematics of Finite Elements and Applications (New York) (J.R. Whiteman, ed.), Academic Press, New York, 1975.

[2] R.E. Bank, *Pltmg: A software package for solving elliptic partial differential equations*, SIAM publications, Philadelphia, 1994.

[3] R.J. Collins, *Bandwidth reduction by automatic renumbering*, Int. J. Numer. Meth. Eng. **6** (1973), 345–356.

[4] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, Proceedings of the ACM National Conference, ACM, 1969.

[5] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control Signals and Systems **2** (1989), 303 – 314.

[6] D.A. Waterman F. Hayes-Roth and D. Lenat (eds.), *Building expert systems*, Addison-Wesley, 1983.

[7] Sidney Fels and Geoffrey Hinton, *GlovetalkII: An adaptive gesture-to-formant interface*, Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems, 1995, pp. 456–463.

[8] S.J. Fenves and K.H. Law, *A two-step approach to finite element ordering*, Int. J. Numer. Meth. Eng. **19** (1983), 891–911.

[9] D. Givoli J.E. Flaherty and M.S. Shephard, *Simulation of czochralski melt flows using parallel adaptive finite element procedures*, Modelling Simul.Mater. Sci. Eng. (to appear).

[10] A. R. Gallant and H. White, *On learning the derivatives on an unknown mapping with multilayer feedforward networks*, Neural Networks **5** (1992), 129–138.

[11] M.R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of np-completeness*, Freeman, 1979.

[12] N.E. Gibbs, *A hybrid profile reduction algorithm*, ACM Trans. Math. Software **2** (1976), 378–387.

[13] L. Manevitz D. Givoli and M. Margi, *Heuristic finite element node numbering*, Computing Systems in Engineering **4** (1993), 159–168.

[14] D. Goldberg, *Genetic algorithms*, Addison-Wesley, 1989.

[15] J.O. Hallquist, *Nike2d — a vectorized, implicit, finite deformation, finite element code for analyzing the static and dynamic response of 2-d solids*, Tech. Report UCID-19677, Rev. 1, University of California, 1967.

[16] T.J.R. Hughes, *The finite element method*, Prentice-Hall, New York, 1987.

[17] B. Irons and S. Ahmad, *Techniques of finite elements*, Ellis Horwood and John Wiley, Chichester, England, 1980.

[18] I.P. King, *An automatic reordering scheme for simultaneous equations derived from network systems*, Int. J. Numer. Meth. Eng. **2** (1970), 523–533.

[19] T. Kohonen, *Self-organization and associative memory, second edition*, Springer-Verlag, Berlin, 1988.

[20] M. Shoham M. Meltser and L. Manevitz, *Approximating functions by neural networks: A constructive solution in the uniform norm*, Neural Networks **9** (1996), no. 6, 965–978.

[21] L. M. Manevitz, *Interweaving kohonen maps of different dimensions to handle measure zero constraints on topological mappings*, Neuroprocessing Letters **5** (1997), 153–159.

[22] Y. Stephan O. Sarzeaud and C. Touzet, *Finite element meshing using kohonen's self-organizing maps*, Artificial Neural Networks (T. Kohonen et al, ed.), North Holland, 1991, pp. 1313–1317.

[23] J.T. Oden, *Adaptive methods in computational fluid dynamics*, Finite Elements in Fluids, Vol. 8 (Washington) (T.J. Chung, ed.), Hemisphere Publishers, Washington, 1992, pp. 3 – 30.

[24] H. L. Pina, *An algorithm for frontwidth reduction*, Int. J. Numer. Meth. Eng. **17** (1981), 1539–1546.

[25] M. Leshno V. Lin A. Pinkus and S. Schocken, *Multilayer feedforward networks with non-polynomial activation functions can approximate any continuous function*, Neural Networks **6** (1993), 861–867.

[26] A. Razzaque, *Automatic reduction of frontwidth for finite element analysis*, Int. J. Numer. Meth. Eng. **15** (1980), 1315–1324.

[27] J. Rosenschein, *Rules of encounter*, MIT Press, 1994.

[28] Sloan and Randolph, *Automatic element reordering for finite element analysis with frontal schemes*, Int. J. Numer. Meth. Eng. **19** (1983), 1153–1181.

[29] S.W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, Int. J. Numer. Meth. Eng. **23** (1986), 239–251.

[30] B. Widrow and M. Lehr, *30 years of adaptive neural networks: perceptron, madaline, and backpropagation*, Proceedings of the IEEE **78** (1990), no. 9, 1415–1442.

[31] B. Widrow and S. Stearns, *Adaptive signal processing*, Prentice Hall, 1985.

[32] R. R. Yager, *Essentials of fuzzy modeling and control*, J. Wiley, 1994.

[33] R. R. Yager and B. Bouchon-Meunier, *Fuzzy logic and soft computing*, World Scientific, 1995.

[34] R. R. Yager and Lofti A. Zadeh (eds.), *Fuzzy sets, neural networks, and soft computing*, World Scientific, 1995.

[35] L. Manevitz Malik Yousef and D. Givoli, *Finite element mesh generation using self-organizing neural networks*, Microcomputers in Civil Engineering **12** (1997), 233–250.

[36] O.C. Zienkiewicz, *The finite element method*, 3 ed., McGraw-Hill, New York, 1977.