A fast multi-pole expansion applied to bending of plates integral equations

A Thesis
Submitted to the
Faculty of Engineering Shoubra, Benha University
In Partial Fulfillment of the Requirements for the
Degree of Master of Science
in
Engineering Mathematics

by

Mohamed Elsayed Mohamed Elsayed Nassar

Demonstrator in Engineering Mathematics and Physics Department
Faculty of Engineering at Shoubra, Benha University

Faculty of Engineering-Shobra
Benha University
2013

A fast multi-pole expansion applied to bending of plates integral equations

by

Mohamed Elsayed Mohamed ElsayedNassar

Demonstrator in Engineering Mathematics and Physics Department
Faculty of Engineering at Shoubra, Benha University

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Engineering Mathematics

Supervisor

Prof. Dr. Abd-Elrahman Saad

Professor of Applied Mathematics
Mathematical and Physical Eng.Dept.
Faculty of Engineering - Shoubra
Banha University

Prof. Dr. Youssef F. Rashed

Professor of Structural Analysis and Mechanics
Structural Eng.Dept.
Faculty of Engineering
Cairo University

Dr. Mohamed Y.Akl

Lecturer of Applied Mathematics
Mathematical and Physical Eng.Dept.
Faculty of Engineering - Shoubra

Banha University

Faculty of Engineering-Shobra
BenhaUniversity
2013

APPROVAL COMMITTEE

The Committee on Final Examination recommends that the thesis by

Mohamed Elsayed Mohamed ElsayedNassar

entitled

A fast multi-pole expansion applied to bending of plates integral equations

be accepted in partial fulfillment of the requirements for the degree of
Master of Science in Engineering Mathematics

EXAMINERS COMMITTEE:

Prof. Dr. Abd-Elrahman Saad ……

Professor of Applied Mathematics — Supervisor
Mathematical and Physical Eng.Dept.
Faculty of Engineering - Shoubra
Benha University

Prof. Dr. Mohamed Mohamed Ali Nassar ……

Professor of Applied Mathematics — External Examiner
Mathematical and Physical Eng.Dept.
Faculty of Engineering
Cairo University

Prof. Dr. Youssef Fawzy. Rashed .....

Professor of Structural Analysis and Mechanics — Supervisor
Structural Eng.Department
Faculty of Engineering
Cairo University

Ass.Prof. Dr. Moawad El-Sharnoby .....

Ass.Professor of Applied Mathematics — Internal Examiner
Mathematical and Physical Eng.Dept.
Faculty of Engineering - Shoubra
Benha University

Date: …….

To my father,

my mother,

my Brother,

my sisters,

my wife,

and my son.

Without whom,

none of this effort was mint to happen

ACKNOWLEDGEMENT

First of all due thanks go to **God** the most merciful and most graceful. Who without his guidance and inspiration nothing could have been accomplished.

I would like also to thank my professors in my college for the advices and support.I would like to express my gratitude to everyone who contributed, in different ways, to completion of this work. Inevitably some names will be missing here.

I wish to express my great appreciation and thanks to **Prof. Dr. Abd-Elrahman Saad**, Professor of Applied Mathematics, Mathematical and Physical Engineering Deptartement,Faculty of Engineering Shoubra, Benha University, for his kind guidance, valuable advice, sincere fatherhood, and continuous caring during this research.

I also wish to express my deep indebtedness to **Prof. Dr. Youssef Fawzy Rashed,** Professor of Structural Analysis and Mechanics, Structural Engineering Department, Faculty of Engineering, Cairo University, for his generous guidance and encouraging, sincere help, consistent support by all means and asking, valuable suggestions, and precise advice through all stages of this research work, I express my true thanks and gratitude for opening my mind to the true values of sincere and creativity. I have learned many lessons in working under his guidance and leadership that I will remember for an extremely long time.

Sincere thanks go **Dr. Mohamed Y. Akl**, Applied Mathematics, Mathematical and Physical Engineering Deptartement, Faculty of Engineering - Shoubra, Benha University, for his encouragement and advice in theearly stages of my work: the privilege of studying under him will always remain amemorable part of my life.

I am indebted to **Dr. Moawad El-Sharnoby** for his kind and spontaneous help with the integrations, without which the completion of this work would have to be delayed.

I would like to express my very great appreciation to **Dr. Morcos saman** for his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time so generously has been very much appreciated.

My thanks also go to my colleagues, especially **Eng.Mohamed Ahmed kamal**, **Eng.Taha Hassan, Eng. Mostafa Mobasher** and all friends who are supported me all the way to achieve this work.

A fast multi-pole expansion applied to bending of plates integral equations

by

Mohamed Elsayed Mohamed ElsayedNassar

An Abstract of

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Engineering Mathematics

Faculty of Engineering -Shobra
Benha University

This Thesis presents a new fast multi-pole boundary element formulation for the solution of Reissner's plate bending problems. The solution of Reissner's plate bending problems using the conventional direct boundary element method leads to a non-symmetric fully populated system of matrices. The complexity of the solution then becomes of the order O(N^3) mathematical operations, where N is the total number of problem unknowns. Hence, the use of fast multi-pole technique becomes practically essential in case of solving large-scale problems by the direct boundary element method.

The suggested formulation is based on representing the fundamental solutions as function of potentials. These potentials and their relevant fundamental solutions are expanded by means of Taylor series expansions. In the present formulation, equivalent collocations are based on the first shift expansion of kernels. This is achieved by representation of far field integrations by series expansions and carrying out summations of far clusters, whereas the near field integrations are kept to be computed directly.

In the presented implementation, the fast multi-pole boundary element method is coupled with the iterative solver: Generalized Minimal Residual System (GMRES). The computational complexity is rapidly reduced to be O(N log N). Numerical examples are presented to demonstrate the efficiency, time saving, and accuracy of the

formulation against the conventional direct BEM. The accuracy of the results is traced by cutting Taylor series to few terms. It was proven via numerical examples that three terms are enough to produce sufficient accuracy with substantial reduction of solution time.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction and Background

1.1 Introduction

Many engineering problems can be expressed using partial differential equations (PDEs). In early stages, solutions for PDES could only be obtained using analytical solution. However, analytical solutions are only applicable in some simple cases. In cases of complicated problems, analytical methods become increasingly tedious or even impossible. After the invention of modern computers, researchers focused on applying numerical methods in solutions of PDES. Such methods transform PDEs to algebraic equations that computer codes can solve. Many famous numerical methods have been used, such as finite difference method (FDM), finite element method (FEM) and boundary element method (BEM). The boundary element method is one of the strongest numerical methods in this field, but its application is limited only for small scale problems with simple geometry. In case of large scale problems, the BEM needs large memory for storage of coefficients matrices and a fully populated coefficient matrix is required to be solved. In other words, BEM produces dense and non-symmetric matrices, require $O(N^2)$ operations for computing the coefficient and $O(N^3)$ operations for solving the system by using direct solvers N is the number of unknowns. This disadvantage is not found in the finite element method, so that FEM is widely used in programming. In order to enable BEM to lead against FEM, many researches focused their works to overcome the previously mentioned problem concerned large scale problems solutions. So that the fast multi-pole method (FMM) that originally invented to solve N-particle large scale problems was applied in the BEM different applications. With the help of the FMM, the BEM can now solve large scale problems that are beyond the reach of other methods because its capable of reduce the computer processing unit (CPU) elapsed time and accelerate conventional BEM calculations to $O(N)$. However one of these applications, which is the solution of Ressiner's plate bending problems was not carried out yet. In this thesis, a novel technique which uses the Fast Multi-pole Method in solving Ressiner's plate bending problems is introduced. To justify the selection of BEM into this thesis work, a review

is introduced for different numerical methods and their advantages and disadvantages in next sections.

1.2 Numerical methods

In this section the numerical methods that were widely used are reviewed including their advantage and disadvantage. Later, a comparison is made to demonstrate why the BEM is selected in this thesis work.

1.2.1 Finite difference method (FDM)

The FDM is a method used in the solution of boundary value problems for PDES. The FDM is based on a mathematical discretization of the plate continuum. For this purpose, the plate is covered by a two-dimensional mesh; the partial derivatives in the governing plate equation are replaced by corresponding finite difference quotients at each mesh point. In this way, the differential equation governing the displacements is transformed into algebraic equations.

Advantage of the FDM:

a) Conceptual simplicity.
b) Mathematical simplicity.
c) Ease of programming.

Disadvantage of the FDM:

a) It requires more work to achieve problem modeling and simulation.
b) The matrix of the approximating system of linear algebraic equations if asymmetric, causing some difficulties in numerical solution of this system.
c) An application of the FDM to domains of complicated geometry may run into serious difficulties.
d) Inaccuracies due to many assumptions and approximations.

1.2.2 The finite element method (FEM)

According to the FEM [1], the whole domain of the plate under consideration is discretized into small elements which are only connected at their corner nodes. The unknowns of the problem are the deflections and rotations in the directions of prescribed degrees of freedom (generalized displacements). The values of these unknown are obtained from the solution of equilibrium and compatibility equations

assembled from all elements. Due to the wide usage of the FEM, there are many established commercial programs that based on the method such as (SAP2000) [2] …etc.

Advantage of the FEM:

a) Ability of modeling different geometries and nonlinear materials.
b) Obtained system matrices are positive definite, banded, and sparse.
c) Widely tested approach.
d) Commercial availability.
e) Flexibility.

Disadvantage of the FEM:

a) The FEM requires the use of powerful computers of considerable speed and storage capacity due to domain discretization in large scale problems.
b) It is difficult to ascertain the accuracy of numerical results when large structural systems are analyzed.
c) The method is poorly adapted to a solution of the so-called singular problems (e.g., plates and shells with cracks, corner points, discontinuity internal actions, etc.), and of problems for unbounded domains.
d) The method presents many difficulties associated with problems of C^1continuity and nonconforming elements in plate (and shell) bending analysis.
e) Large effort and time consume in discretization of the domain and no flexibility in modification.
f) Stress concentration.
g) Element values average at vertex.

1.2.3 The boundary element method (BEM)

The boundary element method is a numerical computational method for solving linear partial differential equations which have been formulated as boundary integral equations. It can be applied in many areas of engineering and science including solid mechanics [3, 4], fluid mechanics [5, 6], acoustics [7, 8], fracture mechanics [9], and plasticity [10].

In recent years, the BEM has emerged as a powerful alternative to the FDM and FEM. While these numerical solution techniques require the discretization of the entire plate

domain, the BEM applies discretization only at the boundary of the continuum. Boundary element methods are usually divided into two categories: direct and indirect BEMs. The direct BEM formulates the problem in terms of variables that have definite physical meanings, such as displacements of the boundary nodes of the plate. In contrast, the indirect BEM uses variables whose physical meanings cannot always be clearly specified. The BEM's history can be traced back to 1963 when Jaswon and Symm [11] proposed an integral method, in which; numerical solutions of 2-D potential problems were attempted by exploiting Green's third identity. Their attempt of using Green's third identity inspired Rizzo [3] in his research on elasticity, in which the Somigliana identity received special attention. Rizzo's first paper [4] on solving elastostatic problem using integral equation approach has much stimulated the modern day development of the boundary element method. Later on, the boundary integral equation method has been explored and extended to many other research areas. The term "boundary element method" was first appeared in [12] in 1977 after the first paper by Lachat who incorporated shape functions, Gauss quadrature techniques borrowed from the FEM into boundary integral equation method. For the acoustic problem governed by Helmholtz equation, the first effort of using integral equation appeared in the same year that Jaswon and Symm published their first paper [11] of integral equation for potential problems. Chen and Schweikert [7] solved 3-D sound radiation problems using Fredholm integral equation of the second kind. Early works on boundary integral equation method for acoustic problems also include Chertock's research [13], in which he predicted sound radiation from vibrating surfaces using integral equation, and Copley's papers [14, 15], in which the non-uniqueness difficulties of the exterior boundary integral equation (BIE) at eigen frequencies associated with corresponding interior problems was first reported. A similar integral-equation-based method, T-Matrix method, was developed by Waterman [8] in 1965 for solving electromagnetic problems. To tackle the fictitious eigen frequency difficulties reported in Ref. [14], Schenck [16] in 1968 came up with an idea of adding some additional Helmholtz integral relations in the interior domain. Burton and Miller [17] in 1971 proposed another technique to overcome the fictitious frequency difficulties by linearly combining the conventional boundary integral equation (CBIE)

and the normal derivative of the BIE (HBIE) to circumvent this problem. Subsequent researches on the BEM include, but not limited to:

1) Improving efficiency, including parallel computation [18], iterative solver [19, 20], fast multi-pole method as will be detailed later [21, 22, 37], and many others. With the intensive research, the BEM continues to be a viable numerical simulation tool for many problems.
2) New technologies and formulations to extend the applicability of the BEM, for example: half-space problems [23], indirect BEM [24], Galerkin BEM [25], transient analysis [26], inverse BEM [27], Eigen frequency determination [28], FEM-BEM coupling [29], and hybrid BEM [30].
3) Effective integral evaluations, especially for the hyper-singular integrals. Various methods have been proposed, for example, direct evaluation in Hadamard finite part sense [31], regularization with Taylor expansions [32, 33] or Fourier–Legendre series [34], transforming into integrals with kernel of tangential derivatives or double surface integrals [17], indirect evaluations [35], and singularity subtraction [36].

Advantage of the BEM:

a) Reduces problem dimension.
b) Less unnecessary information.
c) Focus on the body boundary.
d) Good for incompressible materials and unbounded domains.
e) Easy to define and vary boundary elements.
f) Accuracy.
g) Good for stress concentrations.

Disadvantage of the BEM:

a) Unfamiliar mathematics.
b) Not efficient for nonlinear problems.
c) Fully populated matrices.
d) Matrix is not symmetric, so it is non-convenient for large scale problems.

1.2.4 A Comparison of the FE and BE methods

In this section some major differences between the two methods are outlined. Depending on the application some of these differences can either be considered as advantageous or disadvantageous to a particular scheme.

1) **FEM**: An entire domain mesh is required.

   BEM: A mesh of the boundary only is required.

   Comment: Because of the reduction in size of the mesh, one often hears of people saying that the problem size has been reduced by one dimension. This is one of the major pluses of the BEM - construction of meshes for complicated objects, particularly in 3D, is a very time consuming exercise.

2) **FEM**: Entire domain solution is calculated as part of the solution.

   BEM: Solution on the boundary is calculated first, and then the solution at domain points (if required) are found as a separate step

   Comment: There are many problems where the details of interest occur on the boundary, or are localized to a particular part of the domain, and hence an entire domain solution is not required.

3) **FEM**: Reactions on the boundary typically less accurate than the dependent variables.

   BEM: Both "**u**" and "**q**" of the same accuracy.

4) **FEM**: Differential Equation is being approximated.

   BEM: Only boundary conditions are being approximated.

   Comment: The use of the Green-Gauss theorem and a fundamental solution in the formulation means that the BEM involves no approximations of the differential Equation in the domain - only in its approximations of the boundary conditions.

5) **FEM**: Sparse symmetric matrix generated.

   BEM: Fully populated non symmetric matrices generated.

   Comment: The matrices are generally of different sizes due to the differences in size of the domain mesh compared to the surface mesh. There are problems where either method can give rise to the smaller system and quickest solution -

it depends partly on the volume to surface ratio. For problems involving infinite or semi-infinite domains, BEM is to be favored.

6) **FEM**: Element integrals easy to evaluate.

   BEM: Integrals are more difficult to evaluate, and some contain integrands that become singular.

   Comment: BEM integrals are far harder to evaluate. Also the integrals that are the most difficult (those containing singular integrands) have a significant effect on the accuracy of the solution, so these integrals need to be evaluated accurately.

7) **FEM**: Widely applicable. Handles nonlinear problems well.

   BEM: Cannot even handle all linear problems.

   Comment: A fundamental solution must be found (or at least an approximate one) before the BEM can be applied. There are many linear problems (*e.g.,* virtually any non-homogeneous equation) for which fundamental solutions are not known. There are certain areas in which the BEM is clearly superior, but it can be rather restrictive in its applicability.

8) **FEM**: Relatively easy to implement.

   BEM: Much more difficult to implement.

   Comment: The need to evaluate integrals involving singular integrands makes the BEM at least an order of magnitude more difficult to implement than a corresponding finite element procedure.

1.2.5 The fast multi-pole method (FMM):

The fast multi-pole is not a method to solve a certain problem, but it is a technique that applies in a certain method to improve its efficiency. In conventional BEM, since the formed equation matrix is found to be dense and nonsymmetrical, this characteristic limits the use of the conventional BEM to medium size problems and simple geometries. In other words, the complexity order of conventional BEM calculations is $O(N^3)$ numerical operations, where N is the number of unknowns. However, it was concluded that this order can be reduced to $O(N \log N)$ or $O(N)$ with the help of what is named first shift or second shift of fast multi-pole method (FMM) respectively. Some $O(N \log N)$ and $O(N)$ algorithms were first reported to solve

particle simulation problems in the 1980s. Barnes and Hut [38] used a tree data structure and the concept of multi-pole expansion to calculate the matrix-vector product without forming the matrix explicitly. Their tree-code algorithm reduces the calculation complexity from O(N^3) to O(N log N). By introducing the concept of the second shift, Greengard and Rokhlin's [39], the complexity was further reduced to O(N). The first FMM formulations [38, 39] for the solution of the particle simulation problem have influenced the BEM numerical solution of boundary value problems especially in potential theory [40].

This novel technique for the solution of the N particle simulation problems has immediate implications on the BEM numerical solution of boundary value problems in potential theory, as its discrete linear system is the product of pair-wise interactions between sources. By using the tree structure, we can rearrange the precondition fully populated matrix according to boundary representation by near field and far field elements. The kernel expansion and the grouping technique, facilitate to translate the fully populated matrix which obtained from conventional BEM to sparse matrix in FMM with band width depend on the number of levels in the tree structure but this translation forced us to assume some unknowns and convert form direct solver to iterative solver. Recently, there has been an increasing interest in the analysis of the performance of iterative solutions of the equation sets arising from the BEM formulation [41– 45].

Algorithms that utilize FMM idea may be classified into two categories: spherical harmonic expansions and Taylor series expansions [46].

Several O(N) algorithms for the direct BEM formulation of 3D elasticity problems based on spherical harmonic expansions are available in the literature. Among them, the works of Fu et al. [47] and Hayami and Sauter [48] are worth of special attention. Fu et al. decomposed the original 3D elasticity fundamental solution into five terms in such a way that each of them can be expanded in terms of spherical harmonic series with their corresponding duality principle. They tested the performance of the proposed algorithm by analyzing the problem of the elastic interaction of hundreds of solid spherical particles in which N = O (10^5): Hayami and Sauter [50] on the other hand, proposed an expansion of the fundamental solution in terms of the spherical

harmonic expansion of 1/R and the derivatives of R. Order O(N log N) algorithms in terms of the Taylor series expansion of the fundamental solution are also available in the literature for elasticity problems. Peirce and Napier [49] developed, using a Taylor series multi-pole expansion, an algorithm to solve a 2D problem of multiple cracks in an elastic media.

It has been previously reported that multi-pole boundary element strategy based on Taylor expansions will result in computer codes which require O(N log N) operations for problems with N DOF. Popov and Power [51] presented a multi-pole BEM strategy developed for 3D elasticity problems which is also based on Taylor expansions but requires only O(N) operations. Popov and Power's efficient algorithm results from the use of a clustering technique, first shift, in combination with an additional Taylor series expansion around the collocation points, second shift. The idea of the use of the second shift for a multi-pole scheme based on the Taylor series expansions was original proposed by Gomez and Power [52] to solve a 2D viscous flow problem using an indirect BEM formulation.

In conventional calculations of BEM, the matrices are fully populated and non-symmetric. Therefore, the solution of the system when using direct solvers is expensive in computational costs. To overcome this drawback and to increase the efficiency of fast multi-pole solution, iterative solvers have to be adopted in the FMM formulation. Among these iterative techniques, Krylov subspace iterative methods are acknowledged as very effective iterative techniques for the linear systems arising from BEM formulation, especially, GMRES (generalized minimal residual system) [53, 54]. The GMRES is an implicit iterative algorithm, and it can be applied to FMBEMs in conjunction with FMM.

In this thesis, the application of FMM on the conventional BEM for plate bending problems is presented. In tracing the history of formulations based on conventional BEM for plate bending problems [55], it is recorded the work of Bezine [56] and Stern [57] for the thin plate theory and by Van der Weeen [58] for the thick plate theory. The plate theory according to Reissner [59] represents a general method for solving

both thin and thick plates. In this thesis, the formulation in [58] is considered as a base in applying FMM on the conventional BEM.

1.3 Thesis objectives

The objectives of this thesis are:

1. To create the mathematics of the fast multi-pole formulation applied to direct boundary element method solution of shear-deformable plate bending problems.
2. To produce sparse matrix instead of fully populated matrix developed in conventional boundary element coefficient matrix.
3. To investigate and set the possible strategies for the implementation of the derived formulation into feasible algorithm.
4. To implement the derived formulation in computer code and utilizing the iterative solver Generalized Minimal Residual System (GMRES).
5. To test the proposed formulation by applying the developed code to problems and track the efficiency of the solution.

1.4 Thesis organization

This thesis consists of five chapters after this chapter. These chapters contain the followings:

Chapter 2: The basic unit load and unit displacement (fundamental solutions) are reviewed. Basic fundamentals of the boundary element method (BEM) for analysis of thick plate are presented. The first principles of Reissner plate bending are reviewed including the definitions of stresses, strains, tractions, and boundary integral equation introduced

Chapter 3: In this chapter the mathematical of fast multi-pole method are derived.

Chapter 4: In this chapter, a developed computer code for Thick Plate Bending Problems for the suggested formulations is presented.

Chapter 5: This chapter presents two examples including comparison between results of the fast multi-pole method for the thick plate theory and results of analytical solution.

Chapter 6: In this chapter a summary, conclusions and recommendations for future are given.

1.5 Conclusions

In this chapter, an introduction about history of plate theory development, the numerical methods used in the solution of partial differential equations (PDEs) were presented. The advantages and disadvantages for such a method were reviewed. Short notes about fast multi-pole method also were reviewed. Finally the structure of this thesis was presented. In the next chapter the relevant theoretical basis for conventional BEM of Reissner's plate bending is reviewed.

Chapter 2: Boundary element method for Reissner's plate

2.1 Introduction

In this chapter, the BEM application on Reissner's plate bending problems is reviewed [62]. The conventional BEM formulation of Reissner's plate bending theory is introduced. Then the fundamental solutions are derived based on their potential. Finally a conclusion is made and given.

2.2 Conventional BEM for Reissner's plate

Consider an elastic plate of domain Ω and boundary Γ. The plate has a thickness h and is lying in the x_1-x_2 plane where $x_3=0$ is located at the mid surface of the plate. The indicial notation is used in this thesis where the Greek indices vary from 1 to 2 and the Roman indices vary from 1 to 3. According to Reissner's plate bending theory [59], the differential equations that are governing the equilibrium can be written at any arbitrary point ξ,in absence of body forces, as follows [59]:

$$M_{\alpha\beta,\beta}(\xi) - Q_{3\alpha}(\xi) = 0$$

$$Q_{3\alpha,\alpha}(\xi) = 0 \tag{2.1}$$

where, $M_{\alpha\beta}(\xi)$ and $Q_{3\alpha}(\xi)$are the bending moments and shear forces stress resultants respectively. The stress-resultant generalized displacement relationships can be written at point ξas follows [59]:

$$M_{\alpha\beta}(\xi) = D\frac{1-\nu}{2}\left(u_{\alpha,\beta}(\xi) + u_{\beta,\alpha}(\xi) + \frac{2\nu}{1-\nu}u_{\gamma,\gamma}(\xi)\delta_{\alpha\beta}\right) \tag{2.2}$$

$$Q_{3\beta}(\xi) = D\frac{1-\nu}{2}\lambda^2\left(u_{\beta}(\xi) + u_{3,\beta}(\xi)\right) \tag{2.3}$$

where $D = \dfrac{Eh^3}{12(1-\nu^2)}$ is the plate modulus of rigidity, λ is the shear factor, E and ν are the Young's modulus and Poisson's ratio respectively, $u_\alpha(\xi)$ denotes the rotation in two directions x_1, and x_2, u_3 denotes the deflection in x_3 direction at point ξ (for the positive directions refer to Fig. (2.2)), and the symbol $\delta_{\alpha\beta}$ denotes the identity matrix. Using Eqs. (2.2) and (2.3), Eq. (2.1) can be re-written in condensed form as follows:

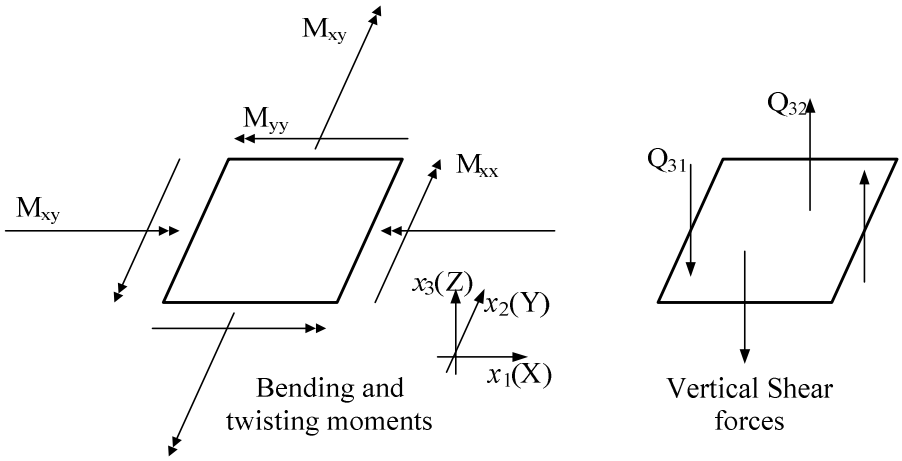

Figure 2.1: The positive directions for the internal straining actions.

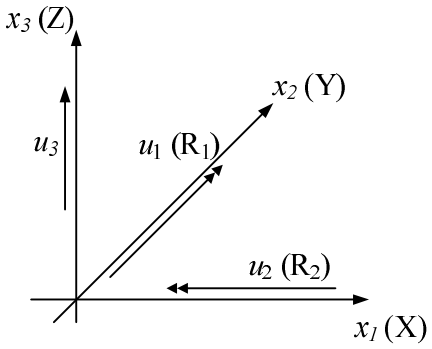

Figure 2.2: Positive directions for the generalized displacement.

$$L_{ij}u_j(\xi) = 0 \tag{2.4}$$

where L_{ij} is the differential operator that can be written as follows:

$$L_{ij} = \begin{bmatrix} \dfrac{D(1-\nu)}{2}\left[\left(\nabla^2 - \lambda^2\right)\delta_{\alpha\beta} + \dfrac{(1+\nu)}{(1-\nu)}\partial_\alpha\partial_\beta\right] & \dfrac{-D(1-\nu)}{2}\lambda^2\partial_\alpha \\ \dfrac{D(1-\nu)}{2}\lambda^2\partial_\beta & \dfrac{D(1-\nu)}{2}\lambda^2\nabla^2 \end{bmatrix} \tag{2.5}$$

in which $\partial_\alpha(\bullet) = \dfrac{\partial(\bullet)}{\partial x_\alpha(\mathrm{x})}$, and ∇^2 is the two-dimensional Laplace operator.

The direct boundary integral equation for Reissner's plate can be written in the following form [58]:

$$\frac{1}{2}u_i(\xi)+\int_{\Gamma(x)}T_{ij}(\xi,x)u_j(x)\,d\Gamma(x)=\int_{\Gamma(x)}U_{ij}(\xi,x)t_j(x)\,d\Gamma(x) \tag{2.6}$$

Where $u_j(x)$ and $t_j(x)$ denote the boundary generalized displacement and traction vectors respectively. ξ and x denote a source point and a field point located on the boundary respectively. $U_{ij}(\xi,x)$ and $T_{ij}(\xi,x)$ denote the two-point fundamental solution kernels [58]:

$$U_{\beta\alpha}(\xi,x)=\frac{1}{8\pi D(1-\nu)}\left\{\left[8B(\lambda R)-(1-\nu)\left(2\ln(\lambda R)-1\right)\right]\delta_{\beta\alpha}-\left[8A(\lambda R)+2(1-\nu)\right]R_{,\beta}R_{,\alpha}\right\} \tag{2.7}$$

$$U_{3\alpha}(\xi,x)=\frac{1}{8\pi D}\left(2\ln(\lambda R)-1\right)RR_{,\alpha} \tag{2.8}$$

$$U_{\alpha 3}(\xi,x)=-U_{3\alpha}(\xi,x) \tag{2.9}$$

$$U_{33}(\xi,x)=\frac{1}{8\pi D(1-\nu)\lambda^2}\left[(1-\nu)(\lambda R)^2\left(\ln(\lambda R)-1\right)-8\ln(\lambda R)\right] \tag{2.10}$$

$$\begin{aligned}T_{\beta\alpha}(\xi,x)=\frac{-1}{4\pi R}\Big[&\left(4A(\lambda R)+2(\lambda R)K_1(\lambda R)+1-\nu\right)\times\left(\delta_{\beta\alpha}R_{,N}+R_{,\alpha}n_\beta\right)\\&+\left(4A(\lambda R)+1+\nu\right)R_{,\beta}n_\alpha-2\left(8A(\lambda R)+2(\lambda R)K_1(\lambda R)+1-\nu\right)R_{,\beta}R_{,\alpha}R_{,n}\Big]\end{aligned} \tag{2.11}$$

$$T_{3\alpha}(\xi,x)=\frac{(1-\nu)}{8\pi}\left[\left(2\left(\frac{1+\nu}{1-\nu}\right)\ln(\lambda R)-1\right)n_\alpha+2R_{,\alpha}R_{,n}\right] \tag{2.12}$$

$$T_{\gamma 3}(\xi,x)=\frac{\lambda^2}{16\pi}\left\{\left[8B(\lambda R)-2(1-\nu)\left(2\ln(\lambda R)-1\right)\right]n_\gamma-\left[8A(\lambda R)+4(1-\nu)\right]R_{,\gamma}R_{,n}\right\} \tag{2.13}$$

$$T_{33}(\xi,x)=\frac{-R_{,n}}{16\pi R}\left[\left(4\ln(\lambda R)-2\right)(\lambda R)^2-\frac{8}{(1-\nu)}\right] \tag{2.14}$$

where :

$$A(\lambda R)=K_0(\lambda R)+\frac{2}{(\lambda R)}\left[K_1(\lambda R)-\frac{1}{(\lambda R)}\right] \tag{2.15}$$

$$B(\lambda R)=K_0(\lambda R)+\frac{1}{(\lambda R)}\left[K_1(\lambda R)-\frac{1}{(\lambda R)}\right] \tag{2.16}$$

in which $K_0(\lambda R)$ and $K_1(\lambda R)$ are modified Bessel functions [61].

In order to solve the boundary integral equation, Eq. (2.6), the boundary is discretized (without losing the generality) into constant elements at which, the generalized displacements and/or the tractions are prescribed at some nodes. The boundary unknowns are obtained by rewriting Eq. (2.6) at a number of collocation nodes equals to the number of unknowns. The solution of such equations will produce all boundary unknowns [58]. Hence, Eq. (2.6) can be rewritten as follows:

$$\frac{1}{2}u_i(\xi) + \sum_{j=1}^{N} u_j(\mathrm{x}) \int_{\Gamma_j} T_{ij}(\xi, \mathrm{x})\, d\Gamma_j(\mathrm{x}) \;\; = \sum_{j=1}^{N} t_j(\mathrm{x}) \int_{\Gamma_j} U_{ij}(\xi, \mathrm{x})\, d\Gamma_j(\mathrm{x}) \tag{2.17}$$

where N is the number of boundary elements. If Eq. (2.17) is applied to all boundary points (i=1 to N), the following system of equations is obtained:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} [\mathbf{H}]_{3N\times 3N} \{\mathbf{u}\}_{3N\times 1} = \sum_{i=1}^{N} \sum_{j=1}^{N} [\mathbf{G}]_{3N\times 3N} \{\mathbf{t}\}_{3N\times 1} \tag{2.18}$$

where {**u**} and {**t**} are the vectors of boundary generalized displacements and tractions, respectively, and [**H**] and [**G**] are the well-known boundary element influence matrices. Reordering Eq. (2.18) for separating boundary unknowns from boundary known values, the following system of equations will be formed as:

$$[\mathbf{A}]_{3N\times 3N} \{\mathbf{x}\}_{3N\times 1} = \{\mathbf{B}\}_{3N\times 1} \tag{2.19}$$

where the vector {**x**} represents all boundary unknowns; generalized displacements or tractions, and the vector {**B**}contains all remaining prescribed boundary conditions. By solving Eq.(2.19), all boundary unknowns are obtained. The matrix [**A**] in Eq. (2.19) is dense and nonsymmetrical, so that when it is solved by any direct solver such as Gauss elimination or LU decomposition, the solution requires O(N^3) numerical operations.

2.3. Fundamental solutions in terms of potentials

The purpose of this section is to define the fundamental solutions $U_{ij}(\xi,\mathrm{x})$ and $T_{ij}(\xi,\mathrm{x})$ in terms of suitable potentials as pre request for the developed fast multi-pole expansion. Following Hörmander steps [60], the fundamental solution could be defined as follows:

$$L_{\ell i} U_{ij}(\xi,\mathrm{x}) = -\delta(\xi,\mathrm{x})\delta_{\ell j} \tag{2.20}$$

Where $\delta(\xi, x)$ is the Paul Dirac delta distribution, and $L_{\ell i}$ is given in Eq. (2.5). In order to solve Eq. (2.20), an operator decoupling scheme is carried out as follows. The co-factor matrix of the original differential operator in Eq. (2.5) can be obtained as follows:

$$L^*_{\ell i} = \begin{bmatrix} \frac{D^2(1-\nu)^2}{4}\lambda^2\left[\nabla^4\delta_{\alpha\beta} - \lambda^2\partial_\alpha\partial_\beta + \frac{(1+\nu)}{(1-\nu)}\nabla^2\left(\nabla^2\delta_{\alpha\beta} - \partial_\alpha\partial_\beta\right)\right] & \frac{D^2(1-\nu)^2}{4}\lambda^2\partial_\alpha\left(\nabla^2 - \lambda^2\right) \\ -\frac{D^2(1-\nu)^2}{4}\lambda^2\partial_\beta\left(\nabla^2 - \lambda^2\right) & \frac{D^2(1-\nu)}{4}\left[2\nabla^4 + (\nu-3)\lambda^2\nabla^2 + (1-\nu)\lambda^4\right] \end{bmatrix} \tag{2.21}$$

The corresponding determinant of the matrix operator in Eq. (2.21) can be obtained as follows:

$$\det\left|L^*_{\ell i}\right| = \frac{D^3(1-\nu)^2}{4}\lambda^2\nabla^4\left(\nabla^2 - \lambda^2\right) \tag{2.22}$$

A suitable potential $\Phi(\xi,x)$ has to be obtained first from the following differential equation:

$$\det\left|L^*_{\ell i}\right|\Phi(\xi,x) = -\delta(\xi,x) \tag{2.23}$$

i.e:

$$\frac{D^3(1-\nu)^2}{4}\lambda^2\nabla^4\left(\nabla^2 - \lambda^2\right)\Phi(\xi,x) = -\delta(\xi,x) \tag{2.24}$$

A possible solution of Eq. (2.24) yields the following potential:

$$\Phi(\xi,x) = \frac{2}{\pi D^3\lambda^6(1-\nu)^2}\left[K_0(\lambda R) + \ln(\lambda R) + \frac{1}{4}\lambda^2\left[R^2\ln(\lambda R)\right]\right] \tag{2.25}$$

where $K_0(\lambda R)$ is the modified Bessel functions [61].

Define:

$$\Phi(\xi,x) = \phi(R) \equiv \phi \tag{2.26}$$

so its relevant derivatives can be obtained as follows:

$$\Phi_{,\alpha} = \phi' R_{,\alpha} \tag{2.27}$$

$$\Phi_{,\alpha\beta} = \frac{\phi'}{R}\left(\delta_{\alpha\beta} - R_{,\alpha}R_{,\beta}\right) + \phi'' R_{,\alpha}R_{,\beta} \tag{2.28}$$

$$\nabla^2\Phi = \phi'' + \frac{\phi'}{R} \tag{2.29}$$

$$\left(\nabla^2\Phi\right)_{,\alpha} = \frac{R_{,\alpha}}{R^2}\left[-\phi' + \phi'' R + \phi''' R^2\right] \tag{2.30}$$

$$\left(\nabla^2\Phi\right)_{,\alpha\beta} = \phi^{(4)} R_{,\alpha} R_{,\beta} + \frac{\phi'''}{R}\delta_{\alpha\beta} + \left(\delta_{\alpha\beta} - 3R_{,\alpha}R_{,\beta}\right)\left(\frac{\phi''}{R^2} - \frac{\phi'}{R^3}\right) \tag{2.31}$$

$$\nabla^4\Phi = \phi^{(4)} + 2\frac{\phi'''}{R} - \frac{\phi''}{R^2} + \frac{\phi'}{R^3} \tag{2.32}$$

$$\text{where } (\bullet)' = \frac{\partial(\bullet)}{\partial R} \tag{2.33}$$

Then, the final expression for U_{ij} (ξ,x) can be obtained as follows:

$$U_{ij}(\xi,\mathrm{x}) = L^*_{\ell i}\delta_{\ell j}\Phi(\xi,\mathrm{x}) \tag{2.34}$$

Considering Eq. (2.21), Eq.(2.34) can be expanded to give:

$$U_{\alpha\beta}(\xi,\mathrm{x}) = \frac{D^2(1-\nu)^2}{4}\lambda^2\left[\nabla^4\delta_{\alpha\beta} - \lambda^2\partial_\alpha\partial_\beta + \frac{(1+\nu)}{(1-\nu)}\nabla^2\left(\nabla^2\delta_{\alpha\beta} - \partial_\alpha\partial_\beta\right)\right]\Phi(\xi,\mathrm{x}) \tag{2.35}$$

$$U_{\alpha 3}(\xi,\mathrm{x}) = -U_{3\alpha}(\xi,\mathrm{x}) = \frac{D^2(1-\nu)^2}{4}\lambda^2\partial_\alpha\left(\nabla^2 - \lambda^2\right)\Phi(\xi,\mathrm{x}) \tag{2.36}$$

$$U_{33}(\xi,\mathrm{x}) = \frac{D^2(1-\nu)}{4}\left[2\nabla^4 + (\nu-3)\lambda^2\nabla^2 + (1-\nu)\lambda^4\right]\Phi(\xi,\mathrm{x}) \tag{2.37}$$

Substituting from Eqs.(2.27) to (2.32) into Eqs.(2.35) to (2.37), the expressions for U_{ij}(ξ,x) can be obtained as follows:

$$\begin{aligned} U_{\alpha\beta}(\xi,\mathrm{x}) = \frac{1}{D}\Bigg\{ & -R_{,\alpha}R_{,\beta}\left[\lambda^2\left(\frac{\phi'}{R} + \phi''\right) + \frac{(1+\nu)}{(1-\nu)}\left(\phi^{(4)} - 3\frac{\phi''}{R^2} + 3\frac{\phi'}{R^3}\right)\right] \\ & + \delta_{\alpha\beta}\left[-\lambda^2\frac{\phi'}{R} + \frac{2}{(1-\nu)}\left(\phi^{(4)} - \frac{\phi''}{R^2} + 2\frac{\phi'''}{R} + \frac{\phi'}{R^3}\right) - \frac{(1+\nu)}{(1-\nu)}\left(\frac{\phi'''}{R} + \frac{\phi''}{R^2} - \frac{\phi'}{R^3}\right)\right]\Bigg\} \end{aligned} \tag{2.38}$$

$$U_{\alpha 3}(\xi,\mathrm{x}) = \frac{R_{,\alpha}}{D}\left[\phi''' + \frac{\phi''}{R} - \phi'\left(\frac{1}{R^2} + \lambda^2\right)\right] \tag{2.39}$$

$$\begin{aligned} U_{33}(\xi,\mathrm{x}) = \frac{1}{(1-\nu)D\lambda^2}\Bigg\{ & 2\phi^{(4)} + 4\frac{\phi'''}{R} + \phi''\left[-\frac{2}{R^2} + (\nu-3)\lambda^2\right] \\ & + \frac{\phi'}{R}\left[\frac{2}{R^2} + (\nu-3)\lambda^2\right] + (1-\nu)\lambda^4\phi\Bigg\} \end{aligned} \tag{2.40}$$

Eqs. (2.38) to (2.40) represent the final expressions for the generalized displacements kernels U_{ij}(ξ,x) in terms of the scalar potential ϕ and its relevant derivatives. Substitution of Φ(ξ,x) from Eq.(2.25) into Eqs.(2.38) to (2.40), the explicit form for the generalized displacement kernels are obtained as given in Eqs.(2.7) to (2.10).

The corresponding expressions for the generalized traction kernels $T_{ij}(\xi,\mathrm{x})$are also derived considering the stress-resultant generalized displacement relationships in Eqs. (2.2) and (2.3).Substituting from Eq.(2.38) into Eq. (2.2) gives:

$$M_{\gamma\alpha\beta}(\xi,\mathrm{x}) = D\frac{1-\nu}{2}\left(U_{\gamma\alpha,\beta}(\xi,\mathrm{x}) + U_{\gamma\beta,\alpha}(\xi,\mathrm{x}) + \frac{2\nu}{1-\nu}U_{\gamma\theta,\theta}(\xi,\mathrm{x})\delta_{\alpha\beta} \right) \tag{2.41}$$

Hence the corresponding traction kernel can be obtained from:

$$T_{\gamma\alpha}(\xi,\mathrm{x}) = M_{\gamma\alpha\beta}(\xi,\mathrm{x})\, n_\beta(\mathrm{x}) \tag{2.42}$$

where $n_\beta(\mathrm{x})$ is the component of the outward normal at the field point (x). The following relevant derivatives can be obtained by considering Eq.(2.38):

$$\begin{aligned} U_{\gamma\alpha,\beta}(\xi,\mathrm{x}) = \frac{1}{DR^4(\nu-1)}\Big\{ & \delta_{\gamma\beta}R_{,\alpha}\Big[\big(3(1+\nu)+\lambda^2R^2(1-\nu)\big)\phi' + \big(-3(1+\nu)+\lambda^2R^2(1-\nu)\big)R\phi''\Big] \\ & + (1-\nu)R^3\phi^{(4)} + \delta_{\gamma\alpha}R_{,\beta}\Big[\big(3(3+\nu)+\lambda^2R^2(\nu-1)\big)\phi' + \big(-3(3+\nu)+\lambda^2R^2(1-\nu)\big)R\phi'' + 6R^2\phi''' \\ & + (\nu-3)R^3\phi^{(4)} - 2R^4\phi^{(5)}\Big] + \delta_{\alpha\beta}R_{,\gamma}\Big[\big(3(1+\nu)+\lambda^2R^2(1-\nu)\big)\phi' + \big(-3(1+\nu)+\lambda^2R^2(1-\nu)\big)R\phi'' \\ & + (1+\nu)R^3\phi^{(4)}\Big] + R_{,\alpha}R_{,\beta}R_{,\gamma}\Big[\big(-15(1+\nu)+3\lambda^2R^2(\nu-1)\big)\phi' + \big(15(1+\nu)+\lambda^2R^2(\nu-1)\big)R\phi'' \\ & + \big(-3(1+\nu)+\lambda^2R^2(1-\nu)\big)R^2\phi''' - 2(1+\nu)R^3\phi^{(4)} + (1+\nu)R^4\phi^{(5)}\Big]\Big\} \end{aligned} \tag{2.43}$$

and:

$$U_{\gamma\theta,\theta}(\xi,\mathrm{x}) = \frac{R_{,\gamma}}{DR^4}\left[\left(\lambda^2R^2-3\right)\phi' + 3\left(1-\lambda^2R^2\right)R\phi'' - \left(\lambda^2R^2+3\right)R^2\phi''' + 2R^3\phi^{(4)} + R^4\phi^{(5)}\right] \tag{2.44}$$

Substituting Eqs.(2.43) and (2.44) into Eqs.(2.41) and (2.42), gives:

$$\begin{aligned} T_{\lambda\alpha}(\xi,\mathrm{x}) = \frac{1}{2R^3}\Big\{ & \left(n_\lambda R_{,\alpha} + \delta_{\lambda\alpha}R_{,\theta}n_\theta\right)\Big[\left(-6\phi'+6\phi''R\right)(2+\nu) + 2\phi''\lambda^2R^3(1-\nu) \\ & + n_\alpha R_{,\lambda}\Big[(1+2\nu)\left(-6\phi'+6\phi''R-2\phi''\lambda^2R^3\right) - 2\phi'\lambda^2R^2(1-2\nu) \\ & - 2\phi'''R^2\nu\left(3+\lambda^2R^2\right) + 2\phi^{(4)}R^3(\nu-1) + 2\nu R^4\phi^{(5)}\Big] \\ & + R_{,\alpha}R_{,\lambda}R_{,\theta}\Big[(1+\nu)\left(30\phi'-30\phi''R+6\phi'''R^2+4\phi^{(4)}R^3-2R^4\phi^{(5)}\right) \\ & + (1-\nu)\left(6\phi'\lambda^2R^2+2\phi''\lambda^2R^3-2\phi'''\lambda^2R^4\right)n_\theta\Big]\Big\} \end{aligned} \tag{2.45}$$

Also, substituting Eq. (2.39) into Eq. (2.2) gives:

$$M_{3\alpha\beta}(\xi,\mathrm{x}) = D\frac{1-\nu}{2}\left(U_{3\alpha,\beta}(\xi,\mathrm{x}) + U_{3\beta,\alpha}(\xi,\mathrm{x}) + \frac{2\nu}{1-\nu}U_{3\theta,\theta}(\xi,\mathrm{x})\,\delta_{\alpha\beta}\right) \tag{2.46}$$

The corresponding traction kernel can be obtained from:

$$T_{3\alpha}(\xi,\mathrm{x}) = M_{3\alpha\beta}(\xi,\mathrm{x})\,n_\beta(\mathrm{x}) \tag{2.47}$$

The following relevant derivatives can be obtained by considering Eq.(39):

$$U_{3\alpha,\beta}(\xi,\mathrm{x}) = \frac{-1}{D}\left\{\delta_{\alpha\beta}\left[\frac{\phi'''}{R} + \frac{\phi''}{R^2} - \frac{\phi'}{R}\left(\frac{1}{R^2} + \lambda^2\right)\right] + R_{,\alpha}R_{,\beta}\left[\phi^{(4)} + \left(\frac{3}{R^2} + \lambda^2\right)\times\left(\frac{\phi'}{R} - \phi''\right)\right]\right\} \tag{2.48}$$

and:

$$U_{3\theta,\theta}(\xi,\mathrm{x}) = \frac{-1}{D}\left\{\phi^{(4)} + 2\frac{\phi'''}{R} + \left(\frac{\phi'}{R} + \phi''\right)\times\left(\frac{1}{R^2} - \lambda^2\right)\right\} \tag{2.49}$$

Substituting Eqs.(2.48) and (2.49) into Eqs.(2.46) and (2.47), gives:

$$\begin{aligned} T_{3\alpha}(\xi,\mathrm{x}) = \frac{1}{2R^3}\Big\{ & n_\alpha\left[\phi'\left(2 - 4\nu + \lambda^2R^2(1+\nu) + \lambda^2R^2(1-\nu)\right) - 2R\phi''\left(1 - 2\nu - \nu\lambda^2R^2\right) - 2\phi'''R^2(1+\nu)\right. \\ & \left. - 2\nu R^3\phi^{(4)}\right] + R_{,\alpha}R_{,\beta}(\nu-1)\left[3\phi'\left(2 + \lambda^2R^2\right) - \phi''R\left(6 + 2\lambda^2R^2\right) + 2R^3\phi^{(4)}\right]n_\beta\Big\} \end{aligned} \tag{2.50}$$

For the generalized shear traction kernels, on the other hand, substituting from Eqs.(2.38) and (2.39) into Eq. (2.3) gives:

$$Q_{\gamma 3\alpha}(\xi,\mathrm{x}) = D\frac{1-\nu}{2}\lambda^2\left(U_{\gamma\alpha}(\xi,\mathrm{x}) + U_{\gamma 3,\alpha}(\xi,\mathrm{x})\right) \tag{2.51}$$

The corresponding traction kernel can be obtained from:

$$T_{\gamma 3}(\xi,\mathrm{x}) = Q_{\gamma 3\alpha}(\xi,\mathrm{x})\,n_\alpha(\mathrm{x}) \tag{2.52}$$

Knowing that (recall Eq.(2.39)):

$$U_{\gamma 3,\alpha}(\xi,\mathrm{x}) = -U_{3\gamma,\alpha}(\xi,\mathrm{x}) \tag{2.53}$$

And substituting Eqs.(2.38) and (2.48) into Eqs.(2.51) and (2.52), gives:

$$\begin{aligned} T_{\gamma 3}(\xi,\mathrm{x}) = \frac{\lambda^2}{R^3}\Big\{ & n_\gamma\left[\phi'\left(1 + \nu + \lambda^2R^2(\nu-1)\right) - \phi''R(1+\nu) + \phi'''R^2(2-\nu) + \phi^{(4)}R^3\right] \\ & + R_{,\gamma}R_{,\theta}n_\theta\left[\phi'(-3\nu) + \phi''R\left(3\nu + \lambda^2R^2(\nu-1)\right) - \phi^{(4)}R^3\nu\right]\Big\} \end{aligned} \tag{2.54}$$

Similar to Eq. (2.54), substituting from Eqs.(2.39) and (2.40) into Eq. (2.3) gives:

$$Q_{33\alpha}(\xi,\mathrm{x}) = D\frac{1-\nu}{2}\lambda^2\left(U_{3\alpha}(\xi,\mathrm{x}) + U_{33,\alpha}(\xi,\mathrm{x})\right) \tag{2.55}$$

and the corresponding traction kernel is given by:

$$T_{33}(\xi,\mathrm{x}) = Q_{33\alpha}(\xi,\mathrm{x})\, n_{\alpha}(\mathrm{x}) \tag{2.56}$$

The following relevant derivatives can be obtained by considering Eq. (2.40):

$$U_{33,\alpha}(\xi,\mathrm{x}) = \frac{R_{,\alpha}}{(1-\nu)D\lambda^2}\left\{2\phi^{(5)} + 4\frac{\phi^{(4)}}{R} + \phi'''\left[\frac{-6}{R^2} + (\nu-3)\lambda^2\right]\right.$$
$$\left. + \frac{\phi''}{R}\left[\frac{6}{R^2} + (\nu-3)\lambda^2\right] + \frac{\phi'}{R^2}\left[\frac{2}{R^2} - 2(\nu-3)\lambda^2\right]\right\} \tag{2.57}$$

And substituting Eqs.(2.39) and (2.57) into Eqs. (2.55) and (2.56), gives:

$$T_{33}(\xi,\mathrm{x}) = \frac{R_{,n}}{R^4}\left\{\phi^{(5)}R^4 + 3\phi^{(4)}R^3 + \phi'''R^2\left[-3 + \lambda^2R^2(\nu-2)\right]\right.$$
$$\left. + \phi''R\left[3 + \lambda^2R^2(\nu-2)\right] + \phi'\left[-3 - \lambda^2R^2(\nu-2) + \lambda^4R^4(1-\nu)\right]\right\} \tag{2.58}$$

Expressions (2.45), (2.50), (2.54) and (2.58) represent the final expressions for the generalized traction kernels $T_{ij}(\xi,\mathrm{x})$ in terms of the scalar potential ϕ and its relevant derivatives. Substitution of $\Phi(\xi,\mathrm{x})$ from Eq. (2.25) into these equations, the explicit forms for traction expressions are obtained as given in Eqs.(2.11) to (2.14).

2.4 Conclusions

In this chapter the shear-deformable plate bending theory was reviewed. The direct boundary integral equations for the Reissner's plate were discussed. Also, derivations of fundamental solution based on their potentials are made. In chapter 3 the fast multi-pole method for the Reissner's plate will be discussed and demonstrate how this will technique overcome the disadvantage of conventional BEM.

Chapter 3: The Proposed Fast Multi-pole Method (FMM)

3.1. Introduction

This chapter introduces the proposed first shift fast multi-pole (FMM) application to the direct boundary element solution of shear-deformable plate bending. The basic idea of the FMM is presented in section 3.2. Then, section 3.3 explains the Taylor series expansions of fundamental solutions and its contributions to the presented FMM. In Sections 3.4 and 3.5, the first shift of FMM and the concept of moments are introduced. The expansion modifications corresponding to successive Taylor series application are illustrated in section 3.6. Finally, section 3.7 presents the final form of the coefficient matrix after the application of FMM.

3.2. The fast multi-pole idea

In FMM technique, the boundary integration can be divided into two parts for each collocation (source) pointξ: The near-field and the far-field. The near-field part represents the integration of the elements that are close to the collocation point, which is computed by the same manner as that of the conventional BEM. Whereas, the far-field part provides the other remaining contribution to the overall value of integral that are corresponding to far boundary elements. This far-field contribution can be indirectly evaluated using equivalent summations via the FMM. It has to be noted that the far-field part represents the expensive part in conventional BEM calculations and it is also the part that spoils the sparsely of the influence matrices. Thus, the purpose of the FMM is to approximately and efficiently compute this expensive far-field part. Taylor series expansion is used here in to provide the ability to compute the far-field part as equivalent summations. Further details regarding FMM will be presented later in next sections.

3.3. Taylor series expansions

As can be seen from section 2.3, both generalized displacement and traction kernels can be represented as derivatives of the potential $\phi(\xi,x)$. In order to obtain the required multi-pole expansions for those kernels at far field zone, an expansion of $\phi(\xi,x)$ is carried out first using Taylor series. Consider the collocation point is ξ, the

field point is **x** and the point at which the series expansion is performed is $\mathbf{x_o}$. Assume that the distance $|\,x\text{-}x_o| << |\,\xi\text{-}x_o|$ (see Fig.(3.1)), then the expansion of $\phi(\xi,x)$ presented in Eq.(2.25) could be formed as follows:

$$\phi\,(\xi,\mathrm{x}) = \phi(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty} \frac{1}{S!} (\mathrm{x}_o - \mathrm{x})_{k_1} \cdots (\mathrm{x}_o - \mathrm{x})_{k_S} \times \phi_{,k_1 \cdots k_S}(\xi,\mathrm{x}_o) \tag{3.1}$$

Where $(\mathrm{x}_o - \mathrm{x})_{k_1}$ represent the k^{th} component of the vector $(\mathrm{x}_o - \mathrm{x})$, and $\phi_{,k_1 \cdots k_S} = \dfrac{\partial^S \phi}{\partial x_{k_1} \cdots \partial x_{k_S}}$.

Substituting Eq. (3.1) into Eq. (2.34), and carrying out the necessary derivations, the expansion form of the generalized displacement kernel $U_{ij}(\xi,x)$ can be obtained as follows:

$$U_{ij}\,(\xi,\mathrm{x}) = U_{ij}\,(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty} \frac{1}{S!} (\mathrm{x}_o - \mathrm{x})_{k_1} \cdots (\mathrm{x}_o - \mathrm{x})_{k_S} \times U_{ij,k_1 \cdots k_S}(\xi,\mathrm{x}_o) \tag{3.2}$$

As it will be discussed later in this thesis, only terms up to the second derivatives for the former expansion are enough to produce excellent accuracy. Therefore, the required derivatives (first and second) are developed as follows: Recall Eq. (2.25), and Eqs.(2.38) to (2.40), the first derivatives of the generalized displacement kernels at point x in Eq. (3.2) can be obtained as follows:

$$\begin{aligned} U_{\alpha\beta,i}(\xi,\mathrm{x}) = \frac{1}{4\pi D\lambda^2(1-\nu)R^3} \Big\{ & \left[4K_1(\lambda R)\lambda^3 R^3 + \left(16K_0(\lambda R) + 2(1-\nu)\right)\lambda^2 R^2 + 32K_1(\lambda R)\lambda R - 32\right] \\ & R_{,\alpha} R_{,\beta} R_{,i} - \left[\left(4K_0(\lambda R) + (1-\nu)\right)\lambda^2 R^2 + 8K_1(\lambda R)\lambda R - 8\right]\left(\delta_{\alpha\beta} R_{,i} + \delta_{i\beta} R_{,\alpha} + \delta_{i\alpha} R_{,\beta}\right) \\ & - \left[4K_1(\lambda R)\lambda^3 R^3\right]\delta_{\alpha\beta} R_{,i} \Big\} \end{aligned} \tag{3.3}$$

$$U_{\alpha 3,i}(\xi,\mathrm{x}) = -U_{3\alpha,i}(\xi,\mathrm{x}) = \frac{1}{8\pi D}\left[\left(2\ln(\lambda R) - 1\right)\delta_{i\alpha} + 2R_{,\alpha} R_{,i}\right] \tag{3.4}$$

$$U_{33,i}(\xi,\mathrm{x}) = \frac{1}{8\pi D(1-\nu)\lambda^2 R}\left[(1-\nu)\lambda^2 R^2\left(2\ln(\lambda R) - 1\right) - 8\right]R_{,i} \tag{3.5}$$

The second derivatives of the generalized displacement kernels can be obtained as follows:

$$U_{\alpha\beta,ij}(\xi,\mathrm{x}) = \frac{1}{4\pi D\lambda^2(1-\nu)R^4}\Big\{\Big[-4K_0(\lambda R)\lambda^4R^4 - 32K_1(\lambda R)\lambda^3R^3 - \big(96K_0(\lambda R)+8(1-\nu)\big)\lambda^2R^2 - 192K_1(\lambda R)\lambda R + 192\Big]R_{,\alpha}R_{,\beta}R_{,i}R_{,j} + \Big[\big(-4K_0(\lambda R)-(1-\nu)\big)\lambda^2R^2 - 8K_1(\lambda R)\lambda R + 8\Big]\big(\delta_{\alpha\beta}\delta_{ij}+\delta_{i\beta}\delta_{j\alpha}+\delta_{i\alpha}\delta_{j\beta}\big) - \Big[4K_1(\lambda R)\lambda^3R^3\Big]\big(\delta_{\alpha\beta}\delta_{ij}\big) + \Big[4K_1(\lambda R)\lambda^3R^3 + \big(16K_0(\lambda R)+2(1-\nu)\big)\lambda^2R^2 + 32K_1(\lambda R)\lambda R - 32\Big]\big(\delta_{\alpha\beta}R_{,i}R_{,j}+\delta_{i\beta}R_{,\alpha}R_{,j}+\delta_{i\alpha}R_{,\beta}R_{,j} \delta_{ij}R_{,\alpha}R_{,\beta}+\delta_{j\beta}R_{,\alpha}R_{,i}+\delta_{j\alpha}R_{,\beta}R_{,i}\big) + \Big[4K_0(\lambda R)\lambda^4R^4 + 8K_1(\lambda R)\lambda^3R^3\Big]\delta_{\alpha\beta}R_{,i}R_{,j}\Big\} \quad (3.6)$$

$$U_{\alpha 3,ij}(\xi,\mathrm{x}) = -U_{3\alpha,ij}(\xi,\mathrm{x}) = \frac{1}{4\pi DR}\Big[\delta_{ij}R_{,\alpha}+\delta_{i\alpha}R_{,j}+\delta_{j\alpha}R_{,i}-2R_{,\alpha}R_{,i}R_{,j}\Big] \quad (3.7)$$

$$U_{33,ij}(\xi,\mathrm{x}) = \frac{1}{8\pi D(1-\nu)\lambda^2R^2}\Big\{\Big[2(1-\nu)\lambda^2R^2+16\Big]R_{,i}R_{,j} - \Big[(1-\nu)\lambda^2R^2\big(1-2\ln(\lambda R)\big)+8\Big]\delta_{ij}\Big\} \quad (3.8)$$

Similar to expansions of generalized displacement kernels, the expansion forms of the moment and shear kernels can be represented as follows:

$$M_{\gamma\alpha\beta}(\xi,\mathrm{x}) = M_{\gamma\alpha\beta}(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty}\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\times M_{\gamma\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o) \quad (3.9)$$

$$M_{3\alpha\beta}(\xi,\mathrm{x}) = M_{3\alpha\beta}(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty}\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\times M_{3\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o) \quad (3.10)$$

$$Q_{\gamma 3\alpha}(\xi,\mathrm{x}) = Q_{\gamma 3\alpha}(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty}\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\times Q_{\gamma 3\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o) \quad (3.11)$$

$$Q_{33\alpha}(\xi,\mathrm{x}) = Q_{33\alpha}(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty}\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\times Q_{33\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o) \quad (3.12)$$

The required derivatives in Eqs.(3.9) to (3.12) for both moment and shear kernels up to the second derivatives are obtained as follows: Recall Eqs.(2.41), (2.46), (2.51), and (2.55), the first derivatives of moment and shear kernels at point x can be obtained as follows:

$$M_{\gamma\alpha\beta,i}(\xi,\mathrm{x}) = \frac{1}{4\pi\lambda^2R^4}\Big\{\Big[-4K_0(\lambda R)\lambda^4R^4 - 32K_1(\lambda R)\lambda^3R^3 - \big(96K_0(\lambda R)+8(1-\nu)\big)\lambda^2R^2 - 192K_1(\lambda R)\lambda R + 192\Big]R_{,\gamma}R_{,\alpha}R_{,\beta}R_{,i} + \Big[-\big(4K_0(\lambda R)-(1+\nu)\big)\lambda^2R^2 - 8K_1(\lambda R)\lambda R + 8\Big]\big(\delta_{\alpha\beta}\delta_{\gamma i}+\delta_{\gamma\alpha}\delta_{\beta i}+\delta_{\gamma\beta}\delta_{\alpha i}\big) + \Big[2\nu\lambda^2R^2 - 2K_1(\lambda R)\lambda^3R^3\Big]\big(\delta_{\gamma\alpha}\delta_{\beta i}+\delta_{\gamma\beta}\delta_{\alpha i}\big) + \Big[4K_1(\lambda R)\lambda^3R^3 + \big(16K_0(\lambda R)+2(1-\nu)\big)\lambda^2R^2 + 32K_1(\lambda R)\lambda R - 32\Big]\big(\delta_{\alpha\beta}R_{,\gamma}R_{,i}+\delta_{\gamma\alpha}R_{,\beta}R_{,i}+\delta_{\gamma\beta}R_{,\alpha}R_{,i} \delta_{\beta i}R_{,\gamma}R_{,\alpha}+\delta_{\alpha i}R_{,\gamma}R_{,\beta}+\delta_{\gamma i}R_{,\alpha}R_{,\beta}\big) + \Big[4K_1(\lambda R)\lambda^3R^3 + 2K_0(\lambda R)\lambda^4R^4\Big]\big(\delta_{\gamma\alpha}R_{,\beta}R_{,i}+\delta_{\gamma\beta}R_{,\alpha}R_{,i}\big) + 4\nu\lambda^2R^2\big(\delta_{\alpha\beta}R_{,\gamma}R_{,i}\big)\Big\} \quad (3.13)$$

$$M_{3\alpha\beta,i}(\xi,\mathrm{x})=\frac{1}{4\pi R}\left\{2(1-\nu)R_{,\alpha}R_{,\beta}R_{,i}-(1-\nu)\left(\delta_{i\beta}R_{,\alpha}+\delta_{i\alpha}R_{,\beta}\right)-(1+\nu)\left(\delta_{\alpha\beta}R_{,i}\right)\right\} \tag{3.14}$$

$$\begin{aligned}Q_{\gamma 3\alpha,i}(\xi,\mathrm{x})=\frac{-1}{2\pi R^3}\Big\{&\left[-K_1(\lambda R)\lambda^3R^3-4K_0(\lambda R)\lambda^2R^2-8K_1(\lambda R)\lambda R+8\right]R_{,\alpha}R_{,\gamma}R_{,i}\\&+\left[K_0(\lambda R)\lambda^2R^2+2K_1(\lambda R)\lambda R-2\right]\left(\delta_{\alpha\gamma}R_{,i}+\delta_{i\gamma}R_{,\alpha}+\delta_{i\alpha}R_{,\gamma}\right)+\left[K_1(\lambda R)\lambda^3R^3\right]\delta_{\alpha\gamma}R_{,i}\Big\}\end{aligned} \tag{3.15}$$

$$Q_{33\alpha,i}(\xi,\mathrm{x})=\frac{1}{2\pi R^2}\left(\delta_{i\alpha}-2R_{,\alpha}R_{,i}\right) \tag{3.16}$$

The second derivatives of moment and shear kernels can be obtained as follows:

$$\begin{aligned}M_{\gamma\alpha\beta,ij}(\xi,\mathrm{x})=\frac{-1}{4\pi\lambda^2R^5}\Big\{&\Big[-2K_1(\lambda R)\lambda^5R^5-24K_0(\lambda R)\lambda^4R^4-144K_1(\lambda R)\lambda^3R^3\\&-\left(384K_0(\lambda R)+24(1-\nu)\right)\lambda^2R^2-768K_1(\lambda R)\lambda R+768\Big]R_{,\gamma}R_{,\alpha}R_{,\beta}R_{,i}R_{,j}\\&+\Big[-2K_1(\lambda R)\lambda^3R^3-\left(8K_0(\lambda R)+(1-\nu)\right)\lambda^2R^2-16K_1(\lambda R)\lambda R+16\Big]\\&\left(\delta_{\gamma\alpha}\delta_{\beta i}R_{,j}+\delta_{\gamma\alpha}\delta_{\beta j}R_{,i}+\delta_{\gamma\beta}\delta_{\alpha i}R_{,j}+\delta_{\gamma\beta}\delta_{\alpha j}R_{,i}+\delta_{\alpha\beta}\delta_{\gamma i}R_{,j}+\delta_{\alpha\beta}\delta_{\gamma j}R_{,i}+\delta_{\gamma i}\delta_{\alpha j}R_{,\beta}\right.\\&\left.+\delta_{\gamma i}\delta_{\beta j}R_{,\alpha}+\delta_{\gamma j}\delta_{\alpha i}R_{,\beta}+\delta_{\gamma j}\delta_{\beta i}R_{,\alpha}+\delta_{ij}\delta_{\gamma\alpha}R_{,\beta}+\delta_{ij}\delta_{\gamma\beta}R_{,\alpha}\right)+\Big[-2K_1(\lambda R)\lambda^3R^3\\&-K_0(\lambda R)\lambda^4R^4\Big]\left(\delta_{\gamma\alpha}\delta_{\beta i}R_{,j}+\delta_{\gamma\alpha}\delta_{\beta j}R_{,i}+\delta_{\gamma\beta}\delta_{\alpha i}R_{,j}+\delta_{\gamma\beta}\delta_{\alpha j}R_{,i}+\delta_{ij}\delta_{\gamma\alpha}R_{,\beta}+\delta_{ij}\delta_{\gamma\beta}R_{,\alpha}\right)\\&-\left(2\nu\lambda^2R^2\right)\left(\delta_{\alpha\beta}\delta_{\gamma i}R_{,j}+\delta_{\alpha\beta}\delta_{\gamma j}R_{,i}\right)+\Big[2K_0(\lambda R)\lambda^4R^4+16K_1(\lambda R)\lambda^3R^3\\&+\left(48K_0(\lambda R)+4(1-\nu)\right)\lambda^2R^2+96K_1(\lambda R)\lambda R-96\Big]\left(\delta_{ij}R_{,\gamma}R_{,\alpha}R_{,\beta}+\delta_{\alpha i}R_{,\gamma}R_{,\beta}R_{,j}\right.\\&\left.+\delta_{\beta i}R_{,\gamma}R_{,\alpha}R_{,j}+\delta_{\alpha j}R_{,\gamma}R_{,\beta}R_{,i}+\delta_{\beta j}R_{,\gamma}R_{,\alpha}R_{,i}+\delta_{\alpha\beta}R_{,\gamma}R_{,i}R_{,j}\right)+8\nu\lambda^2R^2\left(\delta_{\alpha\beta}R_{,\gamma}R_{,i}R_{,j}\right)\Big\}\end{aligned} \tag{3.17}$$

$$\begin{aligned}M_{3\alpha\beta,ij}(\xi,\mathrm{x})=\frac{-1}{4\pi R^2}\Big\{&8(1-\nu)R_{,\alpha}R_{,\beta}R_{,i}R_{,j}-2(1-\nu)\left(\delta_{i\beta}R_{,\alpha}R_{,j}+\delta_{i\alpha}R_{,\beta}R_{,j}+\delta_{j\beta}R_{,\alpha}R_{,i}+\delta_{j\alpha}R_{,\beta}R_{,i}\right.\\&\left.+\delta_{ij}R_{,\alpha}R_{,\beta}\right)-2(1+\nu)\left(\delta_{\alpha\beta}R_{,i}R_{,j}\right)+(1+\nu)\left(\delta_{\alpha\beta}\delta_{ij}\right)+(1-\nu)\left(\delta_{i\beta}\delta_{j\alpha}+\delta_{j\beta}\delta_{i\alpha}\right)\Big\}\end{aligned} \tag{3.18}$$

$$\begin{aligned}Q_{\gamma 3\alpha,ij}(\xi,\mathrm{x})=\frac{-1}{2\pi R^4}\Big\{&\left[K_0(\lambda R)\lambda^4R^4+8K_1(\lambda R)\lambda^3R^3+24K_0(\lambda R)\lambda^2R^2+48K_1(\lambda R)\lambda R-48\right]\\&R_{,\alpha}R_{,\gamma}R_{,i}R_{,j}+\left[-K_1(\lambda R)\lambda^3R^3-4K_0(\lambda R)\lambda^2R^2-8K_1(\lambda R)\lambda R-8\right]\left(\delta_{j\alpha}R_{,\gamma}R_{,i}+\delta_{i\alpha}R_{,\gamma}R_{,j}\right.\\&\left.\delta_{\gamma\alpha}R_{,i}R_{,j}+\delta_{ij}R_{,\gamma}R_{,\alpha}+\delta_{j\gamma}R_{,\alpha}R_{,i}+\delta_{i\gamma}R_{,\alpha}R_{,j}\right)-\left[K_0(\lambda R)\lambda^4R^4+2K_1(\lambda R)\lambda^3R^3\right]\left(\delta_{\gamma\alpha}R_{,i}R_{,j}\right)\Big\}\end{aligned} \tag{3.19}$$

$$Q_{33\alpha,ij}(\xi,\mathrm{x})=\frac{1}{\pi R^3}\left(\delta_{ij}R_{,\alpha}+\delta_{i\alpha}R_{,j}+\delta_{j\alpha}R_{,i}-4R_{,\alpha}R_{,i}R_{,j}\right) \tag{3.20}$$

The expansions for traction kernels can be also developed using Eqs.(2.42), (2.47), (2.52) and (2.56), as follows:

$$T_{\gamma\alpha}(\xi,\mathrm{x})=\left[M_{\gamma\alpha\beta}(\xi,\mathrm{x}_o)+\sum_{S=1}^{\infty}\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\times M_{\gamma\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o)\right]n_\beta(\mathrm{x}) \tag{3.21}$$

$$T_{3\alpha}(\xi,\mathrm{x})=\left[M_{3\alpha\beta}(\xi,\mathrm{x}_o)+\sum_{S=1}^{\infty}\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\times M_{3\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o)\right]n_\beta(\mathrm{x}) \tag{3.22}$$

$$T_{\gamma 3}(\xi,\mathrm{x})=\left[Q_{\gamma 3\alpha}(\xi,\mathrm{x}_o)+\sum_{S=1}^{\infty}\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\times Q_{\gamma 3\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o)\right]n_\alpha(\mathrm{x}) \tag{3.23}$$

$$T_{33}(\xi,\mathrm{x})=\left[Q_{33\alpha}(\xi,\mathrm{x}_o)+\sum_{S=1}^{\infty}\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\times Q_{33\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o)\right]n_\alpha(\mathrm{x}) \tag{3.24}$$

3.4. Moments coefficients

For each collocation boundary point ξ in fast multi-pole procedure, the total boundary (Γ) is divided into two zones; the near-field boundary (Γ_{nf}), and the far-field boundary (Γ_{ff}). Using the expansions mentioned in the previous section, the kernels calculations in the far-field boundary are rapidly converged when only few terms of the expansions are used. It has to be noted that these expansions cannot give the same convergence for near-field elements; therefore, the direct calculations are then used. Thus, using the expansion form for generalized displacement kernel given in Eq. (3.2) for far-field boundaries, the integration of generalized displacement kernel given in Eq. (2.6) can be decomposed into the following terms:

$$\begin{aligned}\int_{\Gamma}U_{ij}(\xi,\mathrm{x})t_j(\mathrm{x})\,d\Gamma(\mathrm{x}) &= \int_{\Gamma_{\mathrm{nf}}}U_{ij}(\xi,\mathrm{x})t_j(\mathrm{x})\,d\Gamma_{\mathrm{nf}}(\mathrm{x})\\ &+\int_{\Gamma_{\mathrm{ff}}}\left(U_{ij}(\xi,\mathrm{x}_o)+\sum_{S=1}^{\infty}\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\times U_{ij,k_1\cdots k_S}(\xi,\mathrm{x}_o)\right)t_j(\mathrm{x})\,d\Gamma_{\mathrm{ff}}(\mathrm{x})\end{aligned} \tag{3.25}$$

Consider the far field boundary to have N_{far} number of constant elements and identifying the traction multi-pole moment coefficients for each element n of them as:

$$\mathrm{C}^n_{to}=\int_{\Gamma_n}t^n_j(\mathrm{x})\,d\Gamma_n(\mathrm{x}) \tag{3.26}$$

$$\mathrm{C}^n_{tk_1\cdots k_S}=\int_{\Gamma_n}\left(\frac{1}{S!}(\mathrm{x}_o-\mathrm{x})_{k_1}\cdots(\mathrm{x}_o-\mathrm{x})_{k_S}\right)t^n_j(\mathrm{x})\,d\Gamma_n(\mathrm{x}) \tag{3.27}$$

Thus, Eq.(3.25) can be rewritten in the following form:

$$\int_{\Gamma} U_{ij}(\xi,\mathrm{x}) t_j(\mathrm{x})\, d\Gamma(\mathrm{x}) = \int_{\Gamma_{\mathrm{nf}}} U_{ij}(\xi,\mathrm{x}) t_j(\mathrm{x})\, d\Gamma_{\mathrm{nf}}(\mathrm{x})$$

$$+ \sum_{n=1}^{N_{far}} \left(U_{ij}(\xi,\mathrm{x}_o) \times \mathrm{C}^n_{to} + \sum_{S=1}^{\infty} U_{ij,k_1\cdots k_S}(\xi,\mathrm{x}_o) \times \mathrm{C}^n_{tk_1\cdots k_S} \right) \tag{3.28}$$

Similarly, by using the expansion forms for moment and shear kernels obtained from Eqs.(3.9) to (3.12), the integration of traction kernels given in Eq.(2.6) can be decomposed into the following terms:

$$\int_{\Gamma} T_{ij}(\xi,\mathrm{x}) u_j(\mathrm{x})\, d\Gamma(\mathrm{x}) = \int_{\Gamma_{\mathrm{nf}}} T_{ij}(\xi,\mathrm{x}) u_j(\mathrm{x})\, d\Gamma_{\mathrm{nf}}(\mathrm{x})$$

$$+ \int_{\Gamma_{\mathrm{ff}}} \left(M_{\gamma\alpha\beta}(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty} \frac{1}{S!} (\mathrm{x}_o - \mathrm{x})_{k_1} \cdots (\mathrm{x}_o - \mathrm{x})_{k_S} \times M_{\gamma\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o) \right) n_\beta u_\alpha(\mathrm{x})\, d\Gamma_{\mathrm{ff}}(\mathrm{x})$$

$$+ \int_{\Gamma_{\mathrm{ff}}} \left(M_{3\alpha\beta}(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty} \frac{1}{S!} (\mathrm{x}_o - \mathrm{x})_{k_1} \cdots (\mathrm{x}_o - \mathrm{x})_{k_S} \times M_{3\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o) \right) n_\beta u_\alpha(\mathrm{x})\, d\Gamma_{\mathrm{ff}}(\mathrm{x})$$

$$+ \int_{\Gamma_{\mathrm{ff}}} \left(Q_{\gamma 3\alpha}(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty} \frac{1}{S!} (\mathrm{x}_o - \mathrm{x})_{k_1} \cdots (\mathrm{x}_o - \mathrm{x})_{k_S} \times Q_{\gamma 3\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o) \right) n_\alpha u_3(\mathrm{x})\, d\Gamma_{\mathrm{ff}}(\mathrm{x})$$

$$+ \int_{\Gamma_{\mathrm{ff}}} \left(Q_{33\alpha}(\xi,\mathrm{x}_o) + \sum_{S=1}^{\infty} \frac{1}{S!} (\mathrm{x}_o - \mathrm{x})_{k_1} \cdots (\mathrm{x}_o - \mathrm{x})_{k_S} \times Q_{33\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o) \right) n_\alpha u_3(\mathrm{x})\, d\Gamma_{\mathrm{ff}}(\mathrm{x})$$

(3.29)

For each element *n*, defining the following generalized displacement multi-pole moment coefficients:

$$\mathrm{C}^n_{uo} = \int_{\Gamma_n} n_\beta\, u^n_j(\mathrm{x})\, d\Gamma_n(\mathrm{x}) \tag{3.30}$$

$$\mathrm{C}^n_{uk_1\cdots k_S} = \int_{\Gamma_n} \left(\frac{1}{S!} (\mathrm{x}_o - \mathrm{x})_{k_1} \cdots (\mathrm{x}_o - \mathrm{x})_{k_S} \right) n_\beta\, u^n_j(\mathrm{x})\, d\Gamma_n(\mathrm{x}) \tag{3.31}$$

Hence, Eq. (3.29) can be rewritten in the following form:

$$\int_{\Gamma} T_{ij}(\xi,\mathrm{x}) u_j(\mathrm{x})\, d\Gamma(\mathrm{x}) = \int_{\Gamma_{\mathrm{nf}}} T_{ij}(\xi,\mathrm{x}) u_j(\mathrm{x})\, d\Gamma_{\mathrm{nf}}(\mathrm{x})$$

$$+\sum_{n=1}^{N_{far}}\left(M_{\gamma\alpha\beta}(\xi,\mathrm{x}_o)\times \mathrm{C}^n_{uo}+\sum_{S=1}^{\infty} M_{\gamma\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o)\times \mathrm{C}^n_{uk_1\cdots k_S}\right)$$

$$+\sum_{n=1}^{N_{far}}\left(M_{3\alpha\beta}(\xi,\mathrm{x}_o)\times \mathrm{C}^n_{uo}+\sum_{S=1}^{\infty} M_{3\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o)\times \mathrm{C}^n_{uk_1\cdots k_S}\right)$$

$$+\sum_{n=1}^{N_{far}}\left(Q_{\gamma 3\alpha}(\xi,\mathrm{x}_o)\times \mathrm{C}^n_{uo}+\sum_{S=1}^{\infty} Q_{\gamma 3\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o)\times \mathrm{C}^n_{uk_1\cdots k_S}\right)$$

$$+\sum_{n=1}^{N_{far}}\left(Q_{33\alpha}(\xi,\mathrm{x}_o)\times \mathrm{C}^n_{uo}+\sum_{S=1}^{\infty} Q_{33\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o)\times \mathrm{C}^n_{uk_1\cdots k_S}\right) \qquad (3.32)$$

As it can be seen from the expanded integral forms in Eqs.(3.28) and (3.32), the multi-pole moment coefficients are independent of the source point, ξ. As a result, these coefficients can be calculated only once for every iteration in the solution of the problem, in which the boundary unknowns are obtained from the earlier cycle of iterative solution. On the other hand, the evaluations of the near-field integrals are carried out using the conventional direct BEM scheme [62], i.e. as in Eq. (2.6) and (2.17).

In order to implement the FMM, a hierarchical tree of clusters (or cells) is needed to be defined. For each collocation point, the far-field elements can be partitioned into many cells, which belong to different levels. Then, the moment coefficients at each element, C, (see Fig.(3.1)) are evaluated with respect to the cell center on the lowest level (the so-called *leaf*). After that, a grouping of multi-pole moment coefficients occurs from the center of leaves (as at point x_o) progressively to the center of higher level cell (as at point x_p) which is known as moment to moment coefficients , O, as can be seen in Fig.(3.2). Then, a direct evaluation for these moment coefficients to the collocation point (local path L) is performed to complete the equivalent summations for far-field kernel expansions. The whole process is then called as "*first shift*" or Barnes and Huts' scheme [38]. In the following sections, the stages of moments grouping and moment to moment transferring will be discussed.

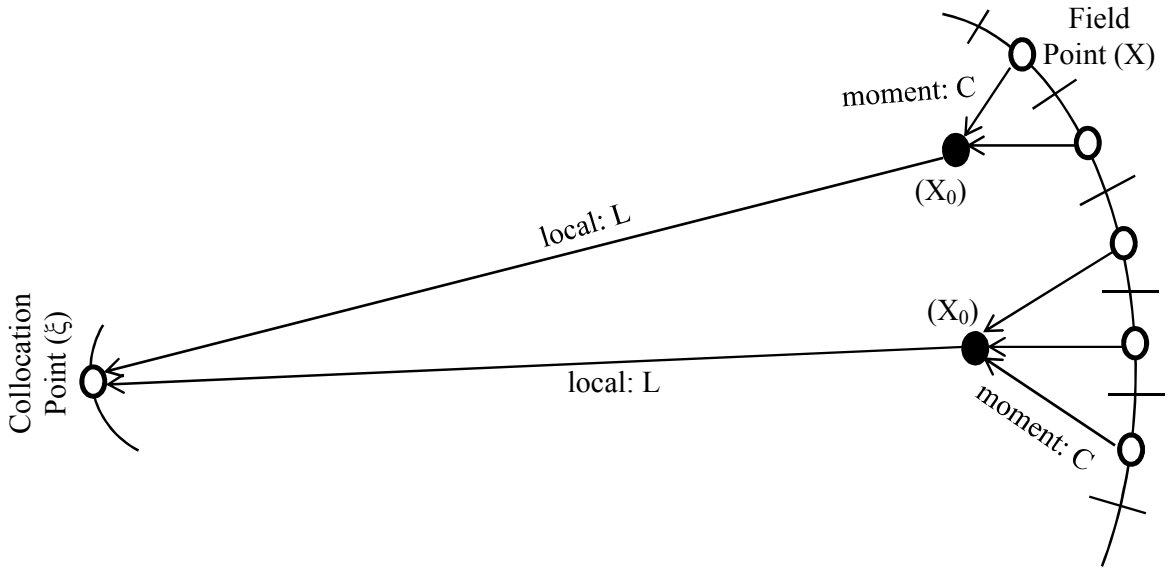

Fig. 3.1: Schematic diagram for the fast multi-pole far-field collocation.

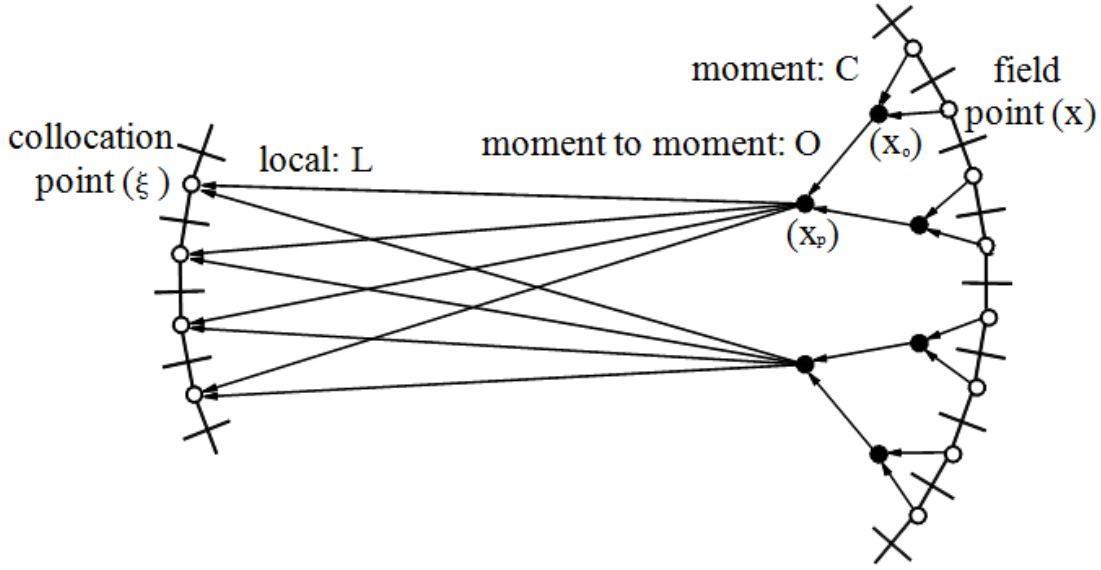

Fig. 3.2: Schematic diagram for the fast multi-pole far-field collocation.

3.5. Moments grouping

The expansion terms of the kernels (moments C) related to elements that are far enough from the collocation point are grouped in such a way, that the corresponding expansions can be carried out without losing accuracy [37]. The grouping starts from the leaves, where only elements are grouped at their lowest cell center as shown in Fig.(3.1). Consider the total number of elements in the leaves is N_l elements, and the total number of cells (leaves) in this level are L, then Eqs.(3.28), and (3.32) can be rewritten as follows:

$$\int_{\Gamma} U_{ij}(\xi,\mathrm{x}) t_j(\mathrm{x})\, d\Gamma(\mathrm{x}) = \int_{\Gamma_{\mathrm{nf}}} U_{ij}(\xi,\mathrm{x}) t_j(\mathrm{x})\, d\Gamma_{\mathrm{nf}}(\mathrm{x}) + \sum_{l=1}^{L} \left(U_{ij}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{to}^{m} + \sum_{S=1}^{\infty} U_{ij,k_1\cdots k_S}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{tk_1\cdots k_S}^{m} \right) \tag{3.33}$$

$$\begin{aligned} \int_{\Gamma} T_{ij}(\xi,\mathrm{x}) u_j(\mathrm{x})\, d\Gamma(\mathrm{x}) = & \int_{\Gamma_{\mathrm{nf}}} T_{ij}(\xi,\mathrm{x}) u_j(\mathrm{x})\, d\Gamma_{\mathrm{nf}}(\mathrm{x}) \\ & + \sum_{l=1}^{L} \left(M_{\gamma\alpha\beta}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{uo}^{m} + \sum_{S=1}^{\infty} M_{\gamma\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{uk_1\cdots k_S}^{m} \right) \\ & + \sum_{l=1}^{L} \left(M_{3\alpha\beta}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{uo}^{m} + \sum_{S=1}^{\infty} M_{3\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{uk_1\cdots k_S}^{m} \right) \\ & + \sum_{l=1}^{L} \left(Q_{\gamma 3\alpha}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{uo}^{m} + \sum_{S=1}^{\infty} Q_{\gamma 3\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{uk_1\cdots k_S}^{m} \right) \\ & + \sum_{l=1}^{L} \left(Q_{33\alpha}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{uo}^{m} + \sum_{S=1}^{\infty} Q_{33\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_o) \times \sum_{m=1}^{N_l} \mathrm{C}_{uk_1\cdots k_S}^{m} \right) \end{aligned} \tag{3.34}$$

Where m denotes the element number within the leaf cell, and l denotes the leaf number.

3.6. Moment to moment transferring

In order to minimize calculation operations, the fast multi-pole tree is designed so that the moment coefficients are obtained at higher levels. Therefore, for each collocation point, the far-field boundary elements which belong to leaves (their centers are at x_o) will then be transformed at the higher level cells (their centers are at

x_p) [37]. This process will be repeated for each two successive levels up to the highest level. Transformation process is obtained by applying extra Taylor expansions at cell centers of higher levels. Therefore, Eqs. (3.33, 3.34) will be modified to be written at higher level as follows:

$$\int_{\Gamma} U_{ij}(\xi,\mathrm{x}) t_j(\mathrm{x})\, d\Gamma(\mathrm{x}) = \int_{\Gamma_{\mathrm{nf}}} U_{ij}(\xi,\mathrm{x}) t_j(\mathrm{x})\, d\Gamma_{\mathrm{nf}}(\mathrm{x}) + \left(U_{ij}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{C}_{to}^{c} + \sum_{S=1}^{\infty} U_{ij,k_1\cdots k_S}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{O}_{tk_1\cdots k_S}^{c} \right) \tag{3.35}$$

$$\begin{aligned} \int_{\Gamma} T_{ij}(\xi,\mathrm{x}) u_j(\mathrm{x})\, d\Gamma(\mathrm{x}) = & \int_{\Gamma_{\mathrm{nf}}} T_{ij}(\xi,\mathrm{x}) u_j(\mathrm{x})\, d\Gamma_{\mathrm{nf}}(\mathrm{x}) \\ & + \left(M_{\gamma\alpha\beta}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{C}_{uo}^{c} + \sum_{S=1}^{\infty} M_{\gamma\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{O}_{uk_1\cdots k_S}^{c} \right) \\ & + \left(M_{3\alpha\beta}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{C}_{uo}^{c} + \sum_{S=1}^{\infty} M_{3\alpha\beta,k_1\cdots k_S}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{O}_{uk_1\cdots k_S}^{c} \right) \\ & + \left(Q_{\gamma 3\alpha}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{C}_{uo}^{c} + \sum_{S=1}^{\infty} Q_{\gamma 3\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{O}_{uk_1\cdots k_S}^{c} \right) \\ & + \left(Q_{33\alpha}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{C}_{uo}^{c} + \sum_{S=1}^{\infty} Q_{33\alpha,k_1\cdots k_S}(\xi,\mathrm{x}_p) \times \sum_{c=1}^{N_c} \mathrm{O}_{uk_1\cdots k_S}^{c} \right) \end{aligned} \tag{3.36}$$

where N_c is the number of cells at the higher levels that have the transferred moments from the lower level of N_l elements. The moments to moment transfer coefficients O_t^c and O_u^c at the higher level are given by [52]:

$$\mathrm{O}_{tk_1\cdots k_S}^{c} = \sum_{m=1}^{N_l} \left(\mathrm{C}_{tk_1\cdots k_S}^{m} + (\mathrm{x}_p - \mathrm{x}_{\mathrm{o}})_{k_S} \mathrm{C}_{tk_1\cdots k_{S-1}}^{m} + \cdots + \frac{(-1)^s}{S!} (\mathrm{x}_p - \mathrm{x}_0)_{k_1} \cdots (\mathrm{x}_p - \mathrm{x}_0)_{k_S} \mathrm{C}_{to}^{m} \right) \tag{3.37}$$

$$\mathrm{O}_{uk_1\cdots k_S}^{c} = \sum_{m=1}^{N_l} \left(\mathrm{C}_{uk_1\cdots k_S}^{m} + (\mathrm{x}_p - \mathrm{x}_{\mathrm{o}})_{k_S} \mathrm{C}_{uk_1\cdots k_{S-1}}^{m} + \cdots + \frac{(-1)^s}{S!} (\mathrm{x}_p - \mathrm{x}_0)_{k_1} \cdots (\mathrm{x}_p - \mathrm{x}_0)_{k_S} \mathrm{C}_{uo}^{m} \right) \tag{3.38}$$

It has to be noted that the infinite series in Eqs. (3.37) and (3.38) are truncated after a certain number of terms to obtain the desired accuracy. As will be proven through numerical examples in chapter (5), only up to three expanded terms (S=3) is enough to obtain an adequate accuracy for original expanded functions.

3.7. Final collocation and matrix form

As illustrated before in the present algorithm, the fundamental solution kernels for generalized displacements and tractions are calculated using two different parts: the near-field element integrations (via the conventional direct BEM), and the far-field element summations (via the FMM). If this process is repeated at every collocation point on the boundary (i.e. N times), the resulted algorithm will reduce the computational complexity from $O(N^3)$to be $O(N \log N)$, and the corresponding modified system of equations can be written as follows:

$$[\mathbf{H}]^{near}_{3N\times 3N}\ \{\mathbf{u}\}^{near}_{3N\times 1} + \{\mathbf{Hu}\}^{far}_{3N\times 1} = [\mathbf{G}]^{near}_{3N\times 3N}\ \{\mathbf{t}\}^{near}_{3N\times 1} + \{\mathbf{Gt}\}^{far}_{3N\times 1} \tag{3.39}$$

where $\{\mathbf{Hu}\}^{far}$ and $\{\mathbf{Gt}\}^{far}$ are vectors denote an implicit evaluation of the matrix-vector multiplication in the far-field regions; **[H]{u}**and **[G]{t}**respectively. Reordering Eq. (3.39) for separating boundary unknowns (appears only at near fields) from boundary known values added with other calculated vectors for far fields, the following system of equations will be formed as:

$$[\mathbf{A}]^{new}_{3N\times 3N}\ \{\mathbf{x}\}_{3N\times 1} = \{\mathbf{B}\}^{new}_{3N\times 1} \tag{3.40}$$

It has to be noticed that the matrix $[\mathbf{A}]^{new}$in Eq. (3.40) is sparse. This advantage facilitates the using of iterative solutions for the resulted system of matrices. Thus, Eq. (3.40) is solved using any suitable iterative solver such as the GMRES [53]. In the first iteration of solution, unknown boundary values in far-fields are set to zeros in order to evaluate initial values of the $\{\mathbf{B}\}^{new}$ vector. After solving the first iteration, the values of **{x}**will be used to describe the values of boundary unknowns at far-fields regions in the second iteration. The procedure is repeated until results of **{x}** converges and reaches a prescribed tolerance.

3.8 Conclusions

In this chapter, the fast multi-pole expansion is applied to the boundary element method for shear deformable plate bending problems, as the conventional BEM is not efficient in solving large-scale problems containing large number of degrees of freedoms. In such problems, the fast multi-pole method when accompanied with iterative solvers (GMRES) has succeeded to substantially decrease the computational time to be O(N log N) instead of O(N^2). In next chapter, the FMM technique is implemented into compute code to solve numerical problems.

Chapter 4: Programing for the proposed FMM

4.1. Introduction

In this chapter, the main structure of the proposed FMM code for solving plate bending problems is introduced. This code is written in FORTRAN language and is provided in Appendix (A).In this program; constant elements are employed to approximate the line integrals. This FMM code for general plate bending problems can be used as a basis to develop FMM programs 2nd shift and for using higher order elements. Section 4.2, represents the program's sequence and the main subroutines in flowchart. In section 4.3, the purpose of each subroutine is explained.

4.2. Work flow process

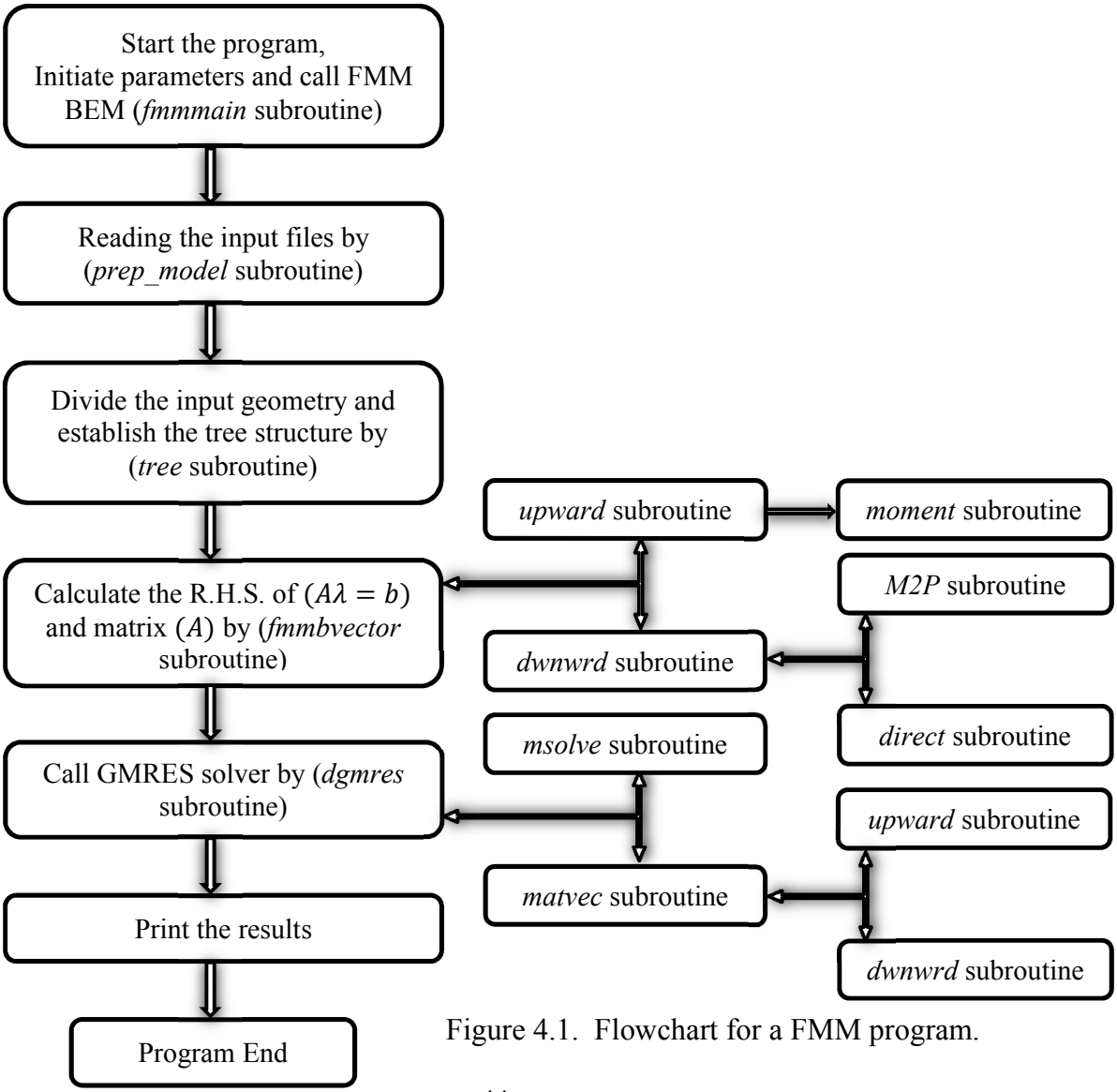

Figure 4.1. Flowchart for a FMM program.

The flowchart of this fast multi-pole BEM code is given in fig.(4.1). The chart shows the main tasks and sequences for the program and the related subroutines. The program for the fast multi-pole BEM is more complicated than the program of conventional BEM because of the tree structure of the cells and various expansions. With the restrictions of the GMRES solver, a large array is needed to be developed in the program to pass the variables to the GMRES solver. A few important subroutines in the program are discussed in the following subsections.

4.2.1 Subroutine *fmmmain*

The fmmmain subroutine starts with calling prep_model, which reads in the data for the boundary nodes, elements, boundary conditions, and field (interior) points from file input.dat(a sample file is given in Appendix (A.1)) and the additional parameters used in the fast multi-pole expansions and solver GMRES from file input.fmm (a sample file is given in Appendix(A.2)). It then generate the tree structure, computes the right hand side {b} vector, solves the system of equation ($A\lambda = b$) using the GMRES solver, computes values at interior points, and finally print the results.

4.2.2 Subroutine *tree*

This subroutine is an essential piece of the entire code. By calling the subroutine tree, the quad tree structure for the elements is created. The information of the tree structure is stored in several arrays in the code. To understand how this subroutine is used to create the tree structure, refer to the BEM model shown in fig.(4.2).

Cells in the tree structure are numbered in the following way: the largest cell at level 0 is called cell 1, the four cells at level 1 are numbered 2,3,4 and 5 respectively, according to order 0,1,2,3 as shown in the side box in fig.(4.2). This operation is continuing in this way to reach all the leaves (leaf is the cell which contains the maximum number of nodes per cell and the maximum number of nodes per cell is user input). This subroutine is automatically ignore all cells have no elements. The model shown in fig.(4.2) consists of 30 nodes and each node refers to one constant element and the numbering of cells according to tree structure can show in fig.(4.3).

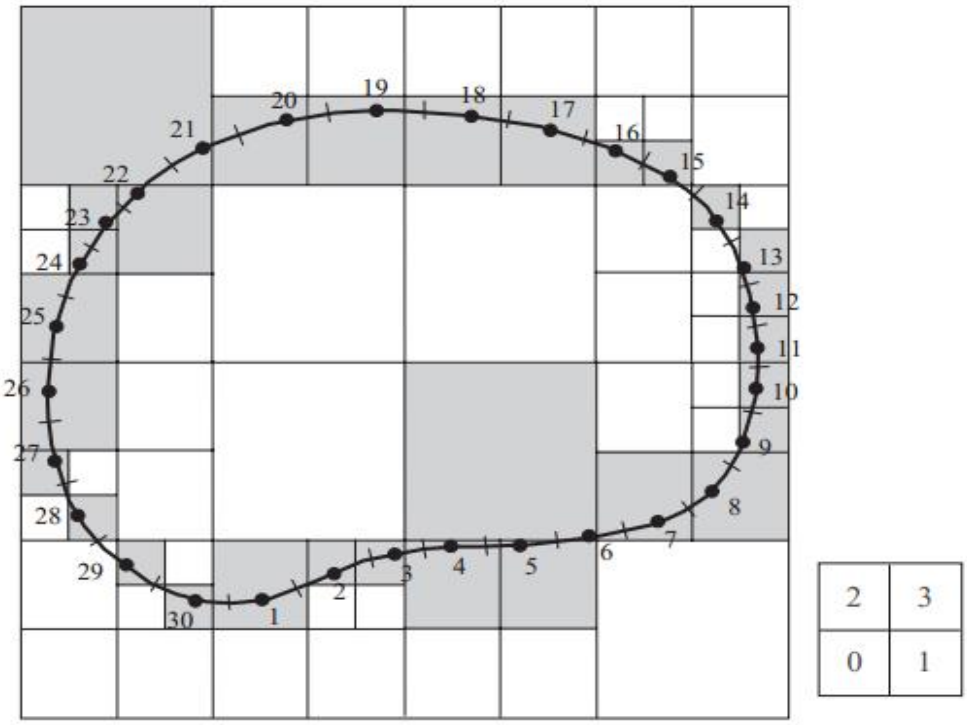

Figure 4.2 A cell structure covering all the boundary elements.
Taken from Liu[37].

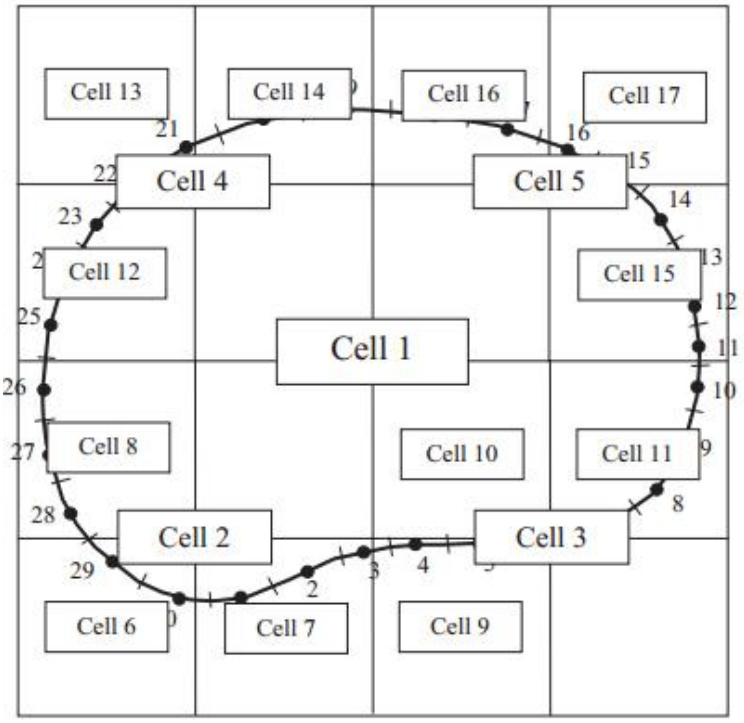

Figure 4.3 Cell number for level 0, 1 and 2 for the model.
Taken from Liu[37].

The tree code sort the coordinates of each node in different arrays, the center of each cell and the relation between cells like which cell is the leaf and which is the parent of it and which is called neighbor of it. This relation between cells can be cleared in the fig. (4.4).All arrays which filled by all this information are sort in a large array and this operation is a step for preparing the information array to the iterative solver GMRES.

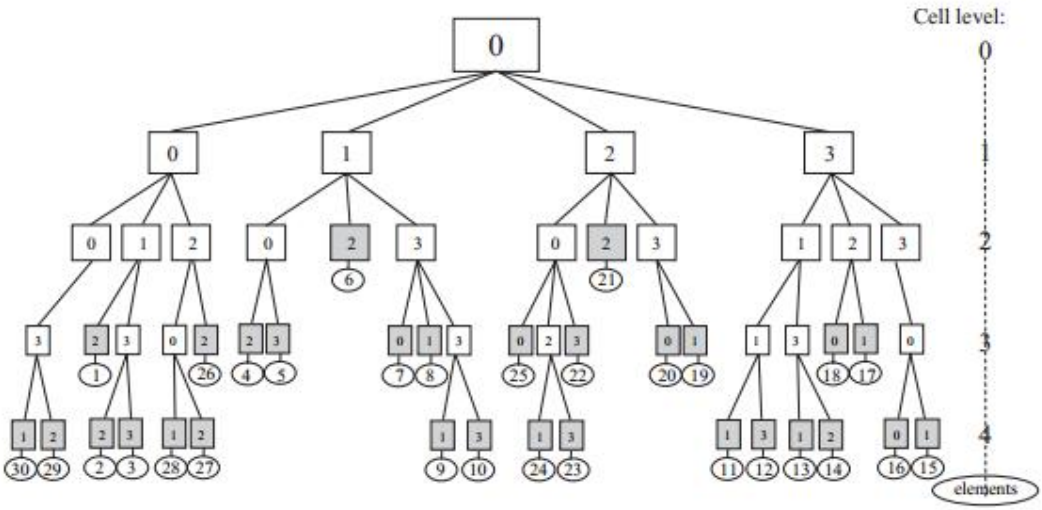

Figure 4.4 The relation between cells in the tree structure.
Taken from Liu[37].

The tree subroutine is main subroutine which addresses the input data to use in the FMM algorism.

4.2.3 Subroutine *fmmbvector*

The main purpose of *fmmbvector* subroutine is to compute the right hand side {**B**} vector by using the fast multi-pole algorism. The right hand side is computed by calling the *upward* subroutine and the *dwnwrd* subroutine. The calculation of the {**B**} vector requires calculating the conventional BEM coefficient by using the FMM algorism. By using the novel algorism in computing the right hand side can save the CPU time and minimize the complexity of code from O(N^2) in the conventional BEM to O(N log N) when using the first shift in FMM algorism and O(N) when using the

second shift in FMM algorism. But with make the complexity of code lower, the use of iterative solver (GMRES) is mandatory.

4.2.4 Subroutine *dgmres*

The *dgmres* subroutine is the GMRES solver in the SLATEC package from www.netlib.org. It is not required to understand the inner workings of this GMRES iterative solver to apply this subroutine. To use this GMRES solver, only two subroutines are needed to be prepared: *msolve* and *matvec*, which are two external subroutines for *dgmres* [37].

4.2.5 Subroutine *msolve*

This subroutine is called by *dgmres* subroutine to prepare the preconditioning matrix that will be used by GMRES solver. The preconditioning matrix is calculated once in the first iteration and the diagonal block matrix stored in the *rwork* array and the related information like size and the location of diagonal block is stored in the *iwork* array.

4.2.6 Subroutine *matvec*

The *matvec* subroutine is providing the algorism for the matrix vector multiplication using the fast multi-pole algorisms by simply calling the *upward* and *dwnwrd* subroutines using the values for the solution vector from the previous iteration [37].

4.2.7 Subroutine *upward*

The *upward* subroutine is calculating the moment expansion in each leaf (the cell which has the maximum number of node defined by user) in the tree structure by calling *moment* subroutine and grouping all these moments by moment to moment expansion and climbing to the parent cell in the higher level and so on to reach level 2 in the tree and store these calculations in **[a]** matrix.

4.2.8 Subroutine *dwnwrd*

The *dwnwrd* subroutine is calculating the coefficient of [**G**] and [**H**] kernels. These calculations achieved by two ways in this subroutine depend on the position of cells.

For near cells, calculation achieved typically as the conventional BEM by calling *direct* subroutine. For far cells, the coefficient calculated by using the FMM expansions by calling *M2P* subroutine when using the first shift which use the values in **[a]** matrix in M2P expansion to get the coefficient which stored in **[b]** matrix.

4.3 Conclusions

In this chapter, the FORTRAN code of the fast multi-pole method was discussed. The operation scheme of the main program was illustrated. Also, the input and output data of each subroutine were explained. The FORTRAN code, its input files and output files are presented in Appendix (A). In the next chapter, the developed program is investigated through some numerical examples in order to test the validity of the proposed FMM against analytical solutions, and conventional BEM solutions.

Chapter 5: Numerical Examples

5.1 Introduction

In this chapter, two different examples with different boundary conditions are selected to demonstrate the accuracy and efficiency of the proposed fast multi-pole formulation. As mentioned before, Taylor expansions using any number of terms could be used, however numerical results proved that up to three terms are enough to obtain results with excellent accuracy. In all problems, the platform for obtaining the present results is a 2.0-GHz Intel® Pentium® Core2Due with 2 GB RAM.

5.2 Cantilever plate

The (20m × 5m) rectangular cantilever plate shown in Fig.(5.1) is considered. The cantilever has thickness of 0.3 m and is fixed along one of its short sides. It is loaded by edge loading of intensity 1.5 t/m as shown in Fig.(5.1). The used material properties are: The Young's modulus $E = 3.0\times10^7$ t/m^2, and the Poisson's ratio ν is set to zero to allow comparison against the analytical solutions of the beam theory. The problem boundary is discretized into many meshes vary from 100 to 1000 constant elements. The results for the generalized displacements at point (A) are evaluated using analytical solution and also obtained from the conventional direct boundary element method as well as obtained from the proposed FMM. The comparison between all results for rotation and deflection are presented in Table (5.1).From Table (5.1), it can be seen that the rotation and deflections values solved by the proposed FMM using 2 or 3 terms are in excellent agreement with those obtained from the conventional BEM. Both results agree with the analytical solutions for all studied cases which illustrate the accuracy of the proposed FMM technique.

In order to demonstrate the strength of the proposed FMM, a comparison is carried out between the conventional BEM and the fast multi-pole BEM using 2 and 3 terms. In this comparison, the CPU time needed to solve problems having different boundary element discretization is computed. The comparison is plotted in Fig.(5.2). As it can be seen from the Fig.(5.2), a substantial saving of elapsed time is achieved

when the problem is solved by the proposed FMM, whether the selected terms are 2 or 3. As the number of elements of the problem increases, the proposed FMM is much more efficient compared to the conventional BEM, which enables the boundary element method to be used in solving large-scale practical plate bending problems.

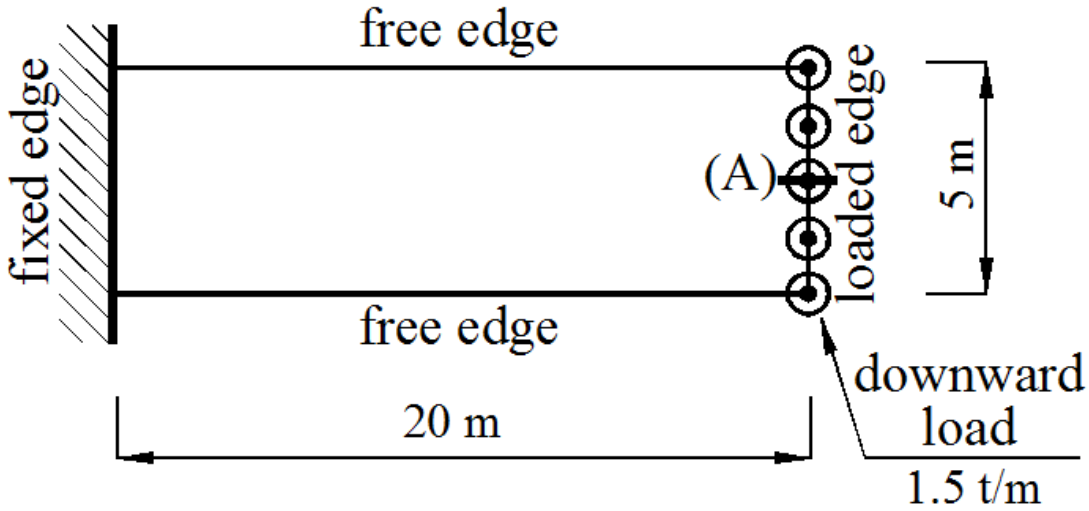

Fig. 5.1: The considered cantilever plate.

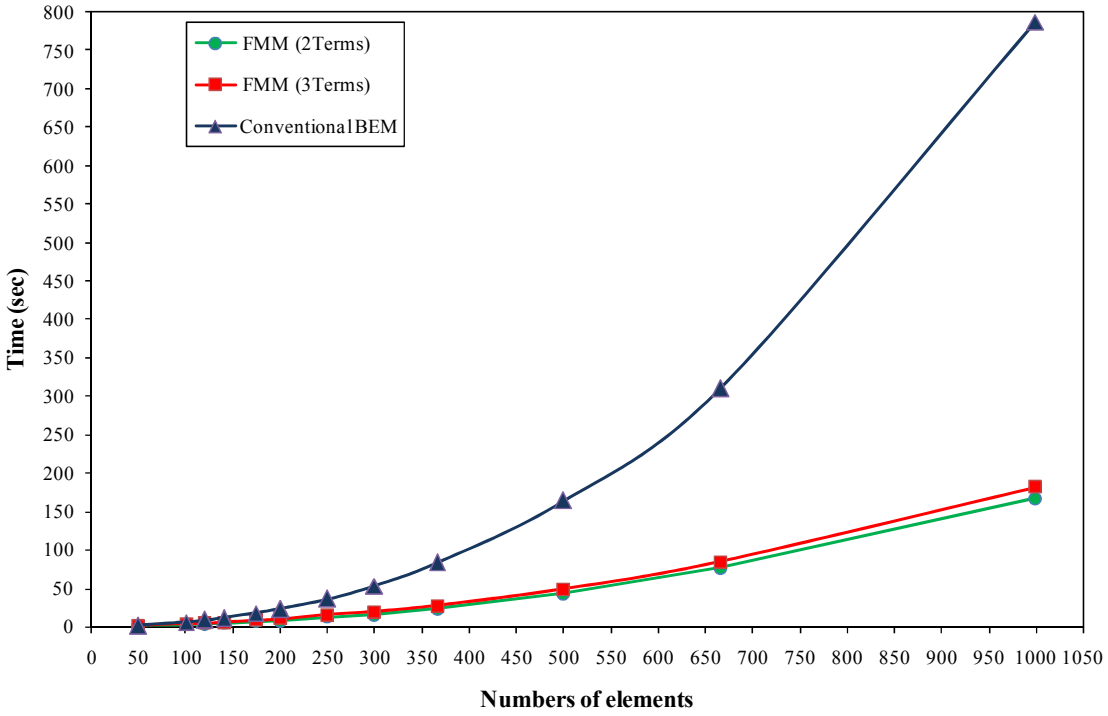

Fig. 5.2: Comparison of CPU time between the proposed FMM and the conventional BEM for the cantilever plate.

Table (5. 1): Results of rotation and deflection of point (A) in the cantilever plate.

Numbers of elements	Mesh	Analytical solution (rotation)	Proposed FMM (rotation)		Conventional BEM (rotation)	Analytical solution (deflection)	Proposed FMM (deflection)		Conventional BEM (deflection)
			2-terms	3-terms			2-terms	3-terms	
50	20x5	0.004444	0.004244	0.004358	0.004414	-0.059259	-0.060112	-0.059364	-0.059837
100	40x10		0.004337	0.004448	0.004489		-0.060460	-0.059713	-0.060066
120	50x10		0.004334	0.004444	0.004492		-0.060265	-0.059559	-0.059969
140	50x20		0.004344	0.004454	0.004502		-0.060395	-0.059694	-0.060106
174	62x25		0.004386	0.004495	0.004537		-0.060766	-0.060070	-0.060431
200	80x20		0.004338	0.004445	0.004491		-0.060153	-0.059443	-0.059837
250	100x25		0.004328	0.004437	0.004484		-0.060021	-0.059330	-0.059738
300	100x50		0.004334	0.004443	0.004491		-0.060078	-0.059395	-0.059804
366	133x50		0.004338	0.004444	0.004491		-0.060089	-0.059382	-0.059780
500	200x50		0.004345	0.004421	0.004465		-0.059777	-0.059117	-0.059497
666	267x66		0.004317	0.004424	0.004469		-0.059852	-0.059148	-0.059535
1000	400x100		0.004303	0.004381	0.004429		-0.059701	-0.058957	-0.059349

5.3 Slab with circular opening

The square slab of side length 6m shown in Fig.(5.3) is considered in this example. The slab is fixed along two opposite sides whereas the other two sides are free and loaded with line load of 10 t/m and line moments of 30 t.m/m. The slab is also having a circular opening on which a line load of 5 t/m is applied as shown in Fig.(5.3). The slab thickness is 0.35 m and its material properties are: $E = 2.5\times10^6$ t/m^2, $\nu = 0.2$. Only one quarter of the slab is solved due to problem symmetry. The problem boundary is discretized into several number of boundary elements vary from 100 to 1000 elements.

Table (5.2) and Table (5.3) demonstrate a comparison for generalized displacements results (rotations about two directions and deflection) at points (A) and (B) respectively. The values obtained based on the proposed FMM are compared against those of conventional BEM. As it can be seen from the results, the proposed FMM achieved accurate evaluation of boundary values against conventional numerical method when expanding the kernel series with only three terms. On the other side, Fig.(5.3) demonstrates the elapsed CPU times for both conventional BEM and the proposed FMM. The figure illustrate that the time needed to solve the problem using the proposed FMM with 2 or 3 terms are extremely small when compared with the time consumed using the conventional BEM. This difference can be easily observed for large problems with large number of boundary elements which appear in case of practical applications.

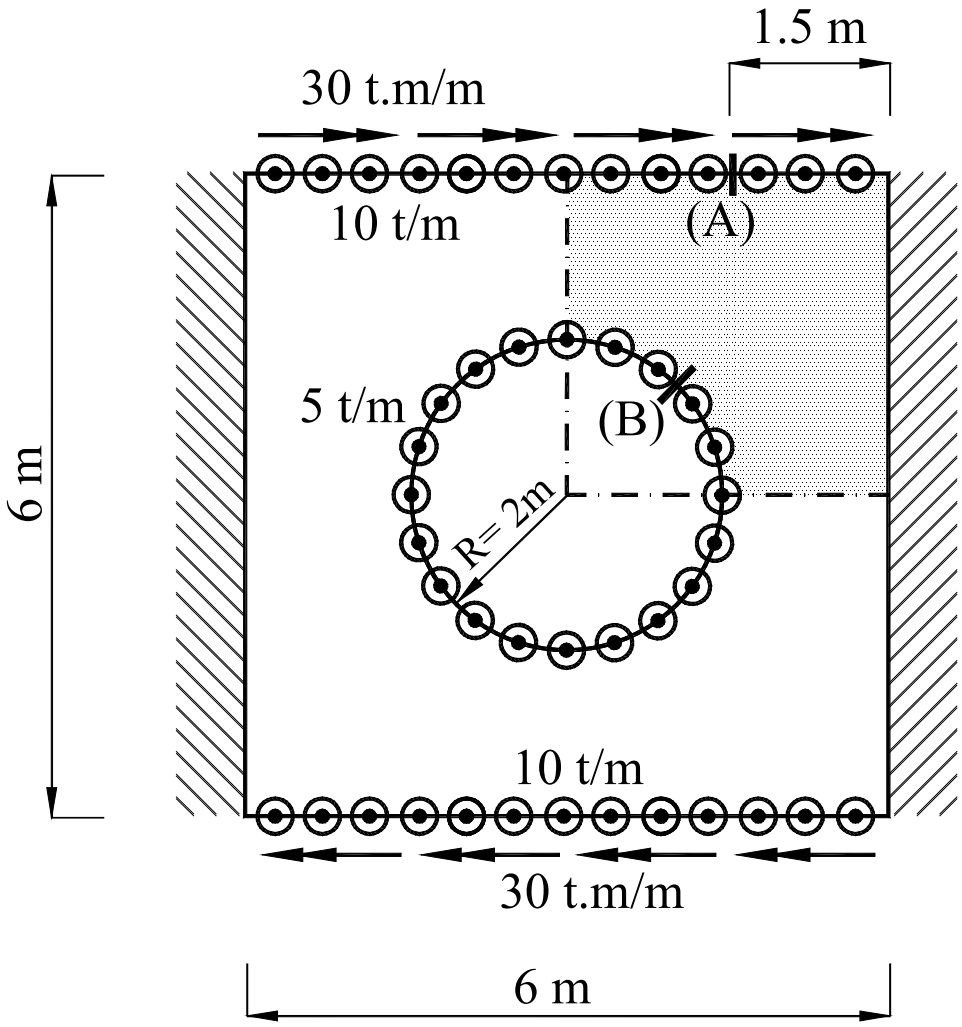

Fig. 5.3: The considered squared slab with circular opening.

Table (5. 2): Generalized displacements of point (A) in the squared slab.

Number of elements	Proposed FMM (U1)		Conventional BEM (U1)	Proposed FMM (U2)		Conventional BEM (U2)	Proposed FMM (U3)		Conventional BEM (U3)
	2-terms	3-terms		2-terms	3-terms		2-terms	3-terms	
86	-0.00003961	-0.00003154	-0.00003162	-0.00001733	0.00008584	0.00009074	-0.00044045	-0.00043475	-0.00043585
218	-0.00004200	-0.00003544	-0.00003451	-0.00001798	0.00004310	0.00008901	-0.00043752	-0.00041660	-0.00043448
490	0.00000689	-0.00002275	-0.00002380	0.00003331	0.00008437	0.00008710	-0.00045897	-0.00044716	-0.00044749
590	0.00001164	-0.00002233	-0.00002336	0.00004306	0.00008469	0.00008692	-0.00045934	-0.00044770	-0.00044794
818	0.00002021	-0.00002141	-0.00002336	0.00005820	0.00008470	0.00008692	-0.00046043	-0.00044770	-0.00044794
980	0.00001998	-0.00002163	-0.00002272	0.00005846	0.00008486	0.00008666	0.00005846	-0.00044853	-0.00044863

Table (5. 3): Generalized displacements of point (B) in the squared slab.

Number of elements	Proposed FMM (U1) ×10-5 rad		Conventional BEM (U1)	Proposed FMM (U2) ×10-3 rad		Conventional BEM (U2)	Proposed FMM (U3)		Conventional BEM (U3)
	2-terms	3-terms		2-terms	3-terms		2-terms	3-terms	
86	-3.96074600	-3.15374270	-3.16187550	-1.73349710	8.58433060	9.07424320	-44.04462800	-43.47479000	-43.58481500
218	-4.19958380	-3.54390100	-3.45054030	-1.79759840	4.30957740	8.90142320	-43.75170900	-41.66021200	-43.44762000
490	0.68921429	-2.27532660	-2.38010790	3.33125430	8.43742790	8.70958820	-45.89665900	-44.71621500	-44.74934500
590	1.16381970	-2.23265740	-2.33645760	4.30606170	8.46881390	8.69158620	-45.93357100	-44.77021100	-44.79433100
818	2.02115950	-2.14077300	-2.33645760	5.81962910	8.47017120	8.69158620	-46.04255200	-44.77021100	-44.79433100
980	1.99768990	-2.16299760	-2.27172010	5.84576820	8.48626480	8.66572580	5.84576820	-44.85285400	-44.86254500

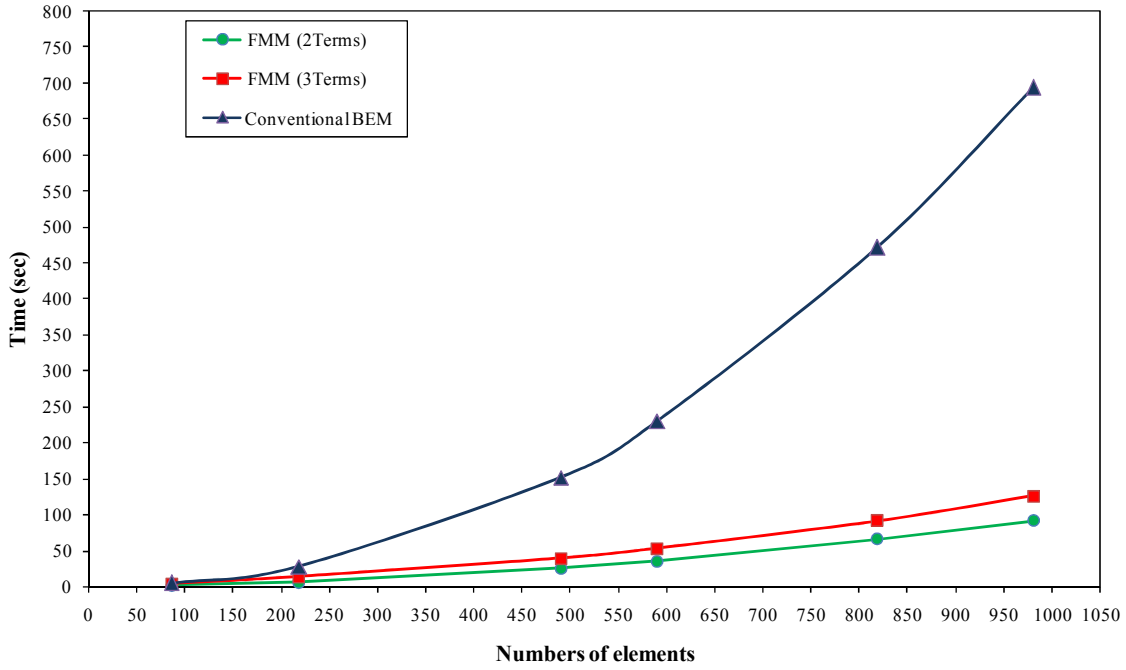

Fig. 5.4: Comparison of CPU time between the proposed FMM and the conventional BEM for the squared slab.

5.4 Conclusions

This chapter discussed, in a practical way, the benefits of applying FMM technique to conventional BEM equations. Results in tables prove that the solution accuracy was not affected by introduction of FMM. Charts prove that FMM technique leads to faster computation than conventional BEM.

Chapter 6: Summary and Conclusions

6.1 Summary

In this thesis, the fast multi-pole expansion technique was applied to the boundary element method for shear deformable plate bending problems. As the conventional method is suitable for medium problems with simple geometry but it is not efficient in solving large-scale problems containing large number of degrees of freedoms. In such problems, the fast multi-pole method when accompanied with iterative solvers (GMRES) had succeeded to substantially decrease the computational time. To validate the proposed formulation, two numerical examples with different boundary conditions were presented. A comparison of the computational time and results accuracy of the proposed FMM against analytical and conventional BEM solutions were carried out. As illustrated in these numerical tests, only few expansion terms (three terms) were needed to obtain results with high accuracy.

6.2 Conclusions

The main idea of the fast multi-pole BEM is to replace the element to element interactions, which are costly to compute, with cell to cell interactions through the introduction of the fast multi-pole expansions of the fundamental solution kernels. Thus, in the proposed application of FMM technique the number of collocation operations is reduced. In addition, when the fast multi-pole technique is applied to the conventional BEM, the coefficient matrix is changed from fully populated matrix to sparse matrix with band width related to the number of nodes per leaf cell. This change enables to minimize the complexity and also saving the CPU time and memory. Application of FMM technique to direct boundary element solution of plates proved to preserve the boundary elements accuracy with improved solution time. This application opens the way for application of parallel programming to BEM.

6.3 Future work

The proposed method is promising -when extended- to open the way for solution of large-scale practical problems. In order to achieve more in this research trend, a future work considering the fast multi-pole local expansions -which is known as "*second shift*"- besides upgrading the boundary elements to be quadratic and add the contribution of domain loads and internal stiffness conditions are suggested to be investigated. Also, the representation of the problem with quadratic elements instead of constant elements will be more accurate and can be one of the future work points. The precondition sparse matrix give a chance to use multi thread programing or GPU programing to save more time and print the result of large scale problem in no time. The fast multi-pole algorism can apply in any method using mesh to simulate its problem like finite element method.

REFERENCES

[1] *Bathe K.J., (1982), "Finite Element Procedure in Engineering Analysis", Englewood Cliffs, NJ, Prentice-Hall.*

[2] *SAP2000 V10, 2004, Structural Analysis Program Software, Integrated Finite Element Analysis and Design of Structures, Computer and Structures, Inc., University Aveneu, Berkeley, California.*

[3] *Rizzo, F., Some integral equation methods for plane problems of classical elastostatics.1964, University of Illinois, Urbana-Champaign. p. 34.*

[4] *Rizzo, F., An integral equation approach to boundary value problems of classical elastostatics. Q. Appl. Math. , Vol. 25 :pp.83-95, 1967*

[5] *Bush, M.B. and R.I. Tanner, Numerical solution of viscous flows using integral equation methods. Int. J. Num. Methods in Fluids, Vol. 3: pp.71–92, 1983*

[6] *Coleman, C.J., On the use of boundary integral methods in the analysis of non-Newtonian fluid flow. J. Non-Newt. Fluid Mech., Vol. 16: pp347–355, 1984.*

[7] *Chen, L. and D. Schweikert, Sound radiation from an arbitrary body. J. Acoust. Soc. Am., Vol. 35: pp1626–1632, 1963.*

[8] *Waterman, P.C., New formulation of acoustic scattering. J. Acoust. Soc. Am., Vol. 45: pp1417-1429, 1969.*

[9] *Cruse, T.A., BIE fracture mechanics analysis: 25 years of developments. Computational Mechanics, Vol. 18(1): pp1-11, 1996.*

[10] *Banerjee, P.K., D.N. Cathie, and T.G. Davies, Two and three-dimensional problems of elastoplasticity, in Deueloprnents in Boundary Elements, P.K. Banerjee and R. Butterfield, Editors. 1979.*

[11] *Jawson, M.A. and G.T. Symm, Integral equation methods in potential theory- I, II. Proc.Royal Soc., Vol. 275A: pp23-46, 1963.*

[12] *Banerjee, P.K. and R. Butterfield, Boundary element method in geomechanics, in Finite element in geomechanics, G. G., Editor. 1977, Wiley: New York. p. 529-70.*

[13] *Chertock, G., Sound radiation from vibrating surfaces. J. Acoust. Soc. Am., Vol. 36: pp1305-1313, 1964.*

[14] *Copley, L.G., Integral equation method for radiation from vibrating bodies. J. Acoust. Soc. Am., Vol. 41: pp807-810, 1967.*

[15] *Copley, L.G., Fundamental Results Concerning Integral Representations in Acoustic Radiation. The Journal of the Acoustical Society of America, Vol. 44(1): pp28-32, 1968.*

[16] *Schenck, H.A., Improved integral formulation for acoustic radiation problems. J. Acoust. Soc. Am., Vol. 44: pp41-58, 1968.*

[17] *Burton, A.J. and G.F. Miller. The application of the integral equation methods to the numerical solution of some exterior boundary-value problems. in Proceedings of the royal society of London, Serious A, Mathemtaic and physical sciences. 1971.*

[18] *Kost, A. and J. Shen, Parallel computation of 3-D nonlinear eddy currents by the Boundary Element Method (BEM) and the Fast Fourier Transform (FFT). COMPEL, 1990. 9, Supplement A: p. 181-184.*

[19] *Valente, F.P. and H.L. Pina, Iterative techniques for 3-D boundary element method systems of equations. Engineering Analysis with Boundary Elements, Vol. 25(6): pp423-429, 2001.*

[20] *Nabors, K., et al., Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory. SIAM J. ScI.COMPUT., Vol. 15(3): pp713-735, 1994.*

[21] *Rokhlin, V., A fast algorithm for the discrete Laplace transformation. Journal of Complexity, Vol. 4(1): pp12-32, 1988.*

[22] *Greengard, L. and V. Rokhlin, A fast algorithm for particle simulations. Journal of Computational Physics, Vol. 73: pp325-348, 1987.*

[23] *Seybert, A.F. and T.W. Wu, Modified Helmholtz integral equation for bodies sitting on an infinite plane. J. Acoust. Soc. Am., Vol. 85(1): pp19-23, 1989.*

[24] *Swedlow, J. and T. Cruse, Formulation of boundary integral equations for three dimensional elasto-plastic flow. Int J Solids Struct Vol. 7: pp1673-1683, 1971.*

[25] *Sirtori, S., General stress anlaysis method by means of integral equations and boundary elements. Meccanica, Vol. 14: pp210-218, 1979.*

[26] *Mansur, W.J., A time-stepping technique to solve wave propagation problems using the boundary element method. 1983, University of Southampton, England.*

[27] *Kim, G.-T. and B.-H. Lee, 3-D sound source reconstruction and field reprediction using the Helmholtz integral equation. Journal of Sound and Vibration, Vol. 136: pp245-261, 1990.*

[28] *Banerjee, P.K., S. Ahmad, and H.C. Wang, A new BEM formulation for the acoustic eigenfrequency analysis. Int. J. Num. Meth. Eng., Vol. 26: pp1299-1309, 1988.*

[29] *Karabalis, D.L. and D.E. Beskos, Dynamic response of 3-D flexible foundations by time domain BEM and FEM. Soil Dyn. Earthquake Eng., Vol. 4: pp91-101, 1985.*

[30] *Dumont, N., The hybrid boundary element method, in Boundary Elements IX. 1987, Springer: Berlin. p. 117-130.*

[31] *Guiggiani, M., Formulation and numerical treatment of boundary integral equations with hypersingular kernels, in Singular Integrals in Boundary Element Methods, V. Sladekand J. Sladek, Editors. 1998, Computational Mechanics Publishers.*

[32] *Krishnasamy, G.L., et al., Hypersingular boundary integral equations: Some applications in acoustic and elastic wave scattering. J. Appl. Mech. , Vol. 57: pp404-414, 1990.*

[33] *Liu, Y.J. and F.J. Rizzo, A weakly singular form of the hypersingular boundary integral equation applied to 3D acoustic wave problems. Comput. Methods Appl. Mech. Engrg., Vol. 96: pp271-287, 1992.*

[34] *Yang, S.A., Evaluation of 2D Green's boundary formula and its normal derivative using Legendre polynomials, with an application to acoustic scattering problems. Int. J.Numer. Methods Eng., Vol. 53: pp905-927, 2002.*

[35] *Meyer, W.L., W.A. Bell, and B.T. Zinn, Boundary integral solutions of three dimensional acoustic radiation problems. Journal of Sound and Vibration, Vol. 59(2): pp245-262, 1978.*

[36] *Crosbie, A.L. and R.G. Schrenker, Radiative transfer in a two-dimensional rectangular medium exposed to diffuse radiation. J. Quant. Spectrosc. Radiat.Transf., Vol. 31: pp339-372, 1984.*

[37] *Liu YJ . Fast multipole boundary element method: theory and applications in engineering. Cambridge University Press, New York, 2009.*

[38] *Barnes J, Hut P. A hierarchical O(N log N) force calculation algorithm, Nature, Vol. 324: pp. 446-449, 1986.*

[39] *Greengrad L, Rokhlin V. A fast algorithm for particle simulations, J Comput. Phys, Vol. 135: pp. 280-292, 1997.*

[40] *Rokhlin V. Rapid solution of integral equations of classical potential theory. J ComputPhys, Vol. 60: pp.187–207, 1985.*

[41] *Mansur WJ, Araujo FC, Malachini JEB. Solution of BEM systems of equations via iterative techniques. Int J Numer Meth Engng, Vol. 33: pp. 1823–41, 1992.*

[42] *Urekew TJ, Rencis J. The importance of diagonal dominance in the iterative solution of equations generated from the boundary element method, Int J Numer Meth Engng, Vol. 36: pp. 3509–27, 1993.*

[43] *Prasad KG, Kane JH, Keyes DE, Balakrishna C. Preconditioned Krylov solvers for BEA. Int J Numer Meth Engng, Vol. 37: pp. 1651–72, 1994.*

[44] *Bulgakov VE, Bialecki RA, Kuhn G. Coarse division transform based preconditioner for boundary element problems, Int J Numer Meth Engng, Vol. 38: pp. 2115– 29, 1995.*

[45] *Hribersek M, Skerget L. Iterative methods in solving Navier–Stokes equations by the boundary element method. Int J Numer Meth Engng, Vol. 39: pp.115 –39, 1996.*

[46] *Zejun Chen, Hong Xiao. The fast multipole boundary element methods (FMBEM) and its applications in rolling engineering analysis, ComputMech, Vol. 50: pp. 513–531, 2012.*

[47] *Fu Y, Klimkowski KJ, Rodin IGJ, Beger E, Browne JC, Singer JK, Van de Geijn RA, Vemaganti KS. A fast solution method for three-dimensional many particle problems of linear elasticity, Int J Numer Meth Engng, Vol. 42: pp.1215–1229, 1998.*

[48] *Hayami K, Sauter SA. Panel clustering for 3-D elastostatics using spherical harmonics, In: Kassab A, Brebbia CA, Chopra M, editors. Proceedings of Boundary Elements XX. Southampton, UK: Computational Mechanics Publications; 1998.*

[49] *Peirce AP, Napier JAL. A spectral multipole method for efficient solution of large scale boundary element models in elastostaics, Int J Numer Meth Engng, Vol. 38: pp. 4009–34, 1995.*

[50] *Hayami K, Sauter SA. Cost estimation of the panel clustering method applied to 3-D elastostatics, In: Brebbia CA, editor. Proceedings of the Second European Boundary Element Method Symposium, EUROBEM 98. Southampton, UK: Computational Mechanics Publications; 1998.*

[51] *Popov V, Power H. An O(N) Taylor series multipole boundary element method for three-dimensional elasticity problems, Engng Anal Bound Elem, Vol.25, pp. 7–18, 2001.*

[52] *Gomez JE, Power H. A multipole direct and indirect BEM for 2D cavity flow at low Reynolds number, Engng Anal Bound Elem, Vol. 19: pp.17–31, 1997.*

[53] *Saad Y, Schultz M. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J Sci Stat Comput, Vol. 7(3): pp. 856–869, 1986.*

[54] *Xiao H, Chen ZJ. Numerical experiments of preconditioned Krylov subspace methods solving the dense nonsymmetric systems arising from BEM, Engng Anal Bound Elem, Vol. 31(12): pp.1013–1023, 2007.*

[55] *Beskos DE, editor, Boundary Element analysis of plates and shells, Berlin, Springer-Verlag, 1991.*

[56] *Bézine, G. Boundary integral formulation for plate flexure with arbitrary boundary conditions, Mech Res Comm, Vol. 5: pp. 197-206, 1978.*

[57] *Stern, M. A general boundary integral formulation for the numerical solution of plate bending problems, Int J Solids Struct, Vol. 15: pp. 769-782, 1979.*

[58] *Vander Weeën F. Application of the boundary integral equation method to Reissner's plate model, Int J Num Meth in Eng, Vol. 18: pp. 1-10, 1982.*

[59] *Reissner, E. The effect of transverse shear deformation on bending of elastic plates, J App Math, Vol. 12: pp. A69-A77, 1945.*

[60] *Hörmander, L., Linear Partial Differential Operators, Springer-Verlag, Berlin, 1963.*

[61] *Abramowitz M, Stegun, IA., eds., Handbook of mathematical functions with formulas, graphs, and mathematical tables, 9th printing, Dover, New York, 1972.*

[62] *Karam V, Telles JCF. On boundary elements for Reissner's plate theory. Engng Anal Bound Elem, Vol. 5: pp21–27, 1988.*

APPENDIX A

Sample computer program

A.1 A fortran code of the fast multi-pole method for plate bending problems.

The following is a list of the source code written in the FORTRAN for the program discussed in section 4.2for plate bending problems using the fast multi-pole BEM.

```
      c Program:Plate_Bending_FMM-Afast multi-pole boundary element
      c Method(BEM)code for analyzing large-scale,general
      c Plate Bending problems using constant elements.
     Program  Plate_Bending


c     implicit real*8 (a-h,o-z)
           IMPLICIT DOUBLE PRECISION (A-H)
     IMPLICIT DOUBLE PRECISION (O-Z)
     integer, allocatable ::ia(:)
     complex*16, allocatable ::am(:)
C== Note that AUU=(no., of ele,no. , of cells ,no.,of terms in equations)
           DOUBLE PRECISION, DIMENSION (3500,16,135)::AUU    !max @ 5 terms of auu=279
           DOUBLE PRECISION, DIMENSION (3500,16,180)::AMM   !max @ 5 terms of amm=372
           DOUBLE PRECISION, DIMENSION (3500,16,90)::AQQ   !max @ 5 terms of aqq=186
           DOUBLE PRECISION, DIMENSION (30000)::coG
           DOUBLE PRECISION, DIMENSION (30000)::coH
           character*80 Prob_Title
           COMMON/MATRIX/AUU,AMM,AQQ
           COMMON/GHdirect/coG,coH
           COMMON/BASICBLOCK/ T,XNU,E,NSUB
           COMMON/BASICLOAD/ QLOAD
     COMMON/FILEBLOCK/ Specr1,Specr2
           QLOAD=0.d0
           Specr1=0.d0 ; Specr2=0.d0



     call CPU_Time(time0)

c          ===========================================
     open (4,file='input.fmm',status='old')
```

```
      open (5,file='input.dat',status='old')
      open (3,file='output.dat',status='unknown')
c     open (7,file='phi_boundary.plt',status='unknown')
c     open (8,file='xy.plt', status='unknown')
c     open (9,file='phi_domain.plt',status='unknown')
c           open (11,file='to_solve.plt',status='unknown')
c           open (12,file='MAINMATRIX.plt',status='unknown')
c           OPEN (13,FILE='GOUT.XLS')
c           open (15,file='to_print.plt',status='unknown')
c Input the parameters
      read(4,*)       maxl, levmx, nexp, ntylr, tolerance
      read(4,*)       maxia, ncellmx, nleafmx, mxl, nwksz
            read(4,*)       maxt
      read(5,'(a80)')  Prob_Title
      read(5,*)       T,XNU,E,NSUB,n,nfield,Qload
      write(3,'(a80)') Prob_Title
      write(*,'(a80)') Prob_Title

c Estimate the maximum numbers of the cells and leaves,
c and size of the preconditioning matrix,etc.
      if(ncellmx.le.0)ncellmx = max(4*n/maxl,100)
      if(nleafmx.le.0)nleafmx = max(ncellmx/2,100)
      if(nwksz.le.0)nwksz = maxl*maxl*nleafmx
      ligw = 3*n
      lrgw = 1+3*n*(mxl+6)+mxl*(mxl+3)
      iwksz = 3*n+3*nleafmx+1
      allocate (ia(maxia))
c Load the addresses (pointers) associated with the locations of the
c variables to be stored in the large array"am"
      call lpointer(lp,ln,maxia,ia,n,nexp,maxt,ntylr,ncellmx,
     &      levmx,ligw,lrgw,nwksz,iwksz,nfield,
     &      l_n,l_x,l_y,l_node,l_dnorm,
     &      l_bc,l_a,l_b,l_xmax,
     &      l_xmin,l_ymax,l_ymin,l_ielem,l_itree,
     &      l_level,l_loct,l_numt,l_ifath,l_lowlev,
     &      l_maxl,l_levmx,l_nexp,l_ntylr,l_tolerance,
     &      l_ncellmx,l_nleafmx,l_mxl,l_u,l_ax,
     &      l_sb,l_sx,l_ligw,l_lrgw,l_igwk,
     &      l_rgwk,l_nwksz,l_iwksz,l_rwork,l_iwork,
     &      l_xfield,l_nfield,l_f,l_maxt)
```

```
c Estimate the memory usage
      maxa = lp
      write(3,100) maxa*16/1.D6
      write(*,100) maxa*16/1.D6
  100 format('  Memory size of the large block am =',f12.1, 'Mb'/)
c Allocate the large block 'am'
      allocate (am(maxa))
c Assign the parameters to the array am()
      call assigni(n,          am(l_n))
      call assigni(maxl,       am(l_maxl))
      call assigni(levmx,      am(l_levmx))
      call assigni(nexp,       am(l_nexp))
            call assigni(maxt,       am(l_maxt))
      call assigni(ntylr,      am(l_ntylr))
      call assignd(tolerance,  am(l_tolerance))
      call assigni(ncellmx,    am(l_ncellmx))
      call assigni(nleafmx,    am(l_nleafmx))
      call assigni(mxl,        am(l_mxl))
      call assigni(ligw,       am(l_ligw))
      call assigni(lrgw,       am(l_lrgw))
      call assigni(nwksz,      am(l_nwksz))
      call assigni(iwksz,      am(l_iwksz))
      call assigni(nfield,     am(l_nfield))
c Call the FMM BEM main program
      call fmmmain(maxa,maxia,am,ia,
     &       am(l_n),am(l_x),am(l_y),am(l_node),
     &       am(l_dnorm),am(l_bc),am(l_a),am(l_b),
     &       am(l_xmax),am(l_xmin),am(l_ymax),am(l_ymin),
     &       am(l_ielem),am(l_itree),am(l_level),am(l_loct),
     &       am(l_numt),am(l_ifath),am(l_lowlev),am(l_maxl),
     &       am(l_levmx),am(l_nexp),am(l_ntylr),am(l_tolerance),
     &       am(l_ncellmx),am(l_nleafmx),am(l_mxl),am(l_u),
     &       am(l_ax),am(l_nfield),am(l_xfield),am(l_f),
     &       am(l_sb),am(l_sx),am(l_igwk),am(l_rgwk),
     &       am(l_ligw),am(l_lrgw),am(l_nwksz),am(l_iwksz),
     &       am(l_rwork),am(l_iwork),am(l_maxt))
c Estimate the total CPU time
      call CPU_Time(time)
      write(3,*)
      write(*,*)
```

```
      write(3,20) time-time0      !'Total CPU time used =',time-time0,'(sec)'
      write(*,20) time-time0      !'Total CPU time used =',time-time0,'(sec)'
   20       format(' Total CPU time used  =',(f12.3),'(sec)')
      stop
      end
c Definition of Variables:
c
c maxa  = maximum size of the array am
c maxia = maximum number of variables allowed
c am    = a large array storing the variables for the SLATEC GMRES solver
c ia    = an array storing the locations of the variables in the array am
c
c n     = number of elements(= number of nodes)
c x     = coordinates of the nodes
c y     = coordinates of the endpoints ofthe elements
c node  = element connectivity
c dnorm = normal at each node
c bc    = BC type and value
c
c a         = multipoleexpansionmoments
c b         = local expansion coefficients
c xmax,xmin = maximum and minimum x coordinate
c ymax,ymin = maximum and minimum y coordinate
c ielem     = ielem(i) gives the original element number for i-the element in
c           the quad-tree structure
c itree     = itree(c)givesthecelllocationofc-thcellwithineach
c           tree level
c loct      = elements included in the c-the cell are listed starting at
c           the loct(c)-the place in the array ielem
c numt      = numt(c)givesthenumberofelementsincludedinthec-thcell
c ifath     = ifath(c)givesthenumberoftheparentcellofthec-thcell
c level     = levellcellsstartatthelevel(l)-thcellinthetree
c lowlev    = numberofthetreelevels
c
c maxl      = maximum number of elements allowed in aleaf
c levmx     = maximum number of levels allowed in the tree structure
c nexp      = number of terms in multipole expansion
c ntylr     = number of terms in local expansion
c tolerance = GMRES solution convergence tolerance
c ncellmx   = maximum number of cells allowed in the tree
```

```
c nleafmx   = maximum number of leaves allowed in the tree
c mxl       = maximumdimensionofKrylovsubspace(usedinGMRES)
c
c u         = firststoresbvector;thensolutionvectorofsystemAx = b
c ax        = resultingvectorofmultiplicationAx
c nfield    = number of the field points inside the domain
c xfield    = coordinates of the field points inside the domain
c f         = valuesofthepotentialatthefieldpointsinsidethedomain
c
c The following variables and arrays are used in the SLATEC GMRES solver:
c      sb,sx,igwk,rgwk,ligw,lrgw,nwksz,iwksz,rwork,iwork

      subroutine lpointer(lp,ln,maxia,ia,n,nexp,maxt,ntylr,ncellmx,
     &           levmx,ligw,lrgw,nwksz,iwksz,nfield,
     &           l_n,l_x,l_y,l_node,l_dnorm,
     &           l_bc,l_a,l_b,l_xmax,
     &           l_xmin,l_ymax,l_ymin,l_ielem,l_itree,
     &           l_level,l_loct,l_numt,l_ifath,l_lowlev,
     &           l_maxl,l_levmx,l_nexp,l_ntylr,l_tolerance,
     &           l_ncellmx,l_nleafmx,l_mxl,l_u,l_ax,
     &           l_sb,l_sx,l_ligw,l_lrgw,l_igwk,
     &           l_rgwk,l_nwksz,l_iwksz,l_rwork,l_iwork,
     &           l_xfield,l_nfield,l_f,l_maxt)
          dimension ia(maxia)
      lp = 1
      l_n        = l_address(1,maxia,ia,lp,4,1)
      l_x        = l_address(2,maxia,ia,lp,8,n*2)
      l_y        = l_address(3,maxia,ia,lp,8,n*2)
      l_node     = l_address(4,maxia,ia,lp,4,n*2)
      l_dnorm    = l_address(5,maxia,ia,lp,8,n*2)
      l_bc       = l_address(6,maxia,ia,lp,8,n*2*3)  !i change *3
      l_a        = l_address(7,maxia,ia,lp,16,(nexp+1)*ncellmx)
      l_b        = l_address(8,maxia,ia,lp,16,(ntylr+1)*ncellmx)
      l_xmax     = l_address(9,maxia,ia,lp,8,1)
      l_xmin     = l_address(10,maxia,ia,lp,8,1)
      l_ymax     = l_address(11,maxia,ia,lp,8,1)
      l_ymin     = l_address(12,maxia,ia,lp,8,1)
      l_ielem    = l_address(13,maxia,ia,lp,4,n)
      l_itree    = l_address(14,maxia,ia,lp,4,ncellmx)
      l_level    = l_address(15,maxia,ia,lp,4,levmx+1)
```

```
      l_loct     = l_address(16,maxia,ia,lp,4,ncellmx)
      l_numt     = l_address(17,maxia,ia,lp,4,ncellmx)
      l_ifath    = l_address(18,maxia,ia,lp,4,ncellmx)
      l_lowlev   = l_address(19,maxia,ia,lp,4,1)
      l_maxl     = l_address(20,maxia,ia,lp,4,1)
      l_levmx    = l_address(21,maxia,ia,lp,4,1)
      l_nexp     = l_address(22,maxia,ia,lp,4,1)
      l_ntylr    = l_address(23,maxia,ia,lp,4,1)
      l_tolerance = l_address(24,maxia,ia,lp,8,1)
      l_ncellmx  = l_address(25,maxia,ia,lp,4,1)
      l_nleafmx  = l_address(26,maxia,ia,lp,4,1)
      l_mxl      = l_address(27,maxia,ia,lp,4,1)
      l_u        = l_address(28,maxia,ia,lp,8,n*3)          !i change *3
      l_ax       = l_address(29,maxia,ia,lp,8,n*3)          !i change *3
      l_sb       = l_address(30,maxia,ia,lp,8,n)
      l_sx       = l_address(31,maxia,ia,lp,8,n)
      l_ligw     = l_address(32,maxia,ia,lp,4,1)
      l_lrgw     = l_address(33,maxia,ia,lp,4,1)
      l_igwk     = l_address(34,maxia,ia,lp,4,ligw)
      l_rgwk     = l_address(35,maxia,ia,lp,8,lrgw)
      l_nwksz    = l_address(36,maxia,ia,lp,4,1)
      l_iwksz    = l_address(37,maxia,ia,lp,4,1)
      l_rwork    = l_address(38,maxia,ia,lp,8,nwksz)
      l_iwork    = l_address(39,maxia,ia,lp,4,iwksz)
      l_xfield   = l_address(40,maxia,ia,lp,8,nfield*2)
      l_nfield   = l_address(41,maxia,ia,lp,4,1)
      l_f        = l_address(42,maxia,ia,lp,8,nfield)
           l_maxt     = l_address(43,maxia,ia,lp,4,1)
c          write(*,*)   lp,ln,maxia,ia,n,nexp,ntylr,ncellmx,
c     &          levmx,ligw,lrgw,nwksz,iwksz,nfield,
c     &          l_n,l_x,l_y,l_node,l_dnorm,
c     &          l_xmin,l_ymax,l_ymin,l_ielem,l_itree,
c     &          l_level,l_loct,l_numt,l_ifath,l_lowlev,
c     &          l_maxl,l_levmx,l_nexp,l_ntylr,l_tolerance,
c     &          l_ncellmx,l_nleafmx,l_mxl,l_u,l_ax,
c     &          l_sb,l_sx,l_ligw,l_lrgw,l_igwk,
c     &          l_rgwk,l_nwksz,l_iwksz,l_rwork,l_iwork,
c     &          l_xfield,l_nfield,l_f
c      pause
      return
```

```
      end
C=============================================================
      integer function l_address(ln,maxia,ia,lp,ibyte,length)
      dimension ia(maxia)
      l_address = lp
      ia(ln) = lp
      iu = 16
      inc = (ibyte*length-1)/iu+1
      lp = lp+inc
      if(ln.gt.maxia) then
      write(*,*) '!Specified # of variables maxia',maxia,'is too small'
      stop
      endif
      return
      end
c           ===========================================================
      subroutine assigni(i,ii)
      integer i,ii
      ii = i
      return
      end
c=============================================================
      subroutine assignd(h,hh)
      real*8 h,hh
        hh = h
      return
      end
c=============================================================
      subroutine fmmmain(maxa,maxia,am,ia,n,x,y,node,dnorm,bc,
     &          a,b,xmax,xmin,ymax,ymin,ielem,itree,level,loct,numt,
     &          ifath,lowlev,maxl,levmx,nexp,ntylr,tolerance,ncellmx,
     &          nleafmx,mxl,u,ax,nfield,xfield,f,sb,sx,igwk,rgwk,
     &          ligw,lrgw,nwksz,iwksz,rwork,iwork,maxt)
      implicit real*8(a-h,o-z)
C     real*8 am(maxa),a,b                                          !I change
      complex*16 am(maxa),a,b
      dimension ia(maxia),ja(1),a(0:nexp,ncellmx),b(0:ntylr,ncellmx),
     &     x(2,n),y(2,n),node(2,n),dnorm(2,n),bc(2,3*n),    !I change
     &     ielem(n),itree(ncellmx),level(0:levmx),loct(ncellmx),
     &     numt(ncellmx),ifath(ncellmx),u(3*n),ax(3*n),sb(3*n),sx(3*n),!I change
```

```
     &     igwk(ligw),rgwk(lrgw),rwork(nwksz),iwork(iwksz),
     &     xfield(2,nfield),f(nfield) !,G(3*N,3*N),H(3*N,3*N)      !I change
C    &     ,AUU(N,N,279)  !I change
      external matvec,msolve
c Input parameters and prepare the BEM model
      call prep_model(n,x,y,node,bc,dnorm,xfield,nfield,maxl,levmx,
     &              nexp,tolerance,maxt,xmin,xmax,ymin,ymax)      !,ntylr

C          WRITE(*,*)xmin,xmax,ymin,ymax
C          DO I=1,N
C          WRITE(*,*) I,'X',dnorm(1,I),'XX',dnorm(2,I)
C          WRITE(*,*) I,'X',BC(1,3*I-1),'XX',BC(2,3*I-1)
C          WRITE(*,*) I,'X',BC(1,3*I),'XX',BC(2,3*I)
C          ENDDO
C          PAUSE

c Generate the quad-tree structure for the elements
      call tree(n,x,xmax,xmin,ymax,ymin,ielem,itree,level,loct,numt,
     &        ifath,lowlev,maxl,levmx,ncellmx,nleafmx,nwksz,iwork)

c          DO I=1,36
c          WRITE(*,*) I,iwork(I)  !ifath(I),
c          ENDDO
C          PAUSE

C          CALL GHMATC(N,Y,X,G,H)       !(N,Y(1,I),Y(2,I),X(1,I),X(2,I),G,H) I change
C          PAUSE
c Compute the right-hand-side vector b with the FMM
      call fmmbvector(n,x,y,node,dnorm,bc,u,ax,a,b,xmax,xmin,ymax,ymin,
     &              ielem,itree,level,loct,numt,ifath,                    !we here
     &              nexp,ntylr,ncellmx,lowlev,maxl,
     &              rwork,iwork,maxt)
c          WRITE(15,*) 'AUU(1,9,1)'
c          WRITE(*,*) AUU(1,9,1)
c          PAUSE
c Solve the BEM system of equations Ax=b with the fast multipole BEM
c Prepare parameters for calling the iterative solver GMRES
c (SLATEC GMRES solver is used,which is available at www.netlib.org.
c See the documentation for the SLATEC GMRES solver for more information
c about the following related parameters)
```

```
C           do i=1,36 !3*n
C           write(*,*)'rwork', rwork(i)
C           enddo
C           pause
c           DO I=1,50
c           WRITE(*,*) ja(I)
c           ENDDO
c           PAUSE
      nelt    = 1
      isym    = 0
      itol    = 0
      tol     = tolerance
      iunit   = 3
      igwk(1) = mxl
      igwk(2) = mxl
      igwk(3) = 0
      igwk(4) = 1
      igwk(5) = 10
      do i=1,n
c           write(*,*)            ax(3*i-2)
c           write(*,*)            ax(3*i-1)
c           write(*,*)            ax(3*i)
      ax(3*i-2)   = 0.d0
            ax(3*i-1)   = 0.d0
            ax(3*i)   = 0.d0
      enddo
c           pause
c           do k=1,324
c           write(*,*) k,iwork(k),rwork(k)
c           enddo
c           pause
      write(*,*) ' Call Equation Solver GMRES...'
      call dgmres(3*n,u,ax,nelt,ia,ja,am,isym,matvec,msolve,itol,tol,        !I CHANGE
     &           itmax,iter,er,ierr,iunit,sb,sx,rgwk,lrgw,igwk,ligw,
     &           rwork,iwork)
      write(3,*) ' Error indicator from GMRES:',ierr
      write(*,*) ' Error indicator from GMRES:',ierr
            WRITE(*,*) '========================================='
            WRITE(3,*) '========================================='
c Output the boundary solution
```

```
      do i=1,n
      u(3*ielem(i)-2) = ax(3*i-2)
            u(3*ielem(i)-1) = ax(3*i-1)
            u(3*ielem(i)) = ax(3*i)
      enddo
      write(3,*)'Fast Multipole BEM Solution:'
c           write(7,*)'Fast Multipole BEM Solution:'
      WRITE(3,1)
    1 FORMAT(' NO.',6X,'Rx',13X,'Ry',13X,'W',13X,'Mx',13X,'My',13X,'Q')
            do i=1,n

            IF(bc(1,3*I-2).EQ.1.AND.bc(1,3*I-1).EQ.1.AND.
           &   bc(1,3*I).EQ.1) THEN
           write(3,2)i,bc(2,3*i-2),bc(2,3*i-1),
           &            bc(2,3*i),u(3*i-2),u(3*i-1),u(3*i)
            ELSEIF(bc(1,3*I-2).EQ.1.AND.bc(1,3*I-1).EQ.1.AND.
           &   bc(1,3*I).EQ.2) THEN
           write(3,2)i,bc(2,3*i-2),bc(2,3*i-1),
           &            u(3*i),u(3*i-2),u(3*i-1),bc(2,3*i)
            ELSEIF(bc(1,3*I-2).EQ.1.AND.bc(1,3*I-1).EQ.2.AND.
           &   bc(1,3*I).EQ.1) THEN
           write(3,2)i,bc(2,3*i-2),u(3*i-1),bc(2,3*i)
           &            ,u(3*i-2),bc(2,3*i-1),u(3*i)
            ELSEIF(bc(1,3*I-2).EQ.1.AND.bc(1,3*I-1).EQ.2.AND.
           &   bc(1,3*I).EQ.2) THEN
           write(3,2)i,bc(2,3*i-2),u(3*i-1),u(3*i)
           &            ,u(3*i-2),bc(2,3*i-1),bc(2,3*i)
            ELSEIF(bc(1,3*I-2).EQ.2.AND.bc(1,3*I-1).EQ.1.AND.
           &   bc(1,3*I).EQ.1) THEN
           write(3,2)i,u(3*i-2),bc(2,3*i-1),bc(2,3*i)
           &            ,bc(2,3*i-2),u(3*i-1),u(3*i)
            ELSEIF(bc(1,3*I-2).EQ.2.AND.bc(1,3*I-1).EQ.1.AND.
           &   bc(1,3*I).EQ.2) THEN
           write(3,2)i,u(3*i-2),bc(2,3*i-1),u(3*i)
           &            ,bc(2,3*i-2),u(3*i-1),bc(2,3*i)
            ELSEIF(bc(1,3*I-2).EQ.2.AND.bc(1,3*I-1).EQ.2.AND.
           &   bc(1,3*I).EQ.2) THEN
           write(3,2)i,u(3*i-2),u(3*i-1),u(3*i),bc(2,3*i-2)
           &            ,bc(2,3*i-1),bc(2,3*i)
            ELSE
```

```
            WRITE(*,*) 'Error in writing output results'
            STOP
            ENDIF
      enddo
  2   FORMAT(I3,6E15.8)
c Evaluate the field inside the domainand output the results
C           subroutine domain_field(nfield,xfield,UIP,SIP,n,x,y,bc,node,UR)
      call domain_field(nfield,xfield,n,x,y,bc,node,u) !UIP,SIP,
      return
      end
c           ================================================
      subroutine prep_model(n,x,y,node,bc,dnorm,xfield,nfield,maxl,
     &                  levmx,nexp,tolerance,maxt,            ! ,ntylr
     &                  xmin,xmax,ymin,ymax)
      implicit real*8(a-h,o-z)
c           INTEGER           NODE
      dimension x(2,*),Y(2,N),node(2,N),bc(2,*),dnorm(2,*)
            + ,xfield(2,*) !I change
            write(*,*) ' Reading input data...'
      write(*,2) n,maxl,levmx,maxt,tolerance  !,nexp
      write(3,2) n,maxl,levmx,maxt,tolerance  !,nexp
    2 format('  Total number of elements          =',I12
     &       /'  Max. number of elements in aleaf   =',I12
     &       /'  Max. number of tree levels        =',I12
     &       /'  Number of terms used in expansions =',I12
     &       /'  Tolerance for convergence         =',D12.3)

      write(3,*) "======================================="
            DO I=1,N
            DO J=1,2
            X(J,I)=0.0
            Y(J,I)=0.0
            NODE(J,I)=0.0
            DNORM(J,I)=0.0
            ENDDO
            ENDDO
            DO I=1,3*N
            DO J=1,2
            BC(J,I)=0.D0
            ENDDO
```

```
            ENDDO
C==Input the mesh data
c           write(*,*) " reading input data..."
      read(5,*)
      do i=1,n
      read(5,*)itemp,Y(1,i),Y(2,i)
c           WRITE(*,*) 'xxx',itemp,y(1,i),y(2,i)
      enddo
      read(5,*)
            write(*,*) " Reading Boundary condions..."
c           write(*,*) ' No., of elements =',n
      do i=1,n
            read(5,*)itemp,node(1,i),node(2,i),bc(1,3*I-2),bc(2,3*I-2),
            +       bc(1,3*I-1),bc(2,3*I-1),bc(1,3*I),bc(2,3*I)      !I change
C           write(*,*) i,bc(1,3*I),bc(2,3*I) !Y(1,i),Y(2,i)
      enddo
C           PAUSE
C==Input the field points inside the domain
      if (nfield.gt.0) then
            write(*,*) ' co-ordinates of points inside domain....'
      read(5,*)
      do i=1,nfield
      read(5,*)itemp,xfield(1,i),xfield(2,i)
            write(*,*) itemp,xfield(1,i),xfield(2,i)
      enddo
      endif
C==Compute mid-nodes and normals of the elements
C           write(*,*) "====mid-nodes and normals of the elements====="
C           write(*,*) 'i,x(1,i),x(2,i),h1,h2,el,dnorm(1,i),dnorm(2,i)'
c           write(11,*) "====mid-nodes and normals of the elements====="
c           write(11,*) 'i, ,x(1,i), ,x(2,i), ,el, ,dnorm(1,i),
c           & ,dnorm(2,i)'
C                     do i=1,n
C            write(*,*) i,node(1,i),Y(1,I) !,'BVC',Y(1,node(1,i)) !,node(2,i)
C           enddo
C           pause

            do i=1,n
c                     write(*,*) i,node(1,i),node(2,i),y(1,node(1,i)),
c           +         y(2,node(1,i)),y(1,node(2,i)),
```

```
c           + y(2,node(2,i))
c           pause
        x(1,i) = (Y(1,node(1,i)) + Y(1,node(2,i)))*0.5
        x(2,i) = (Y(2,node(1,i)) + Y(2,node(2,i)))*0.5
        h1 =   Y(2,node(2,i)) - Y(2,node(1,i))
        h2 = -Y(1,node(2,i)) + Y(1,node(1,i))
        el = sqrt(h1**2 + h2**2)
c           write(*,*) i,node(1,i+1),node(2,i+1)

c           pause
C           endif
        dnorm(1,i) = h1/el
        dnorm(2,i) = h2/el
C           write(*,*) i,x(1,i),x(2,i),h1,h2,el,dnorm(1,i),dnorm(2,i)
c           write(11,*) i,x(1,i),x(2,i),el,dnorm(1,i),dnorm(2,i)
      enddo

c  14 format(i5,2x,f10.4,2x,f10.4,2x,f10.4,2x,f10.4,2x,f10.4)

c     Determine the square bounding the problem domain (Largest cell used in FMM)
          xmin=x(1,1)
          xmax=x(1,1)
          ymin=x(2,1)
          ymax=x(2,1)
      do 10 i=2,n
          if(x(1,i).le.xmin) then
          xmin=x(1,i)
          elseif(x(1,i).ge.xmax) then
          xmax=x(1,i)
          endif
          if(x(2,i).le.ymin) then
          ymin=x(2,i)
          elseif(x(2,i).ge.ymax) then
          ymax=x(2,i)
          endif
    10 continue
c           write(*,*) 'corner coordinates ofthe',
c           & 'square bounding the problem domain= (',xmin,',',ymin,')','&',
c           & '(',xmax,',',ymax,')'
          scale = 1.05d0 !Make the squares lightly larger
```

```
          xyd   = max(xmax-xmin,ymax-ymin)/2.d0
          xyd   = xyd*scale
          cx    = (xmin+xmax)/2.d0
          cy    = (ymin+ymax)/2.d0
          xmin  = cx-xyd
          xmax  = cx+xyd
          ymin  = cy-xyd
          ymax  = cy+xyd
c           write(*,*) 'corner coordinates ofthe',
c           & 'square bounding the problem domain after scaled = (',xmin,',',
c           &  ymin,')','&','(',xmax,',',ymax,')'
c           pause
c           write(11,*) 'corner coordinates ofthe',
c           & 'square bounding the problem domain after scaled = (',xmin,',',
c           &  ymin,')','&','(',xmax,',',ymax,')'
c     Output nodal coordinates for plotting
c      write(8,*) 'nodal coordinates for plotting'
c       do i = 1,n
c         write(8,*) x(1,i),x(2,i)
c       enddo
C           PAUSE
            write(*,*)
            write(*,*) " Reading input file ==> Done"
            write(*,*)
      return
      end
C=======================================================================
c-----------------------------------------------------------------------
      subroutine fmmbvector(n,x,y,node,dnorm,bc,u,ax,a,b,xmax,xmin,
     & ymax,ymin,ielem,itree,level,loct,numt,ifath,
     & nexp,ntylr,ncellmx,lowlev,maxl,rwork,iwork,maxt)
      implicit real*8(a-h,o-z)
      real*8 a,b
            integer flash
c           complex*16 a,b
      dimension a(0:nexp,ncellmx),b(0:ntylr,ncellmx),
     & x(2,*),y(2,*),node(2,*),dnorm(2,*),bc(2,*),u(*),ax(*),      !I change
     & ielem(*),itree(*),level(0:*),loct(*),numt(*),ifath(*),
     & rwork(*),iwork(*)
c Switch the BC type
```

```
      do i=1,3*n      !I change
      if(bc(1,i).eq.1.) then
      bc(1,i) = 2.d0
      else
      bc(1,i) = 1.d0
      endif
      enddo

      do i=1,n                        !I change
      u(3*i-2)  = bc(2,3*ielem(i)-2)
            u(3*i-1)  = bc(2,3*ielem(i)-1)
            u(3*i)  = bc(2,3*ielem(i))
c           write(*,*) ielem(i),u(3*i-2),u(3*i-1),u(3*i)
      ax(3*i-2) = 0.d0
            ax(3*i-1) = 0.d0
            ax(3*i) = 0.d0
      enddo
            flash=1
c           pause
c Apply the FMM to compute the right-hand side vector b

      call upward(u,n,y,node,dnorm,bc,a,xmax,xmin,ymax,ymin,ielem,
     & itree,level,loct,numt,ifath,nexp,ncellmx,lowlev,maxl,maxt)          !we here too
            write(*,*) ' Please wait for calculations...'
      call dwnwrd(u,ax,n,x,y,node,dnorm,bc,a,b,xmax,xmin,ymax,ymin,
     & ielem,itree,level,loct,numt,ifath,nexp,ntylr,ncellmx,
     & lowlev,maxl,rwork,iwork,flash,maxt)
            write(*,*) ' Thanks for Waiting...'
c Store b vector in u and switch the BC type back
c           write(15,*) '=======uuuuuu======'
      do i=1,3*n
c           write(*,*) 'ax',i,ax(i)
      u(i) =- ax(i)
c           write(15,*)i, u(i)
      if(bc(1,i).eq.1.) then
      bc(1,i) = 2.d0
      else
      bc(1,i) = 1.d0
      endif
      enddo
```

```
c           write(15,*) '=======uuuuuu======'
c           pause
      return
      end
c-----------------------------------------------------------------------
      subroutine upward(u,n,y,node,dnorm,bc,a,xmax,xmin,ymax,ymin,ielem,
     & itree,level,loct,numt,ifath,nexp,ncellmx,lowlev,maxl,maxt)
      implicit real*8(a-h,o-z)
      real*8 a !,z0,zi,b
      dimension a(0:nexp,ncellmx),
     & y(2,*),node(2,*),dnorm(2,*),bc(2,*),u(*),
     & ielem(*),itree(*),level(0:*),loct(*),numt(*),ifath(*)
c     write(*,*)'kkkkk', level(lowlev+1)-1
c           pause
           do i=1,level(lowlev+1)-1
      do k=0,nexp
      a(k,i) = (0.d0,0.d0)  !Clear multipole moments
      enddo
      enddo
c           write(*,*) maxt
c           pause
c           iiki=iiki+1
c     write(*,*)iiki, '======U====='
c           do i=1,n
c           write(*,*) I,u(3*i-2), u(3*i-1),u(3*i)
c           enddo
c           pause
c           write(*,*) '======='
      do 10 lev=lowlev,2,-1 !Loop from leaf to level 2 cells(Upward)
      ndivx = 2**lev
      dx = (xmax-xmin)/ndivx !Determine cell size
      dy = (ymax-ymin)/ndivx
      do 20 icell=level(lev),level(lev+1)-1 !Loop for level l cells
        itr  = itree(icell)
        itrx = mod(itr,ndivx)
        itry = itr/ndivx                !Position of the cell
        cx   = xmin+(itrx + 0.5d0)*dx
        cy   = ymin+(itry + 0.5d0)*dy     !Center of the cell
c           WRITE(*,*) ICELL,CX,CY,          numt(icell)
c           PAUSE
```

```
c Multipole expansion
      if(numt(icell).le.maxl.or.lev.eq.lowlev) then !Compute moment
c           WRITE(*,*) ICELL,CX,CY,          numt(icell)
c           PAUSE
      call moment(a(0,icell),y,node,ielem(loct(icell)),
     &numt(icell),nexp,cx,cy,u(3*loct(icell)-2),
     &bc,dnorm,maxt)
      endif
c           do idi=0,nexp-1
c           write(*,*) idi, icell,a(idi,icell)
c           enddo
c           pause

c M2M translation
      if(lev.ne.2) then             !Do M2M translation to form moments
c           WRITE(*,*) 'M2M'
       cxp = xmin+(int(itrx/2)*2 + 1)*dx
       cyp = ymin+(int(itry/2)*2 + 1)*dy !Center of parent cell
            r1= -cx+cxp                          !(z_c - z_c')
            r2= -cy+cyp
       io  = ifath(icell)      !Cell no. of parent cell
c           if(icell.eq.20.or.icell.eq.39) then
c           write(*,*) r1,r2,icell,io
c           do k=0,30 !nexp
c           write(*,*)k,'dffd',a(k,io), a(k,icell)
c           enddo
c           pause
c           endif
c           write(*,*)icell,io !,cx,cy,cxp,cyp  !r1,r2,,a(62,io),a(62,icell)


       do k=0,nexp-1,31          !Use M2M
c           write(*,*) k
            a(k+0,io) = a(k+0,io) + a(k+0,icell)                                    !  c0
            if(maxt.eq.1) goto 24
c           write(*,*) 'C0',a(k+0,io)
            a(k+1,io) = a(k+1,io) + a(k+1,icell)- r1*a(k+0,icell)              !  c1
c           write(*,*) 'C1',a(k+1,io)
     a(k+2,io) = a(k+2,io) + a(k+2,icell)- r2*a(k+0,icell)                     !  c2
            if(maxt.eq.2) goto 24
```

```
c           write(*,*) 'C2',a(k+2,io)
      a(k+3,io) = a(k+3,io) + a(k+3,icell)- r1*a(k+1,icell)                    ! c11
           & +r1**2/2*a(k+0,icell)
c           write(*,*) 'C11',a(k+3,io)
      a(k+4,io) = a(k+4,io) + a(k+4,icell)- (r2*a(k+1,icell)+
           & r1*a(k+2,icell))/2+r1*r2/2*a(k+0,icell)                    !  c12
           write(*,*) 'C12',a(k+4,io)
      a(k+5,io) =a(k+4,io) !a(k+5,io) + a(k+5,icell)- (r1*a(k+2,icell)+
c          & r2*a(k+1,icell))/2+r1*r2/2*a(k+0,icell)                    !  c21
c          write(*,*) 'C21',a(k+5,io)
      a(k+6,io) = a(k+6,io) + a(k+6,icell)- r2*a(k+2,icell)                    ! c22
           & +r2**2/2*a(k+0,icell)
           if(maxt.eq.3) goto 24
c          write(*,*) 'C22',a(k+6,io)
      a(k+7,io) = a(k+7,io) + a(k+7,icell)- r1*a(k+3,icell)                    !   c111
           & +r1**2/2*a(k+1,icell)-r1**3/6*a(k+0,icell)
           write(*,*) 'C111',a(k+7,io)
      a(k+8,io) = a(k+8,io) + a(k+8,icell)+(- r2*a(k+3,icell)+
           & r1*r2/2*a(k+1,icell)- r1*a(k+4,icell)+r1*r2/2*a(k+1,icell)-
     & r1*a(k+5,icell)+r1**2/2*a(k+2,icell))/3

           & -r2*r1**2/6*a(k+0,icell)                                    !        c112
c          write(*,*) 'C112',a(k+8,io)
      a(k+9,io) =a(k+8,io) !a(k+9,io) + a(k+9,icell)+(- r2*a(k+3,icell)+
c          & r1*r2/2*a(k+1,icell)- r1*a(k+4,icell)+r1*r2/2*a(k+1,icell)-
c     & r1*a(k+5,icell)+r1**2/2*a(k+2,icell))/3
c          & -r2*r1**2/6*a(k+0,icell)                                    !        c121
c          write(*,*) 'C121',a(k+9,io)
      a(k+10,io) = a(k+10,io)+ a(k+10,icell)+(- r2*a(k+4,icell)+
           & r2**2/2*a(k+1,icell)- r2*a(k+5,icell)+r1*r2/2*a(k+2,icell)-
     & r1*a(k+6,icell)+r1*r2/2*a(k+2,icell))/3
           & -r1*r2**2/6*a(k+0,icell)                                    !        c122
c          write(*,*) 'C122',a(k+10,io)
      a(k+11,io) = a(k+8,io)!a(k+11,io)+ a(k+11,icell)+(- r2*a(k+3,icell)+
c          & r1*r2/2*a(k+1,icell)- r1*a(k+4,icell)+r1*r2/2*a(k+1,icell)-
c     & r1*a(k+5,icell)+r1**2/2*a(k+2,icell))/3
c          & -r2*r1**2/6*a(k+0,icell)                                    !        c211
c          write(*,*) 'C211',a(k+11,io)
      a(k+12,io) = a(k+10,io)!a(k+12,io)+ a(k+12,icell)+(- r2*a(k+4,icell)+
c          & r2**2/2*a(k+1,icell)- r2*a(k+5,icell)+r1*r2/2*a(k+2,icell)-
```

```
c     & r1*a(k+6,icell)+r1*r2/2*a(k+2,icell))/3
c           & -r1*r2**2/6*a(k+0,icell)                                  !          c212
c           write(*,*) 'C212',a(k+12,io)
      a(k+13,io) =a(k+10,io) !a(k+13,io) + a(k+13,icell)+(- r2*a(k+4,icell)+
c           & r2**2/2*a(k+1,icell)- r2*a(k+5,icell)+r1*r2/2*a(k+2,icell)-
c     & r1*a(k+6,icell)+r1*r2/2*a(k+2,icell))/3
c           & -r1*r2**2/6*a(k+0,icell)                                  !          c221
c           write(*,*) 'C221',a(k+13,io)
      a(k+14,io) = a(k+14,io) + a(k+14,icell)- r2*a(k+6,icell)      !c222
           & +r2**2/2*a(k+2,icell)-r2**3/6*a(k+0,icell)
           if(maxt.eq.4) goto 24
c           write(*,*) 'C222',a(k+14,io)
      a(k+15,io) = a(k+15,io) + a(k+15,icell)- r1*a(k+7,icell)                  !c1111
           & +r1**2/2*a(k+3,icell)-r1**3/6*a(k+1,icell)+ r1**4/24*a(k+0,icell)
c           write(*,*) 'C1111',a(k+15,io)
      a(k+16,io) = a(k+16,io) + a(k+16,icell)+(- r2*a(k+7,icell)+
           & r1*r2/2*a(k+3,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+8,icell)+
     & r1*r2/2*a(k+3,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+9,icell)+
     & r1**2/2*a(k+4,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+11,icell)+
     & r1**2/2*a(k+5,icell)-r1**3/6*a(k+2,icell))/4
           & + r2*r1**3/24*a(k+0,icell)                                                !          c1112
c           write(*,*) 'C1112',a(k+16,io)
      a(k+17,io) =a(k+16,io) !a(k+17,io) + a(k+17,icell)+(- r2*a(k+7,icell)+
c           & r1*r2/2*a(k+3,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+8,icell)+
c     & r1*r2/2*a(k+3,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+9,icell)+
c     & r1**2/2*a(k+4,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+11,icell)+
c     & r1**2/2*a(k+5,icell)-r1**3/6*a(k+2,icell))/4
c           & + r2*r1**3/24*a(k+0,icell)                                  !          c1121
c           write(*,*) 'C1121',a(k+17,io)
      a(k+18,io) = a(k+18,io) + a(k+18,icell)+(- r2*a(k+8,icell)+
           & r2**2/2*a(k+3,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+9,icell)+
     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r1*a(k+10,icell)+
     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+11,icell)+
     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+12,icell)+
     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+13,icell)+
     & r1**2/2*a(k+6,icell)-r2*r1**2/6*a(k+2,icell))/6
           & + r1**2*r2**2/24*a(k+0,icell)                                !          c1122
c           write(*,*) 'C1122',a(k+18,io)
      a(k+19,io) =a(k+16,io) !a(k+19,io) + a(k+19,icell)+(- r2*a(k+7,icell)+
c           & r1*r2/2*a(k+3,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+8,icell)+
```

```
c     & r1*r2/2*a(k+3,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+9,icell)+
c     & r1**2/2*a(k+4,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+11,icell)+
c     & r1**2/2*a(k+5,icell)-r1**3/6*a(k+2,icell))/4
c           & + r2*r1**3/24*a(k+0,icell)                                  !          c1211
c           write(*,*) 'C1211',a(k+19,io)
      a(k+20,io) =a(k+18,io) !a(k+20,io) + a(k+20,icell)+(- r2*a(k+8,icell)+
c           & r2**2/2*a(k+3,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+9,icell)+
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r1*a(k+10,icell)+
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+11,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+12,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+13,icell)+
c     & r1**2/2*a(k+6,icell)-r2*r1**2/6*a(k+2,icell))/6
c           & + r1**2*r2**2/24*a(k+0,icell)                          !          c1212
c           write(*,*) 'C1212',a(k+20,io)
      a(k+21,io) =a(k+18,io) !a(k+21,io) + a(k+21,icell)+(- r2*a(k+8,icell)+
c           & r2**2/2*a(k+3,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+9,icell)+
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r1*a(k+10,icell)+
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+11,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+12,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+13,icell)+
c     & r1**2/2*a(k+6,icell)-r2*r1**2/6*a(k+2,icell))/6
c           & + r1**2*r2**2/24*a(k+0,icell)                          !          c1221
c           write(*,*) 'C1221',a(k+21,io)
      a(k+22,io) = a(k+22,io) + a(k+22,icell)+(- r2*a(k+10,icell)+
c           & r2**2/2*a(k+4,icell)-r2**3/6*a(k+1,icell)- r2*a(k+12,icell)+
     & r2**2/2*a(k+5,icell)-r1*r2**2/6*a(k+2,icell)- r2*a(k+13,icell)+
     & r1*r2/2*a(k+6,icell)-r1*r2**2/6*a(k+2,icell)- r1*a(k+14,icell)+
     & r1*r2/2*a(k+6,icell)-r1*r2**2/6*a(k+2,icell))/4
c           & + r1*r2**3/24*a(k+0,icell)                                  !  c1222
c           write(*,*) 'C1222',a(k+22,io)
      a(k+23,io) =a(k+16,io) !a(k+23,io) + a(k+23,icell)+(- r2*a(k+7,icell)+
c           & r1*r2/2*a(k+3,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+8,icell)+
c     & r1*r2/2*a(k+3,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+9,icell)+
c     & r1**2/2*a(k+4,icell)-r2*r1**2/6*a(k+1,icell)- r1*a(k+11,icell)+
c     & r1**2/2*a(k+5,icell)-r1**3/6*a(k+2,icell))/4
c           & + r2*r1**3/24*a(k+0,icell)                                  !          c2111
c           write(*,*) 'C2111',a(k+23,io)
      a(k+24,io) =a(k+18,io) !a(k+24,io) + a(k+24,icell)+(- r2*a(k+8,icell)+
c           & r2**2/2*a(k+3,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+9,icell)+
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r1*a(k+10,icell)+
```

```
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+11,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+12,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+13,icell)+
c     & r1**2/2*a(k+6,icell)-r2*r1**2/6*a(k+2,icell))/6
c           & + r1**2*r2**2/24*a(k+0,icell)                                  !         c2112
c           write(*,*) 'C2112',a(k+24,io)
      a(k+25,io) =a(k+18,io) !a(k+25,io) + a(k+25,icell)+(- r2*a(k+8,icell)+
c           & r2**2/2*a(k+3,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+9,icell)+
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r1*a(k+10,icell)+
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+11,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+12,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+13,icell)+
c     & r1**2/2*a(k+6,icell)-r2*r1**2/6*a(k+2,icell))/6
c           & + r1**2*r2**2/24*a(k+0,icell)                                  !         c2121
c           write(*,*) 'C2121',a(k+25,io)
      a(k+26,io) =a(k+22,io) !a(k+26,io) + a(k+26,icell)+(- r2*a(k+10,icell)+
c           & r2**2/2*a(k+4,icell)-r2**3/6*a(k+1,icell)- r2*a(k+12,icell)+
c     & r2**2/2*a(k+5,icell)-r1*r2**2/6*a(k+2,icell)- r2*a(k+13,icell)+
c     & r1*r2/2*a(k+6,icell)-r1*r2**2/6*a(k+2,icell)- r1*a(k+14,icell)+
c     & r1*r2/2*a(k+6,icell)-r1*r2**2/6*a(k+2,icell))/4
c           & + r1*r2**3/24*a(k+0,icell)                                           ! c2122
c           write(*,*) 'C2122',a(k+26,io)
      a(k+27,io) =a(k+18,io) !a(k+27,io) + a(k+27,icell)+(- r2*a(k+8,icell)+
c           & r2**2/2*a(k+3,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+9,icell)+
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r1*a(k+10,icell)+
c     & r1*r2/2*a(k+4,icell)-r1*r2**2/6*a(k+1,icell)- r2*a(k+11,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+12,icell)+
c     & r1*r2/2*a(k+5,icell)-r2*r1**2/6*a(k+2,icell)- r1*a(k+13,icell)+
c     & r1**2/2*a(k+6,icell)-r2*r1**2/6*a(k+2,icell))/6
c           & + r1**2*r2**2/24*a(k+0,icell)                                  !         c2211
c           write(*,*) 'C2211',a(k+27,io)
      a(k+28,io) =a(k+22,io) !a(k+28,io) + a(k+28,icell)+(- r2*a(k+10,icell)+
c           & r2**2/2*a(k+4,icell)-r2**3/6*a(k+1,icell)- r2*a(k+12,icell)+
c     & r2**2/2*a(k+5,icell)-r1*r2**2/6*a(k+2,icell)- r2*a(k+13,icell)+
c     & r1*r2/2*a(k+6,icell)-r1*r2**2/6*a(k+2,icell)- r1*a(k+14,icell)+
c     & r1*r2/2*a(k+6,icell)-r1*r2**2/6*a(k+2,icell))/4
c           & + r1*r2**3/24*a(k+0,icell)                                           !         c2212
c           write(*,*) 'C2212',a(k+28,io)
      a(k+29,io) =a(k+22,io) !a(k+29,io) + a(k+29,icell)+(- r2*a(k+10,icell)+
c           & r2**2/2*a(k+4,icell)-r2**3/6*a(k+1,icell)- r2*a(k+12,icell)+
```

```
c     & r2**2/2*a(k+5,icell)-r1*r2**2/6*a(k+2,icell)- r2*a(k+13,icell)+
c     & r1*r2/2*a(k+6,icell)-r1*r2**2/6*a(k+2,icell)- r1*a(k+14,icell)+
c     & r1*r2/2*a(k+6,icell)-r1*r2**2/6*a(k+2,icell))/4
c           & + r1*r2**3/24*a(k+0,icell)                                    !           c2221
c           write(*,*) 'C2221',a(k+29,io)
      a(k+30,io) = a(k+30,io) + a(k+30,icell)- r2*a(k+14,icell)+r2**2/2*
           & a(k+6,icell) -r2**3/6*a(k+2,icell)+ r2**4/24*a(k+0,icell)    !           c2222
c           write(*,*) 'C2222',a(k+30,io)
c           if(icell.eq.20.or.icell.eq.39) PAUSE
   24      continue
      enddo
           endif
c           if(icell.eq.19) then
c           write(*,*) io,icell
c           do k=0,278
c           write(*,*) k,' ',a(k,io),'  ',a(k,icell)
c           enddo
c           write(*,*) r1,r2,icell,io,a(62,io),a(62,icell)
c           pause
c           endif

   20 continue
   10 continue

c           do i=1,level(lowlev+1)-1
c     do k=0,nexp-1
c           write(*,*) a(k,i)
c     write(11,7) a(k,i)
c     enddo
c   7  format(17(f30.26))
c     write(*,*)     i,'========================='
c     write(11,*) i,'========================='
c             pause
c     enddo
c             write(11,*) '===end=====end=====end====='
c           write(*,*) '===end=====end=====end====='
      return
      end
c-----------------------------------------------------------------------
*DECK DGMRES
```

```
      SUBROUTINE DGMRES (N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE,
     +   ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, SB, SX, RGWK, LRGW,
     +   IGWK, LIGW, RWORK, IWORK)
C***BEGIN PROLOGUE  DGMRES
C***PURPOSE  Preconditioned GMRES iterative sparse Ax=b solver.
C            This routine uses the generalized minimum residual
C            (GMRES) method with preconditioning to solve
C            non-symmetric linear systems of the form: Ax = b.
C***LIBRARY   SLATEC (SLAP)
C***CATEGORY  D2A4, D2B4
C***TYPE      DOUBLE PRECISION (SGMRES-S, DGMRES-D)
C***KEYWORDS  GENERALIZED MINIMUM RESIDUAL, ITERATIVE PRECONDITION,
C             NON-SYMMETRIC LINEAR SYSTEM, SLAP, SPARSE
C***AUTHOR  Brown, Peter, (LLNL), pnbrown@llnl.gov
C           Hindmarsh, Alan, (LLNL), alanh@llnl.gov
C           Seager, Mark K., (LLNL), seager@llnl.gov
C            Lawrence Livermore National Laboratory
C            PO Box 808, L-60
C            Livermore, CA 94550 (510) 423-3141
C***DESCRIPTION
C
C *Usage:
C     INTEGER   N, NELT, IA(NELT), JA(NELT), ISYM, ITOL, ITMAX
C     INTEGER   ITER, IERR, IUNIT, LRGW, IGWK(LIGW), LIGW
C     INTEGER   IWORK(USER DEFINED)
C     DOUBLE PRECISION B(N), X(N), A(NELT), TOL, ERR, SB(N), SX(N)
C     DOUBLE PRECISION RGWK(LRGW), RWORK(USER DEFINED)
C     EXTERNAL  MATVEC, MSOLVE
C
C     CALL DGMRES(N, B, X, NELT, IA, JA, A, ISYM, MATVEC, MSOLVE,
C    $     ITOL, TOL, ITMAX, ITER, ERR, IERR, IUNIT, SB, SX,
C    $     RGWK, LRGW, IGWK, LIGW, RWORK, IWORK)
C
C *Arguments:
C N      :IN       Integer.
C        Order of the Matrix.
C B      :IN       Double Precision B(N).
C        Right-hand side vector.
C X      :INOUT    Double Precision X(N).
C        On input X is your initial guess for the solution vector.
```

```
C         On output X is the final approximate solution.
C NELT   :IN       Integer.
C         Number of Non-Zeros stored in A.
C IA     :IN       Integer IA(NELT).
C JA     :IN       Integer JA(NELT).
C A      :IN       Double Precision A(NELT).
C         These arrays contain the matrix data structure for A.
C         It could take any form.  See "Description", below,
C         for more details.
C ISYM   :IN       Integer.
C         Flag to indicate symmetric storage format.
C         If ISYM=0, all non-zero entries of the matrix are stored.
C         If ISYM=1, the matrix is symmetric, and only the upper
C         or lower triangle of the matrix is stored.
C MATVEC :EXT      External.
C         Name of a routine which performs the matrix vector multiply
C         Y = A*X given A and X.  The name of the MATVEC routine must
C         be declared external in the calling program.  The calling
C         sequence to MATVEC is:
C            CALL MATVEC(N, X, Y, NELT, IA, JA, A, ISYM)
C         where N is the number of unknowns, Y is the product A*X
C         upon return, X is an input vector, and NELT is the number of
C         non-zeros in the SLAP IA, JA, A storage for the matrix A.
C         ISYM is a flag which, if non-zero, denotes that A is
C         symmetric and only the lower or upper triangle is stored.
C MSOLVE :EXT      External.
C         Name of the routine which solves a linear system Mz = r for
C         z given r with the preconditioning matrix M (M is supplied via
C         RWORK and IWORK arrays.  The name of the MSOLVE routine must
C         be declared external in the calling program.  The calling
C         sequence to MSOLVE is:
C            CALL MSOLVE(N, R, Z, NELT, IA, JA, A, ISYM, RWORK, IWORK)
C         Where N is the number of unknowns, R is the right-hand side
C         vector and Z is the solution upon return.  NELT, IA, JA, A and
C         ISYM are defined as above.  RWORK is a double precision array
C         that can be used to pass necessary preconditioning information
C         and/or workspace to MSOLVE.  IWORK is an integer work array
C         for the same purpose as RWORK.
C ITOL   :IN       Integer.
C         Flag to indicate the type of convergence criterion used.
```

```
C       ITOL=0  Means the  iteration stops when the test described
C            below on  the  residual RL  is satisfied.  This is
C            the  "Natural Stopping Criteria" for this routine.
C            Other values  of  ITOL  cause  extra,  otherwise
C            unnecessary, computation per iteration and     are
C            therefore  much less  efficient.  See  ISDGMR (the
C            stop test routine) for more information.
C       ITOL=1  Means   the  iteration stops   when the first test
C            described below on  the residual RL  is satisfied,
C            and there  is either right  or  no preconditioning
C            being used.
C       ITOL=2  Implies    that   the  user    is   using    left
C            preconditioning, and the second stopping criterion
C            below is used.
C       ITOL=3  Means the  iteration stops   when  the  third test
C            described below on Minv*Residual is satisfied, and
C            there is either left  or no  preconditioning being
C            used.
C       ITOL=11 is    often  useful  for   checking  and comparing
C            different routines.  For this case, the  user must
C            supply  the  "exact" solution or  a  very accurate
C            approximation (one with  an  error much less  than
C            TOL) through a common block,
C                COMMON /DSLBLK/ SOLN( )
C            If ITOL=11, iteration stops when the 2-norm of the
C            difference between the iterative approximation and
C            the user-supplied solution  divided by the  2-norm
C            of the  user-supplied solution  is  less than TOL.
C            Note that this requires  the  user to  set up  the
C            "COMMON     /DSLBLK/ SOLN(LENGTH)"  in the calling
C            routine.  The routine with this declaration should
C            be loaded before the stop test so that the correct
C            length is used by  the loader.  This procedure  is
C            not standard Fortran and may not work correctly on
C            your   system (although  it  has  worked  on every
C            system the authors have tried).  If ITOL is not 11
C            then this common block is indeed standard Fortran.
C TOL    :INOUT    Double Precision.
C       Convergence criterion, as described below.  If TOL is set
C       to zero on input, then a default value of 500*(the smallest
```

```
C         positive magnitude, machine epsilon) is used.
C ITMAX  :DUMMY    Integer.
C         Maximum number of iterations in most SLAP routines.  In
C         this routine this does not make sense.  The maximum number
C         of iterations here is given by ITMAX = MAXL*(NRMAX+1).
C         See IGWK for definitions of MAXL and NRMAX.
C ITER   :OUT      Integer.
C         Number of iterations required to reach convergence, or
C         ITMAX if convergence criterion could not be achieved in
C         ITMAX iterations.
C ERR    :OUT      Double Precision.
C         Error estimate of error in final approximate solution, as
C         defined by ITOL.  Letting norm() denote the Euclidean
C         norm, ERR is defined as follows..
C
C         If ITOL=0, then ERR = norm(SB*(B-A*X(L)))/norm(SB*B),
C                           for right or no preconditioning, and
C                     ERR = norm(SB*(M-inverse)*(B-A*X(L)))/
C                           norm(SB*(M-inverse)*B),
C                           for left preconditioning.
C         If ITOL=1, then ERR = norm(SB*(B-A*X(L)))/norm(SB*B),
C                           since right or no preconditioning
C                           being used.
C         If ITOL=2, then ERR = norm(SB*(M-inverse)*(B-A*X(L)))/
C                           norm(SB*(M-inverse)*B),
C                           since left preconditioning is being
C                           used.
C         If ITOL=3, then ERR =  Max  |(Minv*(B-A*X(L)))(i)/x(i)|
C                           i=1,n
C         If ITOL=11, then ERR = norm(SB*(X(L)-SOLN))/norm(SB*SOLN).
C IERR   :OUT      Integer.
C         Return error flag.
C              IERR = 0 => All went well.
C              IERR = 1 => Insufficient storage allocated for
C                          RGWK or IGWK.
C              IERR = 2 => Routine DGMRES failed to reduce the norm
C                          of the current residual on its last call,
C                          and so the iteration has stalled.  In
C                          this case, X equals the last computed
C                          approximation.  The user must either
```

```
C                  increase MAXL, or choose a different
C                  initial guess.
C            IERR =-1 => Insufficient length for RGWK array.
C                  IGWK(6) contains the required minimum
C                  length of the RGWK array.
C            IERR =-2 => Illegal value of ITOL, or ITOL and JPRE
C                  values are inconsistent.
C       For IERR <= 2, RGWK(1) = RHOL, which is the norm on the
C       left-hand-side of the relevant stopping test defined
C       below associated with the residual for the current
C       approximation X(L).
C IUNIT  :IN       Integer.
C       Unit number on which to write the error at each iteration,
C       if this is desired for monitoring convergence.  If unit
C       number is 0, no writing will occur.
C SB     :IN       Double Precision SB(N).
C       Array of length N containing scale factors for the right
C       hand side vector B.  If JSCAL.eq.0 (see below), SB need
C       not be supplied.
C SX     :IN       Double Precision SX(N).
C       Array of length N containing scale factors for the solution
C       vector X.  If JSCAL.eq.0 (see below), SX need not be
C       supplied.  SB and SX can be the same array in the calling
C       program if desired.
C RGWK   :INOUT    Double Precision RGWK(LRGW).
C       Double Precision array used for workspace by DGMRES.
C       On return, RGWK(1) = RHOL.  See IERR for definition of RHOL.
C LRGW   :IN       Integer.
C       Length of the double precision workspace, RGWK.
C       LRGW >= 1 + N*(MAXL+6) + MAXL*(MAXL+3).
C       See below for definition of MAXL.
C       For the default values, RGWK has size at least 131 + 16*N.
C IGWK   :INOUT    Integer IGWK(LIGW).
C       The following IGWK parameters should be set by the user
C       before calling this routine.
C       IGWK(1) = MAXL.  Maximum dimension of Krylov subspace in
C         which X - X0 is to be found (where, X0 is the initial
C         guess).  The default value of MAXL is 10.
C       IGWK(2) = KMP.  Maximum number of previous Krylov basis
C         vectors to which each new basis vector is made orthogonal.
```

```
C           The default value of KMP is MAXL.
C         IGWK(3) = JSCAL.  Flag indicating whether the scaling
C           arrays SB and SX are to be used.
C           JSCAL = 0 => SB and SX are not used and the algorithm
C             will perform as if all SB(I) = 1 and SX(I) = 1.
C           JSCAL = 1 =>  Only SX is used, and the algorithm
C             performs as if all SB(I) = 1.
C           JSCAL = 2 =>  Only SB is used, and the algorithm
C             performs as if all SX(I) = 1.
C           JSCAL = 3 =>  Both SB and SX are used.
C         IGWK(4) = JPRE.  Flag indicating whether preconditioning
C           is being used.
C           JPRE = 0  =>  There is no preconditioning.
C           JPRE > 0  =>  There is preconditioning on the right
C             only, and the solver will call routine MSOLVE.
C           JPRE < 0  =>  There is preconditioning on the left
C             only, and the solver will call routine MSOLVE.
C         IGWK(5) = NRMAX.  Maximum number of restarts of the
C           Krylov iteration.  The default value of NRMAX = 10.
C           if IWORK(5) = -1,  then no restarts are performed (in
C           this case, NRMAX is set to zero internally).
C         The following IWORK parameters are diagnostic information
C         made available to the user after this routine completes.
C         IGWK(6) = MLWK.  Required minimum length of RGWK array.
C         IGWK(7) = NMS.  The total number of calls to MSOLVE.
C LIGW   :IN       Integer.
C         Length of the integer workspace, IGWK.  LIGW >= 20.
C RWORK  :WORK     Double Precision RWORK(USER DEFINED).
C         Double Precision array that can be used for workspace in
C         MSOLVE.
C IWORK  :WORK     Integer IWORK(USER DEFINED).
C         Integer array that can be used for workspace in MSOLVE.
C
C *Description:
C       DGMRES solves a linear system A*X = B rewritten in the form:
C
C        (SB*A*(M-inverse)*(SX-inverse))*(SX*M*X) = SB*B,
C
C       with right preconditioning, or
C
```

```
C       (SB*(M-inverse)*A*(SX-inverse))*(SX*X) = SB*(M-inverse)*B,
C
C       with left preconditioning, where A is an N-by-N double precision
C       matrix, X and B are N-vectors,  SB and SX  are diagonal scaling
C       matrices,   and M is  a preconditioning    matrix.   It uses
C       preconditioned  Krylov   subpace  methods  based     on  the
C       generalized minimum residual  method (GMRES).   This routine
C       optionally performs  either  the  full     orthogonalization
C       version of the  GMRES  algorithm or an incomplete variant of
C       it.  Both versions use restarting of the linear iteration by
C       default, although the user can disable this feature.
C
C       The GMRES  algorithm generates a sequence  of approximations
C       X(L) to the  true solution of the above  linear system.  The
C       convergence criteria for stopping the  iteration is based on
C       the size  of the  scaled norm of  the residual  R(L)  =  B -
C       A*X(L).  The actual stopping test is either:
C
C            norm(SB*(B-A*X(L))) .le. TOL*norm(SB*B),
C
C       for right preconditioning, or
C
C            norm(SB*(M-inverse)*(B-A*X(L))) .le.
C                  TOL*norm(SB*(M-inverse)*B),
C
C       for left preconditioning, where norm() denotes the Euclidean
C       norm, and TOL is  a positive scalar less  than one  input by
C       the user.  If TOL equals zero  when DGMRES is called, then a
C       default  value  of 500*(the   smallest  positive  magnitude,
C       machine epsilon) is used.  If the  scaling arrays SB  and SX
C       are used, then  ideally they  should be chosen  so  that the
C       vectors SX*X(or SX*M*X) and  SB*B have all their  components
C       approximately equal  to  one in  magnitude.  If one wants to
C       use the same scaling in X  and B, then  SB and SX can be the
C       same array in the calling program.
C
C       The following is a list of the other routines and their
C       functions used by DGMRES:
C       DPIGMR  Contains the main iteration loop for GMRES.
C       DORTH   Orthogonalizes a new vector against older basis vectors.
```

```
C      DHEQR   Computes a QR decomposition of a Hessenberg matrix.
C      DHELS   Solves a Hessenberg least-squares system, using QR
C              factors.
C      DRLCAL  Computes the scaled residual RL.
C      DXLCAL  Computes the solution XL.
C      ISDGMR  User-replaceable stopping routine.
C
C      This routine does  not care  what matrix data   structure is
C      used for  A and M.  It simply   calls  the MATVEC and MSOLVE
C      routines, with  the arguments as  described above.  The user
C      could write any type of structure and the appropriate MATVEC
C      and MSOLVE routines.  It is assumed  that A is stored in the
C      IA, JA, A  arrays in some fashion and  that M (or INV(M)) is
C      stored  in  IWORK  and  RWORK   in  some fashion.   The SLAP
C      routines DSDCG and DSICCG are examples of this procedure.
C
C      Two  examples  of  matrix  data structures  are the: 1) SLAP
C      Triad  format and 2) SLAP Column format.
C
C      ================== S L A P Triad format ==================
C      This routine requires that the  matrix A be   stored in  the
C      SLAP  Triad format.  In  this format only the non-zeros  are
C      stored.  They may appear in  *ANY* order.  The user supplies
C      three arrays of  length NELT, where  NELT is  the number  of
C      non-zeros in the matrix: (IA(NELT), JA(NELT), A(NELT)).  For
C      each non-zero the user puts the row and column index of that
C      matrix element  in the IA and  JA arrays.  The  value of the
C      non-zero   matrix  element is  placed  in  the corresponding
C      location of the A array.   This is  an  extremely  easy data
C      structure to generate.  On  the  other hand it   is  not too
C      efficient on vector computers for  the iterative solution of
C      linear systems.  Hence,   SLAP changes   this  input    data
C      structure to the SLAP Column format  for  the iteration (but
C      does not change it back).
C
C      Here is an example of the  SLAP Triad   storage format for a
C      5x5 Matrix.  Recall that the entries may appear in any order.
C
C        5x5 Matrix      SLAP Triad format for 5x5 matrix on left.
C                        1  2  3  4  5  6  7  8  9 10 11
```

```
C      |11 12  0  0 15|   A: 51 12 11 33 15 53 55 22 35 44 21
C      |21 22  0  0  0|  IA:  5  1  1  3  1  5  5  2  3  4  2
C      | 0  0 33  0 35|  JA:  1  2  1  3  5  3  5  2  5  4  1
C      | 0  0  0 44  0|
C      |51  0 53  0 55|
C
C      ==================== S L A P Column format ==================
C
C      This routine requires that the matrix A  be stored in  the
C      SLAP Column format.  In this format the non-zeros are stored
C      counting down columns (except for  the diagonal entry, which
C      must appear first in each  "column")  and are stored  in the
C      double precision array A.   In other words,  for each column
C      in the matrix put the diagonal entry in  A.  Then put in the
C      other non-zero  elements going down  the column (except  the
C      diagonal) in order.   The  IA array holds the  row index for
C      each non-zero.  The JA array holds the offsets  into the IA,
C      A arrays  for  the  beginning  of each   column.   That  is,
C      IA(JA(ICOL)),  A(JA(ICOL)) points   to the beginning  of the
C      ICOL-th   column    in    IA and   A.      IA(JA(ICOL+1)-1),
C      A(JA(ICOL+1)-1) points to  the  end of the   ICOL-th column.
C      Note that we always have  JA(N+1) = NELT+1,  where N is  the
C      number of columns in  the matrix and NELT  is the number  of
C      non-zeros in the matrix.
C
C      Here is an example of the  SLAP Column  storage format for a
C      5x5 Matrix (in the A and IA arrays '|'  denotes the end of a
C      column):
C
C         5x5 Matrix      SLAP Column format for 5x5 matrix on left.
C                            1  2  3    4  5    6  7    8    9 10 11
C      |11 12  0  0 15|   A: 11 21 51 | 22 12 | 33 53 | 44 | 55 15 35
C      |21 22  0  0  0|   IA: 1  2  5 |  2  1 |  3  5 |  4 |  5  1  3
C      | 0  0 33  0 35|  JA:  1  4  6    8  9   12
C      | 0  0  0 44  0|
C      |51  0 53  0 55|
C
C *Cautions:
C     This routine will attempt to write to the Fortran logical output
C     unit IUNIT, if IUNIT .ne. 0.  Thus, the user must make sure that
```

```
C     this logical unit is attached to a file or terminal before calling
C     this routine with a non-zero value for IUNIT.  This routine does
C     not check for the validity of a non-zero IUNIT unit number.
C
C***REFERENCES  1. Peter N. Brown and A. C. Hindmarsh, Reduced Storage
C                  Matrix Methods in Stiff ODE Systems, Lawrence Liver-
C                  more National Laboratory Report UCRL-95088, Rev. 1,
C                  Livermore, California, June 1987.
C               2. Mark K. Seager, A SLAP for the Masses, in
C                  G. F. Carey, Ed., Parallel Supercomputing: Methods,
C                  Algorithms and Applications, Wiley, 1989, pp.135-155.
C***ROUTINES CALLED  D1MACH, DCOPY, DNRM2, DPIGMR
C***REVISION HISTORY  (YYMMDD)
C   890404  DATE WRITTEN
C   890404  Previous REVISION DATE
C   890915  Made changes requested at July 1989 CML Meeting.  (MKS)
C   890922  Numerous changes to prologue to make closer to SLATEC
C           standard.  (FNF)
C   890929  Numerous changes to reduce SP/DP differences.  (FNF)
C   891004  Added new reference.
C   910411  Prologue converted to Version 4.0 format.  (BAB)
C   910506  Corrected errors in C***ROUTINES CALLED list.  (FNF)
C   920407  COMMON BLOCK renamed DSLBLK.  (WRB)
C   920511  Added complete declaration section.  (WRB)
C   920929  Corrected format of references.  (FNF)
C   921019  Changed 500.0 to 500 to reduce SP/DP differences.  (FNF)
C   921026  Added check for valid value of ITOL.  (FNF)
C***END PROLOGUE  DGMRES
C        The following is for optimized compilation on LLNL/LTSS Crays.
CLLL. OPTIMIZE
C     .. Scalar Arguments ..
      DOUBLE PRECISION ERR, TOL
      INTEGER IERR, ISYM, ITER, ITMAX, ITOL, IUNIT, LIGW, LRGW, N, NELT
C     .. Array Arguments ..
      DOUBLE PRECISION A(NELT), B(N), RGWK(LRGW), RWORK(*), SB(N),
     +                 SX(N), X(N)

      INTEGER IA(NELT), IGWK(LIGW), IWORK(*), JA(NELT)
C     .. Subroutine Arguments ..
      EXTERNAL MATVEC, MSOLVE
```

```
C     .. Local Scalars ..
      DOUBLE PRECISION BNRM, RHOL, SUM
      INTEGER I, IFLAG, JPRE, JSCAL, KMP, LDL, LGMR, LHES, LQ, LR, LV,
     +        LW, LXL, LZ, LZM1, MAXL, MAXLP1, NMS, NMSL, NRMAX, NRSTS
C     .. External Functions ..
      DOUBLE PRECISION D1MACH, DNRM2
      EXTERNAL D1MACH, DNRM2
C     .. External Subroutines ..
      EXTERNAL DCOPY, DPIGMR
C     .. Intrinsic Functions ..
      INTRINSIC SQRT
C***FIRST EXECUTABLE STATEMENT  DGMRES
      IERR = 0
c           do i=1,36
c           write(*,*) i,rwork(i)
c           enddo
c           pause
c           write(*,*)  n,nelt
c           pause
C ----------------------------------------------------------------------
C       Load method parameters with user values or defaults.
C ----------------------------------------------------------------------

      MAXL = IGWK(1)
      IF (MAXL .EQ. 0) MAXL = 10
      IF (MAXL .GT. N) MAXL = N
      KMP = IGWK(2)
      IF (KMP .EQ. 0) KMP = MAXL
      IF (KMP .GT. MAXL) KMP = MAXL
      JSCAL = IGWK(3)
      JPRE = IGWK(4)
C       Check for valid value of ITOL.
      IF( (ITOL.LT.0) .OR. ((ITOL.GT.3).AND.(ITOL.NE.11)) ) GOTO 650
C       Check for consistent values of ITOL and JPRE.
      IF( ITOL.EQ.1 .AND. JPRE.LT.0 ) GOTO 650
      IF( ITOL.EQ.2 .AND. JPRE.GE.0 ) GOTO 650
      NRMAX = IGWK(5)
      IF( NRMAX.EQ.0 ) NRMAX = 10
C       If NRMAX .eq. -1, then set NRMAX = 0 to turn off restarting.
      IF( NRMAX.EQ.-1 ) NRMAX = 0
```

```
C       If input value of TOL is zero, set it to its default value.
      IF( TOL.EQ.0.0D0 ) TOL = 500*D1MACH(3)
C
C       Initialize counters.
      ITER = 0
      NMS = 0
      NRSTS = 0
c           n=3*n      ! I CHANGE
C   ------------------------------------------------------------------
C       Form work array segment pointers.
C   ------------------------------------------------------------------
      MAXLP1 = MAXL + 1
      LV = 1
      LR = LV + N*MAXLP1
      LHES = LR + N + 1
      LQ = LHES + MAXL*MAXLP1
      LDL = LQ + 2*MAXL
      LW = LDL + N
      LXL = LW + N
      LZ = LXL + N
C         WRITE(*,*) LR,LHES,LDL,LW,LXL,LZ,LRGW
C     PAUSE
C       Load IGWK(6) with required minimum length of the RGWK array.
      IGWK(6) = LZ +N - 1
      IF( LZ+N-1.GT.LRGW ) GOTO 640
C   ------------------------------------------------------------------
C       Calculate scaled-preconditioned norm of RHS vector b.
C   ------------------------------------------------------------------
C         write(*,*) JPRE
C         pause
C         write(*,*) 'N',N
C         pause

      IF (JPRE .LT. 0) THEN
C           PAUSE
        CALL MSOLVE(N, B, RGWK(LR), NELT, IA, JA, A, ISYM,
     $       RWORK, IWORK)
        NMS = NMS + 1
      ELSE
C         DO KK=1,N
```

```
C          WRITE(*,*)'B', B(KK)
C          ENDDO
C          PAUSE
       CALL DCOPY(N, B, 1, RGWK(LR), 1)
C          DO KK=LR,N+LR-1
C          WRITE(*,*) KK,'RGWK',RGWK(KK)
C          ENDDO
C          PAUSE
C          DO I=1,igwk(6)
C          if(rgwk(I).NE.0)THEN
C          WRITE(*,*) I,RGWK(I)
C          ENDIF
C          ENDDO
C          PAUSE
      ENDIF
C          write(*,*) 'NN',N
c          pause

C          write(*,*) JSCAL
C          pause
      IF( JSCAL.EQ.2 .OR. JSCAL.EQ.3 ) THEN
        SUM = 0
        DO 10 I = 1,N
          SUM = SUM + (RGWK(LR-1+I)*SB(I))**2
 10     CONTINUE
        BNRM = SQRT(SUM)
      ELSE
        BNRM = DNRM2(N,RGWK(LR),1)
C            IF(BNRM.EQ.0) THEN  ! I CREATE
C              BNRM=1                                  ! I CREATE
C            ENDIF                                   ! I CREATE
C          write(*,*) BNRM
c          pause

      ENDIF
C          write(*,*) 'NNN',N
C           pause

C     ======  RWORK ARRAY IS CHANGED HERE @ MATVEC  ======
C   ---------------------------------------------------------------
```

```
C        Calculate initial residual.
C   ------------------------------------------------------------------
     CALL MATVEC(N, X, RGWK(LR), NELT, IA, JA, A, ISYM)
c            do I=1,324
c            WRITE(*,*) i,'rwork',rwork(i)  !B(I)
c            ENDDO
C            WRITE(*,*) '============================'
C            do I=1,N
C            WRITE(*,*) RGWK(LR-1+I)
C            ENDDO
C            WRITE(*,*) '============================'
c           pause
     DO 50 I = 1,N
       RGWK(LR-1+I) = B(I) - RGWK(LR-1+I)
  50   CONTINUE
C     do I=1,N
C            WRITE(*,*) "MM",I,RGWK(LR-1+I)
C            ENDDO
C            PAUSE
C           write(*,*) 'NNNN',N
C            pause

C   ------------------------------------------------------------------
C        If performing restarting, then load the residual into the
C        correct location in the RGWK array.
C   ------------------------------------------------------------------
 100  CONTINUE
C      WRITE(*,*) NRSTS,NRMAX
C             PAUSE
     IF( NRSTS.GT.NRMAX ) GOTO 610
     IF( NRSTS.GT.0 ) THEN
C        Copy the current residual to a different location in the RGWK
C        array.
       CALL DCOPY(N, RGWK(LDL), 1, RGWK(LR), 1)
     ENDIF
C          DO I=1,N
C          WRITE(*,*)        RGWK(LR-1+I)
C          ENDDO
C          WRITE(*,*) 'NNNNN',N
```

```
c            do I=1,324
c            WRITE(*,*) i,'rwork',rwork(i)
c            ENDDO
c           PAUSE
C   -------------------------IMPORTANT-----------------------------
C       Use the DPIGMR algorithm to solve the linear system A*Z = R.
C   ----------------------------------------------------------------        MATAHA
      CALL DPIGMR(N, RGWK(LR), SB, SX, JSCAL, MAXL, MAXLP1, KMP,
     $     NRSTS, JPRE, MATVEC, MSOLVE, NMSL, RGWK(LZ), RGWK(LV),
     $     RGWK(LHES), RGWK(LQ), LGMR, RWORK, IWORK, RGWK(LW),
     $     RGWK(LDL), RHOL, NRMAX, B, BNRM, X, RGWK(LXL), ITOL,
     $     TOL, NELT, IA, JA, A, ISYM, IUNIT, IFLAG, ERR)
      ITER = ITER + LGMR
      NMS = NMS + NMSL
c            PAUSE
C           DO I=1,N
C           WRITE(*,*)       I,RGWK(LZ-1+I)
C           ENDDO
C           PAUSE
C
C       Increment X by the current approximate solution Z of A*Z = R.
C
      LZM1 = LZ - 1
      DO 110 I = 1,N
        X(I) = X(I) + RGWK(LZM1+I)
  110 CONTINUE
C     WRITE(*,*) IFLAG
c           PAUSE
      IF( IFLAG.EQ.0 ) GOTO 600
      IF( IFLAG.EQ.1 ) THEN
        NRSTS = NRSTS + 1
        GOTO 100
      ENDIF
      IF( IFLAG.EQ.2 ) GOTO 620
C   -----------------------------------------------------------------
C       All returns are made through this section.
C   -----------------------------------------------------------------
C       The iteration has converged.
C
  600 CONTINUE
```

```
      IGWK(7) = NMS
      RGWK(1) = RHOL
      IERR = 0
      RETURN
C
C     Max number((NRMAX+1)*MAXL) of linear iterations performed.
 610  CONTINUE
      IGWK(7) = NMS
      RGWK(1) = RHOL
      IERR = 1
      RETURN
C
C     GMRES failed to reduce last residual in MAXL iterations.
C     The iteration has stalled.
 620  CONTINUE
      IGWK(7) = NMS
      RGWK(1) = RHOL
      IERR = 2
      RETURN
C     Error return.  Insufficient length for RGWK array.
 640  CONTINUE
      ERR = TOL
      IERR = -1
      RETURN
C     Error return.  Inconsistent ITOL and JPRE values.
 650  CONTINUE
      ERR = TOL
      IERR = -2
      RETURN
C------------ LAST LINE OF DGMRES FOLLOWS ----------------------------
      END
c=======================================================
```

APPENDIX A.1

Sample of first input file

A Square Plate Bending

.3 0 30000000 8 50 1 0 0 !THICKNESS V E NSUB No. of Elements, No. of Field Points

 # Nodes (Node No., x-coordinate, y-coordinate):

1 0 0
2 1 0
3 2 0
4 3 0
5 4 0
6 5 0
7 6 0
8 7 0
9 8 0
10 9 0
11 10 0
12 11 0
13 12 0
14 13 0
15 14 0
16 15 0
17 16 0
18 17 0
19 18 0
20 19 0
21 20 0
22 20 1
23 20 2
24 20 3
25 20 4
26 20 5
27 19 5
28 18 5
29 17 5
30 16 5
31 15 5
32 14 5
33 13 5
34 12 5

35 11 5
36 10 5
37 9 5
38 8 5
39 7 5
40 6 5
41 5 5
42 4 5
43 3 5
44 2 5
45 1 5
46 0 5
47 0 4
48 0 3
49 0 2
50 0 1

 # Elements and Boundary Conditions (Element No., Local Node 1, Local Node 2, BC Type, BC Value):

1	**1**	**2**	**2**	**0**	**2**	**0**	**2**	**0**
2	**2**	**3**	**2**	**0**	**2**	**0**	**2**	**0**
3	**3**	**4**	**2**	**0**	**2**	**0**	**2**	**0**
4	**4**	**5**	**2**	**0**	**2**	**0**	**2**	**0**
5	**5**	**6**	**2**	**0**	**2**	**0**	**2**	**0**
6	**6**	**7**	**2**	**0**	**2**	**0**	**2**	**0**
7	**7**	**8**	**2**	**0**	**2**	**0**	**2**	**0**
8	**8**	**9**	**2**	**0**	**2**	**0**	**2**	**0**
9	**9**	**10**	**2**	**0**	**2**	**0**	**2**	**0**
10	**10**	**11**	**2**	**0**	**2**	**0**	**2**	**0**
11	**11**	**12**	**2**	**0**	**2**	**0**	**2**	**0**
12	**12**	**13**	**2**	**0**	**2**	**0**	**2**	**0**
13	**13**	**14**	**2**	**0**	**2**	**0**	**2**	**0**
14	**14**	**15**	**2**	**0**	**2**	**0**	**2**	**0**
15	**15**	**16**	**2**	**0**	**2**	**0**	**2**	**0**
16	**16**	**17**	**2**	**0**	**2**	**0**	**2**	**0**
17	**17**	**18**	**2**	**0**	**2**	**0**	**2**	**0**
18	**18**	**19**	**2**	**0**	**2**	**0**	**2**	**0**
19	**19**	**20**	**2**	**0**	**2**	**0**	**2**	**0**
20	**20**	**21**	**2**	**0**	**2**	**0**	**2**	**0**
21	**21**	**22**	**2**	**0**	**2**	**0**	**2**	**-1.5**
22	**22**	**23**	**2**	**0**	**2**	**0**	**2**	**-1.5**
23	**23**	**24**	**2**	**0**	**2**	**0**	**2**	**-1.5**

24 24 25 2 0 2 0 2 -1.5

25 25 26 2 0 2 0 2 -1.5

26 26 27 2 0 2 0 2 0

27 27 28 2 0 2 0 2 0

28 28 29 2 0 2 0 2 0

29 29 30 2 0 2 0 2 0

30 30 31 2 0 2 0 2 0

31 31 32 2 0 2 0 2 0

32 32 33 2 0 2 0 2 0

33 33 34 2 0 2 0 2 0

34 34 35 2 0 2 0 2 0

35 35 36 2 0 2 0 2 0

36 36 37 2 0 2 0 2 0

37 37 38 2 0 2 0 2 0

38 38 39 2 0 2 0 2 0

39 39 40 2 0 2 0 2 0

40 40 41 2 0 2 0 2 0

41 41 42 2 0 2 0 2 0

42 42 43 2 0 2 0 2 0

43 43 44 2 0 2 0 2 0

44 44 45 2 0 2 0 2 0

45 45 46 2 0 2 0 2 0

46 46 47 1 0 1 0 1 0

47 47 48 1 0 1 0 1 0

48 48 49 1 0 1 0 1 0

49 49 50 1 0 1 0 1 0

50 50 1 1 0 1 0 1 0

 # Field Points Inside Domain (Field Point No., x-coordinate, y-coordinate):

1 10 2.5

 # End of File

APPENDIX A.2

Sample of second input file

```
500    10    279     279     1.0E-7  ! maxl  levmx   nexp    ntylr  tolerance
50  50000  50000    50   90000000  ! maxia ncellmx nleafmx mxl    nwksz
3                          ! maxt
```

Definitions of the above parameters:

maxt: maximum terms in use from expansion (T =< 5)

maxl: maximum number of elements in a leaf

levmx: maximum number of tree levels

nexp: order of the fastmultipole expansions (p)

ntylr: order of the local expansions (= p, in general)

tolerance: tolerance for convergence used in the iterative solver

maxia: maximum number of parameters

ncellmx: maximum number of cells allowed in the tree

nleafmx: maximum number of leaves allowed in the tree

mxl: maximum dimension of Krylov subspace used in the iterative solver

nwksz: size of the space used to store coefficients in preconditioner

(use default in the code,if value = 0)

c----------