International Journal of Engineering & Applied Sciences (IJEAS)
Vol.7, Issue 2(2015)1-18

OBJECT-ORIENTED PROGRAMMING IN MESHFREE ANALYSIS OF ELASTOSTATIC PROBLEMS

B. Kanber [a] and M.M. Yavuz [b]*

[a] *Mechanical Engineering Department, University of Gaziantep, Gaziantep, Turkey*
[b] *Energy Systems Engineering Department, Osmaniye Korkut Ata University, Osmaniye, Turkey*
[*]*E-mail address*: *kanber@gantep.edu.tr*

Abstract

In this work, the main philosophy behind the object-oriented programming (OOP) of meshfree methods is discussed for solution of elastostatic problems. Objects and classes are constructed with respect to the structure of meshfree methods. Local radial point interpolation method (LRPIM) and meshless local Petrov-Galerkin (MLPG) method are used in local weak form in the program. Basic object oriented programming operators; encapsulation, inheritance and polymorphism are used for increasing modularity. Seven main classes and their subclasses are constructed for decreasing complexity. Additional storage modules and solver functions are implemented. As a result of this, new techniques on interpolations and integrations can be easily adapted to construction of shape functions in meshfree program structure. Objects are defined and implemented for solution of 2D elastostatic problems in MATLAB. Two elestostatic problems are solved in MATLAB OOP and their results are compared with results of a procedural program that is written in FORTRAN. Class designs and their hierarchy are discussed in details.

Keywords: Object-oriented programming (OOP), meshfree methods, MATLAB, 2D elastostatic problems.

1. Introduction

The usage and implementation of numerical methods on computer programs are also important, as well as their developments. Computer coding of numerical methods is a very different kind of study and applicability of numerical methods depends on the coding performance. A well designed program contains less error and is easily handled. There are lots of codes which have different kinds of structures, even if they have small sizes. Procedural programming is commonly used in coding of these programs, which includes less procedure. But addition of more codes, new functions, elements and items causes increasing conflicts, even if subroutines or other auxiliary modules are used. One of the programming techniques, object-oriented programming (OOP) is used for preventing complexity, which has capable of key-lock property and can group similar structures. This key-lock property supports to avoid unnecessary usages of programs. Hence OOP can be capable to use in programming of numerical methods. Developments in numerical methods provide different solution techniques like FDM (finite difference method), BEM (boundary element method), FEM (finite element method) and so on. The usage of FEM is widely known, especially in the solutions of solid mechanics problems. FEM defines the analysed numerical model with small elements, which are called finite elements and all solutions are accomplished with them. Solution procedure includes some structures (interpolations, transformations, construction of shape functions and application of essential and natural boundary conditions) which work in harmony together. Any conflicts may cause wrong results. Hence object-oriented programming techniques are widely used in finite and boundary element in literature. Further developments or extension sections of numerical methods can be easily applied with OOP. This structure is well used in FEM with defining classes and their methods. This method [1-3] provides to better code writing with addition of modularity in FEM. OOP methodology is used for increasing readability, modularity and reusability. C++ PL in OOP is widely used and most of the OO FEA codes are written with this language. When model or any

analysed condition is changed, only changing related object can satisfy the new changes without changing the whole model. The applications of OO FEA with C++ programming are available in the studies [4-21] for different analyses in literature. Duplications in the programs decrease with OOP techniques. Inheritance and polymorphism are mainly used for providing a main framework to programs. Combining different methods can be easily prepared with OOP, like both usage of FEM and FVM (finite volume method) [22-23], FEM and BEM (boundary element method) [24]… the usage of OOP only in BEM [25-26] are available. Programming languages are also important cases. Dubois-Pelerin and Zimmermann [27] transfer the developed OO FEM structure [2-3] from Smalltalk to OO C++ for increasing efficiency. The efficiency is also discussed [28-30] between different programming languages for OOP. It is detected in some cases that OO programs are slower than procedural programs. The slowness is caused from young age of OO programs, absence of a fast numerical library and polymorphism, which requires dynamic binding.

Some studies focus on application of OOP to meshfree methods in literature. Krysl and Belytschko [31] design a library for EFG (element-free Galerkin) shape functions with both OO and procedural programming. The complexity of EFG shape functions is generally greater than FEM shape functions and encapsulation property of OOP can decrease this complexity. Different studies [32-35] are available about OO programming of meshfree methods. However, meshfree techniques are not fully developed and they are the main interest of many researches in literature. Hence their programming techniques must also be considered, when new meshfree methods have been developed. This method mainly focuses on how to prevent predefined model construction for interpolation and integration in numerical analyses. There are different kinds of studies about prevention of predefined elements modelling for construction of shape functions, like SPH [36] (smoothed particle hydrodynamics), DEM [37] (diffuse element method), EFG [38] (element-free Galerkin method), MLPG [39] (meshless local Petrov-Galerkin), PIM [40] (point interpolation method) and RPIM [41] (radial point interpolation method), which are widely used in meshfree analysis. Further stages of these methods have been developed. More extended studies on PIM and RPIM can also be available in literature and researchers are trying to adapt them in meshfree analysis. LC-PIM [42] (linearly conforming point interpolation method) is developed for increasing efficiency, which includes PIM. NI-RPIM [43] (nodal integration radial point interpolation method) is developed by addition of Taylor series expansion in nodal integration. CS-RPIM [44] (cell-based smoothed radial point interpolation method) is developed for solutions of statics and vibration analysis of solids, which contains RPIM. Some of the developments in numerical methods include most of previous literature studies. Hence, OOP of meshfree methods can be a main body for further adaptations of new meshfree techniques. Different integration techniques for construction of shape functions are easily implemented and can provide to work in harmony.

In this work, an object oriented meshfree program is constructed with respect to the meshfree program of the study [45]. Two different meshfree techniques: Local radial basis point interpolation method (LRPIM) and meshless local Petrov-Galerkin (MLPG) method [38-40,45-48] are used in local weak forms. Objects are defined for these methods and implemented for the solution of 2D elastostatic problems in MATLAB. Two case studies are examined. The results are compared with analytical and FEM solutions.

2. Object-oriented Programming (OOP) and MATLAB

OOP is an appropriate programming technique for large codes and based on a hierarchy of classes, which cooperate their objects. It mainly includes three major components; encapsulation, inheritance and polymorphism. Encapsulation hides implementation details of the objects. Input and output processes are mainly concerned at the outside of the class. Inheritance is the methodology to form

new classes in terms of old classes. Polymorphism is the ability to use an operator or method in different ways.

MATLAB is a programming environment for algorithm development, data analysis, visualization, numerical computation and widely used in scientific and engineering applications. It also supports OOP technique with various class operators. There are two different ways to define classes in MATLAB. The first way is using `value` class and the second is using `handle` class. A `value` class constructor returns an instance that is associated with the variable, which it assigned. A `handle` class constructor returns a handle object that is a reference to the object created. In this work, `handle` class is used as a base class of all developed classes in this study. This class also controls hierarchy of its subclasses.

3. Meshfree Classes and Their Implementations in a MATLAB Program

3.1. Classes and their hierarchy

In this work, seven different classes are mainly used. However, three of them are especially related with meshfree techniques; `inte`, `sdo` and `sf` classes. They have their own sub-classes. Some sub-classes have further sub-classes. In the subsections of this chapter, they are discussed in detail. Basic and special class properties are explained. Classes and its properties mainly manage objects and programming schemes. Defined properties in the class will be assigned to objects. Hence required and related properties must be determined for setting efficient object management. Thus the programmed method and its logic must be well known.

The basic concept of elastostatic problems is the solution of equilibrium equations with respect to applied natural and essential boundary conditions. Their equations are given in Eq. 1, 2 and 3, respectively, which are mainly concerned in the application of object-oriented meshfree program for solution of elastostatic problems.

$$L^T\sigma + \mathrm{b} = 0 \quad \text{in}\,\Omega \tag{1}$$

$$\sigma\, n = \bar{t} \qquad \text{on}\,\Gamma_{\mathrm{t}} \tag{2}$$

$$u = \bar{u} \qquad \text{on}\,\Gamma_{\mathrm{u}} \tag{3}$$

There are several methods are available for solution of these equations. Strong and weak form formulations are defined for solution of partial differential equations (PDEs). LRPIM and MLPG methods can handle weak form PDEs. Hence, the equilibrium equation [45] in Eq. 1 can be written in weak form with respect to weighted residual methods for LRPIM as

$$\int_{\Omega_q} \hat{W}_I \left(\sigma_{ij,j} + \mathrm{b_i}\right) d\Omega = 0 \tag{4}$$

where $\hat{W}_I$ is the weight function at related node. The application of natural and essential boundary conditions on weak form of equilibrium equation for local quadrature domain of a related node is given in Eq. 5. This local quadrature domain can intersect with natural and essential boundaries, which need a special formulation on construction of stiffness matrix that related locations. Three boundary locations are determined with respect to natural and essential boundary conditions. Γ_{qi}

represents not intersection locations of local quadrature domain with global boundary, Γ_{qt} represents intersection locations with natural boundary and Γ_{qu} represents intersection locations of essential boundary locations. More detail information can be achieved in the study of [45].

$$\int_{\Gamma_{qi}} \hat{W}_I n_j \sigma_{ij} d\Gamma + \int_{\Gamma_{qu}} \hat{W}_I n_j \sigma_{ij} d\Gamma + \int_{\Gamma_{qt}} \hat{W}_I n_j \sigma_{ij} d\Gamma - \int_{\Omega_q} \left[\hat{W}_{I,j} \sigma_{ij} - \hat{W}_I b_i \right] d\Omega = 0 \tag{5}$$

Eq. 5 can be written with application of stresses. Application of tractions at inside of natural boundary conditions is given in Eq. 6.

$$\int_{\Omega_q} \hat{W}_{I,j} \sigma_{ij} d\Omega - \int_{\Gamma_{qi}} \hat{W}_I t_i d\Gamma - \int_{\Gamma_{qu}} \hat{W}_I t_i d\Gamma = \int_{\Gamma_{qt}} \hat{W}_I \bar{t}_i d\Gamma + \int_{\Omega_q} \hat{W}_I b_i d\Omega \tag{6}$$

Similar equilibrium equation is valid for MLPG method and its local weak formulation is given in Eq. 7. Same applications of natural and essential boundary conditions are valid as LRPIM method. In addition, equilibrium equation of MLPG method includes a curve integral for application of essential boundary conditions where local quadrature domain intersects with essential boundary Γ_{qu}. But, on the contrary, to use RPIM shape functions, MPLG method uses MLS shape functions, which do not support Kronecker delta function property. Hence essential boundary conditions are enforced with penalty method. A penalty method with factor (α) is included, which is used as 10^8.

$$\int_{\Omega_q} \hat{W}_I \left(\sigma_{ij} + b_i \right) d\Omega - \alpha \int_{\Gamma_{qu}} \hat{W}_I \left(u_i - \bar{u}_i \right) d\Gamma = 0 \tag{7}$$

The solution of these equations includes various components, which are classified for decreasing complexity in classes of OOP.

3.1.1 `mc (model constructor)` class

`mc` (model constructor) class is mainly used for construction of basic numerical model, has two main properties: *`young`* and *`anu`*. They represent Young's modulus and Poisson's ratio, respectively. It does not include any methods. Its properties are transferred to classes of `material` and `nodes`. Material properties must be known for calculation of strain and stresses with respect to Hooke's law, which is given in Eq. 8.

$$\sigma = D \varepsilon \tag{8}$$

`material` class is defined for materials of the model and has two own properties: `you` and `dmat`. `dmat` includes material matrix for isotropic materials in its property with respect to plane stress and strain, which are given in Eq. 9 and 10, respectively. In addition to them, different material models can be easily added with `dmat` property.

$$D = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)^2 \end{bmatrix} \tag{9}$$

$$D = \frac{E(1-\nu)}{(1-2\nu)(1+\nu)} \begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{bmatrix} \tag{10}$$

The class of `nodes` is used to read all data, which are related geometrical and meshfree parameter properties for the model and has three different sub-classes: `xlsinputnodes`, `textinputnodes` and `randtextinputnodes`. `xlsinputnodes` class is used to read all its data from Excel files. If data is given in a text file, they can be read from `textinputnodes` class. Node coordinate can be obtained randomly by using MATLAB statements in the class of `randtextinputnodes`.

3.1.2 `inte` (integration cell) class

Integration is one of the basic procedures in meshfree methods. Different integration methods are available and integration domains can be variable. Therefore, `inte` class is defined for local and global quadrature domains. The properties are inherited to the `qdomain`, `gausscoefficient`, `domaingausspoints`, `localquadraturedomain` and `globalquadraturedomain` classes. The size of quadrature domain is calculated in `qdomain` class and recorded in `ds` property. The weights are calculated with respect to the number of sampling points in `gausscoefficient` class and recorded in `gauss` property. A local quadrature domain is determined for a sampling point in `localquadraturedomain` class. The determined local quadrature domain is further subdivided in `subdivisionlqdomain` class. In `domaingausspoints` class, gauss points are calculated for a divided quadrature domain and recorded in *`gss`* property. Global quadrature domain can be used directly with `globalquadraturedomain` class.

3.1.3 `sdo` (support domain) class

The number of nodes in a support domain and their coordinates are determined in `sdo` class. It has a single basic property, `gpos`. It is inherited to its sub-classes of `supportdomain` and `testfunc`. `supportdomain` class determines the number of nodes and their labels in a support domain. In this process, three different support domain geometries can be used. They are given in `rectangularsdomain`, `triangularsdomain` and `circularsdomain` classes. In Eq. 11, size of support domain is given with circular distance (d_s) for a circular support domain. d_c is usually selected as average nodal spacing and α_s is a constant and used as [45,48] between 2.00 and 3.00 generally.

$$d_s = \alpha_s \times d_c \tag{11}$$

The number of nodes and their labels in support domain for integration are stored in *`ndex`* and *`nv`* properties. `testfunc` class computes the weights and stores them in *`w`* property. Different weight functions [45] are available like the cubic spline function (W1), quartic spline weight function (W2) and other weight functions. W1 and W2 are given in Eq. 12 and 13, respectively. Both LRPIM and MPLG methods can use these weight functions in the solution of equilibrium equation.

$$\hat{W}_i(x)=\begin{Bmatrix} 2/3-4\bar{r}_i^2+4\bar{r}_i^3 & \bar{r}_i\le 0.5 \\ 4/3-4\bar{r}_i+4\bar{r}_i^2+4/3\bar{r}_i^3 & 0.5\le\bar{r}_i\le 1.0 \\ 0 & 1<\bar{r}_i \end{Bmatrix} \tag{12}$$

$$\hat{W}_i(x)=\begin{Bmatrix} 1-6\bar{r}_i^2+8\bar{r}_i^3-3\bar{r}_i^4 & \bar{r}_i\le 1.0 \\ 0 & \bar{r}_i>1.0 \end{Bmatrix} \tag{13}$$

Matrices of weight functions ($\hat{W}$) and their derivatives ($\hat{V}_t$) of related point are given in Eq.14 and 15.

$$\hat{W}_I=\hat{W}(x,x_I)_{(2\times 2)}=\begin{bmatrix} \hat{W}(x,x_I) & 0 \\ 0 & \hat{W}(x,x_I) \end{bmatrix} \tag{14}$$

$$\hat{V}_I=\hat{V}(x,x_I)_{(3\times 2)}=\begin{bmatrix} \hat{W}_{,x}(x,x_I) & 0 \\ 0 & \hat{W}_{,y}(x,x_I) \\ \hat{W}_{,y}(x,x_I) & \hat{W}_{,x}(x,x_I) \end{bmatrix} \tag{15}$$

3.1.4 `sf (shape function)` class

Most meshfree methods are classified according to shape function construction. Therefore, `sf` class is defined for RPIM and MLS shape functions. It has seven properties: `ndex`, `mbasis`, `gpos`, `x`, `nv`, `phi` and `numnode`. In `rpim` class, shape functions and their derivatives are obtained by `computeradialbasis` class. `computeradialbasis` includes basis functions and polynomial terms with their derivatives in `rk` property. Polynomial terms are given in Eq. 16, which are mainly derived from binomial expansion.

$$p^T(x)=\{1,x,y,x^2,xy,y^2,...\} \tag{16}$$

Radial basis functions are calculated [45] using one of its sub-classes of `mq` (multi-quadrics), `exp` (Gaussian) or `tsp` (thin plate spline). multi-quadrics (MQ), the Gaussian (Exp), the thin plate spline (TSP) function are given in Eq. 17, 18 and 19, respectively.

$$R_i(x,y)=\exp\left[-\alpha_c\left(\frac{r_i}{d_c}\right)^2\right] \tag{17}$$

$$R_i(x,y)=r_i^n \tag{18}$$

$$R_i(x,y)=r_i^n\log r_i \tag{19}$$

d_c is the average nodal spacing near the point of interest at x; α_c and n are two arbitrary real numbers of dimensionless parameters, which are called shape parameters. It is [45,48] recommended to use q as 1.03 and α_c as 3.00 for MQ basis function. The radial distance is given in Eq. 20 for 2D.

$$r_i(x)=\sqrt{(x-x_i)^2+(y-y_i)^2} \tag{20}$$

Shape functions can be constructed by using a field function, $u(x)$ which includes polynomial and basis functions and is given in Eq 21.

$$u(x)=\sum_{i=1}^{n} R_i(x)\times a_i+\sum_{j=1}^{m} P_j(x)\times b_j=R^T(x)\times a+P^T(x)\times b \tag{21}$$

$R_i(x)$ and $P_j(x)$ represent radial basis and polynomial basis functions, respectively. a_i and b_j are related constants, n is the number of field nodes in the local support domain and m is the number of polynomial terms. Interpolations between nodes are mainly accomplished within the local support domain for each node or point of interests.

The matrix form of the above equation can be expressed in Eq. 22. U_e is the vector of function values at nodes in the local support domain. R_q is the moment matrix and P_m is the polynomial moment matrix [45], which are given in Eq. 23 and 24, respectively.

$$U_e=R_q\times a+P_m\times b=\{u_1\ u_2\ ..u_n\}^T \tag{22}$$

$$R_q=\begin{bmatrix} R_1(r_1) & R_2(r_1) & \dots & R_n(r_1) \\ R_1(r_2) & R_2(r_2) & \dots & R_n(r_2) \\ \dots & \dots & \dots & \dots \\ R_1(r_n) & R_2(r_n) & \dots & R_n(r_n) \end{bmatrix}_{(n\times n)} \tag{23}$$

$$P_m=\begin{bmatrix} 1 & x_1 & y_1 & z_1 & \dots & P_m(x_1) \\ 1 & x_2 & y_2 & z_2 & \dots & P_m(x_2) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & y_n & z_n & \dots & P_m(x_n) \end{bmatrix}_{(n\times m)} \tag{24}$$

a is the vector of unknown coefficients for RBF and b is the vector of unknown coefficients for polynomial basis functions. They are given in Eq. 25 and 26.

$$a^T=\{a_1\ a_2\ ...a_n\} \tag{25}$$

$$b^T=\{b_1\ b_2\ ...b_n\} \tag{26}$$

For solution of field function, unknown parameter a in Eq. 21 must satisfy in polynomial function,

$$\sum_{i=1}^{n} p_j(x_i)\times a_i=P_m^T\times a=0 \qquad \text{j=1, 2, .., m} \tag{27}$$

Combination of Eq. 21 and Eq. 22 supports the Eq. 28.

$$\widetilde{U}_e=\begin{bmatrix} U_e \\ 0 \end{bmatrix}=\begin{bmatrix} R_q & P_m \\ P_m^T & 0 \end{bmatrix}\begin{Bmatrix} a \\ b \end{Bmatrix}=G\times\ a_0 \quad \text{j=1,2,..,m} \tag{28}$$

where

$$\widetilde{U}_e = \begin{bmatrix} U_e \\ 0 \end{bmatrix} = \{a_1\ a_2\ ...a_n\ 0\,0...0\}^T \tag{29}$$

Unique solution is obtained if inverse of matrix G exists:

$$a_0 = \begin{Bmatrix} a \\ b \end{Bmatrix} = G^{-1} \times \widetilde{U}_e \tag{30}$$

Substituting Eq. 30 into Eq. 21, interpolation with respect to field function can be expressed as,

$$u(x) = \left\{R^T(x) \times P^T(x)\right\} \times G^{-1} \times \widetilde{U}_e = \widetilde{\varphi}(x) \times \widetilde{U}_e \tag{31}$$

Finally [45], RPIM shape functions for the corresponding n field nodes can be obtained as

$$\varphi^T(x) = \{\varphi_1(x)\,\varphi_2(x)...\,\varphi_n(x)\} \tag{32}$$

The approximation function can be written as

$$u(x) = \varphi^T(x) \times U_e = \sum_{i=1}^{n} \varphi_i \times u_i \tag{33}$$

The derivatives of $u(x)$ can be easily obtained as

$$u_{i,k}(x) = \varphi_{,k}^T(x) \times U_e \tag{34}$$

where k denotes the coordinates and partial differentials are taken with respect to that defined coordinated by k.

In `mls` class, the `mls` shape functions and their derivatives are obtained by using `compute_basis` and `compute_ab` sub-classes. `compute_basis` class computes base functions and their derivatives. `compute_ab` class computes the weights by using `weight_w1` and `weight_w2` and finally computes shape functions and their derivatives. When MLS shape functions are constructed, the field function $u(x)$ can be written with approximate formation of MLS in Eq. 35. $a(x)$ is unknown coefficients and they are shown in Eq. 36. These coefficients are included to L_2 norm for finding their values in Eq. 37. When minimizing L_2 with applying $\partial \mathrm{J}/\partial\alpha = 0$, a linear equation can be achieved in Eq. 38.

$$u(x) = \sum_{i=1}^{m} P_j(x) \times a_j(x) = P^T(x) \times a(x) \tag{35}$$

$$a^T = \{a_1(x)\,a_2(x)...a_m(x)\} \tag{36}$$

$$J = \sum_{i=1}^{n} \hat{W}(x - x_i)\left[p^T(x_i)a(x) - u_i\right]^2 \tag{37}$$

$$A(x)a(x) = B(x)U_s \tag{38}$$

As same as right hand side of Eq. 22, field functions can be expressed as;

$$U_s = \{u_1 \ u_2 \ \dots \ u_n\}^T \tag{39}$$

In Eq. 38, $A(x)$, $B(x)$ and $a(x)$ can be expressed as in Eq. 40, 41. and 42, respectively;

$$A(x) = \sum_{i=1}^{n} \hat{W}_i(x)p(x_i)p^T(x_i) \tag{40}$$

$$B(x) = \left[\hat{W}_1(x)p(x_1) \ \hat{W}_2(x)p(x_2) \ \dots \ \hat{W}_n(x)p(x_n)\right] \tag{41}$$

$$a(x) = A^{-1}(x)B(x)U_s \tag{42}$$

Also $a(x)$ can supply field function and the field function can be written with respect to shape functions as;

$$u^h(x) = \sum_{i=1}^{n} \phi(x)u_i = \Phi^T(x)U_s \tag{43}$$

$\Phi(x)$ is represents the constructed shape functions. When they are regulated, they can be written as;

$$\Phi^T(x) = \{\phi_1(x) \ \phi_2(x) \ \dots \ \phi_n(x)\}_{(1\times n)} = p^T(x)A^{-1}(x)B(x) \tag{44}$$

All calculated shape functions and their derivatives are stored in the property of `phi`. Strain matrix is constructed with RPIM or MLS shape functions and it is given in Eq. 45.

$$B_{(3\times 2n)} = \begin{bmatrix} \frac{\partial \phi_1}{\partial x} & 0 & \frac{\partial \phi_2}{\partial x} & 0 & \dots & \frac{\partial \phi_n}{\partial x} & 0 \\ 0 & \frac{\partial \phi_1}{\partial y} & 0 & \frac{\partial \phi_2}{\partial y} & \dots & 0 & \frac{\partial \phi_n}{\partial y} \\ \frac{\partial \phi_1}{\partial y} & \frac{\partial \phi_1}{\partial x} & \frac{\partial \phi_2}{\partial y} & \frac{\partial \phi_2}{\partial x} & \dots & \frac{\partial \phi_n}{\partial y} & \frac{\partial \phi_n}{\partial x} \end{bmatrix} \tag{45}$$

Further step will be construction of stiffness and force matrices with respect to constructed shape functions. Stiffness formulation of LRPIM is given in Eq. 46 and it is regulated with Gauss quadrature in Eq. 47 for integration. J_{qi} and J_{qu} are Jacobean matrices at curve boundary locations.

$$K_I = \int_{\Omega_q} \hat{V}_I^T DBd\Omega - \int_{\Gamma_{qi}} \hat{W}_I^T nDBd\Gamma - \int_{\Gamma_{qu}} \hat{W}_I^T nDBd\Gamma \tag{46}$$

$$K_I = \sum_{k=1}^{n_g} \hat{w}_k \hat{V}_I^T(x_{Qk}) DB(x_{Qk}) \left| J_q^D \right| - \sum_{k=1}^{n_{gt}} \hat{w}_k \hat{W}_I^T(x_{Qk}) nDB(x_{Qk}) \left| J_{qi}^B \right| - \sum_{k=1}^{n_{gt}} \hat{w}_k \hat{W}_I^T(x_{Qk}) nDB(x_{Qk}) \left| J_{qu}^B \right| \tag{47}$$

Applied traction and body forces are integrated in Eq. 48 and it is regulated with Gauss quadrature for integration in Eq. 49.

$$f_I = \int_{\Gamma_{qt}} \hat{W}_I^T \bar{t} d\Gamma + \int_{\Omega_q} \hat{W}_I^T b d\Omega \tag{48}$$

$$f_I = \sum_{k=1}^{n_{gt}} \hat{w}_k \hat{W}_I^T(x_{Qk}) \bar{t} \left| J_{qt}^B \right| + \sum_{k=1}^{n_g} \hat{w}_k \hat{W}_I^T(x_{Qk}) b \left| J_q^D \right| \tag{49}$$

Similar stiffness and force matrices can be integrated in MLS shape functions, which are given in Eq. 50 and 51.

$$K_I = \int_{\Omega_q} \hat{V}_I^T DB d\Omega - \int_{\Gamma_{qi}} \hat{W}_I^T nDB d\Gamma - \int_{\Gamma_{qu}} \hat{W}_I^T nDB d\Gamma + \alpha \int_{\Gamma_{qu}} \hat{W}_I^T \Phi d\Gamma \tag{50}$$

$$f_I = \int_{\Gamma_{qt}} \hat{W}_I^T \bar{t} d\Gamma + \int_{\Omega_q} \hat{W}_I^T b d\Omega + \alpha \int_{\Gamma_{qu}} \hat{W}_I^T \bar{u} d\Gamma \tag{51}$$

3.1.5 `bc (boundary condition)` class

`bc` is defined for the essential and natural boundary conditions. It has a single property of `x`. It inherits this property to `integration_bcqt`, `integration_bcquqi` and `essentialbc` classes. `integration_bcquqi` is mainly used integration of Eq. 52, which is inside of stiffness matrix terms in Eq. 46 and 50.

$$- \int_{\Gamma_{qi}} \hat{W}_I^T nDB d\Gamma - \int_{\Gamma_{qu}} \hat{W}_I^T nDB d\Gamma \tag{52}$$

`integration_bcqt` class is used integration of Eq. 53, which represents integration of traction at the inside of Eq. 48 and 51.

$$\int_{\Gamma_{qt}} \hat{W}_I^T \bar{t} d\Gamma \tag{53}$$

In `integration_bcqt` class, applied traction is transferred to the nodes and stored as nodal values in `f` property. Then, `f` is used to calculate force vector. `integration_bcquqi` class is used in the calculation boundary integrals in a local quadrature domain. Essential boundary conditions are handled by using `essentialbc` class and all calculated values are recorded into `ak` property.

3.1.6 `solv (solution)` class

The target in developing `solv` class is encapsulation of all classes that are related with solutions. Therefore, it has no property and it is the base class of `dobmaxcalculator, solverband,`

`sparsestorage` and `getinverse`. `gausseqsolver` and `bandsolver` are subclasses of `solverband`. All these classes are generally used in the basic matrix operations and transfer their results to the class properties where they are called. `bandsolver` class is used solution of banded matrices and `gausseqsolver` class is used as alternatively usage of `bandsolver` for standard gauss elimination. `matlabsolver` class includes direct and different iterative methods for solution of linear equations (A*x=b), which are available in MATLAB library. Some iterative solvers are directly imported as class functions and ready for calling, which are Biconjugate gradients method (bicg), Biconjugate gradients stabilized method (bicgstab), Biconjugate gradients stabilized (l) method (bicgstabl), Conjugate gradients squared method (cgs), Generalized minimum residual method (gmres), LSQR method (lsqr), Minimum residual method (minres), Preconditioned conjugate gradients method (pcg), Quasi-minimal residual method (qmr), Symmetric LQ method (symmlq) and Transpose-free quasi-minimal residual method (tfqmr). These method are used for solution of linear equations (A*x=b) with iterations. Dimensions of matrices must be square (nxn) and symetric for term of A in minres, pcg and symmlq solvers. Other methods can be used with asymmetric matrices. Only lsqr method can be capable to solve mxn matrix dimensions of A. It is pointed [45] that stiffness matrices are generally sparse, banded and asymmetric. Hence all iterative solvers can be used compatible without minres, pcg and symmlq solvers.

Solution of complex geometries mostly causes to increase number of nodes in the solution. This phenomena cause to increase size of global stiffness matrix in meshfree methods. In most of the cases, global stiffness matrix holds zero values and its storage decrease computational effort. `sparsestorage` class is used for construction of sparse matrices, if it includes lots of zero terms. Sparse ratio function can calculate sparsity, which is the ratio of number of zero terms to all terms. Sparse matrices are also constructed with sparse function in MATLAB alternatively. If only a linear equation is solved with small sizes, sparse function can be used without `sparsestorage` class.

3.1.7 `post (results)` class

All methods, which are related with results of solution, are encapsulated in a single class; `post`. It has three basic properties, `x`, `numnode`, `u2`. It inherits its properties to `getdisplacement`, `getnodestress`, `output` and `getengerror` classes. Displacements and stresses are calculated with respect to Eq. 54 and 55.

$$\varepsilon_{(3\times1)} = B_{(3\times2n)} u_{(2n\times1)} \tag{54}$$

$$\sigma_{(3\times1)} = D_{(3\times3)} \varepsilon_{(3\times1)} = D_{(3\times3)} B_{(3\times2n)} u_{(2n\times1)} \tag{55}$$

`getdisplacement` class computes the nodal displacement under action of given loading condition and records them in `u2` and `disp` properties. The nodal stresses are computed in `getnodestress` class and recorded in `stress` property. In the `output` class, calculated displacements and stresses are written in a text file using its methods. The class also inherits all its properties and methods to `getengerror` that computes the energy error in the solution together with `totalgausspoints`.

4 Example solution and discussions

A cantilever beam with an end load (case I) and an axial loaded bar (case II) are solved using RPIM and MLPG methods in developed program. In these solutions, rectangular support domains and local

rectangular integration cells are used. The constructed model has a length of 48 cm and a height of 12 cm. Unit thickness is used in the analyses. The used material properties have Young modulus of 30.10^6 N/cm^2 and Poison's ratio of 0.0. Regularly distributed nodes have a nodal spacing of 4.8 cm in x (horizontal) direction and 3.00 cm in y (vertical) directions. Number of sub-division of quadrature domains is equal to 2 for both x and y directions. Number of gauss points is equal to 4 for each domain. 40 integration cells are used. Influence domain constant is selected as 3.00. Radial basis function parameters are selected as nrbf of 1.00, alfc of 1.00, dc of 3.00 and q of 1.03. The used number of basis functions is equal to 3. In both cases, the force (P) is taken as 1000 N as shown in Fig. 1. The shape parameters are taken from the compared study [45] for getting same solution procedure. The number of regular distributed nodes is equal to 55 in all solutions.

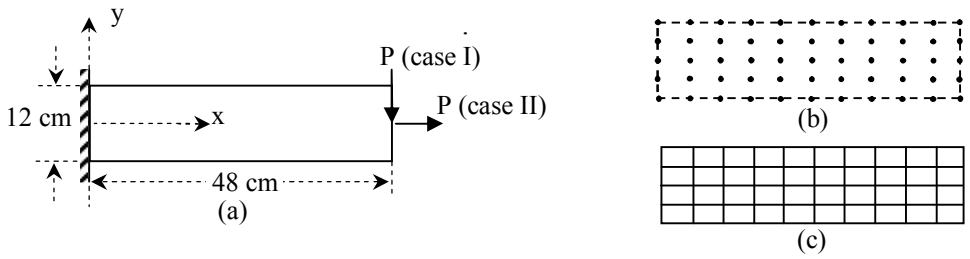


Figure 1: Engineering (a), meshfree (b) and FEM (c) models of axial loaded and cantilever beam problems and their boundary conditions

4.1 Key Design Principles and Solution Procedure

A main framework is constructed to manage all objects and all base classes use the MATLAB `handle` class. The overall scheme and interaction of classes can be seen in Fig. 2. `nodes` and `material` objects are used almost everywhere in the program. Therefore, they are created at the beginning of the solution. Construction of other classes is realized in sequence after these classes. All other classes are created by sending required data to their constructor functions. All computations are carried out inside of member functions and results are stored in objects as class properties. If local variables are not defined as a property of that class, they are cleared at the end of the class function operations. Hence, it prevents unnecessary memory usage. Only related and re-used from outside of class operators are assigned as class property.

Node coordinates, geometry and boundary conditions, meshfree parameters and other properties are read from files with `mc` class and stored in `nodes` object. Then `material` object is created for construction of material matrix with respect to plane stress or plane strain. When model operations in the classes are finished, calculation of quadrature domain of nodes begins with `inte` class. `qdomain` object stores the quadrature domain size and `gausscoefficient` object stores the gauss weights. In the solutions, local quadrature domain integrations are used. Therefore, `localquadraturedomain` object stores the quadrature domain boundaries and it is divided to subdomains. `nodes` object is called inside of this class and node coordinates are directly used without any conflicts. It is commonly used in procedural programming that transferring parameters with assigning new parameter names between programs and subroutines which causes to conflicts, especially in much sizes of codes. However, parameters are stored as properties of class objects and re-assignment is not required. `subdivisionlqdomain` object stores the divided local quadrature domain and gauss node positions are stored in `domaingausspoints` object. The type of support domain geometry is selected as rectangular. The number of nodes and node numbers in a support domain of a gauss point are stored in `supportdomain` object. Cubic spline test functions are used and their weights are stored inside of `testfunc` object.

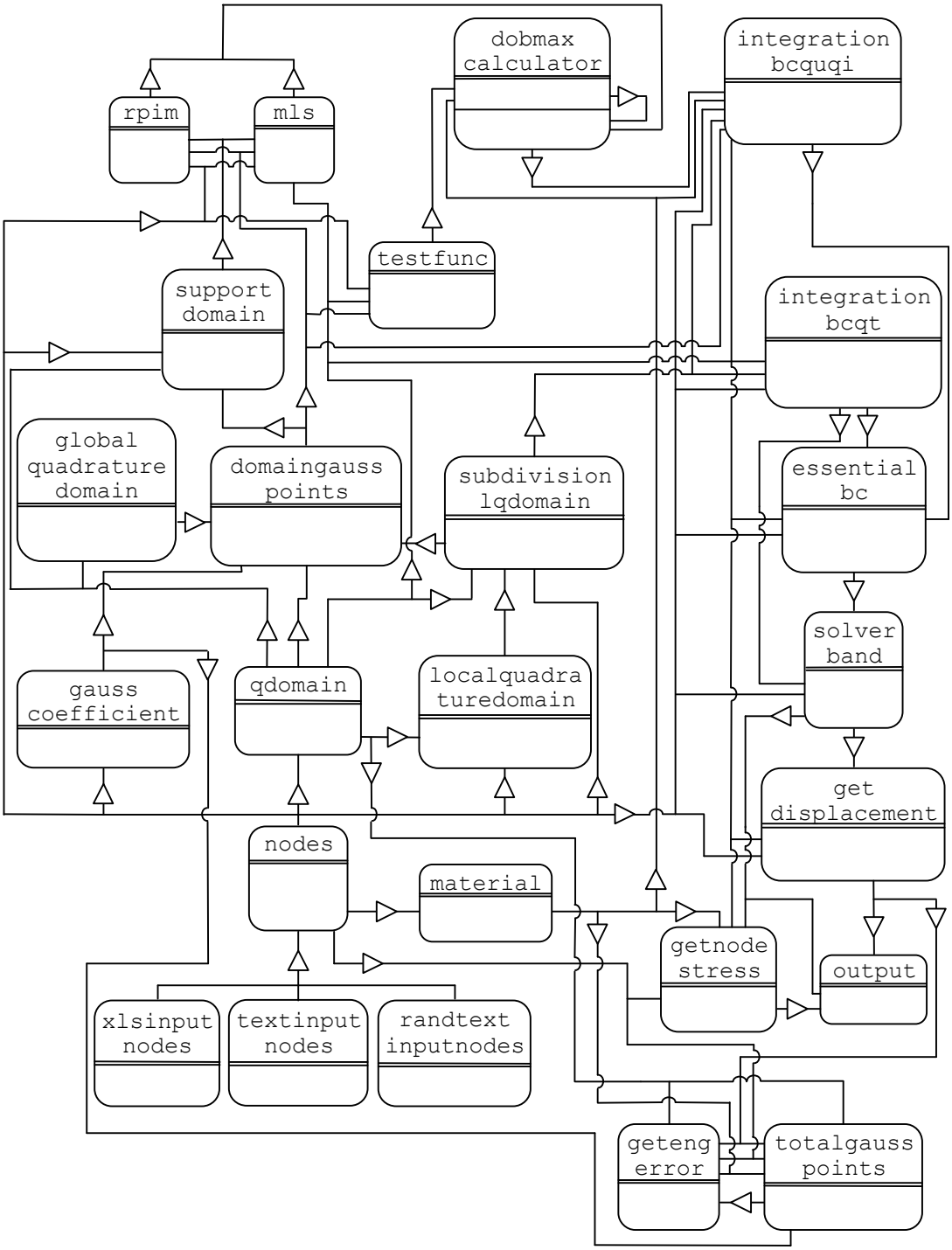


Figure 2: Overall flow and communications of classes during the solution for local quadrature domain

Next step is the calculation of shape functions. They are calculated and stored in `rpim` object. `computeradialbasis` class includes the derivatives of radial basis functions and it is used by `rpim` object. 3 different basis function classes are available; `mq`, `exp` and `tsp`. `mq` class is used in this analysis. It can be also calculated by using `mls` object using MLPG method. If `mls` method is used, the calculated shape functions are stored in `mls` object. Inheritance property has large benefits on the developments of meshfree methods, especially similar structures and their sub-links are available, if object oriented programming technique is used. Superclass properties are directly

transferred as properties of subclasses. Redefining of properties of subclasses is not required. Besides of that subclass includes its own special properties. Including new methods in shape function class can be easily handled by OOP, which can be defined as a subclass. Main properties like shape functions and their derivative values property `phi` directly assigns that new method.

OOP can be able to provide to work RPIM and MLS shape functions together for construction of shape functions. Both methods have their own benefits and accuracy in the calculations. One of them can be used in determined locations and the other can be used in other locations. This procedure can more easily accomplished by OOP. A switch is used to select the method for the construction of shape functions. `dobmaxcalculator` object is used for arrangements and calculations of shape functions. `integration_bcqt` object stores the applied force at related nodes. Also `integration_bcquqi` object stores the calculations of boundary integrals. All the calculations are repeated for each node and stored in the related class object. When the calculations finish, essential boundary conditions are applied and they are stored in `essentialbc` object.

Stiffness and force matrix includes lots of zero terms with respect to interpolation. `sparsestorage` class can be used for construction of sparse stiffness and force matrices. Also sparse function in MATLAB can be directly used without using `sparsestorage` class. However, either using sparse function or using `sparsestorage` class consumes time for construction of sparse matrix. In this study, sparse function is directly used. Because solution model has few nodes and their properties are easily stored in memory. `sparsestorage` class also consumes time, especially %25 greater solution times in `solv` class, when comparing solution with sparse function in this solution. `solverband` object is used for the solution of linear equations. After the solution, the nodal displacements are stored in `getdisplacement` object. `getnodestress` object stores the nodal stresses, which are calculated with respect to nodal displacements. `output` object stores the output of the `getdisplacement` and `getnodestress` object for writing them to an external file. Energy error is calculated by using `totalgausspoints` and `getengerror` objects and stored inside of `getengerror` object.

4.2 Solutions and discussion

Analytical and FEM solutions are available in the examined cases. The deformation and axial stress formulations are given in Eq. 56 and 57 for axial loaded bar problem. A is cross-section area of the bar, E is Young's modulus of material, x is horizontal distance from one end of the bar and P represents applied force.

$$\delta = \frac{P \times x}{A \times E} \tag{56}$$

$$\sigma = \frac{P}{A} \tag{57}$$

For the cantilever beam problem, the displacements are plotted for the upper side where tensional stresses occur. The deflection and axial stresses of the cantilever beam are given in Eq. 58 and 59. I is moment of inertia, L is length of the beam and c is the distance between upper surface and natural axis of the beam.

$$y = \frac{P}{6 \times E \times I}\left(x^3 - 3 \times L \times x^2\right) \tag{58}$$

$$\sigma = \frac{M \times c}{I} \tag{59}$$

In ANSYS, same geometrical model is constructed by using PLANE182 elements. These elements have 4 nodes at the edges for rectangular type. Plane stress assumption is used. Similar boundary conditions are applied as analytical solutions.

The original procedural meshfree program [45] gives the same displacement results as OOP meshfree program as shown in Fig. 3 for cantilever beam and axial loaded bar problems. The analytical and ANSYS solutions are also given and it is seen that they are in good agreements. RPIM solutions are a little bit closer to analytical solutions than MLS results.

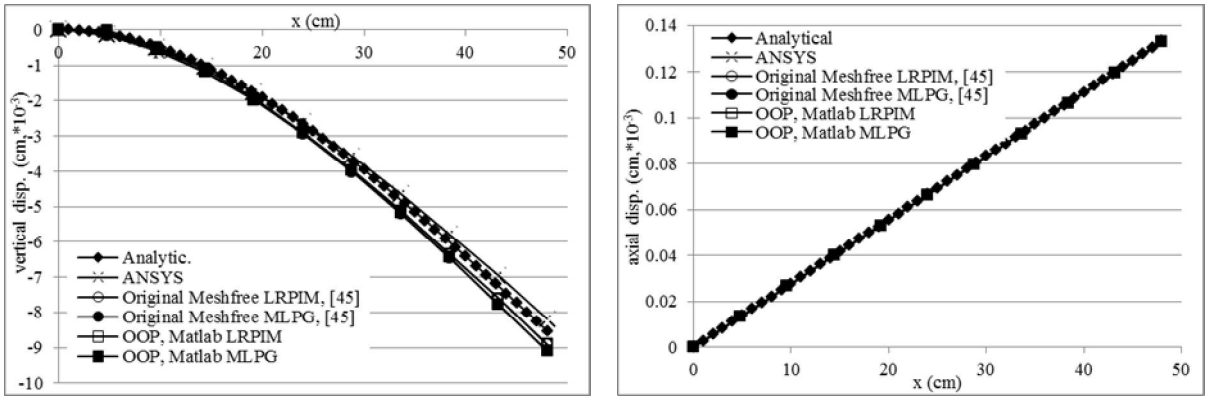
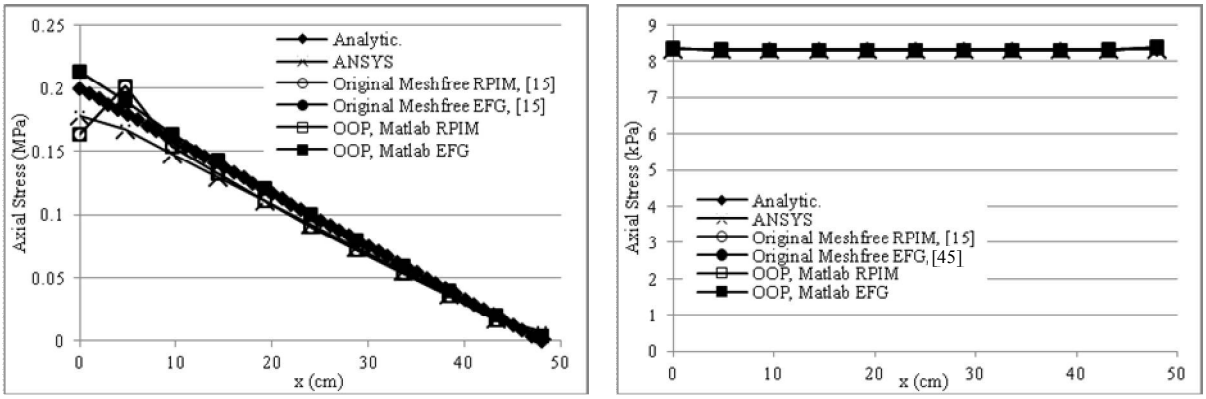


Figure 3: The displacement results of analytic, ANSYS, original meshfree [45] and OOP meshfree programs for cantilever beam problem (a) and for axial loaded beam problem (b)

Stress results are given in Fig. 4 for cantilever beam and axial loaded bar problems. It is seen that both original procedural meshfree program and OOP program nearly give the same results. All numerical results have good agreements with analytical solutions. A little difference can be seen in the results of RPIM method at the left corner, but same difference exists in the original procedural program. All stress results of bar problem are in perfect agreements with analytical and ANSYS results.



(a) (b)

Figure 4: The stress results of analytic, ANSYS, original meshfree [45] and OOP meshfree programs for cantilever beam problem and for axial loaded bar problem

5 Conclusion

Object-oriented programming of meshfree methods are discussed in detail for solution of elastostatic problems. Since new meshfree techniques are continuously improved by different researchers, the goal is developing a program that can be easily adapted to new meshfree techniques. Therefore a program is mainly organised based on types of support domains, integrations and shape functions. It can handle a solution in different ways:

- Using different integration cells; either global or local rectangular integration cells,
- Using different support domains; rectangular, circular or triangular support domains,
- Using different shape functions; either RPIM or MLS shape functions.
- Different MATLAB functions are included for increasing modularity at solution.

A new technique, for example about shape functions, can be easily adapted by creating its classes and objects without changing whole program structure. The source of a problem can be easily detected by only dealing with relevant classes.

In a procedural program, the program is developed using functions and subroutines. The parameters that are transferred to them must be carefully selected and general flow must be designed without causing a "spaghetti code" which is hard to understand and maintain. All variables in a procedural program are initially defined as public. However, in an object oriented program, the variables inside the classes are initially defined as private. The program is constructed based on objects and works by their interactions. Objects are instances of classes. Each class encapsulates its properties and methods. So, an encapsulated class can be modified and improved without considering the whole program structure.

The flexibility and modularity of OOP technique are used during developing of Meshfree code in MATLAB. However, it was shown that MATLAB OOP codes are slower against procedural FORTRAN codes. The reason of slowness of MATLAB is based on slowness of it doing some certain operations such as loops [49-51]. The results of MATLAB OOP program, on the other hand, are in perfect agreement with results of FORTRAN procedural program.

References

[1] Mackie, R.I., Object oriented programming of the finite element method. *International Journal for Numerical Methods in Engineering*, 35, 425-436, 1992.
[2] Zimmermann, T., Dubois-Pelerin, Y. and Bomme, P., Object oriented finite element programming: I. governing principles. *Computer Methods in Applied Mechanics and Engineering*, 98, 291-303, 1992.
[3] Dubois-Pelerin, Y., Zimmermann, T. and Bomme, P., Object oriented finite element programming: II. a prototype program in Smalltalk. *Computer Methods in Applied Mechanics and Engineering*, 98, 361-397, 1992.
[4] Ohtsubo, H. and Kawamura, Y., Development of the object-oriented finite element modeling system – modify. *Engineering with Computers*, 9, 187-197, 1993.
[5] Zimmermann, T., Bomme, P., Eyheramendy, D., Vernier, L. and Commend, S., Aspects of an object-oriented finite element environment. *Computers and Structures*, 68, 1-16, 1998.
[6] Pantale, O., An object-oriented programming of an explicit dynamics code: application to impact simulation. *Advances in Engineering Software*, 33, 297–306, 2002.
[7] Peters, B. and Dziugys, A., Numerical simulation of the motion of granular material using object-oriented techniques. *Comput. Methods Appl. Mech. Engrg.*, 191, 1983–2007, 2002.
[8] Ma, Y. and Feng, W., Object-oriented finite element analysis and programming in VC++.

Applied Mathematics and Mechanics, 23(12), 1437-1443, 2002.
[9] Yu, L. and Kumar, A.V., An object-oriented modular framework for implementing the finite element method. *Computers and Structures*, 79, 919-928, 2001.
[10] Patzak, B. and Bittnar, Z., Design of object oriented finite element code. *Advances in Engineering Software*, 32, 759-767, 2001.
[11] Menetrey, P. and Zimmermann, T., Object-oriented non-linear finite element analysis: application to j2 plasticity. *Computers & Structures*, 49(5), 767-777, 1994.
[12] Dubois-Pelerin, Y. and Pegon, P., Object-oriented programming in nonlinear finite element analysis. *Computers and Structures*, 67, 225-241, 1998.
[13] Zabaras, N. and Srikanth, A., An object-oriented programming approach to the Lagrangian fem analysis of large inelastic deformations and metal forming processes. *Int. J. Numer. Meth. Engng.*, 45, 399–445, 1999.
[14] Lages, E.N., Paulino, G.H., Menezes, I.F.M. and Silva, R.R., Nonlinear finite element analysis using an object-oriented philosophy – application to beam elements and to the cosserat continuum. *Engineering with Computers*, 15, 73–89, 1999.
[15] Commend, S and Zimmermann, T., Object-oriented nonlinear finite element programming: a primer. *Advances in Engineering Software*, 32, 611-628, 2001.
[16] Tabatabai, S.M.R., Object-oriented finite element-based design and progressive steel weight minimization. *Finite Elements in Analysis and Design*, 39, 55–76, 2002.
[17] Wegner, T. and Peczak, A., Implementation of a strain energy-based nonlinear finite element in the object-oriented environment. *Computer Physics Communications*, 181, 520–531, 2010.
[18] Peskin, A.P. and Hardin, G.R., An object-oriented approach to general purpose fluid dynamics software. *Computers chem. Engng*, 20(8), 1043-1058, 1996.
[19] Munthe, O. and Langtangen, H.P., Finite elements and object-oriented implementation techniques in computational fluid dynamics. *Comput. Methods Appl. Mech. Engrg.*, 190, 865-888, 2000.
[20] Sampath, R. and Nicholas, Z., An object oriented implementation of a front tracking finite element method for directional solidification processes. *Int. J. Numer. Meth. Engng.*, 44, 1227-1265, 1999.
[21] Qiao, H., Object-oriented programming for the boundary element method in two-dimensional heat transfer analysis. *Advances in Engineering Software*, 37, 248–259, 2006.
[22] Liu, J., Lin, I., Shih, M., Chen, R. and Hsieh, M., Object-oriented programming of adaptive finite element and finite volume methods. *Applied Numerical Mathematics*, 21, 439-467, 1996.
[23] Phongthanapanich, S. and Dechaumphai, P., EasyFEM - an object-oriented graphics interface finite element/finite volume software. *Advances in Engineering Software*, 37, 797–804, 2006.
[24] Marczak, R.J., Object-oriented numerical integration—a template scheme for FEM and BEM applications. *Advances in Engineering Software*, 37, 172–183, 2006.
[25] Lage, C., The application of object-oriented methods to boundary elements. *Comput. Methods Appl. Mech. Engrg.*, 157, 205-213, 1998.
[26] Wang, W., Ji, X. and Wang, Y., Object-oriented programming in boundary element methods using C++. *Advances in Engineering Software*, 30, 127-132, 1999.
[27] Dubois-Pelerin, Y. and Zimmermann, T., Object-oriented finite element programming: III an efficient implementation in C++. *Computer Methods in Applied Mechanics and Engineering,* 108, 165-183, 1993.
[28] Cary, J.R., Shasharina, S.G., Cummings, J.C., Reynders, J.V.W. and Hinker, P.J., Comparison of C++ and Fortran 90 for object-oriented scientific programming. *Computer Physics Communications*, 105, 20-36, 1997.
[29] Archer, G.C., Fenves, G. and Thewalt, C., A new object-oriented finite element analysis program architecture. *Computers and Structures*, 70, 63-75, 1999.
[30] Kromer, V., Dufosse, F. and Gueury, M., On the implementation of object-oriented philosophy

for the design of a finite element code dedicated to multibody systems. *Finite Elements in Analysis and Design*, 41, 493–520, 2005.
[31] Krysl, P. and Belytschko, T., ESFLIB: a library to compute the element free Galerkin shape functions. *Comput. Methods Appl. Mech. Engrg.*, 190, 2181-2205, 2001.
[32] Seidl, A. and Schmidt, T., Object oriented approach to consistent implementation of Meshless and classical FEM. *Systemics, Cybernetics and Informatics*, 4(1), 58-64, 2005.
[33] Zhang, X. and Subbarayan, G., jNURBS: an object-oriented, symbolic framework for integrated, meshless analysis and optimal design. *Advances in Engineering Software*, 37, 287–311, 2006.
[34] Sánchez, J.M.M. and Gonçalves, P.B., Shape function object modeling for meshfree methods. *Mecánica Computacional*, 29, 4753-4767, 2010.
[35] Barbieri, E. and Meo, M., A fast object-oriented Matlab implementation of the Reproducing Kernel Particle Method. *Comput Mech*, 49, 581–602, 2012.
[36] Lucy, L.B., A numerical approach to the testing of the fission hypothesis. *Astron. J.*, 82, 1013–1024, 1977.
[37] Nayroles, B., Touzot, G. and Villon, P., Generalizing the finite element method: diffuse approximation and diffuse elements. *Comput. Mech.*, 10, 307-318, 1992.
[38] Belytschko, T., Lu, Y.Y. and Gu, L., Element-Free Galerkin methods. *Int. J. Numer. Meth. Engng.*, 37, 229-256, 1994.
[39] Atluri, S.N. and Zhu, T., A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics. *Computational Mechanics*, 22, 117-127, 1998.
[40] Liu, G.R. and Gu, Y.T., A point interpolation method for two-dimensional solids. *Int. J. Numer. Methods Engrg.*, 50, 937-951, 2001.
[41] Liu, G.R. and Gu, Y.T., A local radial point interpolation method (LR-PIM) for free vibration analyses of 2-D solids. *J. of Sound and Vibration*, 246(1), 29-46, 2001.
[42] Liu, G.R., Zhang, G.Y. and Dai, K.Y., A linearly conforming point interpolation method (LC-PIM) for 2D solid mechanics problems. *International Journal of Computational Methods*, 2(4), 645–665, 2005.
[43] Liu, G.R., Zhang, G.Y., Wang, Y.Y., Zhong, Z.H., Li, G.Y. and Han, X., A nodal integration technique for meshfree radial point interpolation method (NI-RPIM). *International Journal of Solids and Structures*, 44, 3840–3860, 2007.
[44] Cui, X.Y., Liu, G.R. and Li, G.Y., A cell-based smoothed radial point interpolation method (CS-RPIM) for static and free vibration of solids. *Engineering Analysis with Boundary Elements*, 34, 144–157, (2010).
[45] Liu, G.R. and Gu, Y.T., *An introduction to Meshfree methods and their programming*, Springer, Berlin, 2005.
[46] Kanber, B. and Bozkurt, O.Y., A diagonal offset algorithm for the polynomial point interpolation method. *Commun. Numer. Meth. Engng.*, 24, 1909-1922, 2008.
[47] Liu, G.R., Zhang, G.Y., Gu, Y.T. and Wang, Y.Y., A meshfree radial point interpolation method (RPIM) for three-dimensional solids. *Comput. Mech.*, 36, 421-430, 2005.
[48] Liu, G.R., *Mesh free methods: moving beyond the finite element method*, CRC press; 2nd edition, New York, 2009.
[49] Berry, A., Bordat, J., Heggernes, P., Simonet, G. and Villanger, Y., A wide-range algorithm for minimal triangulation from an arbitrary ordering. *Journal of Algorithms,* 58, 33-66, 2006.
[50] Eklund, A., Andersson, M. and Knutsson, H., fMRI analysis on the GPU - possibilities and challenges. *Computer Methods and Programs in Biomedicine*, 105, 145-161, 2012.
[51] Wang, J. and Hopke, P.K., Equation-oriented system: an efficient programming approach to solve multilinear and polynomial equations by the conjugate gradient algorithm. *Chemometrics and Intelligent Laboratory Systems*, 55, 13-22, 2001.