An introduction to programming the meshless Element F reeGalerkin method

Article *in* Archives of Computational Methods in Engineering · September 1998

DOI: 10.1007/BF02897874

CITATIONS
275

READS
1,442

2 authors, including:

John Dolbow
Duke University
80 PUBLICATIONS **8,107** CITATIONS

SEE PROFILE

Vol. 5, 3, 207-241 (1998)

Archives of Computational Methods in Engineering
State of the art reviews

An Introduction to Programming the Meshless Element Free Galerkin Method

J. Dolbow and T. Belytschko

Northwestern University
Department of Civil Engineering
2145 Sheridan Road
Evanston, IL 60208-3109, USA

Summary

A detailed description of the Element Free Galerkin(EFG) method and its numerical implementation is presented with the goal of familiarizing scientists and engineers with the new computational technique. In this spirit, an in-depth explanation of the essential concepts which comprise the method is given with specific emphasis on the one-dimensional formulation. First, the EFG algorithm for a one-dimensional problem in linear elastostatics is given; the results are compared to those achievable with standard finite element techniques. A step by step explanation of the MATLAB program used to solve the problem is given with specific references to the EFG method in one-dimension. Next, a simplified two-dimensional implementation to linear elastostatics is described. Results are calculated with the method and the aid of a two-dimensional MATLAB EFG program, and conclusions are drawn about the method and its capabilities. The source programs used to solve both the one-dimensional and two-dimensional problems are provided in the Appendices and are available on the web.

1 INTRODUCTION

The finite element method for the modeling of complex problems in applied mechanics and related fields is well established. It is a robust and thoroughly developed technique, but it is not without shortcomings. The reliance of the method on a mesh leads to complications for certain classes of problems. Consider the modeling of large deformation processes; considerable loss in accuracy arises when the elements in the mesh become extremely skewed or compressed. The growth of cracks with arbitrary and complex paths, and the simulation of phase transformations is also difficult. The use of a mesh in modeling these problems creates difficulties in the treatment of discontinuities which do not coincide with the original mesh lines.

The traditional technique for handling these complications is to remesh the domain of the problem at every step during the evolution of the simulation. This prevents the severe distortion of elements and allows mesh lines to remain coincident with any discontinuities throughout the evolution of the problem. For this purpose, several complex and robust mesh generation routines have been developed. However, this technique requires the projection of field variables between meshes in successive stages of the problem, which leads to logistical problems as well as a degradation of accuracy. In addition, for large three-dimensional problems, the computational cost of remeshing at each step of the problem becomes prohibitively expensive.

To ameliorate these difficulties, a new class of methods have recently been developed which do not require a mesh to discretize the problem. These are methods in which the approximate solution is constructed entirely in terms of a set of nodes, and no elements or characterization of the interrelationship of the nodes is needed to construct the discrete equations. It is then possible to develop discrete equations from a set of nodes and a description of the internal and external surfaces of the model. For the latter purpose, a CAD description, such as a model in three dimensions, may be used, so meshless methods may overcome many of the difficulties associated with meshing for three-dimensional analyses.

ISSN: 1134–3060 Received: June 1996

The principal attraction of meshless methods is the possibility of simplifying adaptivity and problems with moving boundaries and discontinuities, such as phase changes or cracks. In crack growth problems, for example, nodes can be added around a crack tip to capture the stress intensity factors with the desired accuracy; this nodal refinement can be moved with a propagating crack through a background arrangement of nodes associated with the global geometry. Adaptive meshing for a large variety of problems, including linear and nonlinear stress analyses, can be effectively treated by these methods in a simple manner.

This paper presents the details of one form of meshless methods, the element free Galerkin (EFG) method, developed by Belytschko *et al.* [1]. The method shares essential characteristics with many other meshless methods: Reproducing Kernel Particle Methods (Liu *et al.* [8]), hp-clouds (Duarte and Oden [5]), finite point method(Onate et al [10]), and smooth particle hydrodynamics(Monaghan [9]). For a review of these methods, see Belytschko *et al.* [4].

The aim of this paper is to provide the details of the implementation of EFG, including its programming, for scientists and engineers. The details of the method are first explicitly outlined for one-dimensional problems. The essential concepts will be described in the context of a one-dimensional problem in elastostatics, but the differences from other elliptic second order PDE's are insignificant. In addition, an EFG program written in MATLAB for a one-dimensional problem is provided and explained in detail.

Once the basic concepts are understood, the method is easily extended to two and three dimensions. There are some subtle differences between the two-dimensional and one-dimensional formulation, and these will be discussed. Then a problem in two-dimensional linear elastostatics will be solved with the method and the aid of a two-dimensional EFG program. The differences between the 2D and 1D programs will be outlined, and the results will be compared to those achievable with standard finite element methods.

This paper is organized as follows. In Section 2, the details of the EFG method in one dimension are presented. In Section 3, an example problem in one dimensional linear elasticity is provided, along with a comparison between the exact solution and the numerical solution generated by the 1D EFG program. In Section 4, the steps of the program used to solve this problem are outlined, with specific references to the EFG method in one dimension. The extension of the method to two dimensions is outlined in Section 5, and the two-dimensional example is outlined in Section 6. The explanation of the two-dimensional program used to solve this problem is provided in Section 7, along with a flow chart. Conclusions and discussions are given in Section 8. Appendices A and B contain the MATLAB source programs for the 1D and 2D EFG programs respectively. These programs are also available at the website: http://www.tam.nwu.edu/jed/EFG/programs.html.

2 EFG METHOD IN ONE DIMENSION

The element-free Galerkin (EFG) method is a meshless method because only a set of nodes and a description of the model's boundary are required to generate the discrete equations. The connectivity between the nodes and the approximation functions are completely constructed by the method.

The EFG method employs moving least-square (MLS) approximants to approximate the function $u(x)$ with $u^h(x)$. These approximants are constructed from three components: a *weight function of compact support* associated with each node, a *basis*, usually consisting of a polynomial, and a set of *coefficients* that depend on position. The weight function is nonzero only over a small subdomain around a node, which is called its support. The support of the weight function defines a node's *domain of influence*, which is the subdomain over which a particular node contributes to the approximation. The overlap of the nodal domains of influence defines the nodal connectivity.

One attractive property of MLS approximants is that their continuity is related to the continuity of the weight function: if the continuity of the basis is greater than the continuity of the weight function, the resulting approximation will inherit the continuity of the weight function. Therefore, a low order polynomial basis, e.g., a linear basis, may be used to generate highly continuous approximations by choosing an appropriate weight function. Thus, post processing to generate smooth stress and strain fields which is required for C^0 finite element methods is unnecessary in EFG.

Although EFG is considered meshless when referring to shape function construction or function approximation, a mesh will be required for solving partial differential equations (PDE's) by the Galerkin approximation procedure. In order to compute the integrals in the weak form, either a regular background mesh or a background cell structure is used.

This section describes the construction of MLS approximants and the resulting EFG shape functions in one dimension. In the process, the effect of different weight functions on the MLS interpolants is illustrated. The Galerkin procedure to develop the discrete equations for one-dimensional elastostatics is also described.

2.1 MLS Approximants

Consider the discretization of the domain $0 \leq x \leq 1$ given by a set of 11 evenly spaced nodes. Each node has a corresponding 'nodal parameter' or u_I associated with it, which is a parameter governing the function $u(x)$ at that point, but in general $u_I \neq u(x_I)$. This model will be used to both solve the problem in elastostatics and to illustrate the essential features of the EFG method.

The approximation $u^h(x)$ for the function $u(x)$ is posed as a polynomial of order m with non-constant coefficients. The order of the polynomial is defined as the order of the basis. In one dimension, for a linear or a quadratic basis $u^h(x)$ can be written as

$$\begin{aligned} u^h(x) &= a_0(x) + a_1(x)x \quad \text{(linear basis)} \\ u^h(x) &= a_0(x) + a_1(x)x + a_2(x)x^2 \quad \text{(quadratic basis)} \end{aligned}$$

where the unknown parameters $a_j(x)$ vary with x. These approximations are known as moving least squares (MLS) interpolants in curve and surface fitting, and were described by Lancaster and Salkauskas [6]. The local approximation is given by

$$u_L^h(x, \bar{x}) = \sum_{j=0}^{m} p_j(x) a_j(\bar{x}) = \mathbf{p}^T(x)\mathbf{a}(\bar{x}) \tag{1}$$

where p(x) is a complete polynomial of order m

$$\mathbf{p}^T(x) = [1 \quad x \quad x^2, ...x^m] \tag{2}$$

and $\mathbf{a}(x)$ is given by

$$\mathbf{a}^T(x) = [a_0(x) \quad a_1(x) \quad a_2(x), ...a_m(x)] \tag{3}$$

The unknown parameters $a_j(x)$ at any given point are determined by minimizing the difference between the local approximation at that point and the nodal parameters u_I. Let the nodes whose supports include $\mathbf{x}$ be given local node numbers *1* to n. This character of overlapping domains of influence is illustrated in Figure 1 for the method in one dimension.

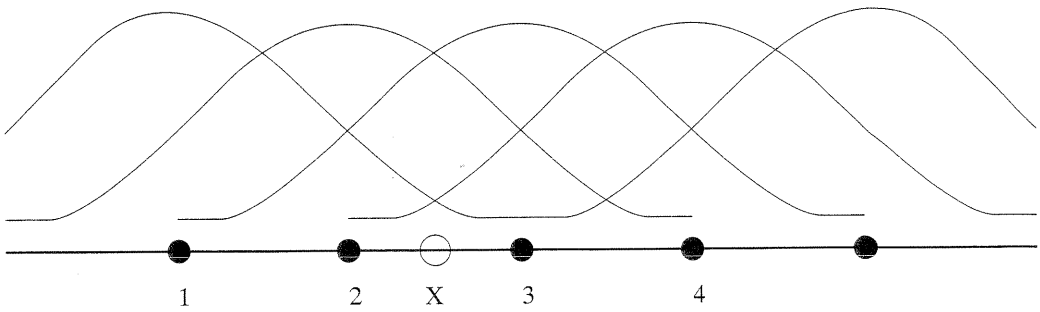

Figure 1. Overlapping domains of influence and local node numbering at point x

To determine $\mathbf{a}(x)$, we then minimize with respect to $\mathbf{a}(x)$ the weighted, discrete L_2 norm given by

$$\begin{aligned} J &= \sum_{I=1}^{n} w(x-x_I)[u_L^h(x_I,x)-u_I]^2 \\ &= \sum_{I=1}^{n} w(x-x_I)[\mathbf{p}^T(x_I)\mathbf{a}(x)-u_I]^2 \end{aligned} \tag{4}$$

where n is the number of nodes in the neighborhood of x for which the weight function $w(x-x_I) \neq 0$, and u_I refers to the nodal parameter of u at $x = x_I$.

The minimum of J in (4) with respect to $\mathbf{a}(x)$ leads to the following set of linear equations:

$$\mathbf{A}(x)\mathbf{a}(x) = \mathbf{B}(x)\mathbf{u} \tag{5}$$

or

$$\mathbf{a}(x) = \mathbf{A}^{-1}(x)\mathbf{B}(x)\mathbf{u} \tag{6}$$

where

$$\begin{aligned} \mathbf{A} &= \sum_{I=1}^{n} w(x-x_I)\mathbf{p}(x_I)\mathbf{p}^T(x_I) \\ &= w(x-x_1)\begin{bmatrix} 1 & x_1 \\ x_1 & x_1^2 \end{bmatrix} + w(x-x_2)\begin{bmatrix} 1 & x_2 \\ x_2 & x_2^2 \end{bmatrix} + ...w(x-x_n)\begin{bmatrix} 1 & x_n \\ x_n & x_n^2 \end{bmatrix} \end{aligned} \tag{7}$$

and

$$\begin{aligned} \mathbf{B}(x) &= [w(x-x_1)\mathbf{p}(x_1), w(x-x_2)\mathbf{p}(x_2), ...w(x-x_n)\mathbf{p}(x_n)] \\ &= \left[w(x-x_1)\begin{bmatrix} 1 \\ x_1 \end{bmatrix} w(x-x_2)\begin{bmatrix} 1 \\ x_2 \end{bmatrix} ...w(x-x_n)\begin{bmatrix} 1 \\ x_n \end{bmatrix}\right] \end{aligned} \tag{8}$$

$$\mathbf{u}^T = [u_1, u_2, ...u_n] \tag{9}$$

The matrices $\mathbf{A}$ and $\mathbf{B}$ have been expanded above for the specific case of a linear basis in one dimension.

By substituting (6) into (1), the MLS approximants can be defined as

$$u^h(x) = \sum_{I=1}^{n} \Phi_I(x)u_I \quad = \mathbf{\Phi}(x)\mathbf{u} \tag{10}$$

where the shape function $\Phi_I(x)$ is defined by

$$\Phi_I(x) = \sum_{j=0}^{m} p_j(x)(\mathbf{A}^{-1}(x)\mathbf{B}(x))_{jI} = \mathbf{p}^T\mathbf{A}^{-1}\mathbf{B}_I \tag{11}$$

where m is the order of the polynomial $p(x)$. To determine the derivatives from the displacement (10), it is necessary to obtain the shape function derivatives. The spatial derivatives of the shape functions are obtained by

$$\begin{aligned} \Phi_{I,x} &= (\mathbf{p}^T\mathbf{A}^{-1}\mathbf{B}_I)_{,x} \\ &= \mathbf{p}^T{}_{,\mathrm{x}}\,\mathbf{A}^{-1}\mathbf{B}_I + \mathbf{p}^T(\mathbf{A}^{-1})_{,\mathrm{x}}\,\mathbf{B}_I + \mathbf{p}^T\mathbf{A}^{-1}\mathbf{B}_{I,\mathrm{x}} \end{aligned} \tag{12}$$

where

$$\mathbf{B}_{I,x}(x) = \frac{dw}{dx}(x - x_I)\mathbf{p}(x_I) \tag{13}$$

and $\mathbf{A}^{-1}{}_{,x}$ is computed by

$$\mathbf{A}^{-1}{}_{,x} = -\mathbf{A}^{-1}\mathbf{A}_{,x}\,\mathbf{A}^{-1} \tag{14}$$

where

$$\begin{aligned} \mathbf{A}_{,x} &= \sum_{I=1}^{n} w(x - x_I)\mathbf{p}(x_I)\mathbf{p}^T(x_I) \\ &= \frac{dw}{dx}(x - x_1)\begin{bmatrix} 1 & x_1 \\ x_1 & x_1^2 \end{bmatrix} + \frac{dw}{dx}(x - x_2)\begin{bmatrix} 1 & x_2 \\ x_2 & x_2^2 \end{bmatrix} \\ &\quad + \ldots\frac{dw}{dx}(x - x_n)\begin{bmatrix} 1 & x_n \\ x_n & x_n^2 \end{bmatrix} \end{aligned} \tag{15}$$

Typical EFG shape functions and their derivatives for the array of eleven nodes in one dimension are shown in Figure 2 and Figure 3, respectively.

It should be noted that EFG shape functions *do not* satisfy the Kronecker delta criterion: $\Phi_I(x_J) \neq \delta_{IJ}$. Therefore they are not interpolants, and the name approximants is used. So $u^h(x_I) \neq u_I$, i.e. the nodal parameters u_I are not the nodal values of $u^h(x_I)$. The approximation to the displacement at the I^th node depends on the nodal parameter u_I as well as the nodal parameters u_1 through u_n corresponding to all other nodes within the domain of influence of node I. This is reflected in the sum given in (10). This property makes the imposition of essential boundary conditions more complicated than with finite elements. Several techniques have been developed to enforce essential boundary conditions, including the use of Lagrange multipliers (Belytschko *et al.* [1]), modified variational principles (Lu *et al.* [8]), and coupling with finite elements (Belytschko *et al.* [2]). In this paper, we will use Lagrange multipliers to enforce the essential boundary conditions.

2.2 Weight Function Description

An important ingredient in the EFG method is the weight function used in (4) through (15). The weight function should be non-zero only over a small neighborhood of x_I, called the domain of influence of node I, in order to generate a set of sparse discrete equations. This is equivalent to stating that the weight functions have compact support. The precise character of $w(x - x_I)$ seems to be unimportant, although it is almost mandatory that it be positive and increase monotonically as $||x - x_I||$ decreases. Furthermore, it is desirable

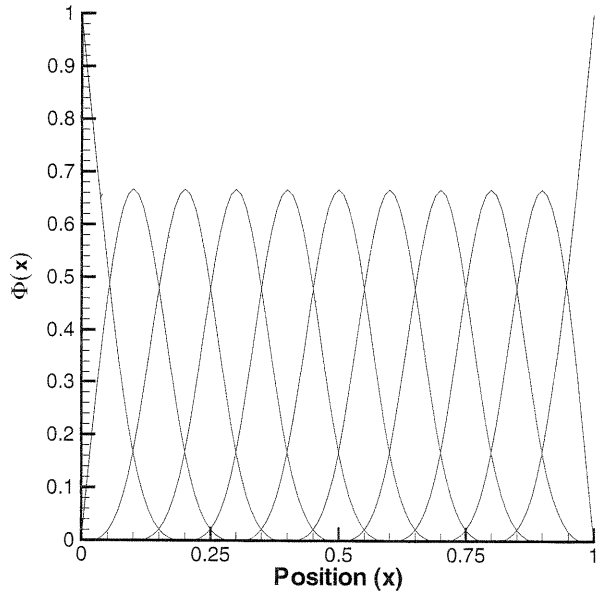

Figure 2. EFG shape functions for an array of eleven nodes in one dimension

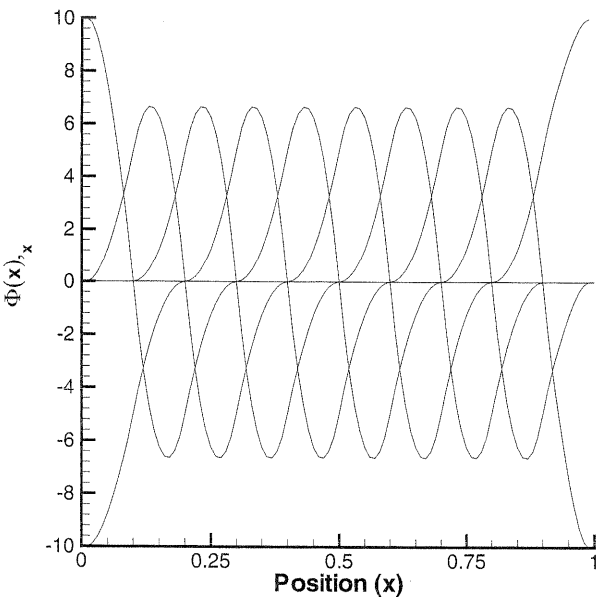

Figure 3. Shape function derivatives for an array of nodes in one dimension

that $w(x - x_I)$ be smooth: if $w(x - x_I)$ is C^1 continuous, then for a polynomial basis, the shape functions $\Phi_I(x)$ are also C^1 continuous (see Lancaster and Saulkauskas [6]).

The choice of the weight function $w(x - x_I)$ affects the resulting approximation $u^h(x)$. As an illustration, consider the three cases shown in Figure 4 where the function $u(x)$ is approximated by $u^h(x)$ using the nodal values at eleven evenly spaced nodes. The MLS approximation $u^h(x)$ is constructed using a linear polynomial basis, $\mathbf{p}^T = [1, x]$. In each case, a representative weight function corresponding to node 6 is plotted along with the

resulting MLS approximation for the entire domain.

In the first example (Figure 4a), the weight function associated with each node is chosen as a constant over the entire domain. The minimization process at each point x involves all points in the domain (n=11) and $J(x)$ in (4) reverts to the standard least squares error norm, resulting in a linear polynomial fit through the data. In this case, the coefficients $\mathbf{a}(x)$ are no longer functions of the space variable x, but are constant everywhere in the domain; thus, the spatial dependence of the coefficients $\mathbf{a}(x)$ is related to the choice of the weight function.

In the second example (Figure 4b), the weights are still constant, but each nodal weight function has compact support since it is only nonzero over a subdomain centered at the node; the size of the subdomain is chosen such that only two nodes will contribute at any point x (n=m=2). These weight functions generate a piecewise linear *interpolation* of the data, equivalent to a linear finite element approximation. By limiting the support size of the weight function, or nodal domain of influence, the minimization becomes a local procedure that only includes the neighbors; the resulting approximation reflects this local character.

In the third example (Figure 4c), the weight functions again possess compact support, but now they are smooth functions that cover larger subdomains such that $(n > m)$. The minimization of $J(x)$ in (4) results in the type of MLS approximations that will be used in the EFG method; the approximation is continuous and smooth even though the polynomial basis is only linear, since the approximation inherits the continuity of the weight function. The approximation possesses local character due to the limited support of the weight function, as in the last example.

Defining $d_I = ||x - x_I||$, and $r = d_I/d_{mI}$, where d_{mI} is the size of the domain of influence of the I^{th} node, the weight function can be written as a function of the normalized radius r. In this paper we will use the cubic spline weight function

$$w(x - x_I) \equiv w(r) = \begin{cases} \frac{2}{3} - 4r^2 + 4r^3 & \text{for} \quad r \leq \frac{1}{2} \\ \frac{4}{3} - 4r + 4r^2 - \frac{4}{3}r^3 & \text{for} \quad \frac{1}{2} < r \leq 1 \\ 0 & \text{for} \quad r > 1 \end{cases} \tag{16}$$

which is shown as a function of r in Figure 5.

The size of the domain of influence at a node, d_{mI} is computed by

$$d_{mI} = d_{max} c_I \tag{17}$$

where d_{max} is a scaling parameter which is typically 2.0-4.0 for a static analysis. The distance c_I is determined by searching for enough neighbor nodes for $\mathbf{A}$ to be regular, i.e. invertible. Referring to (7), it can easily be seen that the matrix $\mathbf{A}$ will be singular if a given node has only one neighbor in its domain of influence. In one dimension with a linear basis, the distance c_I at a node is the maximum distance to the nearest neighbor, insuring that each node will have at least two neighbors in its domain of influence.

Referring to (13) and (15), it can be seen that the spatial derivative of the weight function is necessary to compute the spatial derivative of the $\mathbf{A}$ and $\mathbf{B}$ matrices. This derivative is computed easily using the chain rule of differentiation as

$$\frac{dw_I}{dx} = \frac{dw_I}{dr}\frac{dr}{dx} = \begin{cases} (-8r + 12r^2)\text{sign}(x - x_I) & \text{for} \quad r \leq \frac{1}{2} \\ (-4 + 8r - 4r^2)\text{sign}(x - x_I) & \text{for} \quad \frac{1}{2} < r \leq 1 \\ 0 & \text{for} \quad r > 1 \end{cases} \tag{18}$$

Note that this derivative is continuous over the entire domain.

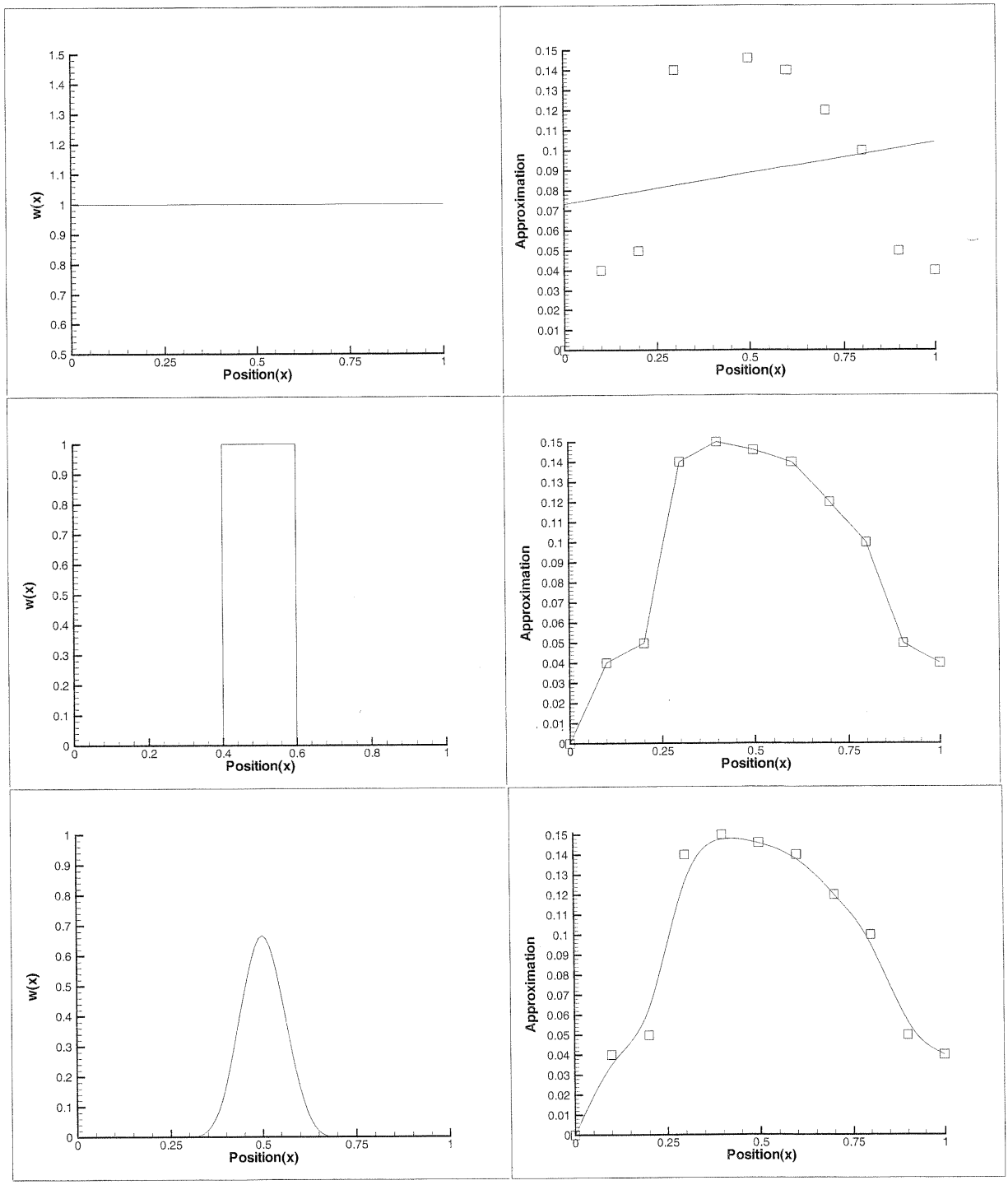

Figure 4. Weight functions and resulting MLS approximations. a) Constant weight function and corresponding approximation for $u^h(x)$. b) Constant weight function with compact support and corresponding approximation. c) Continuous weight function and corresponding approximation

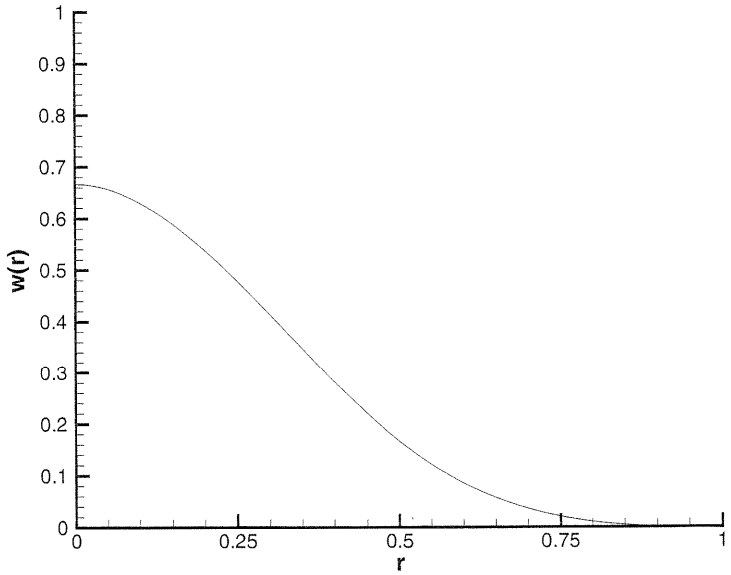

Figure 5. Cubic spline weight function over normalized radius r

2.3 Discrete Equations in 1D

Consider the following one-dimensional problem on the domain $0 \leq x \leq 1$:

$$Eu_{,xx} + b = 0 \quad \text{in} \quad \Omega = (0,1) \tag{19}$$

where $u(x)$ is the displacement, E is Young's modulus, and b is the body force per unit volume. The following specific boundary conditions are chosen:

$$Eu_{,x}\, n = \bar{t} \quad (x = \Gamma_t) \tag{20}$$

$$u = \bar{u} \quad (x = \Gamma_u) \tag{21}$$

To obtain the discrete equations it is first necessary to use a weak form of the equilibrium equation and boundary conditions. The following weak form is used: Let trial functions $\mathbf{u}(x) \in H^1$ and Lagrange multipliers $\lambda \in H^0$ for all test functions $\delta\mathbf{v}(x) \in H^1$ and $\delta\lambda \in H^0$. If

$$\int_0^1 \delta v_{,x}^T\, Eu_{,x}\, dx - \int_0^1 \delta v^T b dx - \delta v^T \bar{t}|_{\Gamma_t} - \delta\lambda^T (u - \bar{u})|_{\Gamma_u} - \delta v^T \lambda|_{\Gamma_u} = 0 \tag{22}$$

then the equilibrium equations (19) and boundary conditions (20) and (21) are satisfied. Note that H^1 and H^0 denote the Hilbert spaces of degree one and zero.

In order to obtain the discrete equations from the weak form, the approximate solution $\mathbf{u}$ and the test function $\delta\mathbf{v}$ are constructed according to (10). The final discrete equations can be obtained by substituing the trial functions and test functions into the weak form (22), yielding the following system of linear algebraic equations;

$$\begin{bmatrix} \mathbf{K} & \mathbf{G} \\ \mathbf{G^T} & 0 \end{bmatrix} \begin{Bmatrix} \mathbf{u} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} \mathbf{f} \\ \mathbf{q} \end{Bmatrix} \tag{23}$$

where

$$\mathbf{K}_{IJ} = \int_0^1 \Phi_I^T{}_{,x} \, E \Phi_{J,x} \, dx \tag{24}$$

$$\mathbf{G}_{IK} = -\Phi_K|_{\Gamma_{uI}} \tag{25}$$

$$\mathbf{f}_I = \Phi_I t_x|_{\Gamma_t} + \int_0^1 \Phi_I b dx \tag{26}$$

$$\mathbf{q}_K = -\bar{u}_K \tag{27}$$

where Γ_u and Γ_t are the essential and natural boundaries, respectively. These equations are assembled by integrating over the domain of the problem using Gauss quadrature.

To evaluate the integrals in (24) and (26), it is necessary to define integration cells over the domain of the problem. These cells can be defined arbitrarily, but a sufficient number of quadrature points must be used to obtain a well conditioned, nonsingular system of equations (23). In one dimension, one example of a cell structure is to set the quadrature cells equal to the intervals between the nodes. Once the cells and corresponding quadrature points are defined, the discrete equations are assembled by looping over each quadrature point.

In the assembly of the discrete equations (23), at each quadrature point the nodes whose domain of influence contain the point x_Q are first determined. These are referred to as 'support' nodes for the quadrature point. The shape functions (11) and shape function derivatives ,(12) corresponding to each of these support nodes are calculated at the point x_Q. Then the contributions to the integrals in (24) and (26) are calculated and assembled. The $\mathbf{G}$ matrix (25) is assembled separately by evaluating the shape functions at each of the essential boundaries. A similar procedure is used at the natural boundaries to evaluate the applied traction contribution to the force vector (26).

3 ONE-DIMENSIONAL NUMERICAL EXAMPLE

Consider the implementation of the EFG method in one dimension for a problem in linear elastostatics. Figure 6 shows the example problem, a one-dimensional bar of unit length subjected to a linear body force of magnitude x.

The displacement of the bar is fixed at the left end, and the right end is traction free. The bar has a constant cross sectional area of unit value, and modulus of elasticity E.

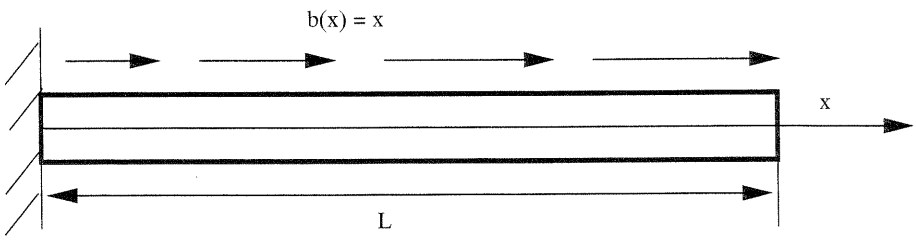

Figure 6. The one-dimensional example problem - a bar of unit length subjected to a linear body force and fixed at the point $x = 0$

The equilibrium equation and boundary conditions for this problem are given by;

$$\begin{aligned} Eu_{,xx} + x &= 0 \qquad 0 < x < 1 \\ u(0) &= 0 \\ u_{,x}(1) &= 0 \end{aligned} \tag{28}$$

The exact solution to the above problem is given by;

$$u(x) = \frac{1}{E}\left[\frac{1}{2}x - \frac{x^3}{6}\right] \tag{29}$$

We shall use this solution to examine the quality of the EFG solution.

Figure 7 is a comparison of the EFG solution to the exact solution for both the displacements and strains along the bar.

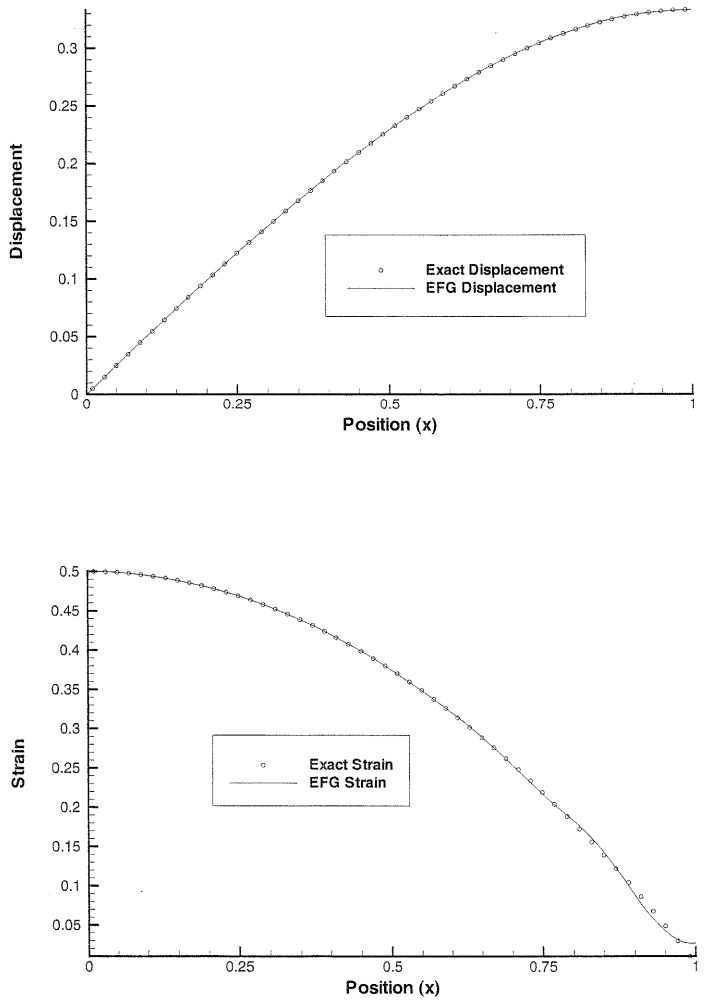

Figure 7. Comparison of EFG and exact results for a) displacements and b) strains for one-dimensional bar problem

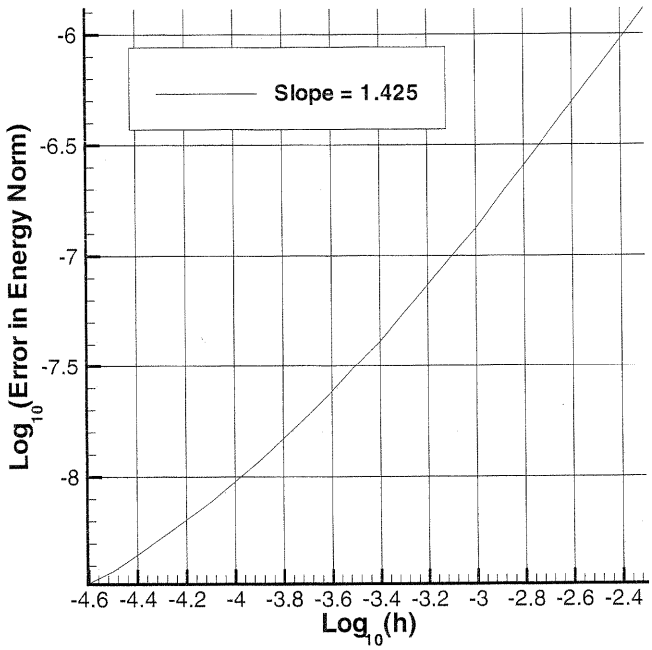

Figure 8. Rate of convergence in energy norm for the one-dimensional problem

These results were obtained using eleven nodes along the length of the bar and one point integration exactly like the program provided in Appendix A. The displacement field calculated with EFG is nearly exact, while there is only a slight error in the strain field. The strain field exhibits an anomaly at the right hand boundary which is typical in EFG: the strain has a marked divergence right at the boundary where it should vanish. It should be noted that these results were plotted using several more sampling points along the bar than there are nodes.

Figure 8 is a plot of the rate of convergence in energy norm for this problem using the EFG method. The rate of convergence in energy is calculated using

$$\text{energy norm} = \left\{ \frac{1}{2} \int_0^1 (u_{,x}^{\text{NUM}} - u_{,x}^{\text{EXACT}})^T E (u_{,x}^{\text{NUM}} - u_{,x}^{\text{EXACT}}) dx \right\}^{\frac{1}{2}} \tag{30}$$

The slope of the line in Figure 8 is approximately 1.425, as indicated in the figure. These results were obtained using one point integration as the number of nodes in the model was increased from 10 to 100. The x-axis is labeled as the cell size, which was always set to the space between the nodes. This rate of convergence is greater than that achievable with linear displacement finite elements (the asymptotic value is 1.0 for linear elements), as was noted before in Belytschko, Lu, and Gu [1].

4 EFG PROGRAM DESCRIPTION

The program provided in Appendix A solves the one-dimensional problem given by Eq. (28). The program has been written in MATLAB, a language which allows matrix manipulations to be performed easily. The language is very similar to FORTRAN, and the program can be adapted easily. The program has been written with a minimum of generality, and it is

aimed at solving the specific problem outlined previously. This shortens the program and also keeps it readable. What follows is a detailed explanation of the steps in the program with specific references to the equations outlined in Section 3.

1	Set up nodal coordinates and integration cells.
2	Set parameters for weight function, material properties.
3	Set up integration points, weights, and Jacobian for each cell.
4	Loop over integration points.
	4.a Calculate weights at each node for given integration point x_G.
	4.b Calculate shape functions and derivatives at point x_G.
	4.c Assemble contributions to K matrix (24) and force vector (26).
	4.d Form G array (25) at first integration point.
5	Construct q vector (27) and assemble discrete equations (23).
6	Solve for nodal parameters u_I.

Table 1. Flowchart of 1D EFG program

Table 1 provides a flow chart which outlines the essential steps of the program. In the first step of the program, the nodal coordinates are set up and the number of integration cells are determined. The nodes are spaced uniformly along the length of the bar, which is not necessary but simplifies the program. Then in the second step the parameters for the weight function are set, including the variable d_{max} which is set to 2.0. This value is used subsequently to determine d_{mI} at each node. Since the nodes are uniformly spaced, the distance c_I is set equal to the distance between the nodes. Therefore, d_{mI} at each node is the same value and is equal to $d_{max}c_I$, in accordance with (17).

In the third step of the program, the integration points, weights, and Jacobian for each integration cell are determined. In the program provided, the integration cells are chosen to coincide with the intervals between the nodes, and one-point quadrature is used to integrate the Galerkin weak form (22). One point quadrature is equivalent to the trapezoidal rule, and it is used in this program for the sake of simplicity. Since the nodes are uniformly spaced, the Jacobian is the same value for each cell, and it is equal to one half the distance between the nodes. The locations of the integration points, their respective weights, and the Jacobian are stored in the array **gg**. Note that the first value in **gg** is set to 0.0. This is the point where the essential boundary condition is enforced, and in the loop over the integration points it will be used to assemble the **G** matrix (25).

To assemble the discrete equations, we loop over each integration point in the fourth step of the program. The distance between the integration point x_G and each node is computed. Then the weight w and its derivative $\frac{dw}{dx}$ are computed for each node, using the normalized distance $r = \frac{\|x-x_I\|}{d_{mI}}$. The weights are calculated in accordance with (16), and the spatial derivatives are calculated using (18). Note that for nodes whose domain of support do not contain the point x_G, the weight and its derivative are set to zero. As a consequence, the contributions from these nodes to the Galerkin form (22) will be zero. In general, setting the weight and its derivative to zero for these nodes can be avoided by only looping over the integration point's support nodes. However, in a one-dimensional program, there is relatively little extra computational cost in looping over all of the nodes in the domain, and it also simplifies the program.

Since a linear basis is used, the **p** vector is set to $[1 \quad x_1]^T$ for each node. Then the **B** matrix is calculated in accordance with (8), and the **A** matrix and its spatial derivative

$\mathbf{A}_{,x}$ are assembled in accordance with (7) and (14) respectively. Note that the $\mathbf{A}$ matrix and its derivative are assembled by looping over all of the nodes in the domain for each integration point.

Once the $\mathbf{A}$ matrix is assembled, its inverse is calculated using the intrinsic MATLAB function inv($\mathbf{A}$). Although the LU decomposition to be outlined in Section 5 could be performed at this step, the computational savings are not significant in one dimension and it is much simpler to directly compute the inverse of the $\mathbf{A}$ matrix. Then the shape functions are calculated using (11), and their derivatives are calculated using (12). Note that since $\mathbf{p}^T = [1 \quad x]$, $\mathbf{p}_{,x}^T = [0 \quad 1]$. The values of the shape functions and their derivatives are stored in the arrays **phi** and **dphi** respectively.

Once the shape functions and derivatives corresponding to each of the nodes are calculated at the integration point, the contributions to the stiffness matrix (24) and the force vector (26) are determined. The exception is at the first integration point, where the shape functions are used to assemble the $\mathbf{G}$ matrix, and then the loop continues. Once all the contributions from each integration point have been assembled into the stiffness matrix and the force vector, the loop ends. Note that since there is no applied traction in this problem, the force vector is completely assembled at this point.

With the $\mathbf{K}$ matrix, $\mathbf{G}$ matrix, and force vector completely assembled, the discrete equations (23) are assembled in the fifth step of the program. Since the prescribed displacement at the left end of the bar is zero, the $\mathbf{q}$ vector is equal to zero as outlined in equation (27). The system of equations is then solved in the final step and the nodal parameters u_I are determined. Note that these are not the nodal displacements. The displacement at each point needs to be calculated using equation (10).

5 EFG IN TWO DIMENSIONS

The essential concepts outlined in Section 2 do not change for the EFG method in two or three dimensions. The scalar variable x simply becomes a vector $\mathbf{x}^T = [x \quad y]$ in equations (1) through (15). For example, for a linear basis in two dimensions, the $\mathbf{p}$ vector is given by

$$\mathbf{p}^T = [1 \quad x \quad y] = \mathbf{p}^T(\mathbf{x}) \tag{31}$$

Similarly, the $\mathbf{A}$ matrix for a linear basis in two dimensions is given by

$$\begin{aligned}
\mathbf{A}(\mathbf{x}) &= \sum_{I=1}^{n} w(\mathbf{x}-\mathbf{x}_I)\mathbf{p}(\mathbf{x}_I)\mathbf{p}^T(\mathbf{x}_I) \\
&= w(\mathbf{x}-\mathbf{x}_1)\begin{bmatrix} 1 & x_1 & y_1 \\ x_1 & x_1^2 & x_1y_1 \\ y_1 & x_1y_1 & y_1^2 \end{bmatrix} + w(\mathbf{x}-\mathbf{x}_2)\begin{bmatrix} 1 & x_2 & y_2 \\ x_2 & x_2^2 & x_2y_2 \\ y_2 & x_2y_2 & y_2^2 \end{bmatrix} + \ldots \\
&\quad w(\mathbf{x}-\mathbf{x}_n)\begin{bmatrix} 1 & x_n & y_n \\ x_n & x_n^2 & x_ny_n \\ y_n & x_ny_n & y_n^2 \end{bmatrix}
\end{aligned} \tag{32}$$

where $\mathbf{x} = [1 \quad x \quad y]$ and $\mathbf{x}_I = [1 \quad x_1 \quad y_1]$. The two-dimensional shape functions are given by

$$\Phi_I(\mathbf{x}) = \sum_{j=0}^{m} p_j(\mathbf{x})(\mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x}))_{jI} = \mathbf{p}^T\mathbf{A}^{-1}\mathbf{B}_I \tag{33}$$

There are, however, some differences and extensions which are not so trivial and consequently need further explanation. These include the use of tensor product weights, a

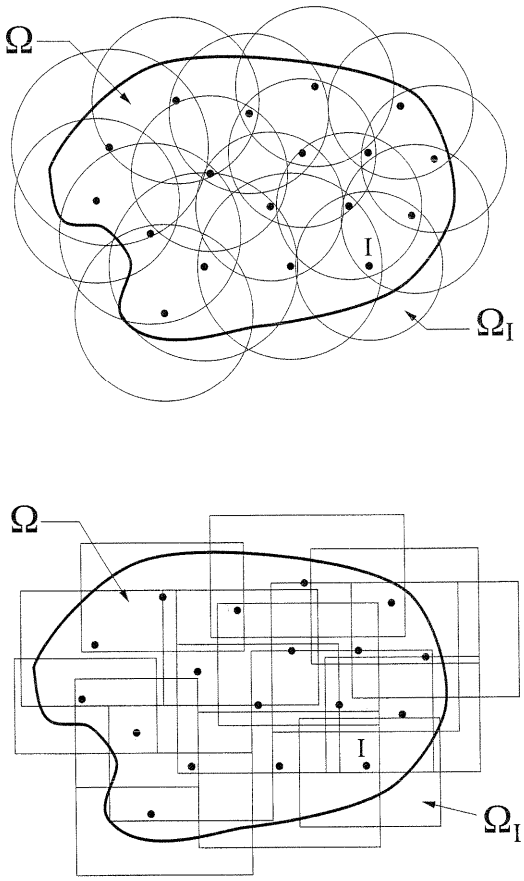

Figure 9. Domains of influence in two dimensions using a) circular and b) tensor product domains

technique for calculating the derivatives of shape functions efficiently, enforcement of the essential boundary conditions, and the development of the discrete equations from the Galerkin weak form. These concepts will be explained in detail in this section.

5.1 Tensor Product Weights

In two dimensions a node's domain of influence covers an area in the domain. The choice of the shape of this domain is arbitrary. However, circular domains or square domains have been used most frequently; anisotropic domains should be useful in many problems but have not yet been studied extensively. The two different schemes are illustrated in Figure 9.

In this paper, tensor product weights will be used with the cubic spline weight function for the implementation of the method in two dimensions.

The tensor product weight function at any given point is given by

$$w(\mathbf{x} - \mathbf{x}_I) = w(r_x) \cdot w(r_y) = w_x \cdot w_y \tag{34}$$

where $w(r_x)$ or $w(r_y)$ is given by (16) with r replaced by r_x or r_y respectively; r_x and r_y are given by

$$r_x = \frac{||x - x_I||}{d_{mx}}$$

$$r_y = \frac{||y - y_I||}{d_{my}} \tag{35}$$

where

$$\begin{aligned} d_{mx} &= d_{max} \cdot c_{xI} \\ d_{my} &= d_{max} \cdot c_{yI} \end{aligned} \tag{36}$$

and c_{xI} and c_{yI} are determined at a particular node by searching for enough neighbor nodes to satisfy the basis in both directions. This is similar to the process outlined in Section 2. The requirement is that the $\mathbf{A}$ matrix in (32) be non-singular everywhere in the domain, and thus invertible, which is necessary to compute the shape function. If the nodes are uniformly spaced, the values c_{xI} and c_{yI} correspond to the distance between the nodes in the x and y directions, respectively.

The derivative of the weight function in (34) is calculated by

$$\begin{aligned} w_{,x} &= \frac{dw_x}{dx} \cdot w_y \\ w_{,y} &= \frac{dw_y}{dy} \cdot w_x \end{aligned} \tag{37}$$

where $\frac{dw_x}{dx}$ and $\frac{dw_y}{dy}$ are given by (18) with r replaced by r_x and r_y respectively.

5.2 Efficient Shape Function Calculation

To compute the shape functions from (33) it is necessary to invert the $\mathbf{A}$ matrix. In one dimension, this operation is not computationally expensive, but in two or three dimensions it becomes burdensome. An alternative approach that is much more computationally efficient was proposed by Belytschko *et al.* [3]. This method involves the LU decomposition of the $\mathbf{A}$ matrix. The shape function in (33) can be written as

$$\mathbf{\Phi}_I(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{A}^{-1}(\mathbf{x})\mathbf{B}_I(\mathbf{x}) = \gamma^T(\mathbf{x})\mathbf{B}_I(\mathbf{x}) \tag{38}$$

which leads to the relationship

$$\mathbf{A}(\mathbf{x})\gamma(\mathbf{x}) = \mathbf{p}(\mathbf{x}) \tag{39}$$

The vector $\gamma(\mathbf{x})$ can be determined using an LU decomposition of the matrix $\mathbf{A}$ followed by back substitution.

The derivatives of $\gamma(\mathbf{x})$ can be computed similarly, which leads to a computationally efficient procedure for computing the derivatives of $u^h(x)$. Taking the spatial derivative of (39), we have

$$\mathbf{A}_{,\mathbf{x}}(\mathbf{x})\gamma(\mathbf{x}) + \mathbf{A}(\mathbf{x})\gamma_{,\mathbf{x}}(\mathbf{x}) = \mathbf{p}_{,\mathbf{x}}(\mathbf{x}) \tag{40}$$

which can be rearranged as

$$\mathbf{A}(\mathbf{x})\gamma_{,\mathbf{x}}(\mathbf{x}) = \mathbf{p}_{,\mathbf{x}}(\mathbf{x}) - \mathbf{A}_{,\mathbf{x}}(\mathbf{x})\gamma(\mathbf{x}) \tag{41}$$

Thus the derivative of $\gamma(\mathbf{x})$ can be calculated using the same LU decomposition obtained from (39). This leads to a simple relationship for the derivative of the shape function in (38) given by

$$\mathbf{\Phi}_{I,\mathbf{x}} = \gamma_{,\mathbf{x}}{}^T(\mathbf{x})\mathbf{B}_I(\mathbf{x}) + \gamma(\mathbf{x})^T\mathbf{B}_{I,\mathbf{x}}(\mathbf{x}) \tag{42}$$

5.3 Enforcement of Essential Boundary Conditions

As mentioned previously, several different techniques have been developed to enforce the essential boundary conditions for EFG. The most efficient and accurate method couples EFG with finite elements along the essential boundary. Finite elements are used along the boundary of the model while EFG nodes are used in the interior. Since finite element shape functions satisfy the identity $N_I(x_J) = \delta_{IJ}$, the imposition of essential boundary conditions is straightforward. This technique also allows the method to be incorporated in standard finite element packages. For a more detailed explanation of this technique, see Belytschko *et al.* [2].

In this paper, Lagrange multipliers will be used to enforce the essential boundary conditions. This method has been selected in this paper because it is the simplest technique which gives acceptably accurate results. It is also a familiar method to many scientists and engineers, and it is easy to program and explain. As this is an introductory paper to the EFG method, these qualities make the use of this technique attractive here. The details of the theoretical implementation of this technique will be developed in the following segment, and the numerical implementation will be outlined in Section 7.

5.4 Discrete Equations in 2D

The discrete equations in two dimensions are very similar to those of the one-dimensional formulation. Line integrals in x become surface integrals in x and y and there are some other subtle differences. For the sake of completeness, they are developed from the Galerkin weak form as follows.

The weak form for the EFG method in two dimensions is posed as follows. Consider the following two-dimensional problem on the domain Ω bounded by Γ:

$$\nabla \cdot \sigma + \mathbf{b} = 0 \quad \text{in} \quad \Omega \tag{43}$$

where σ is the stress tensor, which corresponds to the displacement field $\mathbf{u}$ and $\mathbf{b}$ is a body force vector. The boundary conditions are as follows:

$$\sigma \cdot \mathbf{n} = \bar{\mathbf{t}} \quad \text{on} \quad \Gamma_t \tag{44}$$

$$\mathbf{u} = \bar{\mathbf{u}} \quad \text{on} \quad \Gamma_u \tag{45}$$

in which the superposed bar denotes prescribed boundary values, and $\mathbf{n}$ is the unit normal to the domain Ω.

The variational (or weak) form of the equilibrium equation is posed as follows. Consider trial functions $\mathbf{u}(\mathbf{x}) \in H^1$ and Lagrange multipliers $\lambda \in H^0$ for all test functions $\delta\mathbf{v}(\mathbf{x}) \in H^1$ and $\delta\lambda \in H^0$. Then if

$$\int_\Omega \delta(\nabla_s \mathbf{v}^T) : \sigma d\Omega - \int_\Omega \delta\mathbf{v}^T \cdot \mathbf{b} d\Omega - \int_{\Gamma_t} \delta\mathbf{v}^T \cdot \bar{\mathbf{t}} d\Gamma - \int_{\Gamma_u} \delta^T \cdot (\mathbf{u} - \bar{\mathbf{u}}) d\Gamma - \int_{\Gamma_u} \delta\mathbf{v}^T \cdot \lambda d\Gamma = 0 \quad \forall \delta\mathbf{v} \in H^1, \delta\lambda \in H^0 \tag{46}$$

then the equilibrium equation (43) and the boundary conditions (44) and (45) are satisfied. Here $\nabla_s \mathbf{v}^T$ is the symmetric part of $\nabla \mathbf{v}^T$; H^1 and H^0 denote the Hilbert spaces of degree one and zero, respectively. Note that the trial functions do not satisfy the essential boundary conditions, so that they are imposed with Lagrange multipliers.

In order to obtain the discrete equations from the weak form (46), the approximate solution $\mathbf{u}$ and the test function $\delta\mathbf{v}$ are constructed according to the two-dimensional analogy of equation (10). The Lagrange multiplier λ is expressed by

$$\begin{aligned} \lambda(\mathbf{x}) &= N_I(s)\lambda_I, \quad \mathbf{x} \in \Gamma_u \\ \delta\lambda(\mathbf{x}) &= N_I(s)\delta\lambda_I, \quad \mathbf{x} \in \Gamma_u \end{aligned} \tag{47}$$

where $N_I(s)$ is a Lagrange interpolant and s is the arclength along the boundary; the repeated indices designate summations. The final discrete equations can be obtained by substituting the trial functions, test functions and equations (47) into the weak form (46), which yields

$$\begin{bmatrix} \mathbf{K} & \mathbf{G} \\ \mathbf{G}^{\mathbf{T}} & 0 \end{bmatrix} \begin{Bmatrix} \mathbf{u} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} \mathbf{f} \\ \mathbf{q} \end{Bmatrix} \tag{48}$$

and

$$\mathbf{K}_{IJ} = \int_\Omega \mathbf{B}_I^{\mathbf{T}} \mathbf{D} \mathbf{B}_J d\Omega \tag{49a}$$

$$\mathbf{G}_{IK} = -\int_{\Gamma_u} \mathbf{\Phi}_I \mathbf{N}_{\mathbf{K}} d\Gamma \tag{49b}$$

$$\mathbf{f}_I = \int_{\Gamma_t} \mathbf{\Phi}_I \bar{\mathbf{t}} d\Gamma + \int_\Omega \mathbf{\Phi}_I \mathbf{b} d\Omega \tag{49c}$$

$$\mathbf{q}_K = -\int_{\Gamma_u} N_K \bar{\mathbf{u}} d\Gamma \tag{49d}$$

where

$$\mathbf{B}_I = \begin{bmatrix} \mathbf{\Phi}_{I,x} & 0 \\ 0 & \mathbf{\Phi}_{I,y} \\ \mathbf{\Phi}_{I,y} & \mathbf{\Phi}_{I,x} \end{bmatrix} \tag{50a}$$

$$\mathbf{N}_{\mathbf{K}} = \begin{bmatrix} N_K & 0 \\ 0 & N_K \end{bmatrix} \tag{50b}$$

$$\mathbf{D} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \quad \text{for plane stress} \tag{50c}$$

in which a comma designates a partial derivative with respect to the indicated spatial variable; E and ν are Young's modulus and Poisson's ratio, respectively.

6 TWO-DIMENSIONAL NUMERICAL EXAMPLE

In this section, a classical problem in linear elastostatics will be described and solved using an EFG program written in MATLAB. The test problem is the Timoshenko beam problem. This example serves to illustrate the accuracy of the EFG method in calculating various stress fields in two dimensions. Other classical problems have been examined extensively in previous papers (see Belytschko *et al.* [1] for example), and as such they are not covered here.

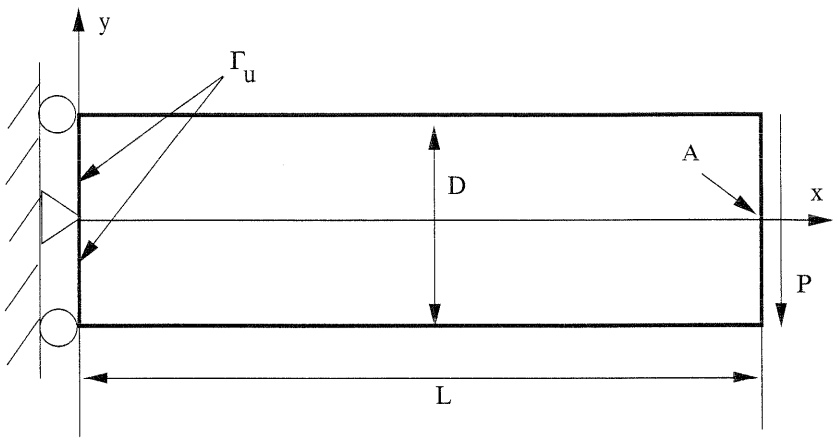

Figure 10. The two-dimensional example problem - a cantilever beam for comparison to the analytic solution from Timoshenko and Goodier

6.1 Timoshenko Beam Problem

Consider a beam of length L subjected to a parabolic traction at the free end as shown in Figure 10. The beam has characteristic height D and is considered to be of unit depth and is assumed to be in a state of plane stress. The exact solution to this problem is given by Timoshenko and Goodier [11]:

$$u_x = -\frac{Py}{6EI}\left[(6L-3x)x + (2+\nu)\left(y^2 - \frac{D^2}{4}\right)\right] \tag{51a}$$

$$u_y = \frac{P}{6EI}\left[3\nu y^2(L-x) + (4+5\nu)\frac{D^2 x}{4} + (3L-x)x^2\right] \tag{51b}$$

where the moment of inertia I of the beam is given by

$$I = \frac{D^3}{12}$$

The stresses corresponding to the displacements (51) are

$$\sigma_x(x,y) = -\frac{P(L-x)y}{I} \tag{52a}$$

$$\sigma_y(x,y) = 0 \tag{52b}$$

$$\sigma_{xy}(x,y) = -\frac{Py}{2I}\left(\frac{D^2}{4} - y^2\right) \tag{52c}$$

The problem was solved for the plane stress case with E = 3.0x10^7, ν = 0.3,D=12,L=48, and P=1000. The regular mesh of nodes and the background mesh used to integrate the Galerkin weak form (46) are shown in Figure 11.

In each integration cell, 4x4 Gauss quadrature was used to evaluate the stiffness of the EFG. The solutions were obtained using a linear basis function with the cubic spline weight function and a d_{max} value of 3.5. These parameters are consistent with the 2D EFG program provided in Appendix B.

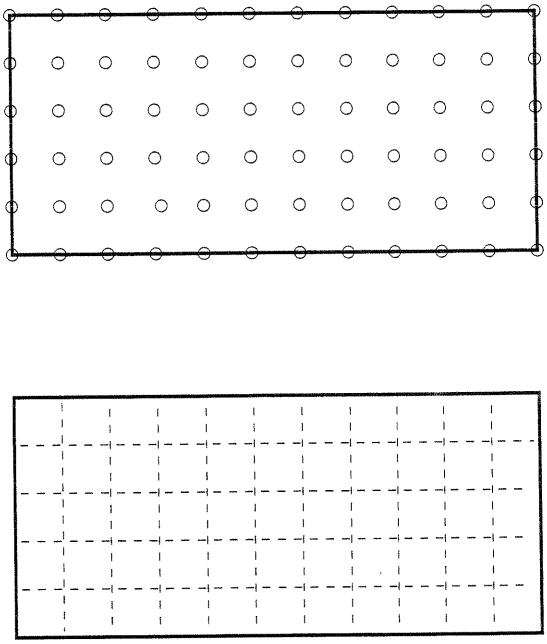

Figure 11. a) Regular nodal arrangement and b) background mesh used for the beam problem

Table 2 compares the EFG calculation for the vertical displacement at point A on the beam in Figure 10 with the exact vertical displacement given in (51b). This calculation was performed for models discretized with 10,27,85, and 175 nodes. This table shows excellent agreement between the exact solution and the EFG solution.

Number of Nodes	u_y exact	u_y EFG	% error
10	-.0089	-.008124	-8.7
27	-.0089	-.008845	-0.6
85	-.0089	-.008897	-0.03
175	-.0089	-.008899	-0.01

Table 2. Comparison of vertical displacement at end of beam

Figure 12 illustrates the comparison between the stresses calculated at the center of the beam (L=24) and the exact stresses (52). These plots show excellent agreement between the EFG results and the analytical solution. Of particular note is the fact that the EFG solution approximates the natural boundary condition ($\tau_{xy} = 0$) at the free surface of the beam very well. This is a characteristic not readily achievable with finite elements.

Figure 13 is a plot of the rate of convergence in energy error for this problem using the EFG method. The rate of convergence in energy is calculated using

$$\text{energy norm} = \left\{ \frac{1}{2} \int_{\Omega} (\varepsilon^{\text{NUM}} - \varepsilon^{\text{EXACT}})^T D (\varepsilon^{\text{NUM}} - \varepsilon^{\text{EXACT}}) d\Omega \right\}^{\frac{1}{2}} \tag{53}$$

where ε is defined as the symmetric gradient of the displacement $\mathbf{u}$. The value h was chosen to be the horizontal distance between the nodes in the model, and in each case a d_{max} of

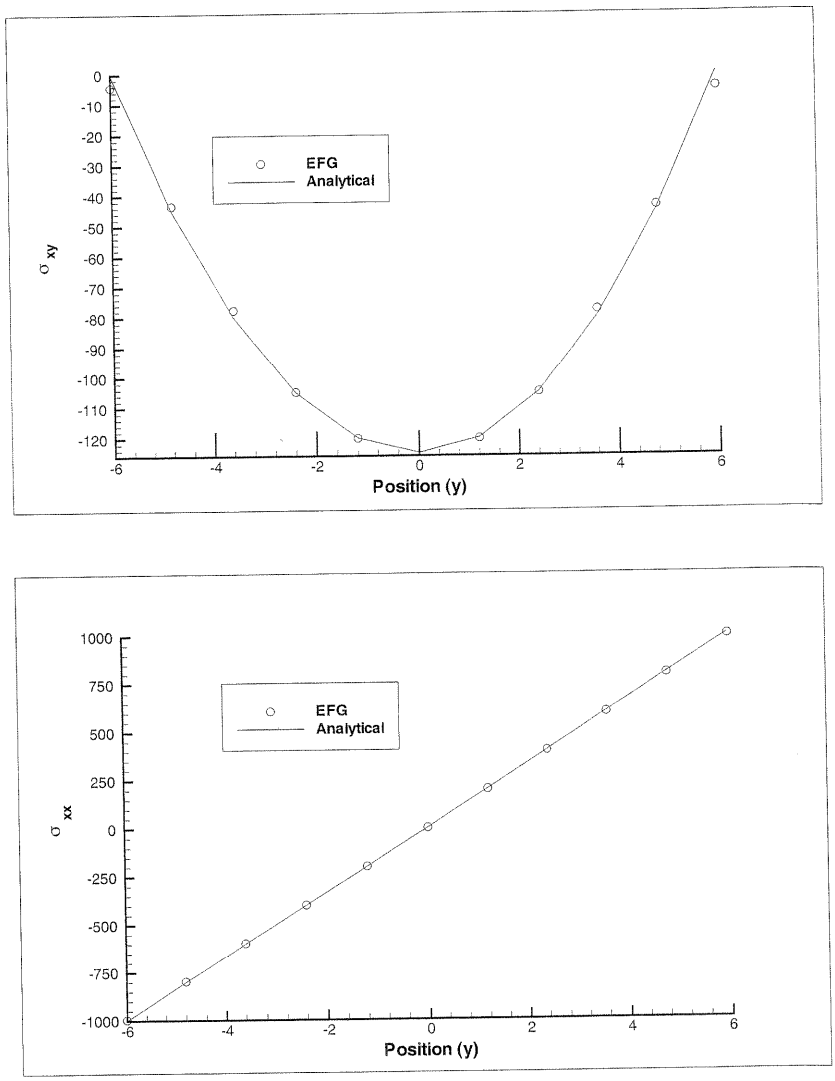

Figure 12. Comparison of a) shear stresses and b)normal stresses in beam

2.0 with the cubic spline weight function (16) was used. In addition, 4x4 Gauss quadrature was used to integrate the Galerkin weak form (46), and 4 point Gauss quadrature was used to integrate the $\mathbf{q}$ vector (49d) and $\mathbf{G}$ matrix (49b) along the essential boundary, as well as the force vector (49c) along the traction boundary. The slope of the line in Figure 13 is approximately 1.71, which is a greater rate of convergence than achievable with linear finite elements. These results have been reported previously in Belytschko *et al.* [1].

It should also be mentioned that the EFG method is not limited to a uniform grid of nodes. In addition, no locking is observed for the case of plane strain when Poisson's ratio is allowed to approach 0.5. These facts were demonstrated in Belytschko *et al.* [1]. This reference also includes several other numerical examples, including patch tests, the infinite plate with the hole problem, and an edge crack problem.

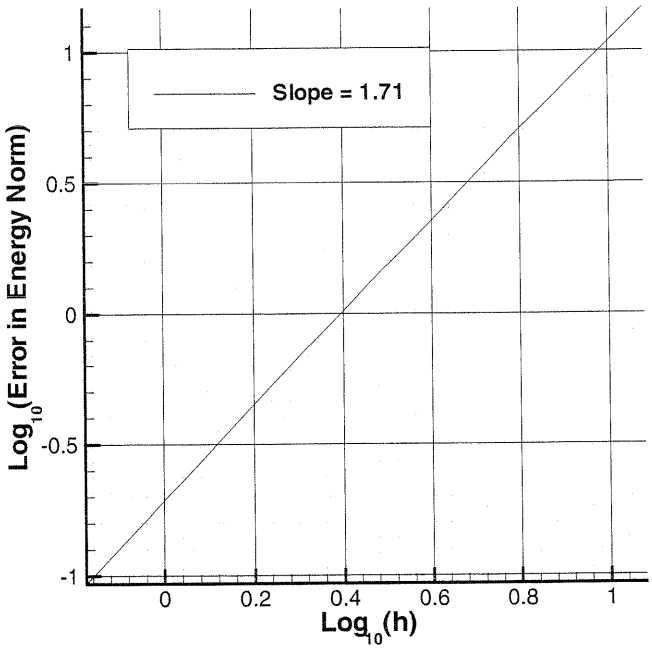

Figure 13. Rate of convergence in energy norm for the two-dimensional problem

7 2D PROGRAM DESCRIPTION

The two-dimensional MATLAB program used to calculate the results for the beam in bending problem in Section 6 is provided in Appendix B. The program is a straightforward extension of the one-dimensional program provided in Appendix A and explained in Section 4. In addition to the solution, the program calculates the error in strain energy for the beam problem. This program has been specialized for the beam problem so as to keep the program as short as possible. The beam problem is attractive because a uniform distribution of nodes and a regular background mesh can be used. While these properties are not required for the method, they simplify the program greatly, much in the same way that uniform nodal spacing simplified the one-dimensional program.

Table 3 provides a flow chart for the 2D EFG program. The reader will notice that the two programs are very similar in structure, and as such only the major differences will be explained in detail in this section. The primary differences include the formation of the $\mathbf{q}$ vector (49d) and $\mathbf{G}$ matrix (49b) at the essential boundary, as well as the calculation of the force vector $\mathbf{f}$ (49c) at the traction boundary. The program also calculates the displacements at each of the nodes using the nodal parameters $\mathbf{u}_I$ and the stresses at each of the quadrature points. These calculations were not performed in the one-dimensional program, but are provided in the 2D program for the sake of illustration. As these calculations are relatively straightforward, they will not be explained in detail in this section.

In the 9^{th} step of the program (see Table 3) Gauss points, weights, and their respective Jacobians are set up along the essential boundary (x=0) and the traction boundary (x=L). 4 point Gauss quadrature is used along each boundary, which simplifies the program. This step is required to integrate the $\mathbf{G}$ matrix, the $\mathbf{q}$ vector, and force vector. In the tenth step of the program, a loop is conducted over the Gauss points set up along the essential boundary. At each point, the contributions to the $\mathbf{G}$ matrix and the $\mathbf{q}$ vector corresponding to the nodal neighbors at that point are determined. Referring to equations (48) and (49b),

1	Define the physical dimensions and material properties.
2	Define the plane stress D matrix (5.4).
3	Set up the nodal coordinates for a uniform mesh.
4	Determine the domain of influence for each node in the mesh.
5	Set up quadrature cells in the domain.
6	Set up Gauss points, weights, and Jacobian for each cell.
7	Loop over Gauss points.
	7.a Determine nodes in the neighborhood of the Gauss point.
	7.b Determine weights (34), shape functions (38), and shape function derivatives (42) for nodes 1 to n.
	7.c Assemble **B** matrix (50a).
	7.d Add contributions to **K** matrix (49a).
8	Determine nodes on traction boundary and essential boundary.
9	Set up Gauss points along traction boundary and essential boundary.
10	Integrate forces along traction boundary to form **f** vector (49c).
11	Integrate Lagrange multipliers along essential boundary to form the **G** matrix (49b) and **q** vector (49d).
12	Enforce essential boundary conditions using Lagrange multipliers.
13	Solve for nodal parameters u_I.
14	Loop over Gauss points.
	14a. Determine exact and analytical stresses at quadrature points.
	14b. Assemble contributions to error norm (53).
15	Print total error in strain energy.

Table 3. Flowchart of 2D EFG program

it can be seen that the rows of the **G** matrix correspond to the nodes affected by the values of the Lagrange multipliers stored in the columns of the **G** matrix. Note that since there are two degrees of freedom at each node, there are two Lagrange multipliers λ_I for each node along the boundary. A similar process is performed along the traction boundary to determine the appropriate nodal forces in the force vector **f** in the 10^{th} step of the program.

After the discrete equations (48) are assembled and the nodal parameters $\mathbf{u}_I$ are calculated, an additional loop over the Gauss points set up over the entire domain of the beam is conducted. This loop calculates the exact and analytical stresses at each of the Gauss points and assembles contributions to the energy norm (53). It should be noted that the data used to generate the plots in Section 6 for the stresses in the beam were obtained using a modification of this segment of the program. All that was required was to set up additional 'sampling points' throughout the center of the beam. In order to obtain convergence data for this program, it is only required to add an external loop which gradually increases the number of nodes in the model.

8 CONCLUSION

The details of the element free Galerkin (EFG) method and its numerical implementation have been presented, with specific emphasis on a one-dimensional formulation. MLS approximants, which constitute the backbone of the method were described, including several examples of the use and effect of different weight functions on the approximations. An example in one-dimensional elastostatics was solved with the aid of a MATLAB program.

The numerical results show the accuracy of the method to be better than that achievable with finite elements. The MATLAB program is provided in Appendix A, and the details of the program with reference to the EFG method were outlined.

The extension of the method to two dimensions was described, and a simple two-dimensional problem in linear elastostatics was presented and solved with the method. The results show the accuracy of the EFG method in computing the stresses in the problem as well as the displacements. The 2D MATLAB program used to solve this problem is provided in Appendix B, and a brief description of the program along with a flow chart is provided. These programs are available at http://www.tam.nwu.edu/jed/EFG/programs.html.

The running time of the two-dimensional EFG program provided in Appendix B is substantially greater than for a comparable finite element program. On a Dell Dimension XPS Pro, with a Pentium Pro 180 MHz processor, the program in Appendix B uses approximately 34 seconds to assemble the stiffness matrix and 34 seconds to perform the post processing; the total running time is 69 seconds. The number of statements in the program exceeds that of a comparable 2D FE program which can be written in about 2 pages of MATLAB. However, the potential for meshless methods for certain classes of problems diminishes the importance of these disadvantages.

The EFG method is by no means limited to solving simple problems in elastostatics, but in the context of an introductory paper these types of problems were chosen for illustration here. In several different papers, the method has been extended to static and dynamic fracture in both two and three dimensions (see Belytschko *et al.* [1], and Belytschko *et al.* [3]). As mentioned previously, the method has also been coupled with finite elements (Belytschko *et al.* [2]) to enforce the essential boundary conditions and for potential use in standard finite element programs. The aim of this paper has been to provide the reader with an introduction to the EFG method, and to facilitate experimentation with the method independently. The EFG method is a new and exciting technique, and it is currently being extended to many different problems in applied mechanics.

ACKNOWLEDGMENTS

The authors are grateful for the support provided by the Office of Naval Research and Army Research Office, to Northwestern University.

REFERENCES

Belytschko, T., Y.Y. Lu, and L. Gu (1994), "Element Free Galerkin Methods", *International Journal for Numerical Methods in Engineering*, **37**, 229-256.

Belytschko, T., D. Organ, and Y. Krongauz (1995), "A Coupled Finite Element- Element free Galerkin Method", *Computational Mechanics*, **17**, 186-195.

Belytschko, T., Y. Krongauz, M. Fleming, D. Organ and W.K. Liu (1996), "Smoothing and Accelerated Computations in the Element-free Galerkin Method", *Journal of Computational and Applied Mechanics*, **74**, 111-126.

Belytschko, T., Y. Krongauz, D. Organ, M. Fleming and P. Krysl (1996), "Meshless Methods: An Overview and Recent Developments", *Computer Methods in Applied Mechanics and Engineering*, **139**, 3-47.

Duarte, C.A. and J.T. Oden (1996), *H-p* Clouds - an *h-p* Meshless Method, *Numerical Methods for Partial Differential Equations*, 1-34.

Lancaster, P. and K. Salkauskas (1981), "Surfaces Generated by Moving Least Squares Methods", *Mathematics of Computation*, **37**, 141-158.

Liu,W.K., S. Jun, and Y.F. Zhang (1995), "Reproducing Kernel Particle Methods", *International Journal for Numerical Methods in Engineering*, **20**, 1081-1106.

Lu, Y.Y., T. Belytschko and L. Gu (1994), "A New Implementation of the Element Free Galerkin Method", *Computer Methods in Applied Mechanics and Engineering*, **113**, 397-414.

Monaghan, J.J. (1992), "Smooth Particle Hydrodynamics", *Annual Review of Astronomy and Astrophysics*, **30**, 543-574.

Onate, E. S. Idelsohn, O.C. Zienkiewicz and R.L. Taylor (1996), "A Finite Point Method in Computational Mechanics. Applications to Convective Transport and Fluid Flow", *International Journal for Numerical Methods in Engineering*, **39**, (22), 3839-3867.

Timoshenko, S.P. and J.N. Goodier (1970), *Theory of Elasticity* (Third ed.). New York, McGraw Hill.

Please address your comments or questions on this paper to:
International Center for Numerical Methods in Engineering
Edificio C-1, Campus Norte UPC
Grand Capitán s/n
08034 Barcelona, Spain
Phone: 34-93-4106035; Fax: 34-93-4016517

APPENDIX A: ONE-DIMENSIONAL EFG PROGRAM

```
% ONE DIMENSIONAL EFG PROGRAM
% SET UP NODAL COORDINATES ALONG BAR, DETERMINE NUMBER OF CELLS
x = [0.0:.1:1.0];
nnodes = length(x);
ncells = nnodes-1;

% SET PARAMETERS FOR WEIGHT FUNCTION, MATERIAL PROPERITES
dmax = 2.0; % RATIO OF DMI TO CI
E=1.0; area=1.0;

% DETERMINE DMI FOR EACH NODE
dm = dmax*(x(2)-x(1))*ones(1,nnodes);

%SET UP GAUSS POINTS, WEIGHTS, AND JACOBIAN FOR EACH CELL
gg = zeros(1,ncells);
jac = (x(2)-x(1))/2;
weight = 2;
gg = -.05:.1:0.95; gg(1) = 0.0;

% INITIALIZE MATRICES
k = zeros(nnodes);
f = zeros(nnodes,1);
GG = zeros(nnodes,1);

% LOOP OVER GAUSS POINTS
for j = 1:length(gg)
   xg = gg(j);

% DETERMINE DISTANCE BETWEEN NODES AND GAUSS POINT
   dif = xg*ones(1,nnodes)-x;;

% SET UP WEIGHTS W AND DW FOR EACH NODE
   clear w dw
   for i=1:nnodes
drdx = sign(dif(i))/dm(i);
r = abs(dif(i))/dm(i);
if r<=0.5
   w(i) = (2/3) - 4*r*r + 4*r^3;
   dw(i) = (-8*r + 12*r^2)*drdx;
elseif r<=1.0
   w(i) = (4/3)-4*r+4*r*r -(4/3)*r^3;
   dw(i) = (-4 + 8*r-4*r^2)*drdx;
elseif r>1.0
   w(i) = 0.0;
   dw(i) = 0.0;
    end
   end

%SET UP SHAPE FUNCTIONS AND DERIVATIVES
```

```
   won = ones(1,nnodes);
   p = [won;x];
   B = p.*[w;w];
   pp = zeros(2);
   A = zeros(2);
   dA = zeros(2);
   for i=1:nnodes
pp = p(1:2,i)*p(1:2,i)';
A = A+w(1,i)*pp;
dA = dA+dw(1,i)*pp;
   end
   Ainv = inv(A);
   pg = [1 xg];
   phi = pg*Ainv*B;
   db = p.*[dw;dw];
   da = -Ainv*(dA*Ainv);
   dphi = [0 1]*Ainv*B+pg*(da*B+Ainv*db);

%ASSEMBLE DISCRETE EQUATIONS
   if j == 1
GG(1:3,1) = -phi(1:3)'
   else if j>1
k = k+(weight*E*area*jac)*(dphi'*dphi);
fbody = area*xg;
f = f+(weight*fbody*jac)*phi';
   end
end
end

% ENFORCE BOUNDARY CONDITION USING LAGRANGE MULTIPLIERS
q=[0];
m = [k GG;GG' zeros(1)];
% SOLVE FOR NODAL PARAMETERS
d = m\[f' q]';
u = d(1:nnodes)
```

APPENDIX B: TWO-DIMENSIONAL EFG PROGRAM

```
% 2D MATLAB EFG CODE - SQUARE DOMAIN
% John Dolbow 2/17/97
clear;
% DEFINE BOUNDARIES/PARAMETERS
Lb = 48;
D=12;
young = 30e6;nu=0.3;
P=1000;


% PLANE STRESS DMATRIX
Dmat = (young/(1-nu^2))*[1 nu 0;nu 1 0;0 0 (1-nu)/2];


% SET UP NODAL COORDINATES
ndivl = 10;ndivw = 4;
[x,conn1,conn2,numnod] = mesh2(Lb,D,ndivl,ndivw);


% DETERMINE DOMAINS OF INFLUENCE - UNIFORM NODAL SPACING
dmax=3.5;
xspac = Lb/ndivl;
yspac = D/ndivw;
dm(1,1:numnod)=dmax*xspac*ones(1,numnod);
dm(2,1:numnod)=dmax*yspac*ones(1,numnod);


% SET UP QUADRATURE CELLS
ndivlq = 10;ndivwq = 4;
[xc,conn,numcell,numq] = mesh2(Lb,D,ndivlq,ndivwq);


% SET UP GAUSS POINTS, WEIGHTS, AND JACOBIAN FOR EACH CELL
quado = 4;
[gauss] = pgauss(quado);
numq2 = numcell*quado^2;
gs = zeros(4,numq2);
[gs] = egauss(xc,conn,gauss,numcell);


% LOOP OVER GAUSS POINTS TO ASSEMBLE DISCRETE EQUATIONS
k = sparse(numnod*2,numnod*2);
for gg=gs
   gpos = gg(1:2);
   weight = gg(3);
   jac = gg(4);


% DETERMINE NODES IN NEIGHBORHOOD OF GAUSS POINT
```

```
   v = domain(gpos,x,dm,numnod);
   L = length(v);
   en = zeros(1,2*L);
   [phi,dphix,dphiy] = shape(gpos,dmax,x,v,dm);
   Bmat=zeros(3,2*L);
   for j=1:L
   Bmat(1:3,(2*j-1):2*j) = [dphix(j) 0;0 dphiy(j);dphiy(j) dphix(j)];
   end
   for i=1:L
   en(2*i-1) = 2*v(i)-1;
   en(2*i) = 2*v(i);
   end
   k(en,en) = k(en,en)+sparse((weight*jac)*Bmat'*Dmat*Bmat);
end


% DETERMINE NODES ON BOUNDARY, SET UP BC'S
ind1 = 0;ind2 = 0;
for j=1:numnod
if(x(1,j)==0.0)
ind1=ind1+1;
nnu(1,ind1) = x(1,j);
nnu(2,ind1) = x(2,j);
end
if(x(1,j)==Lb)
ind2=ind2+1;
nt(1,ind2) = x(1,j);
nt(2,ind2) = x(2,j);
end
end
lthu = length(nnu);
ltht = length(nt);
ubar = zeros(lthu*2,1);
f = zeros(numnod*2,1);


%SET UP GAUSS POINTS ALONG TRACTION BOUNDARY
ind=0;
gauss=pgauss(quado);
for i=1:(ltht-1)
ycen=(nt(2,i+1)+nt(2,i))/2;
jcob = abs((nt(2,i+1)-nt(2,i))/2);
for j=1:quado
        mark(j) = ycen-gauss(1,j)*jcob;
        ind = ind+1;
        gst(1,ind)=nt(1,i);
        gst(2,ind)=mark(j);
        gst(3,ind)=gauss(2,j);
        gst(4,ind)=jcob;
end
end
```

```
%SET UP GAUSS POINTS ALONG DISPLACEMENT BOUNDARY
gsu=gst;
gsu(1,1:ind)=zeros(1,ind);
qk = zeros(1,2*lthu);


%INTEGRATE FORCES ALONG BOUNDARY
Imo = (1/12)*D^3;
for gt=gst
   gpos = gt(1:2);
   weight=gt(3);
   jac = gt(4);
   v = domain(gpos,x,dm,numnod);
   L = length(v);
   en = zeros(1,2*L);
   force=zeros(1,2*L);
   [phi,dphix,dphiy] = shape(gpos,dmax,x,v,dm);
   tx=0;
   ty= -(P/(2*Imo))*((D^2)/4-gpos(2,1)^2);
   for i=1:L
   en(2*i-1) = 2*v(i)-1;
   en(2*i) = 2*v(i);
   force(2*i-1)=tx*phi(i);
   force(2*i) = ty*phi(i);
   end
f(en) = f(en) + jac*weight*force';
   end


% INTEGRATE G MATRIX AND Q VECTOR ALONG DISPLACEMENT BOUNDARY
GG = zeros(numnod*2,lthu*2);
ind1=0;ind2=0;
for i=1:(lthu-1)
   ind1=ind1+1;
   m1 = ind1; m2 = m1+1;
   y1 = nnu(2,m1); y2 = nnu(2,m2);
   len = y1-y2;
   for j=1:quado
   ind2=ind2+1;
   gpos = gsu(1:2,ind2);
   weight = gsu(3,j);
   jac = gsu(4,j);
   fac11 = (-P*nnu(2,m1))/(6*young*Imo);
   fac2 = P/(6*young*Imo);
   xp1 = gpos(1,1);
   yp1 = gpos(2,1);
   uxex1 = (6*Lb-3*xp1)*xp1 + (2+nu)*(yp1^2 - (D/2)^2);
   uxex1 = uxex1*fac11;
   uyex1 = 3*nu*yp1^2*(Lb-xp1)+0.25*(4+5*nu)*xp1*D^2+(3*Lb-xp1)*xp1^2;
   uyex1 = uyex1*fac2;
```

```
   N1 = (gpos(2,1)-y2)/len; N2 = 1-N1;
   qk(2*m1-1) = qk(2*m1-1)-weight*jac*N1*uxex1;
   qk(2*m1) = qk(2*m1) - weight*jac*N1*uyex1;
   qk(2*m2-1) = qk(2*m2-1) -weight*jac*N2*uxex1;
   qk(2*m2) = qk(2*m2) - weight*jac*N2*uyex1;
   v = domain(gpos,x,dm,numnod);
   [phi,dphix,dphiy] = shape(gpos,dmax,x,v,dm);
   L = length(v);
      for n=1:L
      G1 = -weight*jac*phi(n)*[N1 0;0 N1];
      G2 = -weight*jac*phi(n)*[N2 0;0 N2];
      c1=2*v(n)-1;c2=2*v(n);c3=2*m1-1;c4=2*m1;
      c5=2*m2-1;c6=2*m2;
      GG(c1:c2,c3:c4)=GG(c1:c2,c3:c4)+ G1;
      GG(c1:c2,c5:c6)=GG(c1:c2,c5:c6)+G2;
      end
   end
end


% ENFORCE BC'S USING LAGRANGE MULTIPLIERS
f = [f;zeros(lthu*2,1)];
f(numnod*2+1:numnod*2+2*lthu,1) = -qk';
m = sparse([k GG;GG' zeros(lthu*2)]);
d=m\f;
u=d(1:2*numnod);
for i=1:numnod
u2(1,i) = u(2*i-1);
u2(2,i) = u(2*i);
end


% SOLVE FOR OUTPUT VARIABLES - DISPLACEMENTS
displ=zeros(1,2*numnod);
ind=0;
for gg=x
   ind = ind+1;
   gpos = gg(1:2);
   v = domain(gpos,x,dm,numnod);
   [phi,dphix,dphiy] = shape(gpos,dmax,x,v,dm);
   displ(2*ind-1) = phi*u2(1,v)';
   displ(2*ind) = phi*u2(2,v)';
end


% SOLVE FOR STRESSES AT GAUSS POINTS
ind = 0;
enorm=0;
for gg=gs
   ind = ind+1;
   gpos = gg(1:2);
```

```
   weight = gg(3);
   jac = gg(4);
   v = domain(gpos,x,dm,numnod);
   L = length(v);
   en = zeros(1,2*L);
   [phi,dphix,dphiy] = shape(gpos,dmax,x,v,dm);
   Bmat=zeros(3,2*L);
   for j=1:L
   Bmat(1:3,(2*j-1):2*j) = [dphix(j) 0;0 dphiy(j);dphiy(j) dphix(j)];
   end
   for i=1:L
   en(2*i-1) = 2*v(i)-1;
   en(2*i) = 2*v(i);
   end
   stress(1:3,ind) = Dmat*Bmat*u(en);
   stressex(1,ind) = (1/Imo)*P*(Lb-gpos(1,1))*gpos(2,1);
   stressex(2,ind) = 0;
   stressex(3,ind) = -0.5*(P/Imo)*(0.25*D^2 - gpos(2,1)^2);
   err = stress(1:3,ind)-stressex(1:3,ind);
   err2 = weight*jac(0.5*(inv(Dmat)*err)'*(err));
   enorm = enorm + err2;
end
   enorm = sqrt(enorm)
-----------------------------------------------------------------
% MESH GENERATION PROGRAM FOR 2D BEAM IN BENDING
function[x,conn,numcell,numq] = mesh2(length,height,ndivl,ndivw)


% INPUT DATA
numcell= ndivw*ndivl;
numq = (ndivl+1)*(ndivw+1);


% SET UP NODAL COORDINATES
for i = 1:(ndivl+1)
   for j = 1:(ndivw+1)
   x(1,((ndivw+1)*(i-1) +j))= (length/ndivl)*(i-1);
   x(2,((ndivw+1)*(i-1) +j))= -(height/ndivw)*(j-1)+height/2;
   end
end


% SET UP CONNECTIVITY ARRAY
for j = 1:ndivl
   for i = 1:ndivw
   elemn = (j-1)*ndivw + i;
   nodet(elemn,1) = elemn + (j-1);
   nodet(elemn,2) = nodet(elemn,1) + 1;
   nodet(elemn,3) = nodet(elemn,2)+ndivw+1;
   nodet(elemn,4) = nodet(elemn,3)-1;
   end
```

```
end
conn = nodet';
-------------------------------------------------------------------------
function [gs] = egauss(xc,conn,gauss,numcell)
% routine to set up gauss points, jacobian, and weights
index=0;
one = ones(1,4);
psiJ = [-1,+1,+1,-1]; etaJ = [-1,-1,+1,+1];
l = size(gauss);
l = l(2);
for e=1:numcell
% DETERMINE NODES IN EACH CELL
   for j = 1:4
     je=conn(j,e);xe(j)=xc(1,je);ye(j)=xc(2,je);
   end
for i=1:l
     for j=1:l
        index = index+1;
        eta=gauss(1,i);psi=gauss(1,j);
        N = .25*(one+psi*psiJ).*(one+eta*etaJ);
        NJpsi=.25*psiJ.*(one+eta*etaJ);
        NJeta=.25*etaJ.*(one+psi*psiJ);
        xpsi=NJpsi*xe';ypsi=NJpsi*ye';xeta=NJeta*xe';yeta=NJeta*ye';
        jcob=xpsi*yeta-xeta*ypsi;
        xq = N*xe';yq = N*ye';
        gs(1,index) = xq;
gs(2,index) = yq;
gs(3,index) = gauss(2,i)*gauss(2,j);
gs(4,index) = jcob;
     end
   end
end
-------------------------------------------------------------------------
function v = pgauss(k)
% This function returns a matrix with 4 gauss points and their weights
   v(1,1) =-.861136311594052575224;
   v(1,2) =-.339981043584856264803;
   v(1,3) = -v(1,2);
   v(1,4) = -v(1,1);
   v(2,1) =.347854845137453857373;
   v(2,2) =.652145154862546142627;
   v(2,3) = v(2,2);
   v(2,4) = v(2,1);
-------------------------------------------------------------------------
function v = domain(gpos,x,dm,nnodes)
%DETERMINES NODES IN DOMAIN OF INFLUENCE OF POINT GPOS
dif = gpos*ones(1,nnodes)-x;
a = dm-abs(dif);
b = [a;zeros(1,nnodes)];
c = all(b>=-100*eps);
v = find(c);
```

```
------------------------------------------------------------------------
function  [phi,dphix,dphiy] = shape(gpos,dmax,x,v,dm)
% EFG shape function and it's derivative with linear base
L = length(v);
won = ones(1,L);
nv = x(1:2,v);
p = [won;nv];
dif = gpos*won-nv;
t = dm(1:2,v)/dmax;


% WEIGHTS--W and dw are vectors
[w,dwdx,dwdy] = cubwgt(dif,t,v,dmax,dm);
B = p.*[w;w;w];
pp = zeros(3);
aa = zeros(3);
daax = zeros(3);
daay = zeros(3);
for i=1:L
   pp = p(1:3,i)*p(1:3,i)';
   aa = aa+w(1,i)*pp;
   daax = daax+dwdx(1,i)*pp;
   daay = daay+dwdy(1,i)*pp;
end
pg = [1 gpos'];
[L,U,PERM] = lu(aa);
for i=1:3
   if i==1
      C = PERM*pg';
   elseif i==2
      C = PERM*([0 1 0]' - daax*gam(1:3,1));
   elseif i==3
      C = PERM*([0 0 1]' - daay*gam(1:3,1));
   end
   D1 = C(1);
   D2 = (C(2) - L(2,1)*D1);
   D3 = (C(3) - L(3,1)*D1-L(3,2)*D2);
   gam(3,i) = D3/U(3,3);
   gam(2,i) = (D2 - U(2,3)*gam(3,i))/(U(2,2));
   gam(1,i) = (D1 - U(1,2)*gam(2,i)-U(1,3)*gam(3,i))/U(1,1);
end
phi = gam(1:3,1)'*B;
dbx = p.*[dwdx;dwdx;dwdx];
dby = p.*[dwdy;dwdy;dwdy];
dphix = gam(1:3,2)'*B + gam(1:3,1)'*dbx;
dphiy = gam(1:3,3)'*B + gam(1:3,1)'*dby;
------------------------------------------------------------------
function  [w,dwdx,dwdy] = cubwgt(dif,t,v,dmax,dm)
% CUBIC SPLINE WEIGHT FUNCTION
l = length(v);
for i=1:l
```

```
drdx = sign(dif(1,i))/dm(1,v(i));
drdy = sign(dif(2,i))/dm(2,v(i));
rx = abs(dif(1,i))/dm(1,v(i));
ry = abs(dif(2,i))/dm(2,v(i));
if rx>0.5
   wx = (4/3)-4*rx+4*rx*rx -(4/3)*rx^3;
   dwx = (-4 + 8*rx-4*rx^2)*drdx;
elseif rx<=0.5
   wx = (2/3) - 4*rx*rx + 4*rx^3;
   dwx = (-8*rx + 12*rx^2)*drdx;
end
if ry>0.5
   wy = (4/3)-4*ry+4*ry*ry -(4/3)*ry^3;
   dwy = (-4 + 8*ry-4*ry^2)*drdy;
elseif ry<=0.5
   wy = (2/3) - 4*ry*ry + 4*ry^3;
   dwy = (-8*ry + 12*ry^2)*drdy;
end
w(i) = wx*wy;
dwdx(i) = wy*dwx;
dwdy(i) = wx*dwy;
end
```