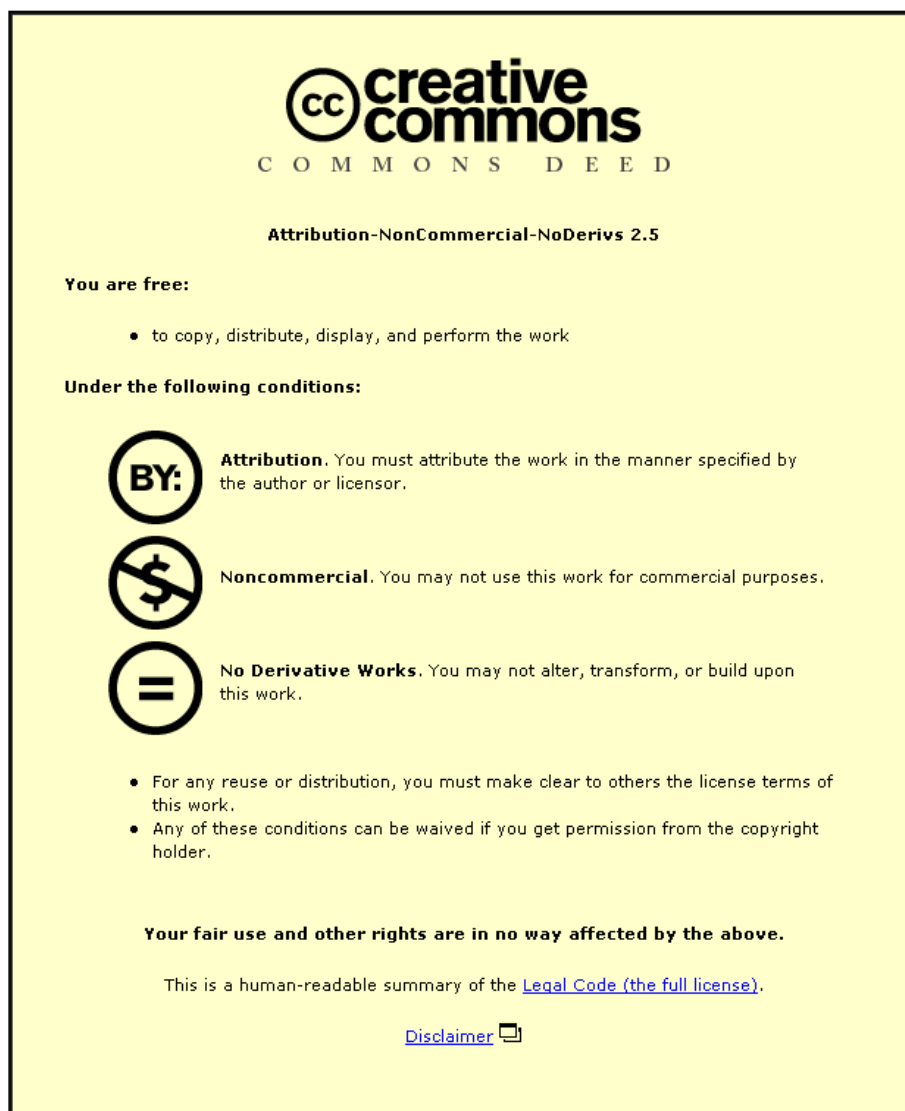


This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (<https://dspace.lboro.ac.uk/>) under the following Creative Commons Licence conditions.



For the full text of this licence, please go to:
<http://creativecommons.org/licenses/by-nc-nd/2.5/>



Pilkington Library

Author/Filing Title MOKHARZADEH, M.R

Accession/Copy No.

Vol. No.

Class Mark T

*Non-copy
open shelf copy*

FOR REFERENCE ONLY

0402085299



BADMINTON PRESS
UNIT 1 BROOK ST
SYSTON
LEICESTER LE7 19

This Thesis is dedicated to :

Mrs. K. Nassehi and Mrs. S. Ghaemi (Saeedi)

A General Global Approximation Method For The Solution Of Boundary Value Problems

by


M.R.Mokhtarzadeh M.Sc.

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for
the award of degree of Doctor of Philosophy of
Loughborough University

August 11, 1998

© by M.R.Mokhtarzadeh 1998

 Loughborough University Book No. <i>1000</i>	
Date	<i>June 99</i>
Class	
Acc No.	<i>040208529</i>

K0648459

Abstract

A general global approximation scheme is developed and its generality is demonstrated by the derivation of classical Lagrange and Hermite interpolation and finite difference and finite element approximations as its special cases. It is also shown that previously reported general approximation techniques which use the idea of moving least square are also special cases of the present method. The combination of the developed general global approximation technique with the weighted residual methods provides a very powerful scheme for the solution of the boundary value problems formulated in terms of differential equations. Although this application is the main purpose of the this project, nevertheless, the power and flexibility of the developed approximation allows it to be used in many other areas. In this study the following applications of the described approximation are developed:

- 1- data fitting (including curve and surface fitting)
- 2- plane mapping (both in cases where a conformal mapping exists and for non-conformal mapping)
- 3- solution of eigenvalue problems with particular application to spectral expansions used in the modal representation of shallow water equations
- 4- solution of ordinary differential equations (including Sturm-Liouville equations, non-homogeneous equations with non-smooth right hand sides and 4th order equations)
- 5- elliptic partial differential equations (including Poisson equations with non-smooth right hand sides)

A computer program which can handle the above applications is developed. This program utilises symbolic, numerical and graphical and the programming language provided by the Mathematica package.

Key words: Diffuse Approximation, Boundary Value Problems, Weighted residual methods, General Global Approximation, Semi-Discretised and Fully Discretised Schemes.

Acknowledgements

I would like to thank my supervisor Dr. V. Nassehi for his help and encouragement during my Ph.D. research.

I am also grateful to Dr. A. Hashemi of Polymer Research Centre Of Iran, Dr. K. D. Watling and Mr. M. Hadian of the Department of Mathematical Science and Dr. A. Kafaee of the Department of Chemical Engineering of Loughborough University for their kind help at various stages of this project.

The main part of this research, which extends the finite element method to new grounds, was carried out at Loughborough during the nine months sabbatical leave which was given to me by the Iran University of Science and Technology (I.U.S.T.) Therefore, although, I.U.S.T. in their wisdom did not agree with my original Ph.D. project I would like to thank them for granting me the leave of absence.

Contents

Contents	i
Abstract	iii
1 Introduction	1
2 A Survey of the Meshless Approximations	5
3 General Global Approximation and Approximation Space	14
3.1 Definition of the General Global Approximation Scheme . . .	15
3.2 Non-Singularity of the Derived ‘Approximation System’	18
3.3 Semi-Discretised and Fully Discretised Schemes	20
3.4 Special Cases of the Defined Approximation	22
3.4.1 Lagrange Interpolation	22
3.4.2 Hermite Interpolation	22
3.4.3 The Finite Element Approximation and Finite Element Space	25
3.4.4 The Finite Difference Approximation	29

3.4.5	Meshless Schemes based on the Moving Least Square Approximation	33
3.5	Implementation of the Developed Approximation	36
3.6	General Diffuse Approximation Method	38
3.7	Computational Strategy	41
4	Applications of the Diffuse Approximation Scheme, Compu- tational Results and Disscusion	44
4.1	Data Fitting	45
4.2	Numerical Mapping	54
4.3	Solution of Ordinary Differential Equations	73
4.3.1	Numerical Solution of Sturm-Liouville Problems	73
4.3.2	Eigenvalue Problems	83
4.3.3	Numerical Solution of Fourth Order Ordinary Differ- ential Equations	87
4.4	Approximation of the Trace Operator	93
4.5	Numerical Solution of Partial Differential Equations	97
4.5.1	Poisson Equation with Homogeneous Boundary Con- ditions	98
4.5.2	Biharmonic Equation with Homogeneous Boundary Con- ditions	106
4.5.3	Three Dimensional Poisson Equation with Homogeneous Boundary Conditions	109
5	Conclusions and Suggestions for Further Developments	111
	References	116
	Appendix	121

Abstract

A general global approximation scheme is developed and its generality is demonstrated by the derivation of classical Lagrange and Hermite interpolation and finite difference and finite element approximations as its special cases. It is also shown that previously reported general approximation techniques which use the idea of moving least square are also special cases of the present method. The combination of the developed general global approximation technique with the weighted residual methods provides a very powerful scheme for the solution of the boundary value problems formulated in terms of differential equations. Although this application is the main purpose of this project, nevertheless, the power and flexibility of the developed approximation allows it to be used in many other areas. In this study the following applications of the described approximation are developed:

- 1- data fitting (including curve and surface fitting)
- 2- plane mapping (both in cases where a conformal mapping exists and for non-conformal mapping)
- 3- solution of eigenvalue problems with particular application to spectral expansions used in the modal representation of shallow water equations
- 4- solution of ordinary differential equations (including Sturm-Liouville equations, non-homogeneous equations with non-smooth right hand sides and 4th order equations)
- 5- elliptic partial differential equations (including Poisson equations with non-smooth right hand sides)

A computer program which can handle the described applications is developed. This program utilises the symbolic language and numerical and graphical capabilities of the Mathematica package.

Chapter 1

Introduction

Mathematical modelling of realistic physical processes usually depends on the formulation of a set of governing differential equations in conjunction with appropriate boundary conditions. In general, the level of complexity of the governing equations in these boundary value problems is such that they can not be solved analytically. Therefore the development of robust and accurate numerical solution schemes for differential equations is of prime importance in most areas of physical sciences. Almost all of the well established numerical solution methods for the differential equations depend on the discretisation of the problem domain into sub-regions. Despite the basic simplicity of formulating piecewise approximations within well defined and finite size sub-regions the process of domain discretisation imposes severe restrictions on the flexibility of these numerical solution schemes. For example, the order of continuity of the approximation depends on the type of the discretisation used and it cannot be improved by mesh refinement. Therefore it is desirable to develop alternative numerical solution strategies for the differential equations which avoid the domain discretisation and instead rely on the global approximation of the unknown functions. In this way suitable approximating

functions which can satisfy required accuracy of the numerical solution can be found in a straightforward manner.

In this project we describe a general global approximation scheme which in conjunction with the weighted residual method provides a very powerful scheme for the numerical solution of differential equations. The power and flexibility of this approximation technique means that it can also be used in a number of other areas besides differential equations such as data fitting, space mapping and eigenvalue problems.

The approximation technique developed in this project is based on the generalisation of the moving least-square method. In recent years a number of 'meshless' numerical solution methods for differential equations have been reported in the literature which also use the moving least square technique. The most notable, and first to appear, amongst these methods are the 'Diffuse Element Method' and the 'Element Free Galerkin Method'. We show that both these methods are special cases of the present 'General Global Approximation scheme'. We further show that the basic finite difference and finite element approximations can also be derived as the special cases of the generalised method described in this thesis.

The present thesis is divided into five chapters. The outline of these chapters is as follows.

Chapter one is an introductory section which briefly describes the main objectives and the achievements of this project.

In chapter two, we review previously reported applications of the moving least-square method and briefly describe the techniques which have been used by other investigators to extend this method to the solution of differential equations. This review provides the basic background for the introduction of our general global approximation scheme.

Chapter three is devoted to the explanation of the mathematical basis of our approximation technique and it includes a thorough description of the development of the present scheme and its associated approximation space. Using theoretical analysis, we show that Lagrange and Hermite interpolation models, finite difference and finite element approximations and previously developed diffuse element and element free Galerkin methods are all special cases of the present scheme. These analyses provide a more fundamental insight about the mathematical basis of these approximation techniques. In chapter three we also describe the details of the computational strategy which is used to carry out the above case studies.

In chapter four we present a number of benchmark case studies to illustrate the applicability of the developed scheme. These applications are as follows:

1. Data Fitting

The developed approximation method is applied to solve a number of curve and surface fitting problems. The flexibility of the scheme is demonstrated by the fitting of non-smooth and discontinuous data. We have proved the accuracy of the approximation by obtaining super-convergent solutions for complex curve fitting problems.

2. Numerical Mapping

We have extended our data fitting algorithms to create a robust plane mapping scheme. This scheme is applied to obtain smooth boundary fitting and domain mappings with high order of continuity. The accuracy of the present mapping scheme is illustrated by the generation of super-convergent mappings in cases where an analytic mapping relation exists.

3. Solution of Differential Equations

The combination of the present general global approximation scheme with the weighted residual techniques provides a powerful method for the solution of differential equations. We have used the Galerkin method to create such a scheme. This scheme is used to solve the following problems

- i* . Sturm-Liouville equations with smooth and non-smooth right hand sides subject to essential (Dirichlet) and mixed boundary conditions,
- ii* . Eigenvalue problems with particular application to the formulation of mathematical models for tidal dynamics based on modal expansions
- iii* . Fourth order ordinary differential equations
- iv* . Transformation of Non-Homogeneous Poisson boundary value problems to homogenous equations using trace theorem
- v* . Solution of two and three dimensional Poisson equations with homogeneous boundary conditions.
- vi* . Solution of biharmonic equations with homogeneous boundary conditions.

In chapter five the conclusions of this research are discussed and a number of suggestions for further extensions of the developed scheme are presented.

The text of the thesis ends with the listing of the references.

The computer programs developed in this research are given as an appendix in this thesis.

Chapter 2

A Survey of the Meshless Approximations

The origin of meshless approximations can be traced back to more than 20 years ago to a paper by L.B.Lucy (1977) who used it to model phenomena without boundaries such as exploding stars and dust clouds. However, because of the limitations of the available computer power at that time the full potential of this concept was not realised until the present decade. In 1991 Nayroles et al developed a method whose aim was the generalisation of the finite element method in a way that it can be used to solve differential equations without relying on domain sub-division. This work was followed by other investigators and a number of approximation techniques which do not depend on the traditional concept of the division of the problem domain to a computational mesh appeared in recent years. In this chapter we present a brief survey of the previously reported meshless approximations and outline their common mathematical background. These approximation methods can be listed in a historical order as

1. 'Diffuse Element Method' developed by (Nayroles, et al, 1991, 1992)
2. 'Element-Free Galerkin Method (EFG)' developed by (Belytschko, et al, 1994-a)
3. 'Reproducing Kernel Particle Methods' (RKPM) developed by (Liu, et al, 1995)
4. 'Hp-clouds' developed by (Duarte and Oden, 1995)
5. 'Partition of Unity Finite Element Method' (PUFEM) developed by (Melenk and Babuska, 1996)
6. 'Finite Point Method' developed (Onate et al, 1996)

Despite the apparent diversity and completely different outlooks of the above methods, it is shown by Belytschko et al (1996) that all of these methods are essentially identical. The only difference is that the partition of unity method provides a more direct route for the generation p-adaptivity (i.e. high order) approximations. However, this is only important for discretised approximations and when a global approach is used, all types of meshless methods can generate approximations with a high degree of smoothness. Because of the theoretical similarity of these methods it can be argued that, essentially, they all have the same basic characteristics. To understand these characteristics it is sufficient to give an analysis of the common mathematical background of these methods. Therefore in this chapter we focus on the explanation of the mathematical foundation of existing meshless approximations. This makes it possible to compare these methods in a general manner with the method developed in this project without being distracted by the details of the individual techniques. Such a comparison is given in chapter three of this thesis.

The main idea under-pinning all meshless approximations is the moving least square method. The moving least square method was developed by Barnhill (1977), McLain (1978), Gordon and Wixom (1978) and Lancaster and Salkauskas (1981). These investigators used the moving least square method to solve curve and surface fitting problems. This method is based on the minimisation of the discrete weighted square of error. Since the weight function can change from point to point in a problem domain, it is called a moving least square technique. In order to develop an approximation scheme on the basis of this method the following procedure was used by Nayroles and his co-workers (1991).

After the selection of points of interest (i.e. nodes) on a problem domain Ω ($\Omega \subseteq R^k$) a global approximation of a function $u : \bar{\Omega} \rightarrow R$ over the nodal points in this domain is formulated by the minimisation of the following functional

$$J[\mathbf{a}(\mathbf{x})] = \sum_{i=1}^n w(\mathbf{x}, \mathbf{x}_i) [\mathbf{p}^T(\mathbf{x}_i) \cdot \mathbf{a}(\mathbf{x}) - u_i]^2 \quad (2.1)$$

In equation (2.1) $\mathbf{x}, \mathbf{x}_i \in R^k$ and $\mathbf{p}(\mathbf{x})$ represents the set of base functions, $\mathbf{a}(\mathbf{x})$ is the M-dimensional array of unknown functions ($[a_1(\mathbf{x}), \dots, a_M(\mathbf{x})]^T$) and n is the number of points in the neighbourhood of \mathbf{x} for which the weight function $w(\mathbf{x}, \mathbf{x}_i) \neq 0$ and u_i is the nodal value of u at $\mathbf{x} = \mathbf{x}_i$. This neighbourhood of \mathbf{x} where $w(\mathbf{x}, \mathbf{x}_i) \neq 0$ is called the domain of influence of \mathbf{x} .

The process of the minimisation of J with respect to $\mathbf{a}(\mathbf{x})$ is shown as

$$\frac{\partial J[\mathbf{a}(\mathbf{x})]}{\partial a_l(\mathbf{x})} = 0 \quad l = 1, 2, \dots, M. \quad (2.2)$$

This leads to the formation of the following set of linear equations

$$\mathbf{A}(\mathbf{x})\mathbf{a}(\mathbf{x}) = \mathbf{B}(\mathbf{x})\mathbf{U} \quad (2.3)$$

Here $\mathbf{A}(\mathbf{x})$ and $\mathbf{B}(\mathbf{x})$ are $M \times M$ and $M \times N$ matrices, respectively, and are expressed as

$$\mathbf{A}(\mathbf{x}) = [\alpha_{rs}(\mathbf{x})] \quad \text{and} \quad \mathbf{B}(\mathbf{x}) = [\beta_{ri}(\mathbf{x})] \quad (2.4)$$

where

$$\alpha_{rs}(\mathbf{x}) = \sum_{i=1}^N w(\mathbf{x}, \mathbf{x}_i) \mathbf{p}_r(\mathbf{x}_i) \mathbf{p}_s(\mathbf{x}_i) \quad r, s = 1, \dots, M \quad (2.5)$$

and

$$\beta_{ri}(\mathbf{x}) = w(\mathbf{x}, \mathbf{x}_i) p_r(\mathbf{x}_i) \quad r = 1, \dots, M \quad i = 1, \dots, N \quad (2.6)$$

The diffuse approximation of u , may now be defined as

$$u^h(\mathbf{x}) = \mathbf{p}(\mathbf{x})^T \mathbf{a}(\mathbf{x}) \quad (2.7)$$

where h represents the discretisation of u using a finite step size h . The substitution of $\mathbf{a}(\mathbf{x})$ from equation (2.3) into equation (2.7) gives

$$u^h(\mathbf{x}) = \sum_{i=1}^N \phi_i(\mathbf{x}) u_i \quad (2.8)$$

Equation (2.8) is the analogous shape function representation of the derived approximation. The incorporation of this approximation with a weighted residual technique provides a variational formulation for field problems.

Nayroles et al (1991, 1992) presented the above method as a generalisation of the finite element technique for the solution of the partial differential equations calling it 'The Diffuse Element Approximation'. This method preserves the local character of finite elements required for getting sparse stiffness matrices. The main disadvantage of this approach is that weight functions with strictly compact support must be used which limits the stability of the numerical solutions. Its main advantages are: no meshing is required, it can

cope with complex geometry of the problem domain generating field variables with regularity, and it is possible to use direct collocation methods due to this regularity.

Following this work, an essentially similar method, was developed by Belytschko et al (1994-a) again as an alternative to the traditional finite element techniques. They called their scheme 'The Element Free Galerkin method'. This method is also applicable to domains with arbitrary shapes and requires only nodal data. The dependent variable and its gradients are assumed to be continuous in the entire domain. The key differences of the element free Galerkin method with the previous formulation introduced by Nayroles and his co-workers are

1. The use of Lagrange multipliers to enforce the essential boundary condition
2. More accurate evaluation of the derivatives
3. The use of higher order quadrature in space

The element free Galerkin method was applied to the solution of a number of problems in heat conduction and elasticity. It was found that the method is very effective for crack growth problems in solid mechanics and results with 1% accuracy were obtained for an example fracture problem. This is because that progressively growing cracks can be easily modelled by an element free method.

Nayroles and his co-workers (1991) and Lu et al (1994) carried out convergence and stability analyses for the meshless approximation techniques. The methods used by these teams of investigators differ in the sense that Nayroles et al based their convergence analysis on the Taylor series expansions and Lu et al. employed Fourier series expansions. Both these analyses showed that

the stability of the meshless approximation schemes mainly depends on the used weight functions.

The following gives the list of weight functions recommended by these investigators. These functions are based on the following definition of the geometrical metric

$$d_i(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_i\| \quad (2.9)$$

i . Conical Weight Function

$$w_i(\mathbf{x}) = \omega_i(\|\mathbf{x} - \mathbf{x}_i\|) = \begin{cases} 1 - [\frac{d_i(\mathbf{x})}{d_{m_i}}]^{2k} & \text{if } d_i(\mathbf{x}) \leq d_{m_i} \\ 0 & \text{if } d_i(\mathbf{x}) > d_{m_i} \end{cases} \quad (2.10)$$

ii . Exponential Weight Function

$$w_i(\mathbf{x}) = \omega_i(\|\mathbf{x} - \mathbf{x}_i\|) = \begin{cases} \frac{e^{-(\frac{d_i(\mathbf{x})}{c})^{2k}} - e^{-(\frac{d_{m_i}}{c})^{2k}}}{1 - e^{-(\frac{d_{m_i}}{c})^{2k}}} & \text{if } d_i(\mathbf{x}) \leq d_{m_i} \\ 0 & \text{if } d_i(\mathbf{x}) > d_{m_i} \end{cases} \quad (2.11)$$

where k, c and d_{m_i} are scalar parameters.

iii . Gauss-Like Weight Function

$$w_i(\mathbf{x}) = \omega_i(\|\mathbf{x} - \mathbf{x}_i\|) = \begin{cases} e^{\ln(\frac{d_i(\mathbf{x})}{d_{m_i}})^p} & \text{if } d_i(\mathbf{x}) \leq d_{m_i} \\ 0 & \text{if } d_i(\mathbf{x}) > d_{m_i} \end{cases} \quad (2.12)$$

iv . Triangular Weight Function

$$w_i(\mathbf{x}) = \omega_i(\|\mathbf{x} - \mathbf{x}_i\|) = \begin{cases} (1 - \frac{d_i(\mathbf{x})}{d_{m_i}}(1 - \epsilon))^k & \text{if } d_i(\mathbf{x}) \leq d_{m_i} \\ 0 & \text{if } d_i(\mathbf{x}) > d_{m_i} \end{cases} \quad (2.13)$$

where k, ϵ , and d_{m_i} are scalar parameters.

Belytschko et al (1994-a) found that exponential weight function, which is essentially the truncated Gauss distribution, performs far better than other

types of weight function. This can be explained by considering the sensitivity of each function to the changes of its domain of influence represented by the value of parameter dm_i .

In a later work Belytschko and co-workers (1994-b) extended their approximation technique to incorporate discontinuous derivatives in meshless formulations. The technique is based on adding an approximation function (shape function) which has a discontinuous first derivative at the discontinuity. The strength of the discontinuity is represented by additional unknowns. In multidimensional problems, the strength of discontinuity is interpolated over the discontinuity line (or surface in three dimensions). The discontinuous approximation has been constructed so that it has compact support, and consequently the resulting discrete equations are sparse and banded. Comparison of the results obtained by this technique with closed-form solutions showed good agreement in both the strength of the discontinuity and the solution away from the discontinuity.

Reproducing kernel moving least-square meshless technique developed by Liu et al (1995) improved the procedure of constructing moving least square interpolation functions. This is based on using the notion of a reproducing kernel which is able to generate any m th order polynomial exactly on an irregular node distribution. These workers also developed an interpolation error estimate to assess the convergence rate of their approximation. They showed that for sufficiently smooth functions the interpolant expansion in terms of sampled values will converge to the original function in the Sobolev norm. An advantage of the incorporation of a reproducing kernel in meshless approximations is that it builds a bridge between the traditional interpolation methods and the spectral methods. This is very important for the construction of modern (next generation) finite element methods. The com-

combination of this procedure with the moving least square creates a powerful facility for the generation of finite elements with C^2 or higher orders of continuity. The main difficulty with the reproducing kernel technique is that it is not computationally efficient.

The extension of the main ideas of the meshless approximation to methods based on partition of unity and hp-clouds are straightforward. Although these methods cannot yet be regarded as mature computational tools, nevertheless, they provide new insight about the capabilities of the meshless approximation. Moreover the use of window functions and wavelets as the weight functions in conjunction with these methods provides promising new techniques for generating convenient solutions for problems involving discontinuities and fast moving fronts.

The main advantages of the described meshless methods can be summarised as

1. In contrast to the traditional numerical approximation methods they do not require preprocessing. In recent years, it has become clear that in linear analysis, mesh generation is a far more time-consuming and expensive task than the assembly and solution of the approximation equations.
2. Because they can generate solutions with a high degree of smoothness they do not need additional postprocessing for the output of field data which are derivatives of the primary-dependent variables. For example in solid analysis very smooth strains and stresses are found directly, whereas in finite element methods, various types of postprocessing, such as L^2 projection, are necessary in order to obtain a smooth stress field suitable for contour plotting.

3. They offer a very powerful method for the accurate incorporation of various types of discontinuities in the solution of wide ranges of problems.
4. These methods have faster rates of convergence than the traditional approximation methods.

In the following chapter we present the generalisation of the approximation scheme based on the moving least square technique. It is self-evident that the present method preserves all of the advantages of the described techniques. Furthermore the developed generalised scheme is computationally efficient and can be implemented using easily-affordable computers.

Chapter 3

General Global Approximation and Approximation Space

In this chapter the mathematical background of the present general global approximation technique and the creation of its corresponding general approximation space are explained. We explore the special cases of this technique and show that the classical interpolation models and mesh-dependent approximations such as the finite difference and the finite element techniques can be derived from this approximation space. We further show that previously reported meshless approximations which are based on moving least square technique can also be derived from the present scheme. These derivations are particularly important from a computational point of view, since they show that a common environment for the implementation of all of the above approximations can be developed. Therefore our developed computational strategy is essentially based upon the concept of the general approximation space which can be easily extended to include other types of numerical schemes.

3.1 Definition of the General Global Approximation Scheme

Let $\Omega \subseteq R^k, k \geq 1$, be an open bounded simply connected domain with a sufficiently smooth boundary $\partial\Omega$ and $\widehat{\Omega} = \Omega \cup \partial\Omega$, called the domain of approximation.

A general approximation space (or simply an approximation space) is a triple of the form

$$(\widehat{\Omega}, \widehat{P}, \widehat{W}) \quad (3.1)$$

where \widehat{P} is a set of M -linearly independent functions in Ω , called the base functions, and $\widehat{W} : R^k \rightarrow R^+$ (real positive numbers), $\widehat{W} \in L^2(\Omega)$, is a weight function generator. Base functions usually consist of polynomials or more generally 'nearly polynomials'. Therefore if we assume that the elements of \widehat{P} are complete polynomials of degree D , at the most, then using the multi-index notation

$$\widehat{P} = \{p_1, \dots, p_M\} \quad (3.2)$$

and

$$p_j(\mathbf{x}) = \sum_{|\alpha| \leq D} c_\alpha \mathbf{x}^\alpha, \quad (3.3)$$

where c_α represents multi-index variable coefficients of the polynomial base functions, α is a k -dimensional array of integers and

$$\begin{aligned} \mathbf{x} &= [x_1, \dots, x_k], \quad \alpha = [\alpha_1, \dots, \alpha_k], \\ \mathbf{x}^\alpha &= x_1^{\alpha_1} \dots x_k^{\alpha_k}, \quad |\alpha| = \alpha_1 + \dots + \alpha_k \end{aligned} \quad (3.4)$$

To define the present general approximation scheme, we consider the following functional

$$J[\mathbf{a}(\mathbf{x})] = \int_{\Omega} W(\mathbf{x}, \xi) [\mathbf{p}(\xi)^T \mathbf{a}(\mathbf{x}) - u(\xi)]^2 d\Omega \quad (3.5)$$

where u is a function such that $u \in L^2(\Omega)$, \mathbf{p} is the array of base functions, and $\mathbf{a}(\mathbf{x})$ is an M -dimensional array of functions in $L^2(\Omega)$, and W is a weight function generated using \widehat{W} . Therefore the approximation scheme based on functional (3.5) can be defined as

$$\text{Find } \mathbf{a}(\mathbf{x}) \text{ such that } J[\mathbf{a}(\mathbf{x})] \text{ is a minimum.} \quad (3.6)$$

Assuming that such an approximation exists, and $\mathbf{a}(\mathbf{x})$ is unique, we define

$$u^a(\mathbf{x}) = \mathbf{p}(\mathbf{x})^T \mathbf{a}(\mathbf{x}) \quad (3.7)$$

where the super-script a in u^a means that u^a is the approximation of u in the approximation space of $(\widehat{\Omega}, \widehat{\mathbf{P}}, \widehat{W})$, subject to the approximation scheme defined by expression (3.6). On the basis of definition (3.6), functional (3.5) may be called a weighted least square of error in $L^2(\Omega)$ sense. In line with the functional (3.5) another functional can be defined which contains the function u , and its first derivatives. In this case we may call it the weighted least square of error in Sobolev sense. Since the weight function W , and functional (3.5), vary from point to point in Ω , the scheme defined by the expression (3.6) may be called a 'Global Moving Least-Square Approximation'.

The first consequence of expression (3.6) is that when $u \in \text{Span}(\widehat{\mathbf{P}})$, then $u^a \equiv u$, where $\text{Span}(\widehat{\mathbf{P}})$ is the set of all linear combinations of the members of $\widehat{\mathbf{P}}$.

In order to find an external value for functional (3.5) in a rigorous manner, we use the notion of strong or Frechet derivative (Linz,1979). Assuming

$\delta \mathbf{a}(\mathbf{x}) \in (L^2(\Omega))^M$ to be an increment of the variable $\mathbf{a}(\mathbf{x})$, we have

$$J[\mathbf{a}(\mathbf{x}) + \delta \mathbf{a}(\mathbf{x})] - J[\mathbf{a}(\mathbf{x})] = T\delta \mathbf{a}(\mathbf{x}) + E(\delta \mathbf{a}(\mathbf{x})) \quad (3.8)$$

where

$$\begin{aligned} T\delta \mathbf{a}(\mathbf{x}) &= 2 \int_{\Omega} W(\mathbf{x}, \xi) \mathbf{p}(\xi)^T \mathbf{a}(\mathbf{x}) \mathbf{p}(\xi)^T \delta \mathbf{a}(\mathbf{x}) d\Omega \\ &\quad - 2 \int_{\Omega} W(\mathbf{x}, \xi) \mathbf{p}(\xi)^T \delta \mathbf{a}(\mathbf{x}) u(\xi) d\Omega \end{aligned} \quad (3.9)$$

and

$$E(\delta \mathbf{a}(\mathbf{x})) = \int_{\Omega} W(\mathbf{x}, \xi) [\mathbf{p}(\xi)^T \delta \mathbf{a}(\mathbf{x})]^2 d\Omega \quad (3.10)$$

It can be easily shown that T is a bounded linear operator. Using Cauchy-Schwarz inequality (Linz, 1979), it can also be shown that

$$E(\delta \mathbf{a}(\mathbf{x})) = o(\|\delta \mathbf{a}(\mathbf{x})\|) \quad (3.11)$$

Therefore, functional (3.5) is strongly differentiable. The sufficient condition for the extremum of J is

$$T\delta \mathbf{a}(\mathbf{x}) = 0 \quad (3.12)$$

for all $\delta \mathbf{a}(\mathbf{x}) \in (L^2(\Omega))^M$. Therefore we have

$$\int_{\Omega} W(\mathbf{x}, \xi) \mathbf{p}(\xi)^T \mathbf{a}(\mathbf{x}) \mathbf{p}(\xi)^T d\Omega = \int_{\Omega} W(\mathbf{x}, \xi) u(\xi) \mathbf{p}(\xi)^T d\Omega \quad (3.13)$$

Equation (3.13) represents a system of $(M \times M)$ linear equations in terms of the unknown function $\mathbf{a}(\mathbf{x})$. This system of equations can be written as

$$\int_{\Omega} W(\mathbf{x}, \xi) \mathbf{p}(\xi) \mathbf{p}(\xi)^T \mathbf{a}(\mathbf{x}) d\Omega = \int_{\Omega} W(\mathbf{x}, \xi) u(\xi) \mathbf{p}(\xi) d\Omega \quad (3.14)$$

or

$$\mathbf{A}(\mathbf{x}) \mathbf{a}(\mathbf{x}) = \mathbf{b}(\mathbf{x}) \quad (3.15)$$

where

$$\mathbf{A}(\mathbf{x}) = \int_{\Omega} W(\mathbf{x}, \xi) \mathbf{p}(\xi) \mathbf{p}(\xi)^T d\Omega \quad (3.16)$$

$$\mathbf{b}(\mathbf{x}) = \int_{\Omega} W(\mathbf{x}, \xi) u(\xi) \mathbf{p}(\xi) d\Omega \quad (3.17)$$

Let

$$\mathbf{A}(\mathbf{x}) = [\hat{a}_{ij}(\mathbf{x})], \quad \mathbf{b}(\mathbf{x}) = [b_1(\mathbf{x}), \dots, b_M(\mathbf{x})]^T, \quad (3.18)$$

and

$$\mathbf{a}(\mathbf{x}) = [a_1(\mathbf{x}), \dots, a_M(\mathbf{x})]^T, \quad (3.19)$$

Equation (3.15) can now be written as

$$\sum_{i=1}^M \hat{a}_{ij}(\mathbf{x}) a_i(\mathbf{x}) = b_j(\mathbf{x}) \quad j = 1 \dots M. \quad (3.20)$$

where

$$\hat{a}_{ij}(\mathbf{x}) = \int_{\Omega} W(\mathbf{x}, \xi) p_i(\xi) p_j(\xi) d\Omega \quad (3.21)$$

$$b_j(\mathbf{x}) = \int_{\Omega} W(\mathbf{x}, \xi) p_j(\xi) u(\xi) d\Omega \quad (3.22)$$

It is obvious that system of equations (3.15) is symmetric. We will refer to this system as the 'General Global Approximation System' or for simplicity the 'Approximation System'.

3.2 Non-Singularity of the Derived 'Approximation System'

Assuming $\mathbf{v} \in R^M$ is an M -dimensional vector, we have

$$\mathbf{v}^T \mathbf{A}(\mathbf{x}) \mathbf{v} = \mathbf{v}^T \left(\int_{\Omega} W(\mathbf{x}, \xi) \mathbf{p}(\xi) \mathbf{p}(\xi)^T d\Omega \right) \mathbf{v}$$

$$\begin{aligned}
&= \int_{\Omega} W(\mathbf{x}, \xi) (\mathbf{v}^T \mathbf{p}(\xi)) (\mathbf{p}(\xi)^T \mathbf{v}) d\Omega \\
&= \int_{\Omega} W(\mathbf{x}, \xi) [\mathbf{v}^T \mathbf{p}(\xi)]^2 d\Omega
\end{aligned} \tag{3.23}$$

which is non-negative (≥ 0). Therefore, for all $\mathbf{x} \in \Omega$, the matrix function $\mathbf{A}(\mathbf{x})$ is a positive semi-definite matrix. Provided that the selected weight function generator is continuous in R^k , we can easily show that the matrix function $\mathbf{A}(\mathbf{x})$ is positive definite. This states that the derived general global approximation exists.

Since the approximation of the function u depends on the created approximation space $(\widehat{\Omega}, \widehat{P}, \widehat{W})$ and the approximation scheme defined by expression (3.6), (i.e. the system of equations (3.15)-(3.17)) it may symbolically be written as

$$(\widehat{\Omega}, \widehat{P}, \widehat{W}, u) \mapsto u^a \tag{3.24}$$

in which u^a is the generalised global approximation of u in the approximation space of $(\widehat{\Omega}, \widehat{P}, \widehat{W})$. Equation (3.14) (or its equivalent system (3.15)-(3.17)), provides a theoretical global approximation scheme. Using this system, in practical problems by the application of a quadrature we can derive a useful approximation scheme. This application can be restricted only to the right hand side of equation (3.15) or it may be extended to the entire equation. The evaluation of the integrals in the governing equations of the present scheme by numerical quadrature imposes a discretisation on these equations. However, the difference between this discretisation, which preserve the global character of the technique, and the domain discretisation, required in mesh dependent approximations, should be emphasised.

3.3 Semi-Discretised and Fully Discretised Schemes

Let \widehat{Q} (or more precisely $\widehat{Q}_{\widehat{\Omega}}$) be a numerical quadrature operator defined on $\widehat{\Omega}$ as

$$\widehat{Q}(f) = \sum_{i=1}^Q \widehat{\omega}_i f(\widehat{\xi}_i) \quad (3.25)$$

If only the right hand side of equation (3.15) (i.e. $\mathbf{b}(\mathbf{x})$ term) are discretised we obtain a semi-discretised approximation scheme. Assuming that $(\widehat{\xi}_i, \widehat{\omega}_i)$, $i = 1, \dots, Q$, are the integration points and weights of the used quadrature, we have

$$\mathbf{b}(\mathbf{x}) = \sum_{i=1}^Q \widehat{\omega}_i W(\mathbf{x}, \widehat{\xi}_i) u(\widehat{\xi}_i) \mathbf{p}(\widehat{\xi}_i) \quad (3.26)$$

This equation can be written as

$$\mathbf{b}(\mathbf{x}) = \widehat{\mathbf{P}}_G \widehat{\mathbf{W}}_D \mathbf{U} \quad (3.27)$$

where $\widehat{\mathbf{P}}_G$ is the Grammian matrix (i.e. a symmetric square matrix), $\widehat{\mathbf{W}}_D$ is a diagonal matrix with diagonal elements $\widehat{\omega}_i W(\mathbf{x}, \widehat{\xi}_i)$ and

$$\mathbf{U} = [u(\widehat{\xi}_1), \dots, u(\widehat{\xi}_Q)]^T, \quad (3.28)$$

is the array of the values of u at the integration points.

In the fully discretised scheme both sides of equation (3.15) are approximated by a numerical quadrature. Thus the fully discretised scheme corresponding to equation (3.15) becomes

$$\mathbf{A}(\mathbf{x}) = \sum_{i=1}^Q \widehat{\omega}_i W(\mathbf{x}, \widehat{\xi}_i) \mathbf{p}(\widehat{\xi}_i) \mathbf{p}(\widehat{\xi}_i)^T \quad (3.29)$$

and

$$\mathbf{b}(\mathbf{x}) = \sum_{i=1}^Q \widehat{\omega}_i W(\mathbf{x}, \widehat{\xi}_i) u(\widehat{\xi}_i) \mathbf{p}(\widehat{\xi}_i) \quad (3.30)$$

$A(x)$ can be factorised to give

$$A(x) = \widehat{P}_G \widehat{W}_D \widehat{P}_G^T \quad (3.31)$$

where \widehat{P}_G is a $(M \times Q)$ Grammian matrix formed by the base functions, \widehat{W}_D is a $(Q \times Q)$ diagonal matrix consisting of the weight functions and weights of the quadrature, and \widehat{P}_G^T which represents the transpose of \widehat{P}_G , is a $(Q \times M)$ matrix.

In the special case of $M = Q$, the non-singularity of the matrix $A(x)$ and the existence of its inverse is guaranteed by the regularity (i.e. all of their components are smooth) of the square Grammian matrix and its transpose, and the positiveness of the weight function. Therefore in this case, at the least, the system defined by equation (15) can be solved to obtain the unknown function $a(x)$ and hence u^a exists. After the discretisation of the original equation by the quadrature operator the approximation space takes the form

$$(\widehat{\Omega}^h, \widehat{P}, \widehat{W}, \widehat{Q}) \quad (3.32)$$

The approximation generated through expression (3.32) is symbolically shown as

$$(\widehat{\Omega}^h, \widehat{P}, \widehat{W}, \widehat{Q}, u) \mapsto u^h \quad (3.33)$$

where u^h is the general discretised diffuse approximation of function u in terms of base functions (analogous to equation 3.7).

3.4 Special Cases of the Defined Approximation

3.4.1 Lagrange Interpolation

Let us consider a domain $\Omega \subseteq R^k, k \geq 1$ and let \hat{N} be the set of nodal points on $\hat{\Omega}$. In this case the approximation space becomes $(\hat{N}, \hat{P}, \hat{W}, \hat{Q})$. We need to determine a relationship for u^h which can provide all information on \hat{N} . Therefore we choose a Reimann sum (Rudin, 1964) as the required quadrature. This automatically transforms all of the information from quadrature points to the nodal points and u^h is defined by

$$u^h(\mathbf{x}) = \mathbf{p}(\mathbf{x})^T \mathbf{a}(\mathbf{x}) \quad (3.34)$$

where $\mathbf{p}(\mathbf{x}) = [p_1(\mathbf{x}), \dots, p_N(\mathbf{x})]^T$. Equation (3.34) provides nodal information for the approximated function and hence it is the Lagrange interpolation model.

3.4.2 Hermite Interpolation

In a generalised Hermite interpolation (Linz, 1979) the values of function and its derivatives up to a certain order must be interpolated at the nodal points. We therefore, need to define a general approximation scheme which is based on a global weighted least square of the error in Sobolev sense. This can be done by the definition of a system of functionals as follows.

Let J_0, J_1 be two functionals defined as

$$J_0[\mathbf{a}(\mathbf{x})] = \int_{\Omega} W_0(\mathbf{x}, \xi) [\mathbf{p}(\xi)^T \mathbf{a}(\mathbf{x}) - u(\xi)]^2 d\Omega \quad (3.35)$$

$$J_1[\mathbf{a}(\mathbf{x})] = \int_{\Omega} W_1(\mathbf{x}, \xi) [\mathbf{p}'(\xi)^T \mathbf{a}(\mathbf{x}) - u'(\xi)]^2 d\Omega \quad (3.36)$$

where W_0 , and W_1 are two weight functions generated from their corresponding weight function generators \widehat{W}_0 , and \widehat{W}_1 . Therefore the approximation scheme which is based on functionals (3.35) and (3.36) may be defined as

$$\text{Find } \mathbf{a}(\mathbf{x}), \text{ such that } J_0[\mathbf{a}(\mathbf{x})] + J_1[\mathbf{a}(\mathbf{x})], \text{ are both minima.} \quad (3.37)$$

or

$$\text{Find } \mathbf{a}(\mathbf{x}), \text{ such that } J_0[\mathbf{a}(\mathbf{x})], \text{ and } J_1[\mathbf{a}(\mathbf{x})], \text{ are a minimum.} \quad (3.38)$$

Assuming that such minimisation exists and $\mathbf{a}(\mathbf{x})$ is unique, then we define

$$u^a(\mathbf{x}) = \mathbf{p}(\mathbf{x})^T \mathbf{a}(\mathbf{x}) \quad (3.39)$$

In this section we use an approximation scheme which is based on formulation (3.38) because it is more robust than the alternative scheme defined by expression (3.37). In expression (3.38) the minimisation of J_0 forces u^a to be "close" to u at the same time that the minimisation of J_1 forces the derivative of u^a to be "close" to the derivative of u . Again using the notion of Frechet derivative we derive two sets of equations analogous to equation (3.15). Thus a global approximation scheme similar to the approximation scheme given by equations (3.16) and (3.17) can be written as

$$\mathbf{A}_0(\mathbf{x})\mathbf{a}(\mathbf{x}) = \mathbf{b}_0(\mathbf{x}) \quad (3.40)$$

$$\mathbf{A}_1(\mathbf{x})\mathbf{a}(\mathbf{x}) = \mathbf{b}_1(\mathbf{x}) \quad (3.41)$$

where

$$\mathbf{A}_0(\mathbf{x}) = \int_{\Omega} W_0(\mathbf{x}, \xi) \mathbf{p}(\xi) \mathbf{p}(\xi)^T d\Omega \quad (3.42)$$

$$\mathbf{A}_1(\mathbf{x}) = \int_{\Omega} W_1(\mathbf{x}, \xi) \mathbf{p}'(\xi) \mathbf{p}'(\xi)^T d\Omega \quad (3.43)$$

$$\mathbf{b}_0(\mathbf{x}) = \int_{\Omega} W_0(\mathbf{x}, \xi) u(\xi) \mathbf{p}(\xi) d\Omega \quad (3.44)$$

$$\mathbf{b}_1(\mathbf{x}) = \int_{\Omega} W_1(\mathbf{x}, \xi) u'(\xi) \mathbf{p}'(\xi) d\Omega \quad (3.45)$$

A discretisation technique identical to the full discretisation method described in section (3.4), can now be applied to obtain

$$\mathbf{A}_0(\mathbf{x}) = \sum_{i=1}^Q \hat{\omega}_i W_0(\mathbf{x}, \hat{\xi}_i) \mathbf{p}(\hat{\xi}_i) \mathbf{p}(\hat{\xi}_i) \quad (3.46)$$

$$\mathbf{A}_1(\mathbf{x}) = \sum_{i=1}^Q \hat{\omega}_i W_1(\mathbf{x}, \hat{\xi}_i) \mathbf{p}'(\hat{\xi}_i) \mathbf{p}'(\hat{\xi}_i) \quad (3.47)$$

$$\mathbf{b}_0(\mathbf{x}) = \sum_{i=1}^Q \hat{\omega}_i W_0(\mathbf{x}, \hat{\xi}_i) u(\hat{\xi}_i) \mathbf{p}(\hat{\xi}_i) \quad (3.48)$$

$$\mathbf{b}_1(\mathbf{x}) = \sum_{i=1}^Q \hat{\omega}_i W_1(\mathbf{x}, \hat{\xi}_i) u'(\hat{\xi}_i) \mathbf{p}'(\hat{\xi}_i) \quad (3.49)$$

An approximation system can now be defined by the following assembly

$$\mathbf{A}(\mathbf{x}) = [\mathbf{A}_0(\mathbf{x}), \mathbf{A}_1(\mathbf{x})]^T \quad (3.50)$$

and

$$\mathbf{b}(\mathbf{x}) = [\mathbf{b}_0(\mathbf{x}), \mathbf{b}_1(\mathbf{x})]^T \quad (3.51)$$

We now consider an approximation space based on the nodal arrangement \hat{N} as

$$(\hat{N}, \hat{P}, \hat{W}_0, \hat{W}_1, \hat{Q}) \quad (3.52)$$

where $\hat{N} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ and $\hat{P} = \{1, x, \dots, x^{2M-1}\}$. It should be noted that in this case \hat{P} is an assembly shown as $[\mathbf{P}_0, \mathbf{P}_1]$. Again using a Reimann sum

as a quadrature operator, we can set up a suitable discretisation scheme in this case as

$$u^h(x) = p_0(x)a(x) \quad (3.53)$$

and

$$u'^h(x) = p_1(x)a(x) \quad (3.54)$$

The above equations give an approximation for the function and its derivative on the nodes and hence they represent the Hermite interpolation model.

3.4.3 The Finite Element Approximation and Finite Element Space

Before we derive the finite element approximation as a special case of the described general global approximation technique we present a brief outline of the basic principles of the finite element procedure. Using this approach a more fundamental insight about the nature of the finite element approximation is gained which makes its relation with the present method clearer. In order to maintain the simplicity of this derivation, without loss of generality, we restrict ourselves to the finite element approximation based on the Lagrange interpolation model.

In a local elemental sense a finite element is a triple (e, π^e, Σ^e) , which is characterised by

- fe 1.* The component e in the above triple is a closed subset of R^k with non-empty interior and smooth boundary. The geometrical space of e is called an element or an element domain

fe 2. The component π^e , in the above triple is a space of real-valued functions defined over the set e . This function space is called the space of shape functions.

fe 3. The third component Σ^e is a finite set of linearly independent linear functionals $\Phi_j, j = 1, \dots, n^e$, defined over the space π^e . These functionals are called nodal variables or the degrees of freedom.

From a geometrical point of view, in most applications e is either a n -simplex, or a n -quadrilateral. Accordingly, the finite elements corresponding to them are called simplex or tensor product finite elements.

The set Σ^e , should have one of the following forms

$$i. \Phi_j(p) = p(\mathbf{a}_j^0), \quad p \in \pi^e$$

$$ii. \Phi_j(p) = Dp(\mathbf{a}_j^1)\xi_{j,r}^1, \quad p \in \pi^e$$

$$iii. \Phi_j(p) = D^2p(\mathbf{a}_j^2)(\xi_{j,r}^2, \xi_{j,s}^2) \quad p \in \pi^e.$$

where the points $\mathbf{a}_j^r, r = 0, 1, 2$, belong to the element e and the non-zero vectors $\xi_{j,r}^1, \xi_{j,r}^2$, and $\xi_{j,s}^2$ are either constructed from the geometry of the finite element, or they are fixed vectors in R^k . The points $\mathbf{a}_j^r, r = 0, 1, 2$, are called nodes of the finite element and in general are denoted by N^e . When all of the degrees of freedom of a finite element are of the form (i) the associated finite element is called a Lagrange element. If they are of the forms (ii), or (iii), they are called Hermite elements.

On a finite element (e, π^e, Σ^e) a sufficiently smooth function u can be approximated as

$$I^e u = \sum_{j=1}^{n^e} \Phi_j(u) p_j. \quad (3.55)$$

The function $I^e u$ is called π^e -interpolation or the local interpolation of the function u .

A global finite element space and its corresponding finite element approximation can now be constructed using the described interpolation scheme. This global approximation is characterised by

fes 1. The first aspect of a global finite element space (which is its most important aspect) is that the set $\widehat{\Omega} = \Omega \cup \partial\Omega$ usually represents a division into a finite number of finite subsets.

fes 2. In a global finite element space π^e contains polynomials or 'nearly polynomial' functions.

fes 3. There exists at least one canonical basis in the global finite element space X^h that its corresponding basis has a minimal support.

Let

$$N^h = \cup_{e \in \tau^h} N^e \quad (3.56)$$

where N^e denotes the set of nodes of e , and N^h is the set of all nodes on $\widehat{\Omega}$, and

$$\tau^h = \{e_1, \dots, e_K\} \quad (3.57)$$

Usually $\widehat{\Omega} \neq \cup_{i=1}^K e_i$, except in the case where $\widehat{\Omega}$ itself is a polyhedral. However, in general the domain $\widehat{\Omega}$ is not a polyhedral and is approximated by a polyhedral $\widehat{\Omega}^h$, $\widehat{\Omega}^h = \cup_{i=1}^K e_i$. The approximated domain $\widehat{\Omega}^h$ is called the discretised domain. This shows that even the use of an accurate finite element interpolation on $\widehat{\Omega}$ does not mean that the order of error is guaranteed to remain equal to the order of the error of interpolation. Now let $(\widehat{\Omega}^h, \widehat{P}^h, \widehat{W}^h)$

be the discretised global approximation space. We consider the following sets of discrete base and weight functions

$$\widehat{P}^h = \{\widehat{P}_1^h, \dots, \widehat{P}_K^h\} \quad (3.58)$$

$$\widehat{W}^h = \{\widehat{W}_1^h, \dots, \widehat{W}_K^h\} \quad (3.59)$$

where \widehat{P}_i^h , and \widehat{W}_i^h are the base and the weight function generators for the element e_i , respectively. Therefore over an element the triple

$$(e_i, \widehat{P}_i^h, \widehat{W}_i^h) \quad i = 1, \dots, K \quad (3.60)$$

defines elemental approximation space. The comparison of equation (3.60) with equation (3.1) reveals essential similarity of the approximation space. Therefore we consider the following functionals

$$J[\mathbf{a}(\mathbf{x})] = \int_{\Omega} W(\mathbf{x}, \xi) [\mathbf{p}(\xi)^T \mathbf{a}(\mathbf{x}) - u(\xi)]^2 d\Omega \quad (3.61)$$

and

$$J^h[\mathbf{a}(\mathbf{x})] = \int_{\Omega^h} W(\mathbf{x}, \xi) [\mathbf{p}(\xi)^T \mathbf{a}(\mathbf{x}) - u_{\Omega^h}^h(\xi)]^2 d\Omega \quad (3.62)$$

where Ω^h is the interior of the $\widehat{\Omega}^h$. We now define an elemental scheme as

$$\text{Find } \mathbf{a}(\mathbf{x}) \text{ such that } J^h[\mathbf{a}(\mathbf{x})] \text{ is a minimum.} \quad (3.63)$$

We can choose a discrete weight function such as

$$w_i^h(\mathbf{x}, \xi) = \begin{cases} 1 & \text{if } \mathbf{x} \in e_i \\ 0 & \text{if } \mathbf{x} \notin e_i \end{cases} \quad (3.64)$$

Thus we have

$$J_i^h[\mathbf{a}(\mathbf{x})] = \int_{\Omega^h} [\mathbf{p}(\xi)^T \mathbf{a}(\mathbf{x}) - u_{\Omega^h}^h(\xi)]^2 d\Omega \quad (3.65)$$

Therefore the approximation scheme can be simplified to

$$\text{Find } \mathbf{a}(\mathbf{x}) \text{ such that } J_i^h[\mathbf{a}(\mathbf{x})] \text{ is a minimum.} \quad (3.66)$$

In general the use of a numerical quadrature say \hat{Q}_i gives a discretised approximation scheme. Thus the use of Reimann sum at nodal points N^{e_i} , which forces the integration points to coincide with nodal points, satisfies the basic properties of the finite element approximation. The result is again an approximation based on a local Lagrange interpolation.

3.4.4 The Finite Difference Approximation

In the development of general global approximation when both sides of equation (3.15) are found by quadrature, the approximated form of a function can be written as a linear combination of function values as

$$u^h(x) = \sum_1^N \phi_i(x) u_i \quad (3.67)$$

where $\phi_i, i = 1, \dots, N$, are called shape functions. This is known as a shape function representation of the approximate function. In the finite element approach a similar linear combination of shape functions is used to represent the function approximations. The main difference between these two techniques is that in the finite element scheme the shape functions originate from interpolating polynomials but as it is shown in the section dealing with the derivation of the general scheme, in the latter scheme the shape functions are found by the minimisation of the originally formed functional. However, because of this similarity the application of both of these approximations to a problem gives a discretised equivalent of the original problem which is transformed to a linear combination of finite dimensional spaces (or sets).

There are other types of discretisation in which the discrete approximation

does not lie in a set (or space of functions). The primary example of such approximations is the finite difference scheme where the approximate function lies in R^K .

To illustrate this point let us consider the following analysis.

Let X, Y be two Banach spaces and $L : X \rightarrow Y$ be a linear operator. Consider the following problem

$$\text{Find } u \in X \text{ such that } Lu = a, \quad a \in Y \quad (3.68)$$

In general the approximate solution u^h lies in a finite dimensional space X^h hence the discretised equivalent of the problem defined by expression (3.68) is

$$\text{Find } u^h \in X^h \text{ such that } L_h u^h = a^h, \quad (3.69)$$

where $L_h : X^h \rightarrow X^h$ is a linear operator. In this analysis the space X^h is not necessarily a subspace of X and the relationship between them must be rigorously defined. This can be done by the use of 'restriction' and 'prolongation' operators (Linz, 1979).

For simplicity, let us assume that Z is a Banach space and X, Y are two subspaces of Z and $L : X \rightarrow Y$ is a linear operator. The sequence of spaces X_n will be assumed to be finite dimensional with $\dim(X_n) = n$. The linear operators $r_n : Z \rightarrow X_n$ and $p_n : X_n \rightarrow Z$ are called restriction and prolongation, respectively, provided that they satisfy the following conditions

$$\sup_n \|r_n\| \leq r, \quad r \in R^+$$

$$\sup_n \|p_n\| \leq p, \quad p \in R^+$$

$$r_n p_n = I_n$$

$$p_n r_n x \rightarrow x$$

$$\|r_n x\| \rightarrow \|x\|$$

An example of these operators are:

$$\begin{aligned} Z &= C[a, b], \quad X_n = R^n \\ a &\leq t_1 < \dots < t_n \leq b \\ r_n x &= x_n = [x(t_1), x(t_2), \dots, x(t_n)]^T \end{aligned}$$

Linear interpolation provides a definition for p_n , that is, $p_n x_n$ is a piecewise linear function such that $p_n x_n(t_j) = x_n(t_j)$.

The above analysis indicates that, unlike the finite element approximation, the derivation of the finite difference scheme from the present general global approximation space cannot be regarded as straightforward and requires the approximation of the differential operators. Therefore, as it is shown in the following, the basic finite difference formulas such as the forward and central differences can only be derived from the general approximation space provided that we use appropriate input.

We consider the approximation of the functions $Du, u \in C^1[a, b]$ and $D^2u, u \in C^2[a, b]$.

In the case of the first function we have the following

$$(Du(x))^a = \mathbf{p}(x)^T \mathbf{a}(x) \quad (3.70)$$

where $\mathbf{a}(x)$ is found by equation (3.15). We again define an approximation space by $(\widehat{\Omega}, \widehat{P}, \widehat{W})$ the required input data for this approximation are given as

$$\widehat{\Omega} = [a, b], \quad \widehat{P} = \{1\}, \quad M = 1,$$

$$w(\mathbf{x}, \xi) = \begin{cases} 1/h & \text{if } \xi \in (x_0, x_0 + h) \\ 0 & \text{if } \xi \notin (x_0, x_0 + h) \end{cases}$$

The substitution of above input in equation (3.15) gives $A(x) = 1$ and

$$b(x) = \int_{x_0}^{x_0+h} \frac{u'(x)}{h} dx = \frac{u(x_0+h) - u(x_0)}{h}$$

which is the forward difference formula.

In the case of the central difference formula we have

$$(D^2 u(x))^a = p(x)^T a(x)$$

and again $a(x)$ is found from equation (3.15). The required input for the approximation space of $(\widehat{\Omega}, \widehat{P}, \widehat{W})$ are

$$\widehat{\Omega} = [a, b], \quad \widehat{P} = \{1\}, \quad M = 1,$$

$$w(x, \xi) = \begin{cases} \frac{\xi - x_0 + h}{h^2} & \text{if } \xi \in (x_0 - h, x_0) \\ \frac{x_0 + h - \xi}{h^2} & \text{if } \xi \in (x_0, x_0 + h) \\ 0 & \text{if } \xi \notin (x_0 - h, x_0 + h) \end{cases}$$

The insertion of these data into equation (3.15) gives $A(x) = 1$ and

$$b(x) = \int_{x_0}^{x_0+h} W(x, \xi) u''(x) d\xi$$

Integration by part gives

$$b(x) = - \int_{x_0}^{x_0+h} W'(x, \xi) u'(x) d\xi$$

and

$$b(x) = \frac{u(x_0+h) - 2u(x_0) + u(x_0-h)}{h^2}$$

which is the central difference formula.

3.4.5 Meshless Schemes based on the Moving Least Square Approximation

A thorough description of the previously developed meshless methods is given in chapter two of the present thesis. The most notable, and first to appear, amongst these methods are the 'Diffuse Element' and 'Element Free Galerkin' schemes. Both these techniques are based on the moving least square method and therefore their relation with the present global approximation is obvious. The derivation of these schemes from the present technique is hence a trivial matter and only requires the application of the Riemann sum to functional (3.5). However, it is important to investigate the consequences of such an approach. Therefore, the following gives a comparison between these methods and the general approach adopted in the present study.

Previously developed meshless methods start from the minimisation of a functional represented as the summation of pointwise functions instead of an integral of weighted least square of error in L^2 . It can hence be said that they are seeking an approximation in l^2 (i.e. discrete L^2 space of real sequences). This apparently minor difference with the present scheme has fundamental implications which severely restrict the generality of the above methods. These limitations can be summarised as:

1. The use of the pointwise sum for the original functional imposes a pre-determined discretisation on the working equations of the scheme and therefore semi-discretised technique cannot be derived from it. In conjunction with this loss of generality, in the pre-discretised schemes the existence of the approximation is not assured and theoretically it should always be based on the use of smooth weight and base functions.

In the present method, however, one can base the entire scheme on L^2 functions. This significantly improves the availability of information required for the convergence and error analyses of the scheme.

2. In order to prove the generality of the present scheme we have considered a number of special cases. Amongst these cases, the derivation of the finite difference operators is particularly significant. The finite difference method represents an entirely distinct type of approximation from the methods which yield function approximations and the derivation of its formulas depends on the integration of the non-discretised form of equation (3.15). Therefore pre-discretised meshless methods, which do not start with the integral form of the original functional, cannot generate finite difference formulas.

A tentative result of the derivation of the finite difference operators from the moving least square scheme is that it may be possible to develop a new and direct proof for the Taylor series expansion on the basis of this analysis.

3. Depending on the extent of the support for the selected weight functions the general approximation can yield a global or a local scheme. If weight functions with compact support are used the scheme generates a localised approximation. Starting from a pre-discretised functional one can only obtain a discrete local technique if weight functions with compact support are used. However, the use of weight functions with compact support in the context of a continuous functional results in a continuous local approximation. This can be utilised to develop a more flexible practical scheme.

4. Although after full discretisation we obtain an apparently identical scheme to the 'Diffuse Element' or 'Element Free Galerkin' approximations, nevertheless, the fundamental difference between these methods and the present scheme remain in force. This is because that despite discretisation we can still use a wider range of weight and base functions. This facility is particularly important in the application of the diffuse approximation to the solution of non-homogenous differential equations requiring generalised solutions. The use of non-smooth weight and base functions, necessary for the generation of singular or generalised solutions for differential equations, cannot be justified in the pre-discretised schemes.
5. Starting from a pre-discretised functional the analysis of the relationship between various nodal arrangements and the accuracy of the final result becomes impossible. Previous investigators using such methods have noticed this problem concluding that the best choice in all problems is a uniform node distribution. The reason for this becomes clear if we consider that the working equations of the discretised scheme is obtained after converting the original relationship expressed in terms of an integral to a summation. Therefore the result obtained using uniformly spaced sampling points can be as accurate as using any other type of nodal distributions in a summation operation.
6. In the present technique we are free to choose the quadrature method before discretisation. Therefore we can select a technique which is most suitable to a particular problem. In the pre-discretised schemes this is not possible because the initial discretisation of the original functional excludes the use of any quadrature method other than Reimann sum.

At the practical level, after the definition of the original functional as a summation and its subsequent minimisation, we obtain the pre-discretised approximation as

$$u^d(\mathbf{x}) = \mathbf{p}(\mathbf{x})^T \mathbf{a}(\mathbf{x}) \quad (3.71)$$

where superscript d in u^d is used to distinguish this result from equation (3.33).

3.5 Implementation of the Developed Approximation

In order to generate the diffuse approximation of a function u i.e. u^h the following set of input data is required:

1. Total number of nodes (N) and the co-ordinates of each node on $\hat{\Omega}$
2. The algebraic dimension of the base functions (M), and
3. The weight function generator (\widehat{W}) and the generated weight function (W).

In the computational algorithm developed in this work we can use the following options regarding the necessary input data

1. Total number of nodes (N) in conjunction with the following nodal arrangements
 - i. Uniformly distributed nodes.
 - ii. Nodes located at positions corresponding to the roots of Legendre polynomials.

- iii. Nodes located at positions corresponding to the roots of Chebyshev polynomials.
2. Algebraic dimension of the base functions (M) in conjunction with
- i. Complete polynomials of degree D .
 - ii. Complete polynomials of degree D , with values zero on $\partial\Omega$.
 - iii. Polynomials combined with other types of functions such as trigonometric, Logarithmic, unit step function etc.
3. Weight functions generated using appropriate weight function generators
- i. Unit weight function, generated from $\widehat{W}(\mathbf{x}) = 1$.
 - ii. Power weight functions, generated from

$$\widehat{W}(\mathbf{x}) = [1 - (\frac{\|\mathbf{x}\|}{d_m})^{k_1}]^{k_2} \quad (3.72)$$

where $d_m \in R^+$, $k_1, k_2 \in R^+$.

- iii. Gauss-like weight functions, generated by

$$\widehat{W}(\mathbf{x}) = \text{Exp}[-(\frac{\|\mathbf{x}\|}{d_m})^k], \quad (3.73)$$

where $d_m \in R^+$, $k \in R^+$.

- iv. Rational weight functions, generated by

$$\widehat{W}(\mathbf{x}) = [1 + (\frac{\|\mathbf{x}\|}{d_m})^{k_1}]^{-k_2} \quad (3.74)$$

where $d_m \in R^+$, $k_1, k_2 \in R^+$.

In the above definitions $\|\cdot\|$, denotes the Euclidean norm of the space R^k . As it is described in section 3.2 the existence and the smoothness of the diffuse

approximations primarily depend on the selected weight functions which are positive, i.e. $\widehat{W} : R^k \rightarrow R^+$. If this condition is fulfilled then it can be shown that there exists at least one family of weight functions which can be used to generate a diffuse approximation for the unknowns. In practice, in order to increase the efficiency and of the computations and the smoothness of the approximation we also impose the following conditions on the weight functions.

- a. The weight functions should be normalised, i.e. $\widehat{W}(0) = 1$
- b. The weight functions should be smooth enough to ensure the continuity of the approximation, i.e. the required continuous differentiability of u^h should be guaranteed.

3.6 General Diffuse Approximation Method

The generality of the present approximation scheme means that it can be used in a number of different families of problems. These applications are listed in the introductory chapter and they are explained in detail in chapter four of this thesis. However, the main application of the developed diffuse approximation scheme is in the solution of differential equations representing boundary value problems. Therefore in this section we describe the incorporation of this technique and the weighted residual schemes which results in the creation of a very powerful numerical solution method for differential equations. This method is explained through the following abstract variational problem.

Consider the general variational problem defined as (Ciarlet, 1978),

$$\text{Find } u \in V, \text{ such that } a(u, v) = F(v) \text{ for all } v \in V \quad (3.75)$$

where V is a Hilbert space with inner product $\langle \cdot, \cdot \rangle$, a is a continuous V -elliptic bi-linear form on $V \times V$ and F is a continuous linear form on V . The existence and uniqueness of the variational problem is guaranteed by the Lax-Milgram theorem (Kesavan, 1989). The discretised variational problem corresponding to the problem defined by the expression (3.80) can be defined as

$$\text{Find } u^h \in V^h, \text{ such that } a(u^h, v^h) = F(v^h) \text{ for all } v^h \in V^h \quad (3.76)$$

where V^h is a sub-space of V . Again the existence and uniqueness of the discretised variational problem can be proved by the Lax-Milgram theorem. In most realistic problems the bi-linear form a , contains the derivatives of u^h , and v^h . Therefore to solve problem (3.76) a closed form for the evaluation of the derivatives of u^h is required. We recall the following equations derived in chapter 3

$$u^h = \mathbf{p}(\mathbf{x})\mathbf{a}(\mathbf{x}) \quad \text{and} \quad \mathbf{A}(\mathbf{x})\mathbf{a}(\mathbf{x}) = \mathbf{B}(\mathbf{x})\mathbf{U}$$

Let us, for simplicity, consider the one-dimensional case we have

$$\mathbf{a}(x) = \mathbf{A}(x)^{-1}\mathbf{B}(x)\mathbf{U} \quad (3.77)$$

and

$$u^h(x) = \mathbf{p}(x)^T \mathbf{A}(x)^{-1} \mathbf{B}(x) \mathbf{U} \quad (3.78)$$

The differentiation of the right hand side of equation (3.78) involves finding the derivative of $\mathbf{A}(x)^{-1}$. This derivative can be found indirectly by the differentiation of the product $\mathbf{A}(x)\mathbf{A}(x)^{-1} = \mathbf{I}$ as

$$\frac{d}{dx} \mathbf{A}(x)^{-1} = -\mathbf{A}(x)^{-1} \left(\frac{d}{dx} \mathbf{A}(x) \right) \mathbf{A}(x)^{-1} \quad (3.79)$$

A more rigorous approach based on the use of the strong or Frechet differentiation yields an identical result for the derivative of $\mathbf{A}(x)^{-1}$. Therefore the function u^h , and its successive derivative is expressed as

$$u^h(x) = \sum_{j=1}^N \phi_j(x) u_j \quad (3.80)$$

and

$$\frac{d}{dx} u^h(x) = \frac{d}{dx} \sum_{j=1}^N \phi_j(x) u_j \quad (3.81)$$

and

$$\frac{d^2}{dx^2} u^h(x) = \frac{d^2}{dx^2} \sum_{j=1}^N \phi_j(x) u_j \quad (3.82)$$

Assuming that

$$V^h = \text{Span}\{\phi_j, j = 1, \dots, N\} \quad (3.83)$$

the discretised variational problem can now be written as

$$\sum_{j=1}^N a(\phi_i, \phi_j) = F(\phi_j) \quad j = 1, \dots, N \quad (3.84)$$

Equation (3.83) is the working equation of the present diffuse approximation method. In cases where the original variational problem involves differential operators equation (3.83) represents a weak formulation describing a weighted residual statement. For example if v is selected to be identical to the shape functions $(\phi_j(x))$ equation (3.83) gives the standard Galerkin method (Ciarlet, 1978). In general, the integration of the discretised differential operators in such cases involves the application of a quadrature. We refer to this integration as the secondary quadrature. In the present scheme the secondary quadrature can be based on a different method to the quadrature required in the generation of the approximation itself.

The symmetry and positive definiteness of the coefficient matrix in the working equation (3.83) is assured by the symmetry and positive definiteness of the bi-linear form in definition (3.75). This means that the described method is guaranteed to give a unique solution for the unknowns u_i .

3.7 Computational Strategy

The diffuse approximation scheme offers a wide range of options for the use of different combinations of base and weight functions in conjunction with various nodal point arrangements in any desired application. The availability of many degrees of freedom, implied in the definition of the diffuse approximation space as $(\widehat{\Omega}, \widehat{P}, \widehat{W})$, enhances the attraction and usefulness of the scheme. In practice, however, the use of trial and error to find a combination of $(\widehat{\Omega}, \widehat{P}, \widehat{W}, \widehat{Q})$ which can be sure to provide the most suitable input for a given problem becomes very tedious. Our aim in the present work has been to develop a computational strategy which resolves this problem. Such a strategy enables the user to find the best combination of input parameters with certainty and ease in every application. Therefore the designed computational algorithm includes an automatic checking routine. This routine is based on a nested loop algorithm for the random selection and combination of the parameters during the implementation of the scheme. The user initiates the test loops by assigning the problem category and the range of the parameters in $(\widehat{\Omega}, \widehat{P}, \widehat{W}, \widehat{Q})$. These computations are terminated after a prescribed number of tests. At the end of this routine the account of the performance of the scheme is given as the output. Guided by this output the user can make a decision either to continue the application by further interactive refinement of the input parameters or he can abandon the application

and start with a new set of $(\widehat{\Omega}, \widehat{P}, \widehat{W}, \widehat{Q})$.

In cases where an analytical solution is known the main reason for the implementation of the scheme is the evaluation of the accuracy of the diffuse approximation. In these applications the described checking algorithm shows the numerical errors found by particular combinations of the model parameters. This gives a simple guide to judge the performance of the scheme. We have used discrete uniform, discrete and discrete norms of error to give more than one measure for the accuracy of the computations. In general, however, the analytical solutions are not known and in order to have a logical basis for the evaluation of the performance of the scheme a different approach is adopted. The main idea is to compare the results found using the formulated equation for $u^h(\mathbf{x})$ with those found directly through the minimisation of the original functional. This approach consists of the following steps.

Step1. Shown symbolically as

$$(\widehat{N}, \widehat{P}, \widehat{W}, \widehat{U}) \leftrightarrow (\widehat{N} \cup \widehat{A}, \widehat{R})$$

where represents additional points in which using equation (2.6) the results are calculated.

Step2. Shown symbolically as

$$(\widehat{N}, \widehat{P}, \widehat{W}, \widehat{U}) \leftrightarrow (\widehat{N} \cup \widehat{A}, \widehat{R})$$

Step3. The convergence criteria is found as $\| \widehat{R}_1 - \widehat{R} \|$

If it is required the above steps can be repeated for a number of cycles.

The described computational environment provides a simple data input facility and yields the output both in numerical and graphical forms. An additional advantage of the random checking algorithm is that it enables the

user to test the applicability and accuracy of the scheme with relative ease in a large number of different types of practical problems.

In order to implement the described computational strategy in the present study we required an environment capable of providing facilities for symbolic, numerical and graphical manipulations as well as a language for programming. Such an environment can be created by Mathematica or any other computer package specially designed to have these facilities. We have utilised Mathematica to develop the computer codes required in this research. The main structure of the developed program is shown in figure (3.1).

Chapter 4

Applications of the Diffuse Approximation Scheme, Computational Results and Discussion

In this chapter details of the application of the developed diffuse approximation technique to various problems are explained. In all of these applications it has been shown that one of the following three different cases may occur.

Case 1. The use of the diffuse approximation leads to a super-convergent result. This means that by using sufficient precision in computations the numerical error of approximation can be made to be as small as desired. In the special case that the computations are based on the rational mode calculations, the approximation yields analytical solutions.

Case 2. When the real mode calculations are used the magnitude of the numerical error depends on the input data, e.g. N , \hat{N} , etc. However, it is important to note that in this case the outcome of the approximation can be

controlled and depending on the availability of the computer power by the selection of appropriate input, an acceptable solution can be found.

Case 3. There are instances where the use of the diffuse approximation in its standard form fails to generate a correct solution. However, using the general approximation method developed in this work, we can always find theoretical reasons for such failures. In this chapter a number of benchmark problems have been included to demonstrate the failure of the standard scheme. It is shown that by the extension of the present scheme it is still possible to generate satisfactory results for these problems.

4.1 Data Fitting

After the construction of a diffuse approximation space any data fitting problem in this space can be defined in the form of the following object

$$object = \{das, data, rda\} \quad (4.1)$$

In this definition $das = (\hat{N}, \hat{P}, \hat{W})$, is the diffuse approximation space consisting of the ordered set of distinct nodal points, the set of the base functions and the weight function generator, respectively. $data = \hat{U}$, is the given numerical data in the problem (i.e. function values). $rda = \{\tilde{X}, \tilde{R}\}$, is the result of the approximation consisting of the nodal point co-ordinates and the computational results, respectively. We can choose a wide variety of nodal point arrangements, base functions and weight functions to carry out the described approximation. However, these selections should not be made arbitrarily and the nature of the problem to be solved should be used as a guide to make the most appropriate choices. This does not restrict the flexibility of the technique and as it is shown later, a large number of options are

always available which make it possible to find a good solution in majority of cases.

Using symbolic notations the entire data fitting scheme can be shown as

$$(\hat{N}, \hat{P}, \hat{W}, u) \mapsto \{\tilde{X}, \tilde{R}\} \quad (4.2)$$

where \tilde{X} represents the points at which a result for the function is sought and \tilde{R} is the set of the values of the fitted function at these points. The insertion of one, two or three-dimensional nodal data into \hat{N} in symbolic relationship (4.2) leads to curve, surface, and volume fittings, respectively.

Test Problem No. 1 Application of the diffuse approximation technique for obtaining a fit for

$$f(x) = \sin(\pi x^2), \quad 0 \leq x \leq 1 \quad (4.3)$$

Based on the described computational procedure the following options are used to solve this problem:

$$\begin{cases} \hat{W} = \text{power function} \\ \hat{P} = \text{polynomial functions} \\ \hat{N} = 15 \end{cases} \quad (4.4)$$

After the refinement of the nodal array the diffuse approximation results are found in 29 nodes. Figure 4.1a shows the comparison of the numerical results with the analytical curve using polynomial base functions of degree 4. The use of the other types of weight functions does not affect the results. However, as it can be seen in figure 4.1b the numerical results improve very significantly after using polynomials of degree 9 as the base functions. This test gives a simple example in which the sought result is smooth.

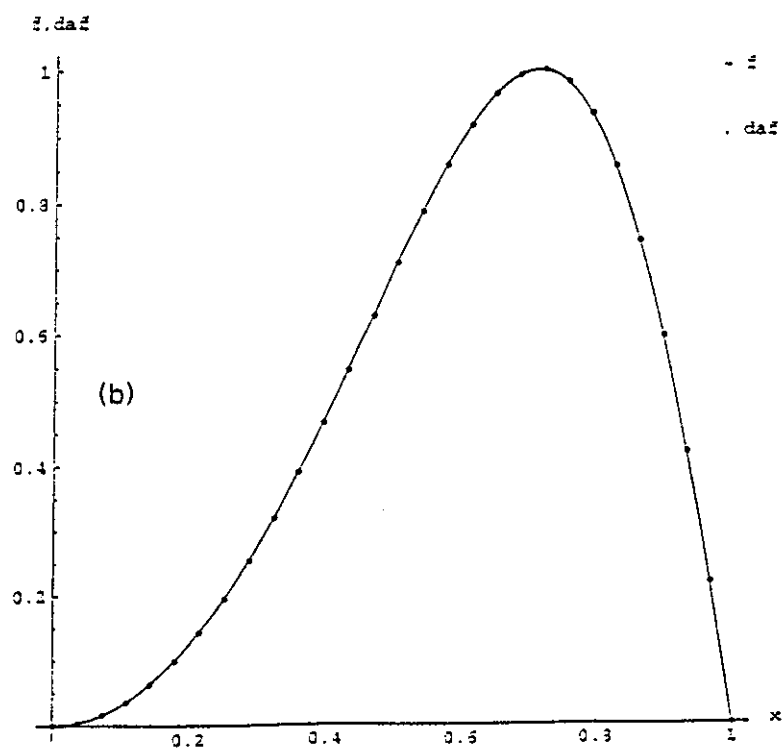
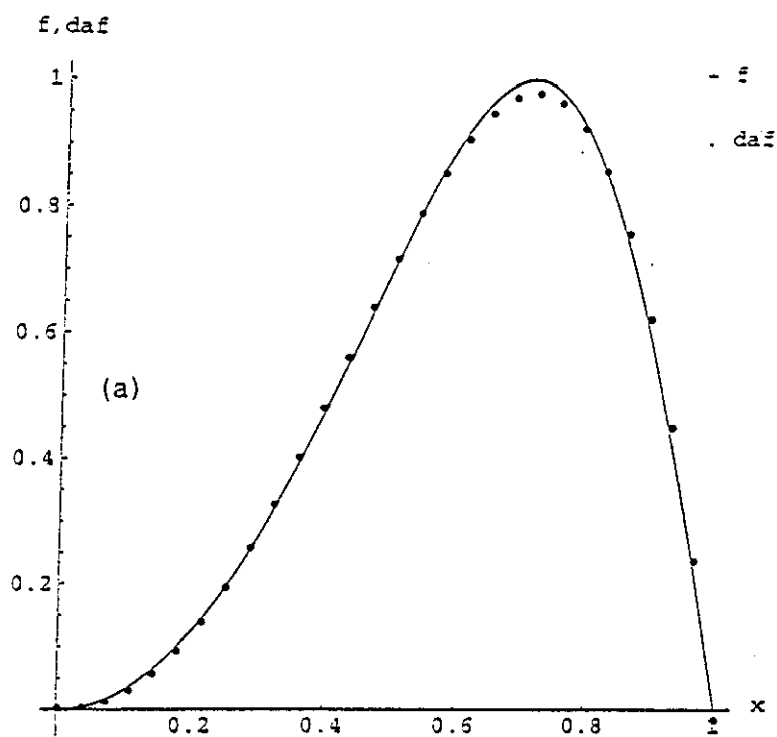


Fig. 4.1

Test Problem No. 2 Application of the diffuse approximation technique for obtaining a fit for

$$f(x) = \text{Sin}(\pi x^2) + U_{\frac{1}{2}}(x), \quad 0 \leq x \leq 1 \quad (4.5)$$

This problem represents a difficult case in which the sought result is not smooth and at point $x = \frac{1}{2}$ it has a discontinuity. Therefore, the use of standard diffuse approximation based on the polynomial base functions cannot yield an acceptable solution in this case.

Figure 4.1c shows the comparison of the diffuse approximation results with the analytical curve using the same input as in the previous test with polynomial base functions of degree 9. In order to obtain a super-convergent result in this case a set of base functions formed by the addition of unit step function with polynomials of degree 4 were used. Figure 4.1d shows the super-convergent result obtained by the application of the diffuse approximation technique in this problem. The main purpose of this example is to demonstrate the power of the diffuse approximation method in dealing with non-regular and discontinuous functions.

The generalised solution of most types of differential equations representing boundary value problems are expected to have non-regular forms.

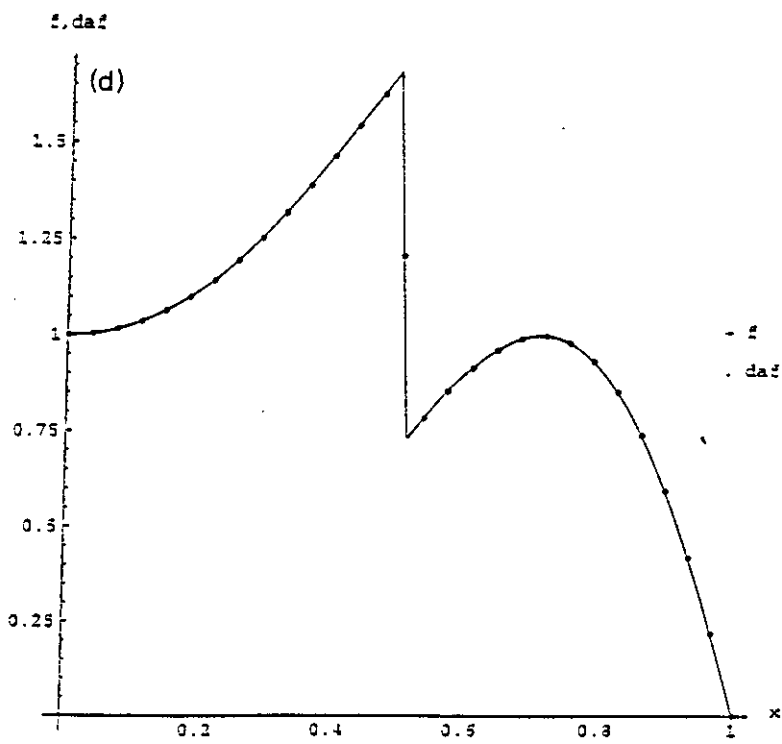
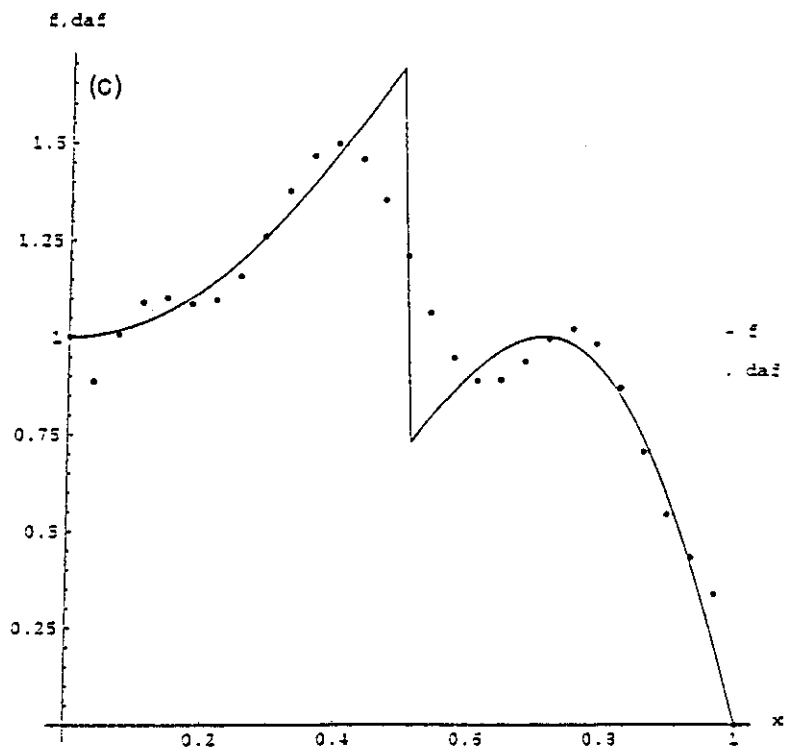


Fig. 4.1

Test Problem No. 3 Application of the diffuse approximation technique for obtaining a fit for

$$f(x_1, x_2) = x_1^2 + x_1x_2 + x_2^2, \quad 0 \leq x_1, x_2 \leq 1 \quad (4.6)$$

Figure 4.1e shows the defined surface.

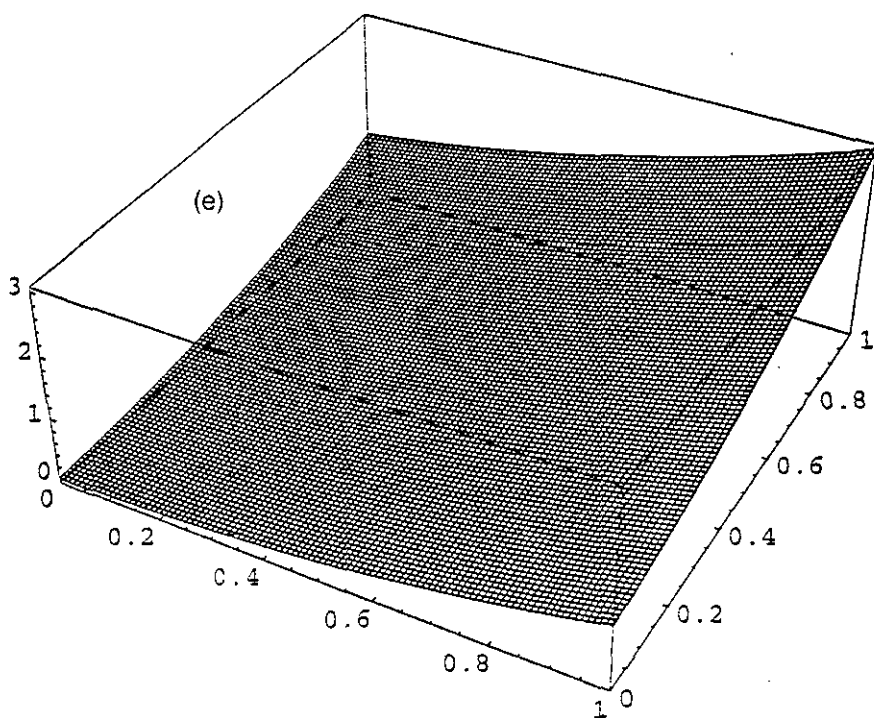


Fig. 4.1

The problem specifications and options used to solve this problem are shown in table 4.1. This table is directly printed from the computer screen and provides an example for the information which in addition to the numerical and graphical results is yielded by the developed program.

Table 4.1: Problem Specifications, Error norms and Computations time. Test Problem No. 3

```

MINIMUM VALUE OF THE VARIABLE[x]          minx1:= 0
MAXIMUM VALUE OF THE VARIABLE[x]          maxx1:= 1
MINIMUM VALUE OF THE VARIABLE[x]          minx2:= 0
MAXIMUM VALUE OF THE VARIABLE[x]          maxx2:= 1
dom:={{0, 1}, {0, 1}}
TOTAL NUMBER OF NODES ON x1 DIRECTION [tnnx1]  tnnx1:= 3
TOTAL NUMBER OF NODES ON x2 DIRECTION [tnnx2]  tnnx2:= 4
tnn:= 12
REFINEMENT FACTOR FOR THE GENERATION OF NEW NODES [$reff]
$reff:={$reffx1,$reffx2} $reff:= {2, 2}
DEGREE OF BASE FUNCTIONS WITH RESPECT TO x1 [degx1] degx1:= 2
DEGREE OF BASE FUNCTIONS WITH RESPECT TO x2 [degx2] degx2:= 2
dim:=9
SELECTED WEIGHT FUNCTION [wf] IS: pwf
EXPLICIT FORM OF THE FUNCTION [f] IS:

$$f[x1,x2]:= x1^2 + x1 x2 + x2^2$$


```

```

DU-error:=4.94049 10-13
DU-d1error:=9.76996 10-13
DU-d2error:=1.47904 10-12
mindet:=7.48682 10-8

```

1186.18 Second

In order to give a clearer comparison between the numerical and the actual results, in figures 4.1f-4.1j we show the cross sections of the above surface with the points found by the diffuse approximation scheme. As it can be seen from table 4.1 the super-convergent results are found using the base functions formed by the tensor product of quadratic polynomials in x_1 and x_2 .

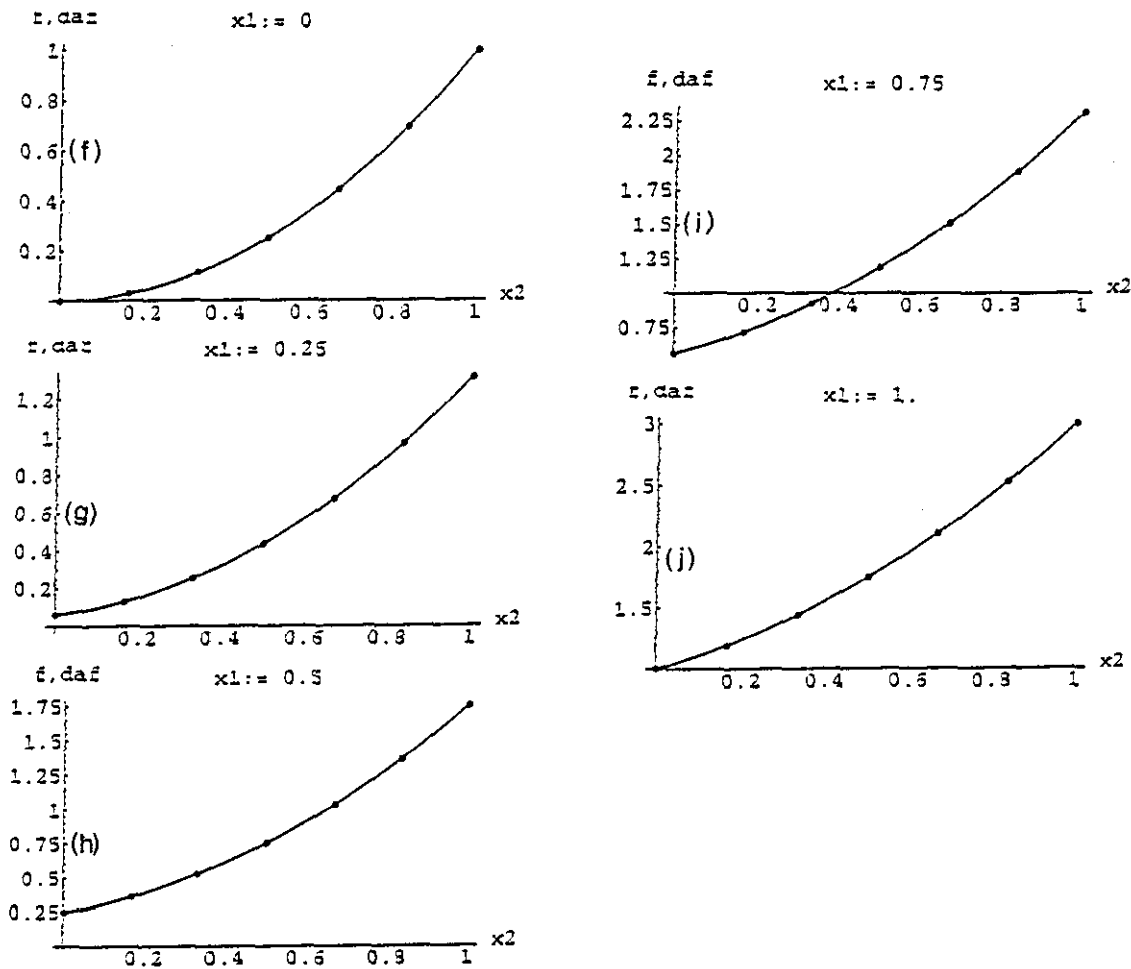


Fig. 4.1

If we use linear polynomials in x_1 and x_2 the obtained fit becomes very inaccurate. Figures 4.1k-4.1m give examples of the latter results. The facility to use symbolic manipulations in all stages of the developed program provides a very convenient means to carry out the required trial and error procedures for obtaining more accurate results in a very short time.

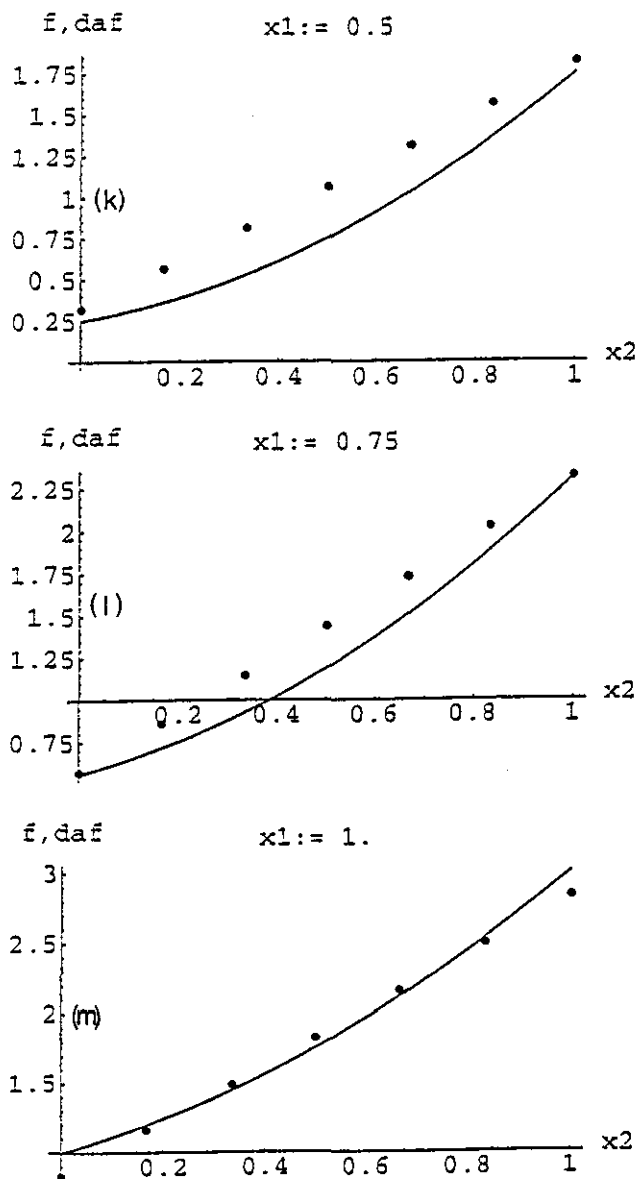


Fig. 4.1

4.2 Numerical Mapping

There are many instances in the numerical modelling of engineering problems that the transformation (mapping) of the problem domain to another system provides a means for obtaining a better and more accurate solution for the governing equations of the model. However, it is seldom possible to find explicit analytical transformations for the generation of mappings from an original problem domain to a suitable solution space. Numerical techniques offer an alternative method for the construction of useful transformations required in most types of engineering problems. Using these methods the numerical values of the mapping functions are found and a point-wise transformation between the original and the target domains is established.

The development of reliable numerical mapping schemes which can transform irregular physical domains to simpler computational domains has been an important topic of research for many investigators in the past. A substantial review article by Thompson and Warsi (1982) list more than 300 research papers published on this subject during the three decades before the 80 s. Apparently, the wide-spread application of the finite element method which through the use of isoparametric elements can cope with curved boundaries has been a disincentive for research in this area in more recent years. However, previously developed methods have continued to be used in numerical grid generation schemes (Han et al, 1992).

Isoparametric finite elements are the most commonly used numerical tools for dealing with irregular problem domains. In the context of the finite element technique these elements are generated by a non-conformal mapping

between an element with curved boundaries and a master element of regular shape (Zienkiewicz and Taylor, 1994). However, in the majority of finite element applications the used isoparametric mapping is based on the Lagrange elements which have a low order of smoothness. The complexity of the development of similar transformations by Hermite elements has prevented the wide-spread utilisation of smoother mappings in the finite element context. Theoretical analysis of C^1 continuous isoparametric Hermite elements are given by Ciarlet and Raviart (1972), Ciarlet (1978) and Brenner and Scott (1994). However, these investigators do not propose a practical algorithm for the generation of these elements. Lapidus and Pinder (1982) give an explicit representation for a type of isoparametric Hermite element which is only C^0 continuous. Thus it is doubtful that this element can be used to construct smooth mappings. Petera and Pittman (1994) combined a Lagrange mapping with a minimisation to develop a numerical method for the generation of continuous isoparametric Hermite elements. The technique proposed by these investigators is complicated and only gives a transformation for the interior of a problem domain. On the boundaries of the domain the numerical values of the functions and their derivatives should be found by the use of a mollifier. This presents an additional difficulty and imposes a restriction on the ease of the application of the method.

In this section we present a new method for the generation of smooth numerical mappings which is based on the use of the diffuse approximation technique. This technique is directly based on the described data fitting scheme.

In order to simplify the explanation of the application of the described data fitting scheme in mapping problems we introduce the following operator equa-

tion

$$u^h(\mathbf{x}) = da[\widehat{N}, \widehat{P}, \widehat{W}, \widehat{U}] \quad (4.7)$$

Therefore the set of the results, \widetilde{R} , can be interpreted as the values of the function u^h generated by the approximation at every point of the set \widetilde{X} (corresponding to data fitting scheme represented by equation 4.2). Consider a domain such as Ω_S defined over a k dimensional space ($\Omega_S \subseteq R^k$). we refer to Ω_S as the source domain. We also define a corresponding target domain called Ω_T , ($\Omega_T \subseteq R^k$). The boundaries of Ω_S and Ω_T are denoted by $\partial\Omega_S$ and $\partial\Omega_T$, respectively. The mapping between the source and the target domains is defined in terms of a node-to-node transformation. If we wish to generate a boundary-fitted transformation then we choose an identical number of nodal points on $\partial\Omega_S$ and $\partial\Omega_T$. These nodes are designated, respectively, as N_S and N_T according to a pre-specified order. This is achieved by the repeated implementation of the data fitting operator (4.7). In this operation, $\widehat{N} = N_S$ and \widehat{U} is the j th component of the desired map. The described mapping can be shown symbolically as

$$da : \{\widehat{N}_S, \widehat{P}, \widehat{W}, \widehat{U}_j\} \mapsto \{\widehat{N}_S, \widetilde{U}_j\} \quad j = 1, \dots, k. \quad (4.8)$$

A similar approach should be adopted if the aim of the mapping is to generate a computational grid which covers the target domain. In this case the source and the target nodes i.e. N_S and N_T should be selected to be within Ω_S and Ω_S as well as on $\partial\Omega_S$ and $\partial\Omega_S$. In plane mapping problem, $k = 2$, and hence we need to apply the operator twice. Similarly in the volume fitting problems the operator is used three times ($k = 3$). The most important application of the described mapping technique is in the generation of computational grids which represent the entire problem domain and boundary fitting is only envisaged to have a limited use in the solution of boundary integral problems.

The freedom to use smooth weight functions is the most important aspect of the present scheme which allows the generation of smooth mappings. This becomes specially important in the generation of space mapping of order C^1 and higher.

Test Problem No. 1 Application of the diffuse approximation technique in plane mapping.

We consider an annular domain as is shown in figure 4.2a with its corresponding target domain shown in figure 4.2b.

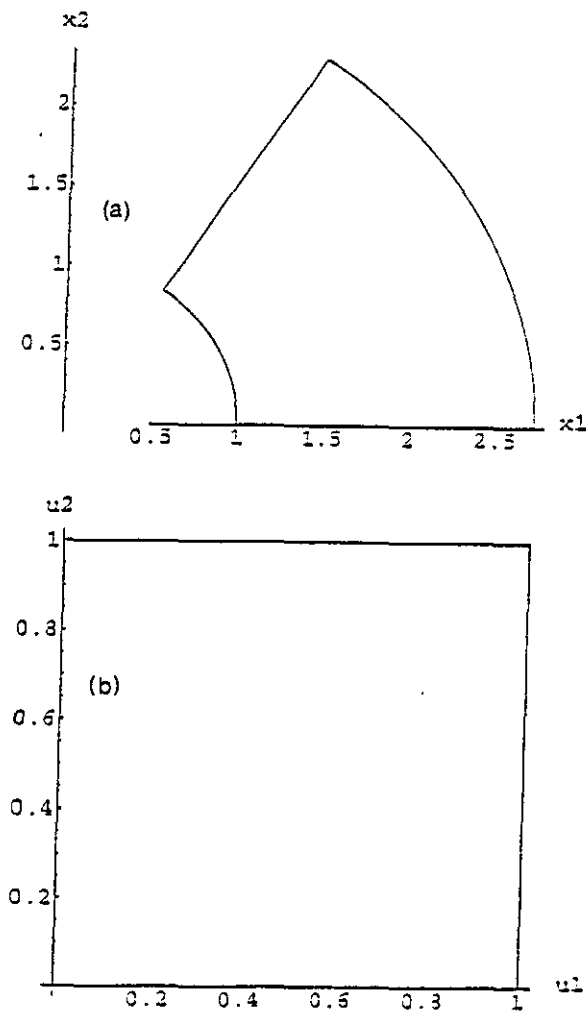


Fig. 4.2

After the definition of the source and the target domains in order to evaluate the effects of the nodal refinement on the accuracy of the results, different arrangements of the source and the target nodes (or if boundary fitting is desired the source and the target boundary nodes) are automatically generated and used by the program.

Typical examples of such arrangements are shown in figures 4.2c-4.2h.

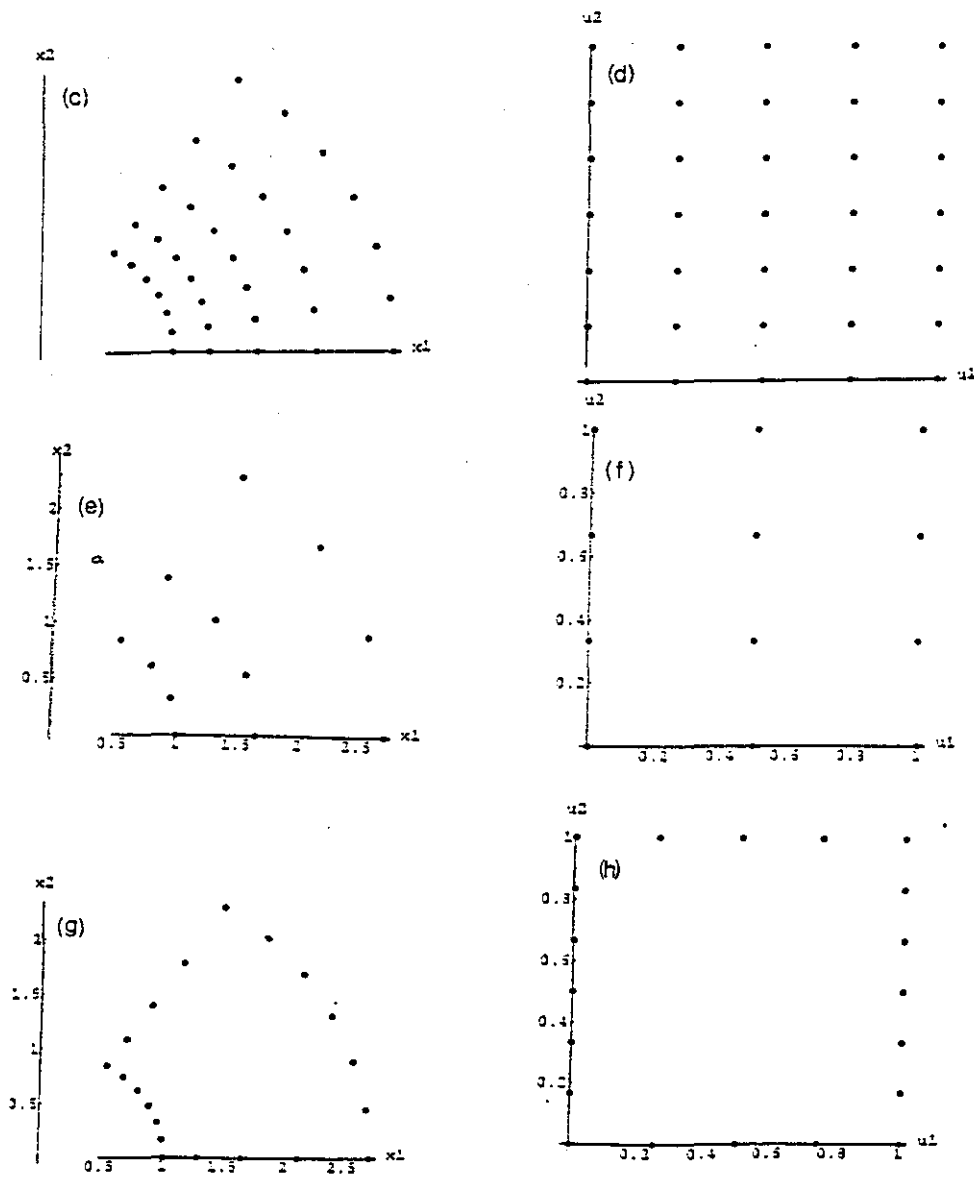


Fig. 4.2

Using bilinear polynomials in x_1 and x_2 as the base functions and the power function as the weight function the result of the diffuse approximation mapping from N_S to N_T , respectively, corresponding to figures 4.2c and 4.2d is shown in figure 4.2i. In this figure we have used lines connecting the intended target nodes to their corresponding points found by the numerical scheme to give a graphical measure of the error of the computations.

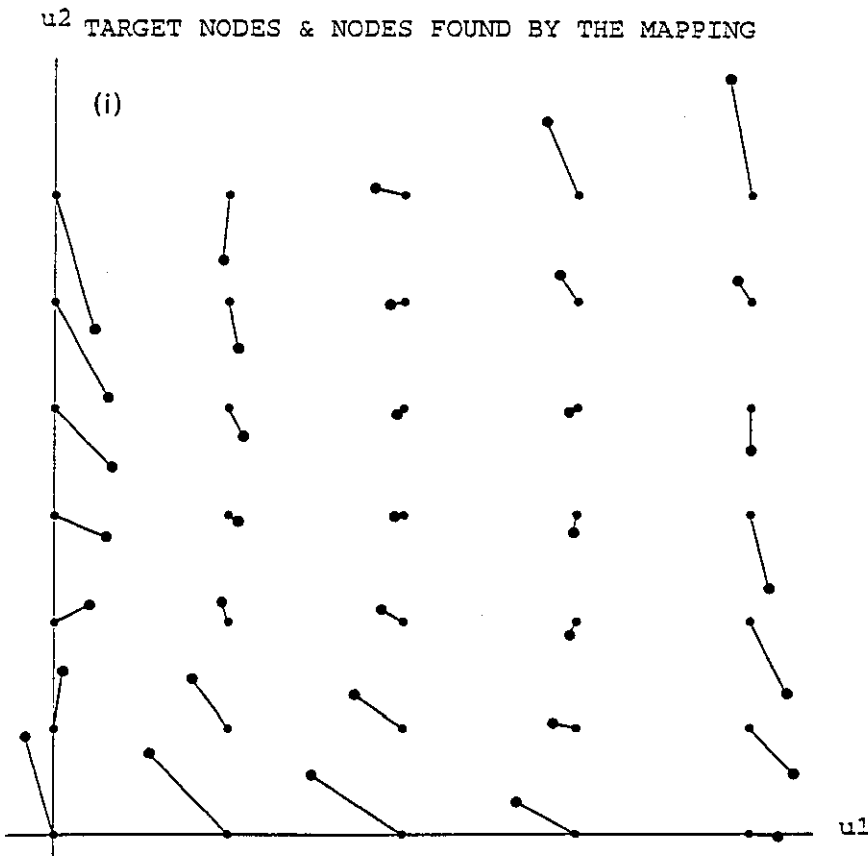


Fig. 4.2

To improve this result we have used biquadratic base functions and at the same time reduced the number of nodes while keeping the other parameters constant. In this case the result of the mapping corresponding to the source and the target nodes, shown in figures 4.2e and 4.2f, is given in figure 4.2j.

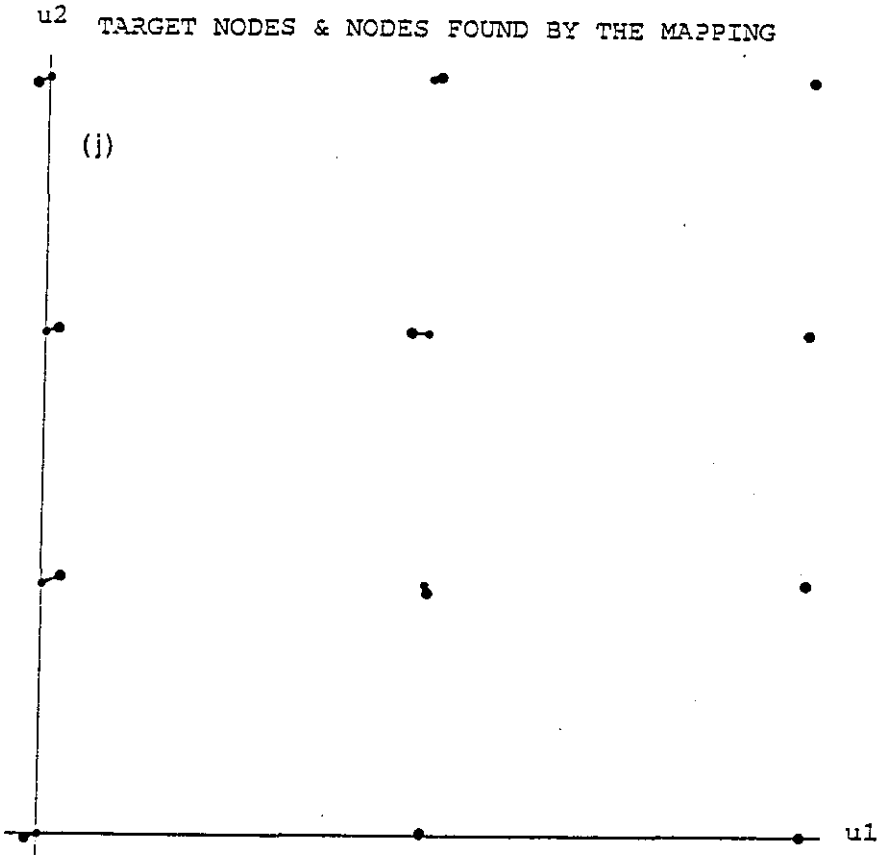


Fig. 4.2

Next the number of nodes is increased but all other parameters are kept constant. This result is shown in figure 4.2k.

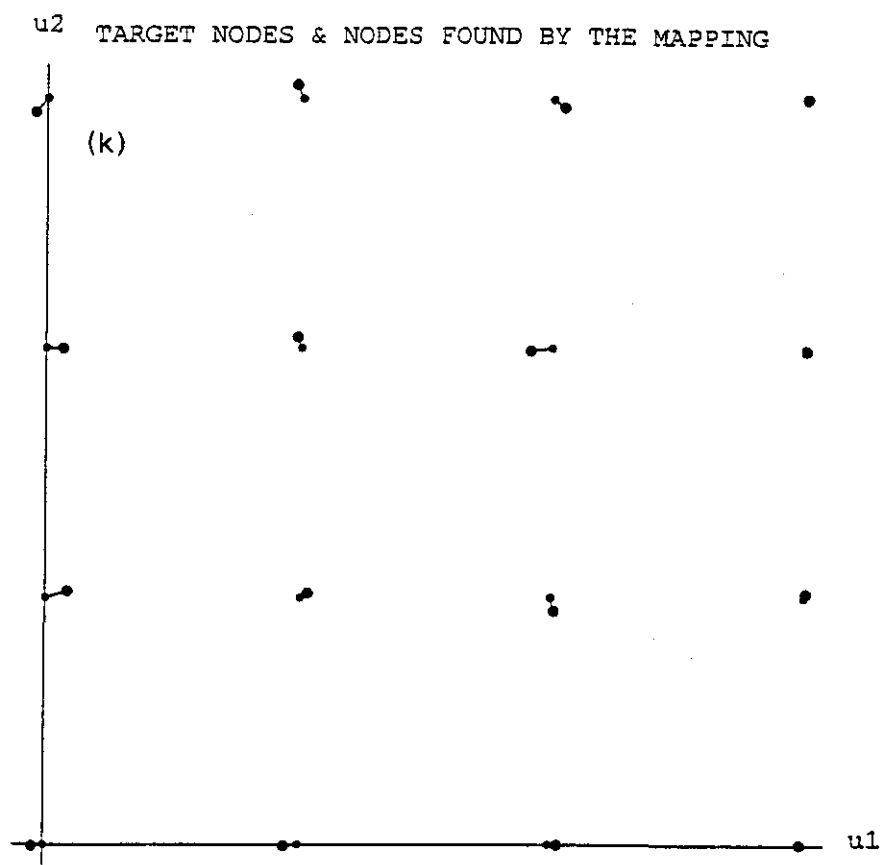


Fig. 4.2

The comparison of the figures 4.2j and 4.2k shows that the increasing of the number of nodes on their own does not improve the result. Finally we have combined a moderate increase in the number of nodes with using the base functions found by the tensor product of cubic polynomials in x_1 and x_2 to carry out the diffuse approximation mapping from the annular section to the square domain. The super-convergent result found using this input is shown in figure 4.2l. In this figure we have used clear circles to show the points found by the scheme which exactly coincide with the predetermined target nodes. The error norms for this calculation are also shown in figure 4.2l.

```

DU-error1:=1.55431 10-15
DU-error2:=9.99201 10-16
DU-error:=1.55431 10-15
mindet:=211.9865980220

```

TARGET NODES & NODES FOUND BY THE NUMERICAL MAPPING

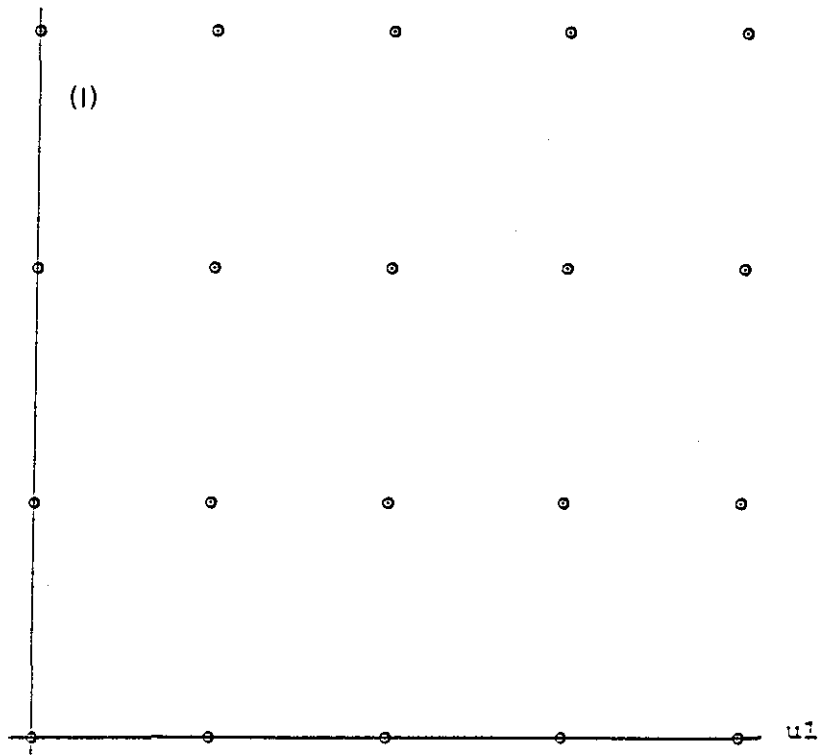


Fig. 4.2
62

The described automatic trial and error procedure is also used to generate a mapping which only includes the boundary nodes in the source and the target domains. In figure 4.2m we show the result obtained by the use of biquadratic base functions in conjunction with a 5×4 arrangement for the boundary nodes. The improved super-convergent result found using tensor product of 3rd order polynomials in x_1 and x_2 as the base function in combination with the 6×5 nodal array is shown in figure 4.2n.

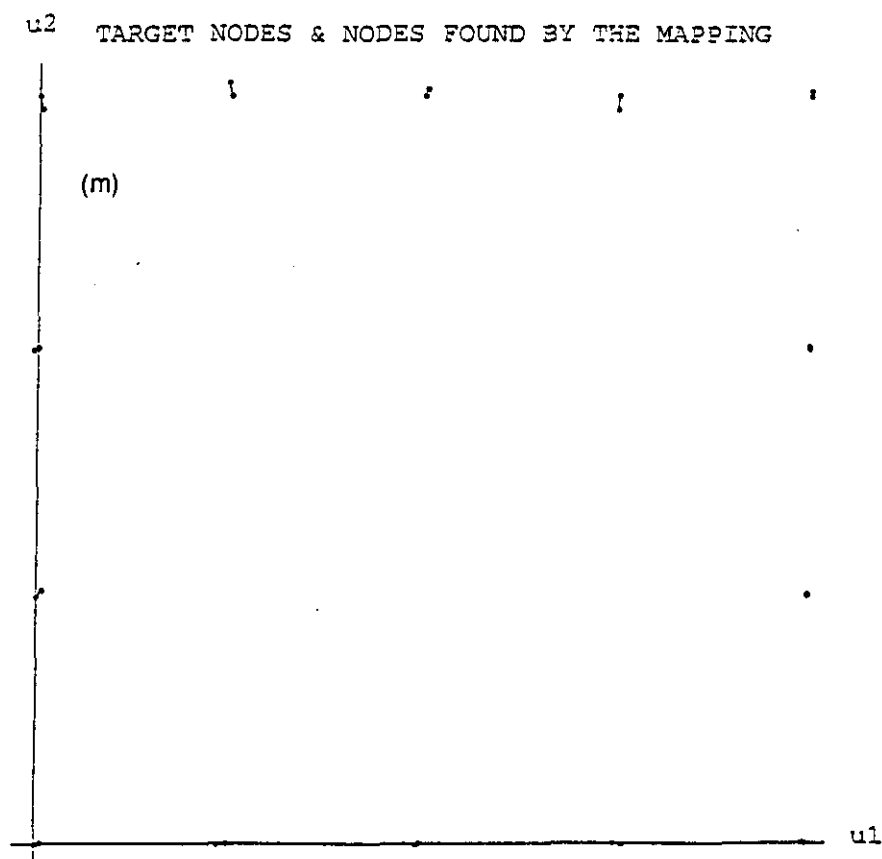


Fig. 4.2

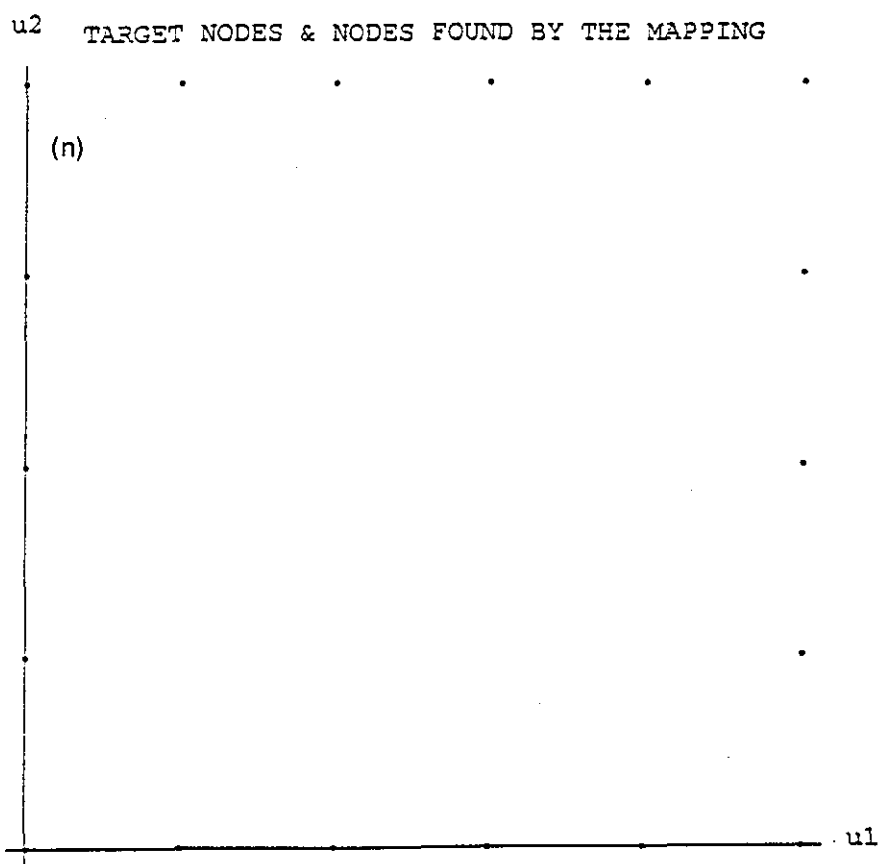


Fig. 4.2.

The validity of the calculated mapping along the entire boundary of the problem domain can be proved by the trace theorem (Kesavan, 1989). However, it is crucial to note that a node-by-node numerical mapping which is used to generate a boundary fitting cannot give an overall mapping for the problem domain. To demonstrate this point we have calculated the nodal positions inside the domain via equation (3.34) and compared them with their corresponding target nodes. This result is shown in figure 4.2o which indicates that the mapping is not valid for the interior nodes. Therefore the recommended use of the boundary mapping is in problems which only involve boundary operations such as the calculation of the boundary integrals.

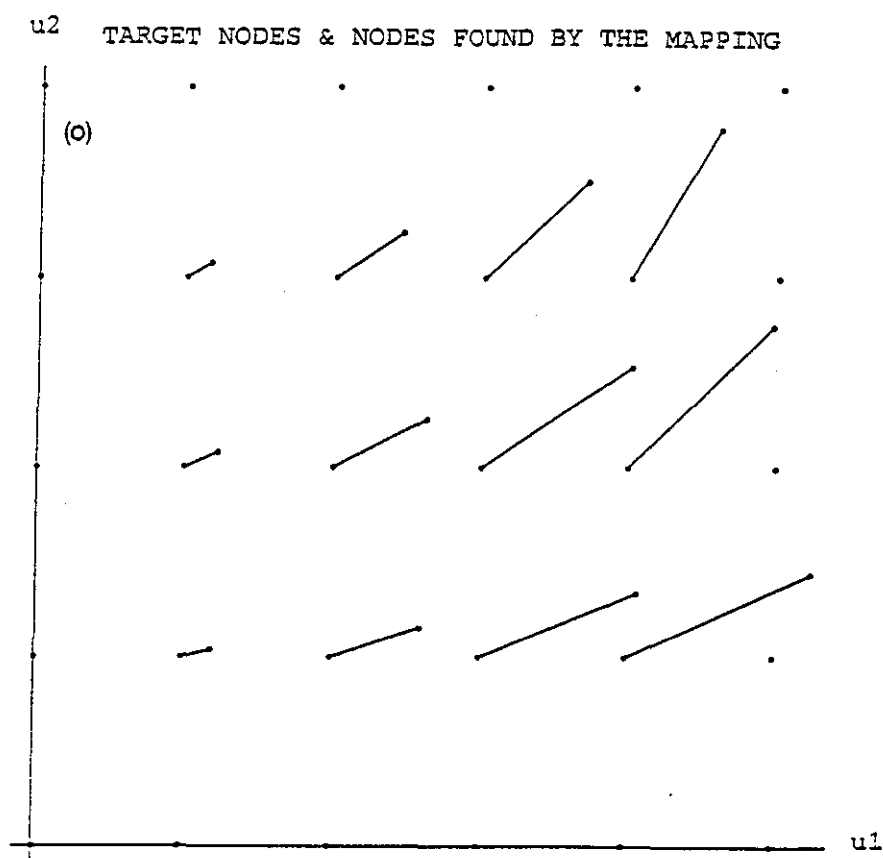


Fig. 4.2

Test Problem No. 2 Application of the diffuse approximation technique in plane mapping.

In the previous test problem a conformal mapping between the source and the target domains can be constructed. Therefore it may be considered as a straightforward mapping problem which may not be adequate for exploring the strength and weaknesses of the developed method to a full extent. In the present test we consider the mapping of a triangular domain with a curved boundary to a triangle with straight sides.

Such a mapping changes the angles and therefore there is no equivalent conformal transformation in this case. This calculation is carried out for two cases. In case 1 we considered the mapping of a quarter of a circle to a triangle. Thus in the source triangle, the intersection points of the curved side with the straight sides are ordinary points. In case 2 we analysed a very complex situation in which the curved side in the source triangle is tangent to the straight sides and hence the corresponding vertices are not ordinary single points.

In the generation of finite element meshes based on triangular elements the only possibility to cope with a situation such as this example is to simplify the problem by assuming that the sides of the element are not tangent to each other. The degree of discretisation error resulting from this assumption depends on the overall shape of the problem domain and in some cases it may be significant. The first case is a special form of the second problem and presents a simplified situation therefore in this discussion we only consider the second case.

In order to be sure that the source nodes in our calculations exactly correspond to the true geometry of the problem domain we start with the regeneration of the entire domain as the arrangement of a very high number of

nodal points. Figures 4.2p and 4.2q show two successive nodal arrangements converging towards the selected source domain. The total number of nodes in figure 4.2p is 1275 and in figure 4.2q is 5050, respectively. After finding a sufficiently large number of nodes which can be said to preserve the main characteristics of the source domain we choose a limited number of nodes from that arrangement to carry out the diffuse approximation mapping.

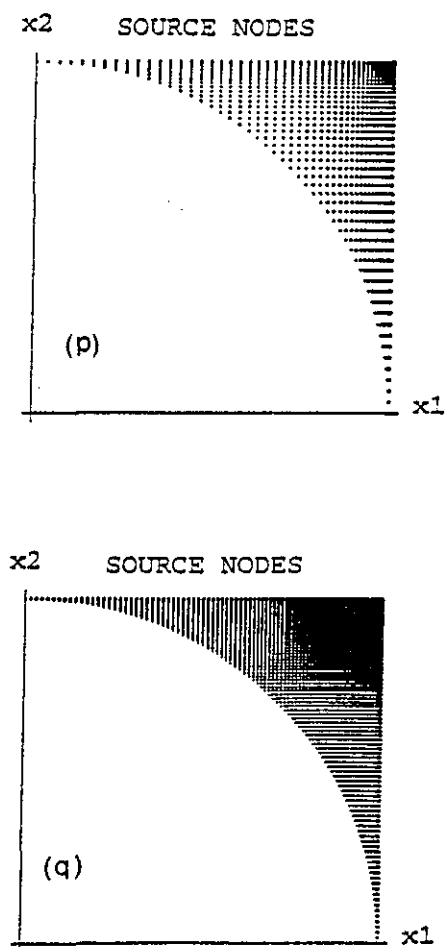


Fig. 4.2

The complexity of this problem is such that we needed to consider many different combinations of input parameters in order to obtain accurate results. The precision of the numerical calculations also proved to be very important in this case and we needed to operate with 20 significant places after the decimal point to obtain accurate results. However, the flexibility of the diffuse approximation technique and its main strength in the generation of high order smooth mappings was demonstrated through this problem and we were able to find super-convergent results in many cases. This is achieved by using base functions which include transcendental as well as polynomial terms. This represents a modification of the standard diffuse approximation technique which is based on the use of polynomial base functions.

An example which includes 7 nodes is shown in figures 4.2r (the source nodes) and 4.2s (the target and the calculated nodes). The best weight function in this case was again found to be the power function and the selected base function is

$$\hat{P} = \{1, x_1, x_2, x_1^{\frac{1}{2}}, x_2^{\frac{1}{2}}, x_1^{\frac{1}{2}} x_2^{\frac{1}{2}}, x_1 x_2\} \quad (4.9)$$

This combination gave a mapping which has the following error norms :

Discrete uniform error in u_1 direction = 8.63274×10^{-9}

Discrete uniform error in u_2 direction = 9.13834×10^{-9} .

Wherein the discrete uniform norm represents maximum absolute magnitude of error found at nodal points.

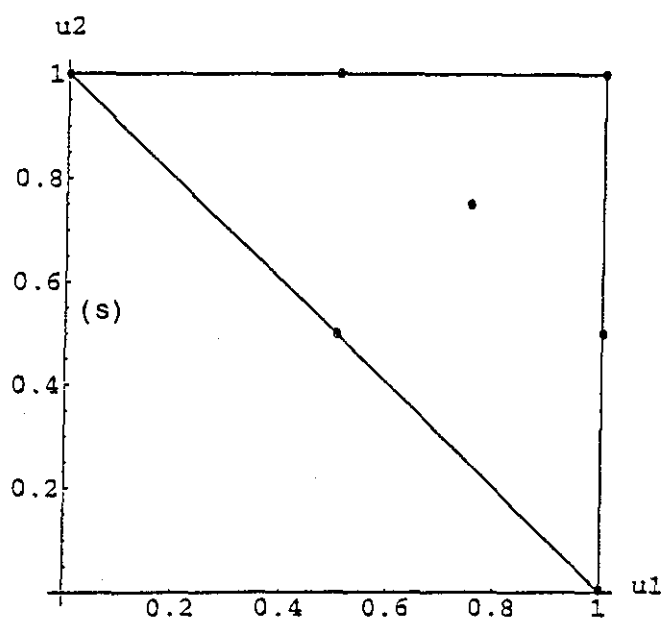
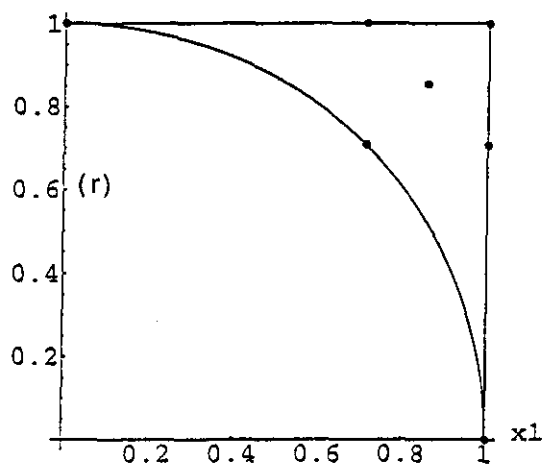


Fig. 4.2

Test Problem No. 3 Application of the diffuse approximation technique in generating transformations in boundary value problems.

The developed mapping scheme may be used as a preliminary step for the simplification of the solution of the boundary value problems. In general, a boundary value problem can be expressed as

$$\begin{cases} Lu = f & \text{in } \Omega \\ Bu = g & \text{on } \partial\Omega \end{cases} \quad (4.10)$$

where L is a differential operator and B stands for the boundary operator. Using the diffuse approximation scheme the mapping function and its derivatives for a desired transformation which maps the physical domain Ω , to a simpler domain Ω^* , is found.

We apply the developed mapping scheme to a number of examples based on the described boundary value problem which have a known analytic transformation in a specific target domain. Thus we can generate a corresponding numerical transformation for the problem and compare it with its analytic counter-part.

We consider the following Poisson equation defined in the source domain shown in figure 4.2t,

$$\Delta v = 8(-x_1 + x_1^2 + x_2^2) \quad (4.11)$$

subject to homogeneous Dirichlet boundary conditions. The analytical solution of this equation is

$$v = (x_1 - x_2)(x_1 + x_2)(1 - x_1 + x_2)(1 - x_1 - x_2) \quad (4.12)$$

Using the following transformation

$$\begin{cases} u_1 = x_1 - x_2 \\ u_2 = x_1 + x_2 \end{cases} \quad (4.13)$$

we find the corresponding transformed equation in the mapped target domain, shown in figure 4.2.u, as

$$\Delta v = 4(u_1^2 + u_2^2 - u_1 - u_2) \tag{4.14}$$

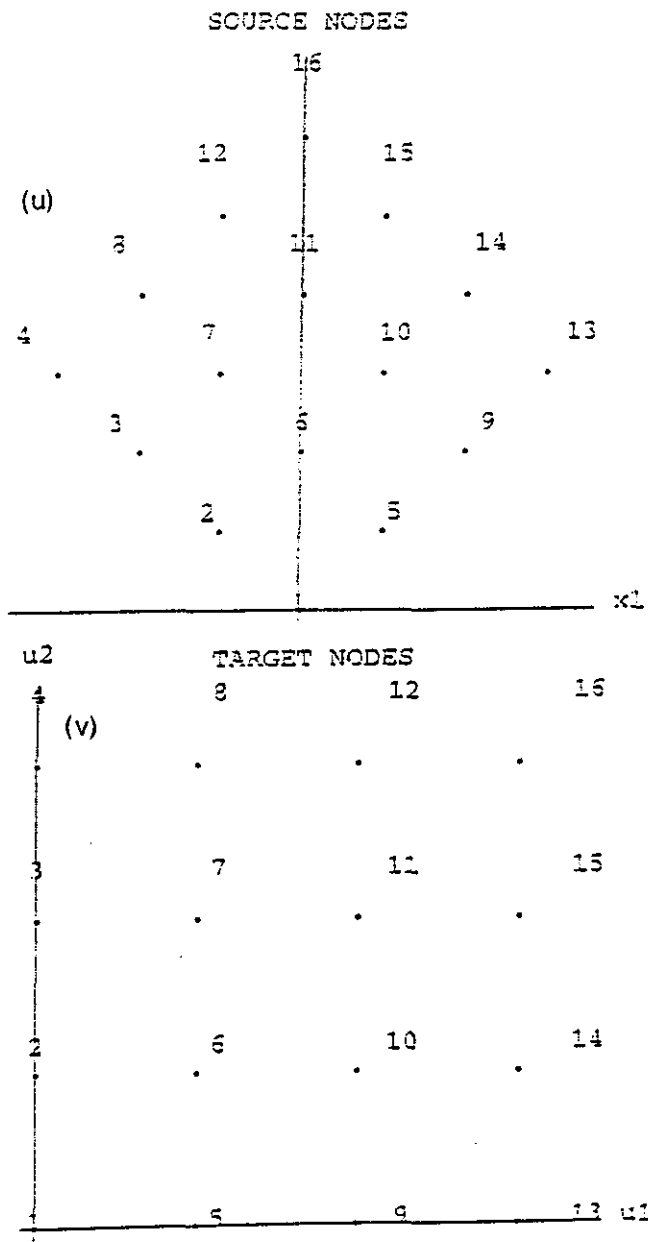


Fig. 4.2

Equation (4.14) is again subject to homogeneous Dirichlet boundary conditions. The analytical solution of the transformed equation in the target domain is

$$v = u_1 u_2 (1 - u_1)(1 - u_2) \quad (4.15)$$

By the application of the diffuse approximation technique to the analytical solution given by equation (4.12) we find the numerical counter-part of the equation (4.15). The comparison of these results is used to investigate the accuracy of the numerical transformation of the boundary value problem. In the above problem this comparison shows the same order of accuracy as the super-convergent mapping which transforms the source domain onto the target domain (the discrete uniform error in both transformations is of the order of 10^{-15}). However, the main factor affecting the accuracy of the mapping of the boundary value problem is the smoothness of its solution. This is proved by considering different examples in which the source and the target domains remain the same (i.e. as are shown in figures 4.2u and 4.2v) but the right hand side of equation (4.11) is changed to generate non-smooth solutions. In the case that the analytical solution of the equation is the square root of the solutions given in the above example the discrete uniform error of the transformation is found to be of the order of 10^{-8} . This is 7 orders of magnitude larger than the error of the domain mapping itself. This error is further increased to be of the order of 10^{-3} in another problem which its analytical solution is the 5th root of the original solution shown as equation (4.12). We also investigated the effects of a small perturbation in the domain mapping on the transformation of the boundary value problem. In this case again if the boundary value problem has a smooth solution, a small error in the domain mapping has a negligible effect on its transformation. However, in cases that the solution is non-smooth a very small error in the domain map-

ping gives a large transformation error for the problem. This has significant implications for the solution of complicated problems by numerical methods such as the finite element technique which mainly rely on an approximate C^0 continuous isoparametric mapping of the curved elements.

4.3 Solution of Ordinary Differential Equations

4.3.1 Numerical Solution of Sturm-Liouville Problems

Sturm-Liouville equations are one of the most frequently used class of ordinary differential equations in physical science. The most important characteristic of these equations is that they represent generalised eigen-systems arising in spectral expansions. These expansions provide a very powerful technique for the solution of the mathematical models of a wide range of engineering processes. Numerous examples of such applications in structural analysis and computational fluid dynamics can be found in the literature. In recent years, in addition to the previously established applications of the spectral expansion methods, new avenues for the utilisation of these techniques have also been developed. The application of the spectral expansions to the governing equations of tidal dynamics by Smith (1995, 1995, 1997), which gives a set of modal equations for water velocity and concentration of pollutants in estuaries, is such an example. These modal equations can be used to create robust and very cost effective pollutant dispersion models in tidal water systems under realistic conditions without using costly full three-dimensional computations. However, the Sturm-Liouville equations arising

in tidal dynamics, as well as many other realistic problems, cannot be solved analytically. Therefore the development of accurate and robust numerical schemes for the solution of this type of ordinary differential equations is essential in problems involving spectral expansions.

We have used the diffuse approximation technique to develop a convenient and robust scheme for the solution of Sturm-Liouville equations. An important new aspect of our work is the utilisation of the power of the diffuse approximation method in combining different types of weight and base functions for the solution of singular Sturm-Liouville problems. The accuracy of the developed method is demonstrated by obtaining super-convergent solutions in a number of test problems in which analytical solutions are known.

Consider the standard form of the Sturm-Liouville equation expressed as

$$-\frac{d}{dx}\left[p(x)\frac{du}{dx}\right] + q(x)u(x) - \lambda w(x)u(x) = f(x); \quad \alpha < x < \beta \quad (4.16)$$

subject to the following 'unmixed type' boundary conditions

$$B_\alpha u = \alpha_1 u(\alpha) + \beta_1 u'(\alpha) = 0 \quad (4.17)$$

and

$$B_\beta u = \alpha_2 u(\beta) + \beta_2 u'(\beta) = 0 \quad (4.18)$$

where λ is a scalar and,

- i.* p, p', q , and w are real-valued and continuous functions in (α, β) , and
- ii.* p , and w are positive in (α, β) (Renardy, and Rogers, 1993).

A Sturm-Liouville problem is called regular if α and β are finite and the conditions *i* and *ii* are also true at the end points of the domain. Otherwise the problem is called singular. In the following sections the solution of the

regular Sturm-Liouville problems by the diffuse approximation method is described. It is shown that this method can be extended to singular problems. We define the following operator

$$\mathcal{L}u = \frac{1}{w} \left[-\frac{d}{dx} \left[p(x) \frac{du}{dx} + q(x)u(x) \right] \right] \quad (4.19)$$

Therefore equation (4.16) can be written as

$$\mathcal{L}u - \lambda u = \frac{f}{w} \quad (4.20)$$

The domain of \mathcal{L} is defined as

$$D(\mathcal{L}) = \{u | u \in H^2(\alpha, \beta), B_\alpha u = B_\beta u = 0\} \quad (4.21)$$

Using the above definitions the solution of a regular Sturm-Liouville problem can be regarded as an eigenvalue analysis in which the scalar λ is the eigenvalues of the operator \mathcal{L} . Therefore the existence and uniqueness of the solution of equation (4.16) depends on λ (Renardy, and Rogers, 1993).

The weak variational formulation of equation (4.16) can be expressed as

$$\text{Find } u \in D(\mathcal{L}), \text{ such that } a(u, v) = F(v) \text{ for all } v \in D(\mathcal{L}) \quad (4.22)$$

where

$$a(u, v) = \int_\alpha^\beta \left[p(x) \frac{du}{dx} \frac{dv}{dx} + Q(x)uv \right] dx + p(u, v) \Big|_\alpha^\beta \quad (4.23)$$

and

$$F(v) = \int_\alpha^\beta v f dx \quad (4.24)$$

and

$$Q(x) = q(x) - \lambda w(x) \quad (4.25)$$

Equation (4.23) can be discretised in different ways. In this study we have used the standard Galerkin method (Brenner and Scott, 1994) to carry out the required discretisation. Therefore assuming that V^h is a finite-dimensional sub-space of the defined operator domain we have

$$\begin{aligned} \text{Find } u^h \in V^h, \quad \text{such that} \\ a(u^h, v^h) = F(v^h) \quad \text{for all, } v^h \in V^h \end{aligned} \quad (4.26)$$

Therefore the substitution of u^h from the described diffuse approximation technique as

$$u^h = \sum_{j=1}^N \phi_j u_j \quad (4.27)$$

gives

$$\sum_{i=1}^N a(\phi_i, \phi_j) u_i = F(\phi_j) \quad j = 1, \dots, N \quad (4.28)$$

Equation (4.28) is the working equation of the present scheme which corresponds to the standard Galerkin method where $v^h = \phi_j, j = 1, \dots, N$.

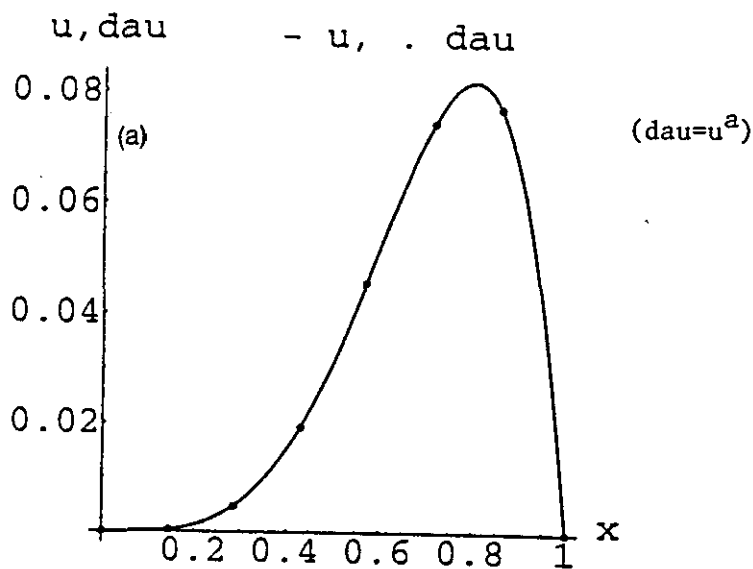
Test problem No. 1 The following regular Sturm-Liouville problem is considered

$$-\frac{d}{dx} \left[(1+x^2) \frac{du}{dx} \right] + xu = -x^2(x^4 - 31x^3 + 20x^2 - 20x + 12) \quad (4.29)$$

subject to $u(0) = 0$ and $u(1) = 0$. This represents a regular problem with a smooth right hand side. To solve this problem we used the following set of input parameters

$$\begin{cases} \hat{N} = 8 \\ \hat{P} = \{(1-x)x, \dots, (1-x)^6 x\} \\ \hat{W} = (1 - \frac{14x^2}{15})^2 \\ \hat{Q} = \text{Newton-Cotes order}=13 \end{cases} \quad (4.30)$$

In figure 4.3.1a the numerical results obtained by the diffuse approximation method and the analytical solution of this problem are compared. As it can be seen from figure 4.3.1a the diffuse approximation method easily generates a super-convergent result for this problem. The computations are carried out using the rational mode in order to show the perfect accuracy of the results. In this case all three error norms were found to be 0 (integer). This solution was found after 50.5 s using a SUN SPARC work-station.



DL1-error[f] := 0

DL2-error[f] := 0

DU-Error[f] := 0

Fig. 4.3.1

Test problem No. 2 Sturm-Liouville problems arising in practical applications such as the representation of the velocity and concentration modes in an estuary, generally depend on the imposition of the natural boundary condition on the free surface of the water. We used the following problem to impose essential and natural conditions at the boundaries of the solution domain

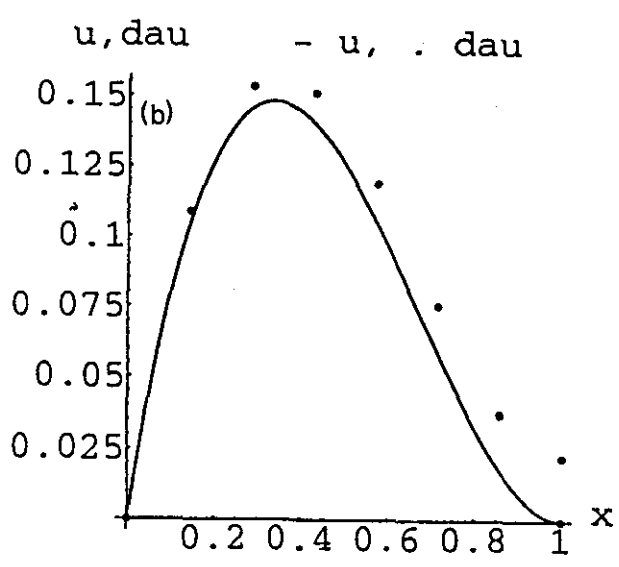
$$-\frac{d}{dx} \left[(1+x^2) \frac{du}{dx} \right] + xu = x^4 - 14x^3 + 13x^2 - 8x + 4 \quad (4.31)$$

subject to $u(0) = 0$ and $u'(1) = 0$. The following sets of input were used

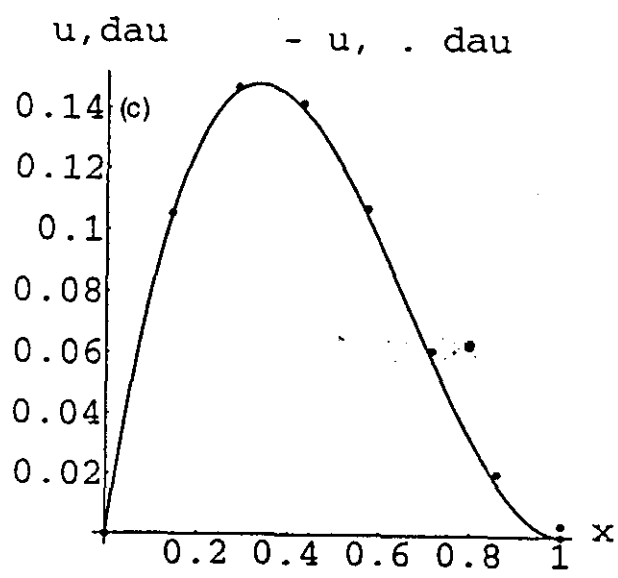
$$\begin{cases} \hat{N} = 8 \\ \hat{P} = \{x, \dots, x^6\} \\ \hat{W} = (1 - \frac{14x^2}{15})^2 \\ \hat{Q} = \text{Gauss-Legendre order}=20, \text{ order}=50, \text{ order}=78 \end{cases} \quad (4.32)$$

In this test the results were obtained by the real mode computations with a precision of 30 significant places. These solutions and their corresponding error norms are shown in figures 4.3.1b, 4.3.1c and 4.3.1d and compared with the analytical solution of the problem. In this case, since one of the boundary conditions is included within the weak statement of the problem, the accuracy of the results is mainly dependent on the order of the quadrature. As it is shown in figures 4.3.1b-4.3.1d at this boundary only after using a very high order quadrature the numerical and the analytical solutions converge. It took about 90 s in our work-station to generate the solution shown in figure 4.3.1d.

$L1-error[f] := 0.0118079$
 $L2-error[f] := 0.0136872$
 $U-Error[f] := 0.0224495$



$L1-error[f] := 0.00197132$
 $L2-error[f] := 0.00228422$
 $U-Error[f] := 0.00373509$



$L1-error[f] := 0.000817028$
 $L2-error[f] := 0.000946683$
 $U-Error[f] := 0.00154759$

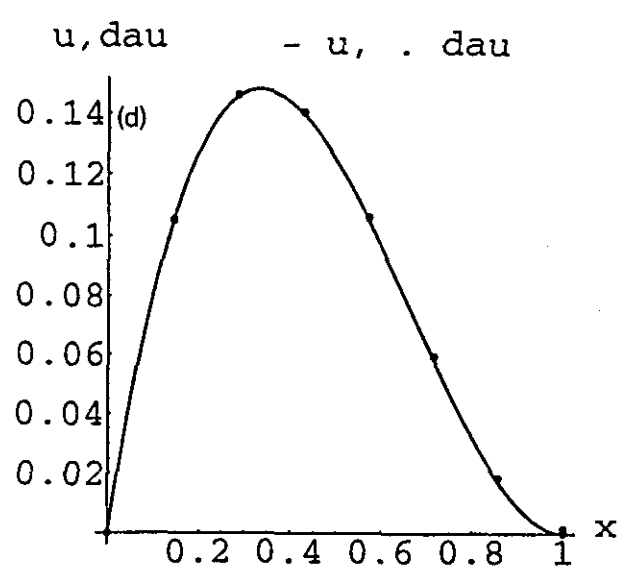


Fig. 4.3.1
 78A

Test problem No. 3 In this test we considered the following regular problem with a non-smooth right hand side

$$-\frac{d}{dx} \left[-\frac{du}{dx} \right] + u = 5x + 6x \ln x + x^3 \ln x, \quad 0 < x < 1 \quad (4.33)$$

subject to $u(0) = 0$ and $u(1) = 0$. Despite the apparent form of the equation (4.33), in which the coefficient $p(x)$ is negative, this equation still represents a regular problem since its sides can be multiplied by -1 to show that it satisfies all of the conditions of the regularity. To solve this problem we used the following set of input parameters

$$\begin{cases} \hat{N} = 10 \\ \hat{P} = \{(1-x)x, \dots, (1-x)x^7\} \\ \hat{W} = (1 - \frac{14x^2}{15})^2 \\ \hat{Q} = \text{Gauss-Legendre order}=20 \end{cases} \quad (4.34)$$

The results of this test problem, obtained using real mode computations with a precision of 50 significant places, and the corresponding error norms are shown in figure 4.3.1e.

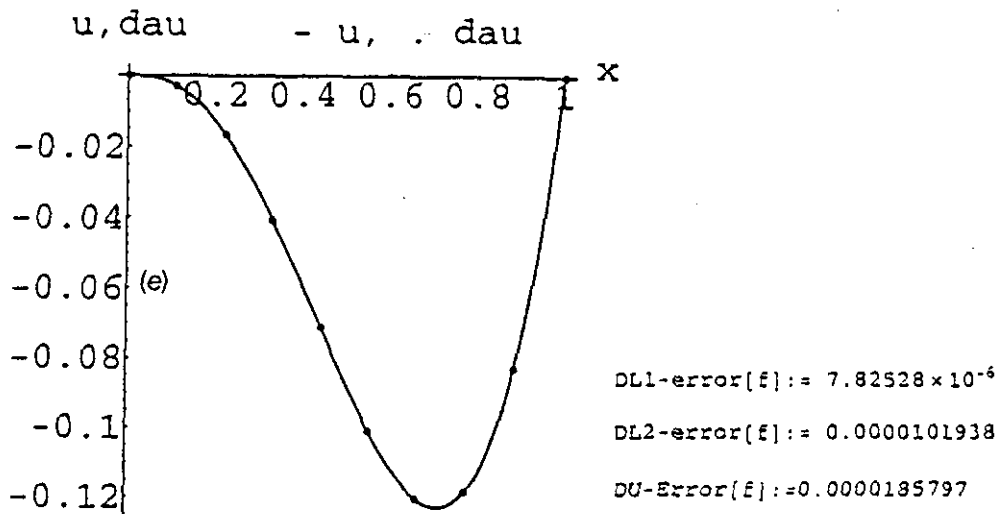


Fig. 4.3.1

Test problem No. 4 In this test we again considered a regular problem with non-smooth right hand side, expressed as

$$-\frac{d}{dx} \left[-\frac{du}{dx} \right] + u = U_{\frac{1}{2}}(x) \quad (4.35)$$

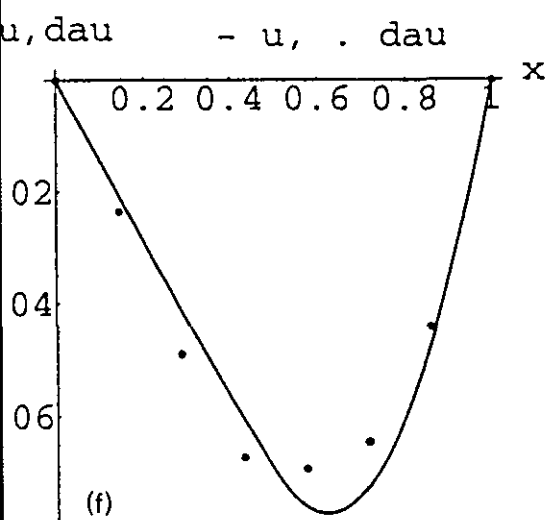
where

$$U_{\frac{1}{2}}(x) = \begin{cases} 0 & \text{if } 0 \leq x < \frac{1}{2} \\ \frac{1}{2} & \text{if } x = \frac{1}{2} \\ 1 & \text{if } \frac{1}{2} < x \leq 1 \end{cases} \quad (4.36)$$

subject to $u(0) = 0$ and $u(1) = 0$. In this case an analytical solution is found using the Green function. Different combinations of input parameters were used to generate the diffuse approximation solution for equation (4.35). Polynomial base functions were proved to be inadequate for the solution of this equation. Therefore we used the following sets of input parameters

$$\left\{ \begin{array}{l} \hat{N} = 8 \text{ and } 6 \\ \hat{P} = \{(1-x)x, (1-x)x^2, \sigma \sin x, \\ \sigma \sin x(1 - \cos(x - \frac{1}{2}))U_{\frac{1}{2}}(x), -\cos(x - \frac{1}{2})U_{\frac{1}{2}}(x)\} \\ \hat{W} = (1 - \frac{14x^2}{15})^2 \\ \hat{Q} = \text{Newton-Cotes order}=13, \text{ Gauss-Legendre} \\ \text{order}=10, \text{ order}=22, \text{ order}=78 \end{array} \right. \quad (4.37)$$

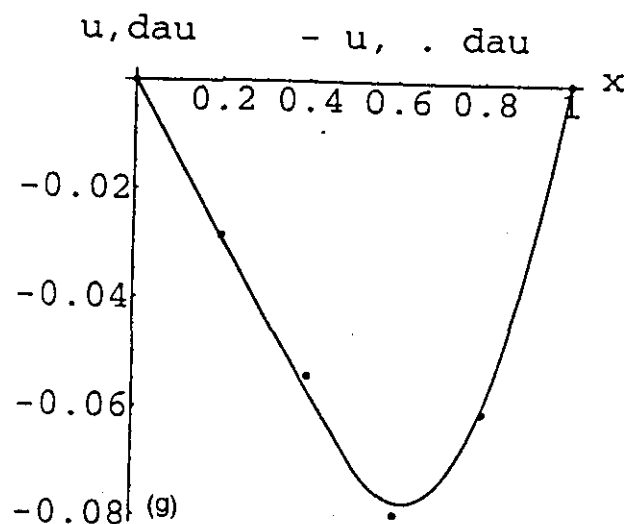
In the selection of the above base functions we were guided by the form of the analytical solution of the problem. In this set of base functions the coefficient σ is a constant. The diffuse approximation results in this test correspond to a generalised solution of a Sturm-Liouville equation with highly non-smooth right hand side. The numerical results obtained by various combinations of the input options given in expression (4.37) are shown in figures 4.3.1f to 4.3.1i.



-error[f] := 0.00491025

-error[f] := 0.00597915

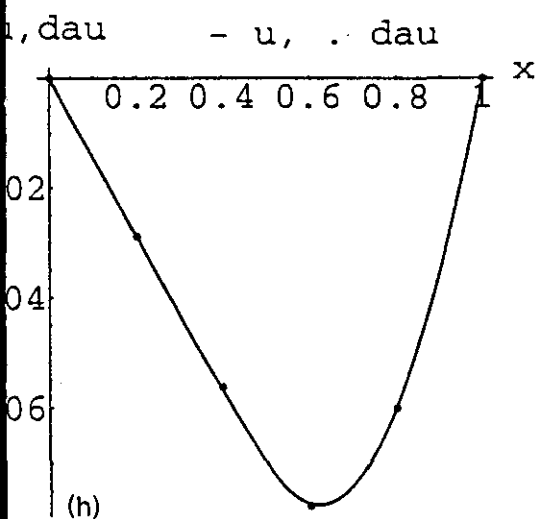
Error[f] := 0.00792814



DL1-error[f] := 0.00121167

DL2-error[f] := 0.00168661

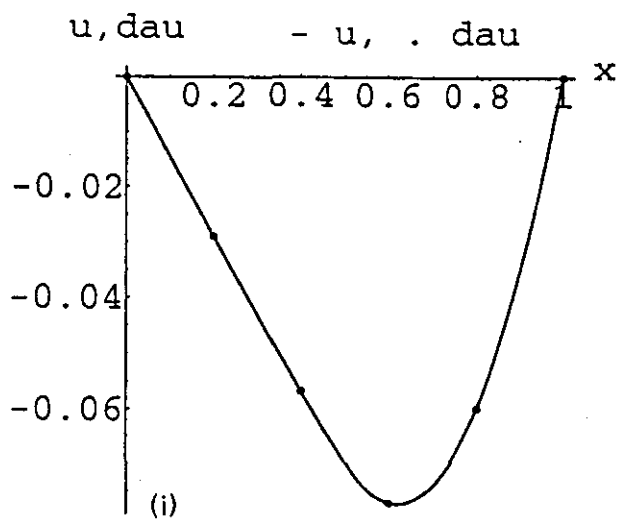
DU-Error[f] := 0.00267475



1-error[f] := 0.000208338

2-error[f] := 0.000294356

-Error[f] := 0.000487815



DL1-error[f] := 0.0000162063

DL2-error[f] := 0.0000230127

DU-Error[f] := 0.0000385992

Fig. 4.3.1

Test problem No. 5 We finally considered the following singular Sturm-Liouville problem

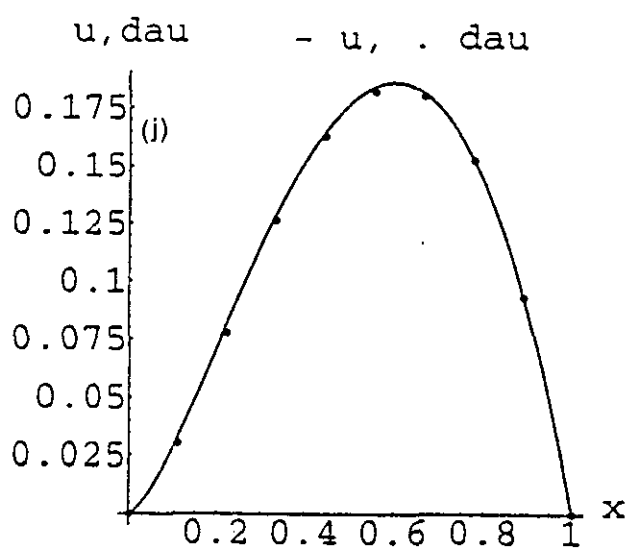
$$-\frac{d}{dx} \left[-x^2 \frac{du}{dx} \right] + xu = -\frac{1}{4}x^{\frac{3}{2}}(4x^2 + 31x - 15) \quad (4.38)$$

subject to $u(0) = 0$ and $u(1) = 0$.

It should be noted that by the multiplication of the sides of equation 3.10 one cannot show that it is a regular equation. This is because that the coefficient of the second order derivative in this equation does not satisfy the conditions of regularity at the boundaries. We used the following set of input to find the solution of this problem

$$\begin{cases} \hat{N} = 10 \\ \hat{P} = \{x^{\frac{3}{2}}, (1-x)x, \dots, (1-x)x^7\} \\ \hat{W} = (1 - \frac{14x^2}{15})^2 \\ \hat{Q} = \text{Newton-Cotes order}=20 \end{cases} \quad (4.39)$$

In this test we used real mode computations with a precision of 50 significant places. The comparison of the numerical and the analytical results and the error norms are shown in figure 4.3.1j. The analytical solution of equation (4.38) is $u = x^{\frac{3}{2}}(1-x)$. The uniqueness of this solution can be proved using Frobenius theorem (Coddington and Levinson, 1955).



DL1-error[f] := 0.0014116

DL2-error[f] := 0.00196431

DU-Error[f] := 0.00378673

Fig. 4.3.1

As it is indicated by the above results in the solution of Sturm-Liouville equations we always could generate accurate solutions by using power weight functions. Therefore it seems that this type of weight function is adequate in these problems. However, the selection of other types of weight functions may prove to be necessary in the solution of partial differential equations by the diffuse approximation method.

4.3.2 Eigenvalue Problems

In the previous section the application and importance of the Sturm-Liouville equations in the solution of eigenvalue problems are described. We recall that an eigenvalue problem is of the form

$$\text{Find } (\lambda, u) \text{ such that} \quad (4.40)$$

$$Lu = \lambda u, \quad u \in D(L) \quad u \neq 0 \quad \text{and} \quad \lambda \in R$$

where

$$Lu = -\frac{d}{dx} \left[p(x) \frac{du}{dx} \right] + q(x)u \quad \alpha < x < \beta \quad (4.41)$$

subject to the boundary condition $u(\alpha) = 0$ and $u(\beta) = 0$ or $u(\alpha) = 0$ and $u'(\beta) = 0$.

There is an infinite sequence of real eigenvalues which satisfy the problem defined by expression (4.40). These can be shown as

$$\lambda_1 \leq \dots \leq \lambda_j \leq \dots \quad (4.42)$$

Associated with these eigenvalues there is a complete set of orthonormal eigenfunctions expressed as

$$(u_i, u_j) = \int_{\alpha}^{\beta} u_i(x) u_j(x) dx = \delta_{ij} \quad (4.43)$$

These eigenfunctions are also orthonormal in the 'energy inner product' sense i.e.

$$(Lu_i, u_j) = (\lambda_i u_i, u_j) = \lambda_j \delta_{jk} \quad (4.44)$$

Using the bilinear form we have

$$a(u_j, u_k) = \int_{\alpha}^{\beta} (pu'_j u'_k + qu_j u_k) dx = \lambda_j \delta_{jk} \quad (4.45)$$

Consider the Rayleigh quotient, defined by

$$R(v) = \frac{a(v, v)}{(v, v)} = \frac{\int [p(v')^2 + qv^2]}{\int v^2} \quad (4.46)$$

Using strong or Frechet derivative the stationary functions of functional $R(v)$ are found which are identical with the eigenfunctions of the standard eigenvalue problem defined by expression (4.40). Alternatively we have

$$\text{Find } (\lambda, u) \quad u \in V \text{ and } \lambda \in R \quad (4.47)$$

$$\text{such that } a(u, v) = \lambda(u, v) \quad \text{for all } v \in V$$

where V is the space of admissible functions and $\lambda(u, v)$ represents an inner product multiplied by λ .

The fundamental (or leading) frequency λ_1 and its associated normal mode, are found as the stationary point (or minimum) and hence

$$\lambda_1 = \min_{v \in V} R(v) \quad (4.48)$$

If we assume that E_{l-1} is the space spanned by the eigenfunctions u_1, \dots, u_{l-1} then

$$\lambda_l = \min_{v \perp E_{l-1}} R(v) \quad (4.49)$$

The discretised eigenvalue problem corresponding to expression (4.47) is

$$\begin{aligned} \text{Find } (\lambda^h, u^h) \quad u^h \in V^h, \lambda \in R \\ \text{such that } a(u^h, v^h) = \lambda^h(u^h, v^h) \quad \text{for all } v^h \in V^h \end{aligned} \quad (4.50)$$

Using the diffuse approximation scheme the approximate function u^h can be written in terms of the shape functions as

$$u^h = \sum_{j=1}^N \phi_j U_j \quad (4.51)$$

Therefore equation (4.50) becomes

$$\sum_{i=1}^N a(\phi_i, \phi_j) U_i = \lambda^h \sum_{i=1}^N (\phi_i, \phi_j) U_i \quad j = 1, \dots, N \quad (4.52)$$

or

$$\mathbf{M}\mathbf{U} = \lambda^h \mathbf{M}_0 \mathbf{U} \quad (4.53)$$

which represents a generalised algebraic eigenvalue problem. The solution of this system of equations yields the approximate eigensystems which are used to generate spectral expansions in problems such as tidal dynamics (Smith, 1995). The essential step in the solution of eigenvalue problems is to find λ^h by setting the characteristic polynomial of equation (4.53) to zero. After the insertion of the eigenvalues found by the solution of characteristic equation into the standard eigenvalue problem it becomes identical to the solution of the Sturm-Liouville equations. However, it should be noted that equation (4.53) is in general sensitive to small perturbations of its coefficients. Therefore small errors in the calculation of eigenvalues may give rise to unacceptably large errors in the calculation of eigenfunctions. In the following examples the standard eigenvalue problem is solved using Dirichlet and Neumann boundary conditions and in each case by the comparison with the analytical eigenvalues the error of the numerical solutions are found.

Test problem No. 1 Consider standard eigenvalue problem given as

$$-\frac{d}{dx} \left[\frac{du}{dx} \right] - \lambda u = 0 \quad (4.54)$$

subject to the boundary conditions $u(0) = 0$ and $u(1) = 0$. To solve this problem the following set of input were used

$$\left\{ \begin{array}{l} Precision = 50 \\ \hat{N} = \{0, 0.1, \dots, 1.\} \\ \hat{P} = \{x(1-x), \dots, (1-x)^9x\} \\ \hat{W} = (1. - 0.907029x^2)^2 \\ \hat{Q} = Gauss-Legendre \text{ order} = 18 \end{array} \right. \quad (4.55)$$

By the comparison of the analytical and numerical eigenvalues we have

$$\begin{aligned} Error[\lambda_1] &:= 9.9458210^{-12} \\ Error[\lambda_2] &:= 4.7614810^{-8} \\ Error[\lambda_3] &:= 0.000191657 \\ Error[\lambda_4] &:= 0.0432937 \\ Error[\lambda_5] &:= 1.70621 \\ Error[\lambda_6] &:= 21.1687 \\ Error[\lambda_7] &:= 135.23 \\ Error[\lambda_8] &:= 666.435 \\ Error[\lambda_9] &:= 4174.58 \end{aligned}$$

The increasing error of eigenvalues renders the members of eigenfunctions corresponding to eigenvalues beyond the first three leading values useless. However, in most practical eigenfunction expansions only leading members are used. As it is shown in the solution of Sturm-Liouville equations, if it is needed, we can reduce the error of the intermediate and final calculations in this application by increasing the order of quadrature.

Test problem No. 2 Solution of standard eigenvalue problem given as

$$-\frac{d}{dx} \left[\frac{du}{dx} \right] - \lambda u = 0 \quad (4.56)$$

subject to the mixed boundary conditions of $u(0) = 0$ and $u'(1) = 0$. To solve this problem the following set of input were used

$$\left\{ \begin{array}{l} Precision = 50 \\ \hat{N} = \{0, 0.142875, \dots, 0.857143, 1.\} \\ \hat{P} = \{x, x^2, \dots, x^7\} \\ \hat{W} = (1. - 0.907029x^2)^2 \\ \hat{Q} = Gauss-Legendre \text{ order}=78 \end{array} \right. \quad (4.57)$$

In this case again the leading eigenfunctions were found with sufficient accuracy. The corresponding error for eigenvalues are

$$Error[\lambda_1] := 0.000332956$$

$$Error[\lambda_2] := 0.00206973$$

$$Error[\lambda_3] := 0.0650484$$

$$Error[\lambda_4] := 1.36772$$

$$Error[\lambda_5] := 24.2389$$

$$Error[\lambda_6] := 161.331$$

$$Error[\lambda_7] := 1263.99$$

This test problem corresponds to the generation of the eigenfunctions required in the modal expansion of tidal dynamics equations. Under practical conditions only two or at most three modal tidal flow equations are used and hence only the error of the first three eigenvalues affect the outcome of the spectral expansions in these problems.

4.3.3 Numerical Solution of Fourth Order Ordinary Differential Equations

Consider the standard fourth order differential equation expressed as

$$\frac{d^2}{dx^2} \left[p(x) \frac{d^2 u}{dx^2} \right] + q(x)u(x) = f(x), \quad x \in [\alpha, \beta] \quad (4.58)$$

subject to boundary conditions

$$u(\alpha) = u'(\alpha) = 0, \quad u(\beta) = u'(\beta) = 0 \quad (4.59)$$

In classical case, where the right hand side, i.e. the function f , is smooth we define

$$a(u, v) = \int_{\alpha}^{\beta} \left[\frac{d^2}{dx^2} \left[p(x) \frac{d^2 u}{dx^2} \right] + q(x)u \right] v dx \quad (4.60)$$

integration by part gives

$$\begin{aligned} \int_{\alpha}^{\beta} \left[\frac{d^2}{dx^2} \left[p(x) \frac{d^2 u}{dx^2} \right] v dx = \right. \\ \left. - \int_{\alpha}^{\beta} \left[\frac{d}{dx} \left[p(x) \frac{d^2 u}{dx^2} \right] \frac{dv}{dx} dx \frac{d^2}{dx^2} \left[p(x) \frac{d^2 u}{dx^2} \right] v \right]_{\alpha}^{\beta} \end{aligned} \quad (4.61)$$

After the application of the boundary conditions and simplification and again using integration by part we have

$$\begin{aligned} \int_{\alpha}^{\beta} \left[\frac{d^2}{dx^2} \left[p(x) \frac{d^2 u}{dx^2} \right] v dx = \right. \\ \left. \int_{\alpha}^{\beta} \left[p(x) \frac{d^2 u}{dx^2} \frac{d^2 v}{dx^2} dx + - \left[p(x) \frac{d^2 u}{dx^2} \frac{dv}{dx} \right]_{\alpha}^{\beta} \right] \end{aligned} \quad (4.62)$$

Again using boundary conditions we have

$$a(u, v) = \int_{\alpha}^{\beta} \left[p(x) \frac{d^2 u}{dx^2} \frac{d^2 v}{dx^2} + q(x)uv \right] dx \quad (4.63)$$

Using bilinear form defined by equation (4.63) and the following linear functional

$$F(v) = \int_{\alpha}^{\beta} v f dx \quad (4.64)$$

the variational formulation of the defined boundary value problem is now represented as

$$\text{Find } u \in H_0^2(\alpha, \beta), \text{ such that} \quad (4.65)$$

$$a(u, v) = F(v) \quad \text{for all } v \in H_0^2(\alpha, \beta)$$

which after the discretisation in usual manner gives the following working equations

$$\sum_{j=1}^N a(\phi_i, \phi_j) c_j = F(\phi_i) \quad j = 1, \dots, N \quad (4.66)$$

Test problem No. 1 Solution of a fourth order O.D.E. with a smooth r.h.s.

$$\frac{d^2}{dx^2} \left[(1+x^2) \frac{d^2 u}{dx^2} \right] + xu = x^2(360 - 1680x + 2580x^2 - 3528x^3 + 3136x^4 + x^5 - 2x^6 + x^7) \quad (4.67)$$

subject to $u(0) = u'(0) = 0$ and $u(1) = u'(1) = 0$. To solve this problem the following set of input were used

$$\left\{ \begin{array}{l} \text{Precision} = 50 \\ \hat{N} = \{0., 0.166667, \dots, 0.833333, 1.\} \\ \hat{P} = \{(1-x)^2 x^2, \dots, (1-x)^2 x^6\} \\ \hat{W} = (1 - 1.416666x^2)^2 \\ \hat{Q} = \text{Gauss-Legendre order}=10 \end{array} \right. \quad (4.68)$$

The discrete uniform norm of error for this approximation is

$$DU\text{-Error} = 0. \quad (4.69)$$

This corresponds to the super-convergent result shown in figure 4.3.3a.

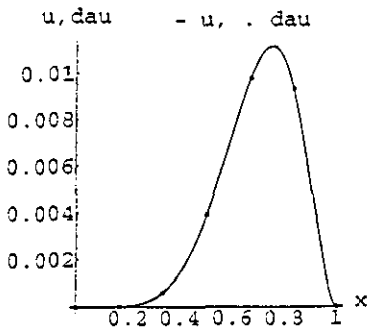


Fig. 4.3.3a.

Test problem No. 2 Solution of a fourth order O.D.E. with r.h.s. corresponding to a function with a singular point

$$\frac{d^2}{dx^2} \left[\frac{d^2 u}{dx^2} \right] + u = x^{-\frac{3}{2}} (-15 - 210x + 945x^2 + 16x^4 - 32x^5 + 16x^6) \quad (4.70)$$

subject to $u(0) = u'(0) = 0$ and $u(1) = u'(1) = 0$. To solve this problem the following set of input were used

$$\left\{ \begin{array}{l} \text{Precision} = 50 \\ \hat{N} = \{0., 0.166667, \dots, 0.833333, 1.\} \\ \hat{P} = \{(1-x)^2 x^2, \dots, (1-x)^2 x^6\} \\ \hat{W} = (1 - 1.416666x^2)^2 \\ \hat{Q} = \text{Gauss-Legendre order}=10 \end{array} \right. \quad (4.71)$$

The result of this approximation is shown in figure 4.3.3b.

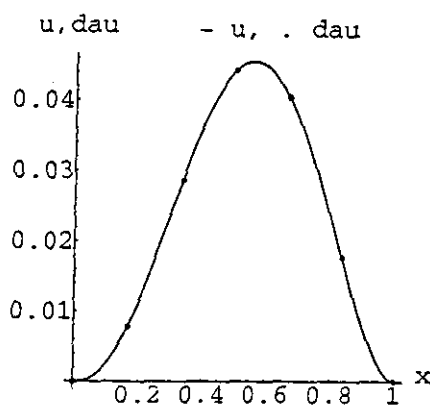


Fig. 4.3.3b.

The discrete uniform error for this solution is

$$DU\text{-Error} = 0.0000523096 \quad (4.72)$$

This error can be reduced by replacing the base function $(1-x)^2x^6$ with $(1-x)^2x^{\frac{5}{2}}$ and increasing the order of quadrature to 15 which gives

$$DU\text{-Error} = 0.0000218514 \quad (4.73)$$

Test problem No. 3 Solution of a fourth order O.D.E. with r.h.s. corresponding to a function with two singular points

$$\frac{d^2}{dx^2} \left[\frac{d^2 u}{dx^2} \right] + u = \frac{1}{16} x^{-\frac{3}{2}} (1-x)^{-\frac{1}{2}} (-15 - 360x + 4320x^2 - 9600x^3 + 5776x^4 - 64x^5 + 96x^6 - 64x^7 + 16x^8) \quad (4.74)$$

subject to $u(0) = u'(0) = 0$ and $u(1) = u'(1) = 0$. To solve this problem the following set of input were used

$$\left\{ \begin{array}{l} Precision = 50 \\ \hat{N} = \{0., 0.166667, \dots, 0.833333, 1.\} \\ \hat{P} = \{(1-x)^2x^2, \dots, (1-x)^2x^6\} \\ \hat{W} = (1 - 1.416666x^2)^2 \\ \hat{Q} = \text{Gauss-Legendre order}=10 \end{array} \right. \quad (4.75)$$

The analytical solution of the given differential equation is

$$u = (1-x)^{\frac{7}{2}} x^{\frac{5}{2}} \quad (4.76)$$

As it is clear from the input parameters in this problem we used the standard diffuse approximation method with polynomial base and weight functions and a relatively low order quadrature. However, despite the existence of two singular points an accurate result with the following small error was obtained.

$$DU\text{-Error} = 0.0000642917 \quad (4.77)$$

The numerical and the analytical solutions of this test problem are compared in figure 4.3.3c.

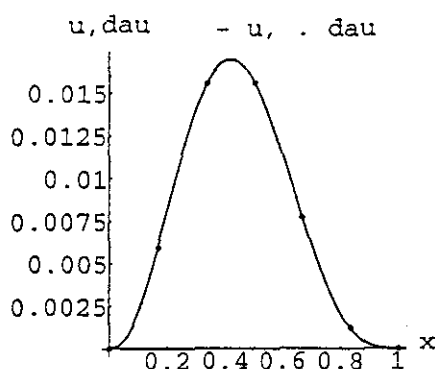


Fig. 4.3.3c.

Test problem No. 4 Solution of a fourth order O.D.E. with r.h.s. corresponding to a function with strong singularity

$$\frac{d^2}{dx^2} \left[\frac{d^2 u}{dx^2} \right] + u = f(x) \quad (4.78)$$

subject to $u(0) = u'(0) = 0$ and $u(1) = u'(1) = 0$. The right hand side is given as

$f(x) =$

$$\begin{aligned} & \frac{1375000 (1-2x)^2 (-1+x)^2}{(13-50x+50x^2)^4} - \frac{20000 (-1+x)^2}{(13-50x+50x^2)^3} - \frac{60000 (-1+x) (-1+2x)}{(13-50x+50x^2)^3} + \frac{600}{(13-50x+50x^2)^2} - \\ & \frac{7500000 (-1+x)^2 (-1+2x)^3 (\text{ArcTan}[5] + \text{ArcTan}[10(-\frac{1}{2}+x)])}{(13-50x+50x^2)^4} + \\ & \frac{400000 (1-2x)^2 (-1+x) (\text{ArcTan}[5] + \text{ArcTan}[10(-\frac{1}{2}+x)])}{(13-50x+50x^2)^3} + \\ & \frac{300000 (-1+x)^2 (-1+2x) (\text{ArcTan}[5] + \text{ArcTan}[10(-\frac{1}{2}+x)])}{(13-50x+50x^2)^3} - \\ & \frac{8000 (-1+x) (\text{ArcTan}[5] + \text{ArcTan}[10(-\frac{1}{2}+x)])}{(13-50x+50x^2)^2} - \frac{6000 (-1+2x) (\text{ArcTan}[5] + \text{ArcTan}[10(-\frac{1}{2}+x)])}{(13-50x+50x^2)^2} + \\ & (-1+x)^2 (\text{ArcTan}[5] + \text{ArcTan}[10(-\frac{1}{2}+x)])^2 \end{aligned}$$

The analytical solution of equation (4.78) is

$$u = (1 - x)^2 (\text{ArcTan}[5] + \text{ArcTan}[10(x - \frac{1}{2})])^2 \quad (4.79)$$

Using the following input corresponding to the standard diffuse approximation method a numerical result which is very close to the analytical solution is found ($DU\text{-Error} = 0.$). As it can be seen from the following data a very high order quadrature was used to find such an accurate result.

$$\left\{ \begin{array}{l} \text{Precision} = 50 \\ \hat{N} = \{0., 0.166667, \dots, 0.833333, 1.\} \\ \hat{P} = \{(1 - x)^2 x^2, \dots, (1 - x)^2 x^6\} \\ \hat{W} = (1 - 1.4166666x^2)^2 \\ \hat{Q} = \text{Gauss-Legendre order}=78 \end{array} \right. \quad (4.80)$$

The numerical and the analytical solutions of this test problem are compared in figure 4.3.3d.

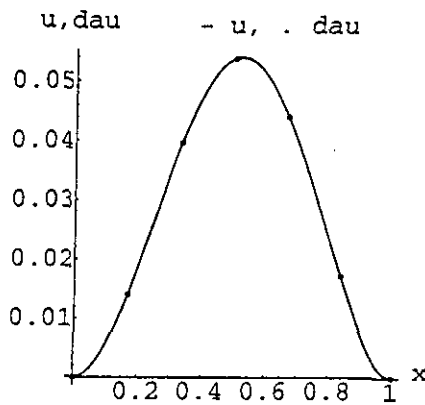


Fig. 4.3.3d.

4.4 Approximation of the Trace Operator

The trace of a smooth function u is denoted by $u|_{\partial\Omega}$, which defines the restriction of u on the boundary given by $\partial\Omega$. The main applications of this

operator are in the theory of partial differential equations, derivation of the Green's formula and in the characterisation of the Hilbert spaces such as $H^1(\Omega)$ and $H^2(\Omega)$. Mathematically rigorous definition of the trace operator is based on the following expression which characterises the operator as a continuous linear mapping (Ciarlet, 1978)

$$\gamma_0 : H^1(\Omega) \rightarrow L^2(\Omega) \quad (4.81)$$

In order to develop a diffuse approximation for the trace operator we consider the following problem.

$$\text{Find } u : \widehat{\Omega} \rightarrow R \text{ such that } \gamma_0 u = g \quad (4.82)$$

This is similar to the mapping problem described earlier in this chapter. Therefore the approximation of the trace operator can essentially be treated as a surface fitting application. The details of the generation of the diffuse approximation scheme for a surface fitting problem is described in section 4.1 and is not given here. However, in this section we give a new application for this scheme which can be used to solve differential equations with non-homogeneous boundary conditions. For simplicity we describe this technique in the context of the solution of the Poisson equation with non-homogeneous Dirichlet boundary conditions and hence we restrict ourselves to the trace of the function value itself. Consider the following Poisson equation

$$\begin{cases} -\Delta u(\mathbf{x}) = f & \text{in } \Omega \\ u = g & \text{on } \partial\Omega \end{cases} \quad (4.83)$$

where $g \in H^{\frac{1}{2}}(\partial\Omega)$ and $f \in L^2(\Omega)$.

According to the trace theorem there exists $\tilde{u} \in H^1(\Omega)$ such that

$$\tilde{u}|_{\partial\Omega} = \gamma_0(\tilde{u}) = g \quad (4.84)$$

Now we define

$$K = \{v | v \in H^1(\Omega), v - \tilde{u} \in H_0^1(\Omega)\} \quad (4.85)$$

It can be easily shown that K is a closed convex set in $H^1(\Omega)$ (Kesavan, 1989). The corresponding weak fomulation of this problem is

$$\begin{aligned} & \text{Find } u \in H^1(\Omega) \quad \text{such that} \\ & a(u, v) = F(v) \quad \text{for all } v \in H_0^1(\Omega) \end{aligned} \quad (4.86)$$

where

$$a(u, v) = \int_{\Omega} \nabla u \nabla v d\Omega \quad (4.87)$$

and

$$F(v) = \int_{\Omega} f v d\Omega \quad (4.88)$$

The solution depends on the continuity of u with respect to f and g . We assume that $g \in H^{\frac{1}{2}}(\partial\Omega)$ therefore there exits a constant C independent of g such that

$$\| u \|_{1,\Omega} \leq C(|f|_{0,\Omega} + |g|_{\frac{1}{2},\partial\Omega}) \quad (4.89)$$

The required continuity of u in terms of the f and g , is stated by inequality (4.89). This inequality is always true, in Hadamard sense (Stoer and Bulirsch 1980), and hense the described variational problem is well posed and the existence and uniqueness of its solution is guaranteed. The application of the trace operator to the right hand side of the variational problem (i.e. equation 4.88) gives

$$F_0(v) = \int_{\Omega} f v d\Omega - \int_{\Omega} \nabla \tilde{u} \cdot \nabla v d\Omega \quad (4.90)$$

which corresponds to a problem with homogeneous boundary conditions. However, in general it is not possible to find the trace function \tilde{u} to carry out such a simplification. Accurate calculation of a numerical function \tilde{u}^h by the diffuse approximation technique resolves this problem. In the following section we describe the diffuse approximation solution of partial differential equations with homogeneous boundary conditions. The combination of the approximation of trace operator with such solutions provides an algorithm for the solution of differential equations with non-homogeneous boundary conditions.

Test problem No. 1 Let Ω be a unit square domain with a function g defined on its boundary $\partial\Omega$ as

$$g : \partial\Omega \rightarrow R$$

where

$$g(\mathbf{x}) = x_1(1 - x_1)(1 - x_2)[ArcTan(15) + ArcTan(30(x_2 - \frac{1}{2}))] \quad (4.91)$$

The approximation of the trace operator is based on a diffuse approximation mapping carried out using the following input data

$$\left\{ \begin{array}{l} Precision = 50 \\ \hat{N} = \{\{0, 0\}, \{0, \frac{1}{2}\}, \dots, \{1, \frac{1}{2}\}, \{1, 1\}\} \\ \hat{P} = \{x_1^i x_2^j, \quad i, j = 0, 2\} \\ \widehat{W} = (1 - 0.565685 \|\mathbf{x}\|^2)^2 \end{array} \right. \quad (4.92)$$

In order to make this approximation accurate the mapping is not restricted to the boundary nodes (see section 4.2, test problem no. 1). However, this does not result in loss of generality the present solution. This is because that according to expression (4.82) the only objective to be satisfied is to make the traced function (u) equal to g on the boundary of the domain and in the

interior nodes it can take arbitrary values. In this example in the interior nodes u is assumed to be one. As it can be seen in figure 4.4a, although, the defined function has a sharp bend in the interior of the domain, nevertheless it is smooth along the boundary (it is uniformly zero on the boundary). In this case a very accurate result for this mapping (trace approximation) is generated which its error on random points located on the boundary is of the order of 10^{-14} . We considered a counter example in which

$$g(\mathbf{x}) = (1 - x_1)(1 - x_2)[ArcTan(15) + ArcTan(30(x_2 - \frac{1}{2}))] \quad (4.93)$$

This function corresponds to figure 4.4b and as it can be seen it has a sharp knee on the boundary as well as bending within the domain. The maximum error on random points located on the boundary in this case, using the same input as before, is 0.55. A conclusion which can be drawn from this example is that in the solution of nonhomogeneous boundary value problems by the present method it may be necessary to use two different sets of input for trace approximation and the solution of the resulting homogeneous problem.

4.5 Numerical Solution of Partial Differential Equations

Let $\Omega \subseteq R^k$ be an open bounded domain with a sufficiently smooth boundary $\partial\Omega$ and $\widehat{\Omega} = \Omega \cup \partial\Omega$. We consider the problem

$$-\sum_{i,j=1}^n \frac{\partial}{\partial x_i} (a_{ij} \frac{\partial u}{\partial x_j}) + \sum_{i=1}^k a_i \frac{\partial u}{\partial x_i} + a_0 u = f \quad \text{in } \Omega \quad (4.94)$$

$$u = 0 \quad \text{on } \partial\Omega \quad (4.95)$$

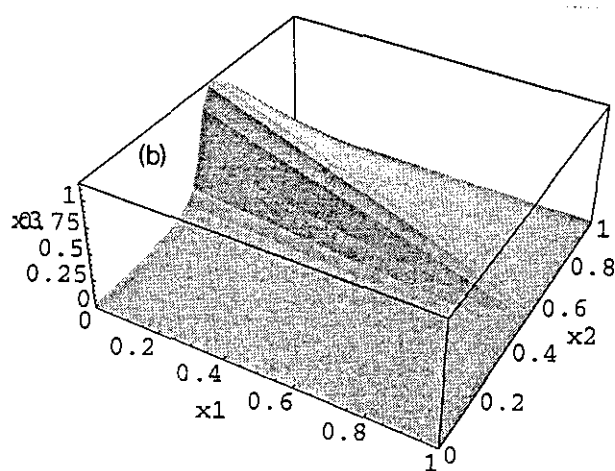
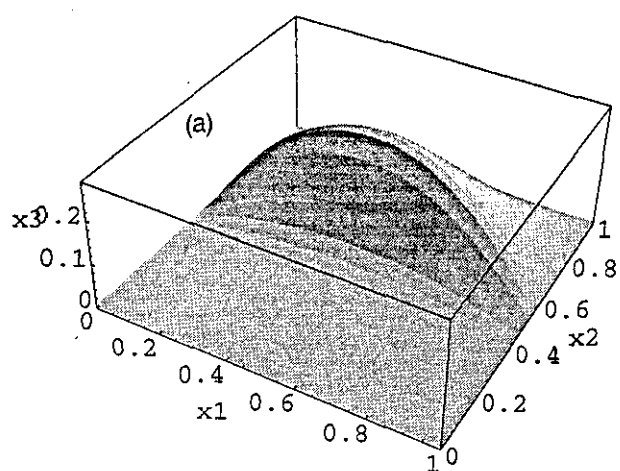


Fig. 4.4.

The variational formulation corresponding to this equation can be expressed as

$$\begin{aligned} & \text{Find } u \in H_0^1(\Omega) \text{ such that} \\ & a(u, v) = F(v) \text{ for all } v \in H_0^1(\Omega). \end{aligned} \quad (4.96)$$

where

$$a(u, v) = \int_{\Omega} \left[\sum_{i,j=1}^k a_{ij} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_j} + \sum_{i=1}^k a_i \frac{\partial u}{\partial x_i} v + a_0 uv \right] d\Omega \quad (4.97)$$

and

$$F(v) = \int_{\Omega} f v d\Omega \quad (4.98)$$

The bilinear form $a(u, v)$ cannot, in general, be assumed to be symmetric and hence the existence and uniqueness of a solution for this equation is not guaranteed. However, the investigation of the solution of this equation in its most general form is outside the scope of this project and to avoid the existence and uniqueness problems we only focus on the solution of elliptic equations.

4.5.1 Poisson Equation with Homogeneous Boundary Conditions

The Poisson equation is a special case of the problem given by equation (4.94) and is expressed as

$$-\sum_{i,j=1}^n \frac{\partial}{\partial x_i} \left(a_{ij} \frac{\partial u}{\partial x_j} \right) + a_0 u = f \quad \text{in } \Omega \quad (4.99)$$

subject to the following boundary condition

$$u = 0 \quad \text{on } \partial\Omega \quad (4.100)$$

In equation (4.97) the coefficient $a_{ij} \in C^1(\widehat{\Omega})$ satisfies the ellipticity condition and $a_0 \in C(\widehat{\Omega})$. If $f \in L^2(\Omega)$ then a weak solution of this problem is $u \in H_0^1(\Omega)$ satisfying

$$a(u, v) = F(v), \quad \text{for all } u \in H_0^1(\Omega) \quad (4.101)$$

where

$$a(u, v) = \int_{\Omega} \left[\sum_{i,j=1}^k a_{ij} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_j} + a_0 uv \right] d\Omega \quad (4.102)$$

and

$$F(v) = \int_{\Omega} f v d\Omega \quad (4.103)$$

To derive the working equation of the present scheme we consider

$$\begin{cases} -\Delta u(\mathbf{x}) = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \quad (4.104)$$

where $\Omega \subseteq R^2, k \geq 1$ is a bounded open set with a smooth boundary $\partial\Omega$.

It can be shown that if $\partial\Omega$ is smooth enough and $f \in L^2(\Omega)$ then $u \in H^2(\Omega) \cap H_0^1(\Omega)$ and if $f \in H^m(\Omega)$, then $u \in H^{m+2}(\Omega)$ (Kesavan 1989). In this case there exists a constant $C > 0$ depending on Ω such that

$$\| u \|_{H^{m+2}(\Omega)} \leq C \| f \|_{H^m(\Omega)} \quad (4.105)$$

Inequality (4.105) gives the continuity of u in terms of f . Because the existence, uniqueness and continuity of the solution are always guaranteed therefore the stated problem is 'well-posed' in Hadamard sense (Stoer and Bulirsch 1980).

Analogous to the ordinary differential equations the main steps in the derivation of the working equation of the partial differential equations are (i) the discretisation of variational problem by a weighted residual technique such

as the Galerkin method and (ii) substitution of u^h in the weak formulation in terms shape functions given as

$$u^h = \sum_{ij} \phi_{ij} u_{ij} \quad (4.106)$$

This leads to the derivation of the working equation of the diffuse approximation method for the Poisson equations expressed as

$$\sum_{i,j} a(\phi_{ij}, \phi_{rs}) u_{ij} = F(\phi_{rs}) \quad (4.107)$$

Test problem No. 1 Solution of a Poisson equation with a smooth right hand side subject to homogeneous boundary condition

$$\begin{cases} -\Delta u(\mathbf{x}) = -2(x_1^2 + x_2^2 - x_1 - x_2) & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \quad (4.108)$$

where $\mathbf{x} = \{x_1, x_2\}$ and $\Omega = [0, 1] \times [0, 1]$. The analytical solution of this equation is

$$u = x_1(1 - x_1)x_2(1 - x_2) \quad (4.109)$$

In this test problem the given input data are

$$\begin{cases} Precision = 10 \\ \hat{N} = \{\{0, 0\}, \{0, \frac{1}{3}\}, \dots, \{1, \frac{2}{3}\}, \{1, 1\}\} \\ \hat{P} = \{x_1^i(1 - x_1)x_2^j(1 - x_2), \quad i, j = 1, 2\} \\ \hat{W} = (1 - 0.222222 \|\mathbf{x}\|^2)^2 \\ \hat{Q} = \text{Gauss-Legendre } orderx1=4, orderx2=4 \end{cases} \quad (4.110)$$

where the two dimensional quadrature is constructed by the use of two quadratures of orders $orderx1$ and $orderx2$. The discrete uniform error norm in this solution is

$$DU - Error[u] := 0.0100796 \quad (4.111)$$

The numerical and the analytical solutions in the interior nodes of this domain are compared in the following table

NODE NO.	ANALYTICAL SOLUTION	NUMERICAL SOLUTION
1	0.0493827	0.0393030
2	0.0493827	0.0413390
3	0.0493827	0.0413390
4	0.0493827	0.0434812

Test problem No. 2 Solution of a Poisson equation with two singular points subject to homogeneous boundary conditions. Consider the following equation

$$\begin{cases} -\Delta u(x) = f(x) \\ u = 0 \quad \text{on} \quad \partial\Omega \end{cases} \quad (4.112)$$

where $x = \{x_1, x_2\}$, $\Omega = [0, 1] \times [0, 1]$ and the r.h.s. is

$$f(x) = - (4 (-10 x_1 (-1 + x_2)^3 x_2^3 + (-1 + x_2)^2 x_2^4 + x_1^6 (1 - 10 x_2 + 10 x_2^2) + 2 x_1^5 (-1 + 15 x_2 - 32 x_2^2 + 18 x_2^3) + x_1^2 x_2^2 (20 - 100 x_2 + 135 x_2^2 - 64 x_2^3 + 10 x_2^4) + 2 x_1^3 x_2 (5 - 50 x_2 + 108 x_2^2 - 81 x_2^3 + 18 x_2^4) + x_1^4 (1 - 30 x_2 + 135 x_2^2 - 162 x_2^3 + 54 x_2^4))) / (9 ((-1 + x_1) x_1 (-1 + x_2) x_2 (x_1 + x_2))^{2/3})$$

The analytical solution of this equation is

$$u = (x_1(1 - x_1)x_2(1 - x_2)(x_1 + x_2))^{\frac{4}{3}} \quad (4.113)$$

In this test problem the given input data are

$$\left\{ \begin{array}{l} \text{Precision} = 50 \\ \hat{N} = \{\{0, 0\}, \{0, \frac{1}{3}\}, \dots, \{1, \frac{2}{3}\}, \{1, 1\}\} \\ \hat{P} = \{x_1^i(1 - x_1)x_2^j(1 - x_2), \quad i, j = 1, 2\} \\ \hat{W} = (1 - 0.222222 \|\mathbf{x}\|^2)^2 \\ \hat{Q} = \text{Gauss-Legendre order}x_1=4 \text{ or } 10, \text{ order}x_2=4 \text{ or } 10 \end{array} \right. \quad (4.114)$$

Because of the existence of two singular points this equation is more difficult to solve than the previous problem. However, as it is shown in the following table the diffuse approximation result can be improved by increasing the order of quadrature.

NODE NO.	ANALYTICAL SOLUTION	NUMERICAL SOLUTION order of quadrature = 4x4	NUMERICAL SOLUTION order of quadrature = 10x10
1	0.0105514	0.0107744	0.0108028
2	0.0181175	0.0168931	0.0170962
3	0.0181175	0.0168931	0.0170962
4	0.0265879	0.0236961	0.0240897

The improvement of the result of the numerical solution by increasing the

order of the quadrature is reflected on the discrete uniform error norm for the solutions shown in the above table. These norms of error are

$$DU - Error[u] := 0.00289178$$

and

$$DU - Error[u] := 0.00249819,$$

respectively.

Test problem No. 3 Generalised solution of the Poisson equation subject to homogeneous boundary conditions. To show the flexibility of the diffuse approximation method in solving Poisson equations with highly non-smooth right hand sides we consider the following problem. This problem was originally proposed by Rachford and Wheeler, (1974) to test the convergence property of the H^{-1} -Galerkin method, and was used again by Babuska et al. (1977) to test the mixed-hybrid finite element method.

$$\begin{cases} -\Delta u(\mathbf{x}) = f(\mathbf{x}) & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \quad (4.115)$$

where $\mathbf{x} = \{x_1, x_2\}$ and $\Omega = [0, 1] \times [0, 1]$ and

$$f(\mathbf{x}) =$$

$$\begin{aligned} & - \left(8 \left((-150 + 145 x_1 - 288 x_1^2 + 264 x_1^3 - 144 x_1^4 + 48 x_1^5 + 145 x_2 + 260 x_1 x_2 - 192 x_1^2 x_2 + 176 x_1^3 x_2 - \right. \right. \\ & \quad 96 x_1^4 x_2 + 32 x_1^5 x_2 - 288 x_2^2 - 192 x_1 x_2^2 + 264 x_2^3 + 176 x_1 x_2^3 - \\ & \quad \left. 144 x_2^4 - 96 x_1 x_2^4 + 48 x_2^5 + 32 x_1 x_2^5) \operatorname{ArcTan}\left[\frac{1}{2}\right] - \right. \\ & \quad (-1 + x_1) (5 - 4 x_1 + 4 x_1^2)^2 (3 + 2 x_2) \operatorname{ArcTan}\left[\frac{1}{2} - x_1\right] - \\ & \quad \left. (3 + 2 x_1) (-1 + x_2) (5 - 4 x_2 + 4 x_2^2)^2 \operatorname{ArcTan}\left[\frac{1}{2} - x_2\right] \right) \right) / \\ & ((5 - 4 x_1 + 4 x_1^2)^2 (5 - 4 x_2 + 4 x_2^2)^2) \end{aligned}$$

The analytical solution of this equation is

$$u = (1 - x_1)(1 - x_2) \left(\text{ArcTan}\left[\frac{1}{2}\nu\right] - \text{ArcTan}\left[\nu\left(\frac{1}{2} - x_1\right)\right] \right) \quad (4.116)$$

$$\left(\text{ArcTan}\left[\frac{1}{2}\nu\right] - \text{ArcTan}\left[\nu\left(\frac{1}{2} - x_2\right)\right] \right)$$

In this test problem the given input data are

$$\left\{ \begin{array}{l} \text{Precision} = 50 \\ \hat{N} = \{\{0, 0\}, \{0, \frac{1}{3}\}, \dots, \{1, \frac{2}{3}\}, \{1, 1\}\} \\ \hat{P} = \{x_1^i(1 - x_1)x_2^j(1 - x_2), \quad i, j = 1, 2\} \\ \widehat{W} = (1 - 0.222222 \|\mathbf{x}\|^2)^2 \\ \hat{Q} = \text{Gauss-Legendre } \text{order}x1=8, \text{ order}x2=8 \end{array} \right. \quad (4.117)$$

where the two dimensional quadrature is constructed from two quadratures of orders *orderx1* and *orderx2*. The result of this approximation for $\nu = 1$ is given in the following table

NODE NO.	ANALYTICAL SOLUTION	NUMERICAL SOLUTION
1	0.0163479	0.0353117
2	0.1172240	0.1653911
3	0.1172240	0.1653911
4	0.8405630	0.5665346

In order to show the improvement of the numerical results by increasing the order of quadrature we solved a more difficult equation of this type in which $\nu = 10$. Improved numerical results obtained using a 10×10 quadrature for this equation are shown in the following table

NODE NO.	ANALYTICAL SOLUTION	NUMERICAL SOLUTION
1	0.0396007	0.0393402
2	0.0417100	0.0413617
3	0.0417100	0.0413617
4	0.0439316	0.0434875

4.5.2 Biharmonic Equation with Homogeneous Boundary Conditions

Let $\Omega \in R^2$ be an open bounded domain with boundary $\partial\Omega$ and $\widehat{\Omega} = \Omega \cup \partial\Omega$.

We consider the problem

$$\begin{cases} \Delta^2 u = f & \text{in } \Omega \\ u = \frac{\partial u}{\partial \nu} = 0 & \text{on } \partial\Omega \end{cases} \quad (4.118)$$

where $f \in L^2(\Omega)$ is a given function. We define the following bilinear form

$$a(u, v) = \int_{\Omega} \Delta^2 u v d\Omega \quad (4.119)$$

Using Green's formula twice and simplifying the result by the application of boundary conditions we have

$$a(u, v) = \int_{\Omega} \Delta u \Delta v d\Omega \quad (4.120)$$

We define the linear functional F as

$$F(v) = \int_{\Omega} f v d\Omega \quad (4.121)$$

The weak formulation corresponding to the biharmonic problem (4.118) is

$$\text{Find } u \in H_0^2(\Omega) \text{ such that} \quad (4.122)$$

$$a(u, v) = F(v), \text{ for all } u \in H_0^2(\Omega) \quad (4.123)$$

The regularity of the weak solution of the biharmonic problem depends on the geometry of domain Ω . If Ω belongs to the class of C^∞ domains then $u \in H_0^4(\Omega)$. However for a polygonal domain (or generally for a *Lipschitz* domain) $u \in H_0^3(\Omega)$ (Konderatev, 1967). Using the described diffuse approximation u^h is expressed in terms of shape functions as

$$u^h = \sum_{ij} \phi_{ij} u_{ij} \quad (4.124)$$

The substitution of u^h from equation (4.124) into the discretised form of expression (4.123) and carrying out the weighted residuals procedure, such as the Galerkin method, gives the working equation of the present scheme expressed as

$$\sum_{i,j} a(\phi_{ij}, \phi_{rs}) u_{ij} = F(\phi_{rs}) \quad (4.125)$$

There is an alternative way in which a weak formulation for the biharmonic equation can be developed. This method is suggested by Ciarlet and Raviart (1974) and is based on an earlier work by Glowinski (1973). Using this method the solution of biharmonic problem can be reduced to a sequence of discrete Poisson equations with homogeneous and nonhomogeneous Dirichlet boundary conditions. However, in practice the application of this method involves the handling of nonhomogeneous boundary conditions which in general is not a trivial matter. In the context of the present diffuse approximation method we can use this technique through the method described in section 4.4 for the approximation of the trace operator.

Test problem No.1 Solution of the biharmonic equations with smooth right hand side subject to homogeneous boundary conditions. Consider the example

$$\begin{cases} \Delta^2 u = 8(1 - 6x_1^3 + \dots + 3x_2^4) & \text{in } \Omega \\ u = \frac{\partial u}{\partial \nu} = 0 & \text{on } \partial\Omega \end{cases} \quad (4.126)$$

The analytical solution of this problem is

$$u = (x_1(1 - x_1)x_2(1 - x_2))^2 \quad (4.127)$$

In this test problem the given input data are

$$\left\{ \begin{array}{l} \text{Precision} = 50 \\ \hat{N} = \{\{0, 0\}, \{0, \frac{1}{4}\}, \dots, \{1, \frac{3}{4}\}, \{1, 1\}\} \\ \hat{P} = \{x_1^i(1-x_1)x_2^j(1-x_2), \quad i, j = 2, 3\} \\ \widehat{W} = (1 - 0.3950617 \| \mathbf{x} \|^2)^4 \\ \hat{Q} = \text{Gauss-Legendre order} x_1=20 \text{ or } 35, \text{ order} x_2=20 \text{ or } 35 \end{array} \right. \quad (4.128)$$

The analytical and the numerical solutions found in the interior nodes of the problem domain using 20×20 and 35×35 quadrature are shown in the following table. As it is expected by increasing the order of quadrature the numerical solution is improved.

NODE NO.	ANALYTICAL SOLUTION	NUMERICAL SOLUTION order of quadrature =20x20	NUMERICAL SOLUTION order of quadrature =35x35
1	0.0012359	0.0021264	0.0009819
2	0.0021972	0.0018199	0.0031134
3	0.0012359	0.0021264	0.0009819
4	0.0021972	0.0018199	0.0031134
5	0.0039062	0.0089011	0.0048196
6	0.0021972	0.0018199	0.0031134
7	0.0012359	0.0021264	0.0009819
8	0.0021972	0.0018199	0.0031134
9	0.0012359	0.0021264	0.0009819

4.5.3 Three Dimensional Poisson Equation with Homogeneous Boundary Conditions

The derivation of the working equation of the diffuse approximation scheme for the three dimensional Poisson equation with homogeneous boundary conditions is very similar to the two dimensional case and is not repeated here. The main purpose of the following example was to evaluate computational time requirement for a benchmark three dimensional problem.

Test problem No.1 Consider the following equation

$$\begin{cases} -\Delta u(\mathbf{x}) = f(\mathbf{x}) & \text{on } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \quad (4.129)$$

where $\mathbf{x} = \{x_1, x_2, x_3\}$ and $\Omega = [0, 1] \times [0, 1] \times [0, 1]$ and

$$\Delta u = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} + \frac{\partial^2 u}{\partial x_3^2}$$

and

$$\begin{aligned} f(\mathbf{x}) = 2((-1 + x_2)x_2(-1 + x_3) + x_1^2(-x_2 + x_2^2 + \\ (-1 + x_3)x_3) + x_1(x_2 - x_2^2 + x_3 - x_3^2)) \end{aligned} \quad (4.130)$$

The analytical solution of this equation is

$$u = x_1 x_2 x_3 (1 - x_1)(1 - x_2)(1 - x_3) \quad (4.131)$$

In this test problem the given input data are as

$$\left\{ \begin{array}{l} \text{Precision} = 50 \\ \hat{N} = \{\{0, 0\}, \{0, \frac{1}{3}\}, \dots, \{1, \frac{2}{3}\}, \{1, 1\}\} \\ \hat{P} = \{x_1^i x_2^j x_3^k (1 - x_1)(1 - x_2)(1 - x_3), \quad i, j, k = 1, 2\} \\ \hat{W} = (1 - 0.480000 \|\mathbf{x}\|^2)^2 \\ \hat{Q} = \text{Gauss-Legendre} \\ \text{order}x1=4, \text{ or } 10 \quad \text{order}x2=4, \text{ or } 10 \quad \text{order}x3=4 \text{ or } 10 \end{array} \right. \quad (4.132)$$

where the three dimensional quadrature is constructed from three 1-dimensional quadratures of orders $orderx1$, $orderx2$ and $orderx3$. The analytical and the numerical results for this problem using $4 \times 4 \times 4$ and $10 \times 10 \times 10$ quadratures for the interior nodes in the cubical domain are given in the following table. The computational time for these runs were 2.5 and 46 hours, respectively. This indicates that for practical problems it may be necessary to use sparse matrix techniques in conjunction with parallel processing to obtain accurate diffuse approximation results.

The comparison of results obtained using $4 \times 4 \times 4$ and $10 \times 10 \times 10$ quadratures indicates that the accuracy of solution cannot simply be increased by only increasing the order of the quadrature used. In order to achieve better results order of quadrature, number of nodes and degree of polynomial base function should be changed in combination.

NODE NO.	ANALYTICAL SOLUTION	NUMERICAL SOLUTION order of quadrature = $4 \times 4 \times 4$	NUMERICAL SOLUTION order of quadrature = $10 \times 10 \times 10$
1	0.0109739	0.0110009	0.0110006
2	0.0109739	0.0109781	0.0109771
3	0.0109739	0.0109781	0.0109780
4	0.0109739	0.0110009	0.0110006
5	0.0109739	0.0110009	0.0110006
6	0.0109739	0.0109781	0.0109771
7	0.0109739	0.0109781	0.0109771
8	0.0109739	0.0110009	0.0110006

Chapter 5

Conclusions and Suggestions for Further Developments

The main topic of the present research has been the generalisation of the diffuse approximation technique and its application to the derivation of a meshless scheme for the boundary value problems. These objectives have, essentially, been fulfilled and the theoretical foundation of the 'General Diffuse Approximation' has been established.

A consequence of the theoretical generalisation of the diffuse approximation scheme is the derivation of the classical interpolation models and the finite difference and the finite element procedures as the special cases of this method. Therefore the effort has been focused on the development of a computational environment where by the utilisation of appropriate algorithms any of the described approximations, as well as the general scheme itself, can be used. The potential benefit of adopting such an approach is that, in any given problem, the most suitable approximation technique can be readily generated. Fundamental requirements for the establishment of such an environment are: the definition of a general approximation space in which, by

the manipulation of the input, the nature of the approximation scheme can be determined, and the use of symbolic programming to achieve the required flexibility in the computational algorithms. The general diffuse approximation technique developed in this project satisfies these conditions and hence the research has lead to the creation of a powerful practical scheme which has a very wide range of applicability.

The applications of this scheme to data fitting, plane mapping, solution of ordinary differential equations, eigenvalue problems and the solution of elliptic partial differential equations have been investigated. The analyses of the results obtained by the described applications provide the basis for the main conclusions of this project. These conclusions are as follows.

1. The developed diffuse approximation scheme provides a robust technique for dealing with non-smooth and irregular functions in all of the above described problems. The flexibility of the scheme allows the utilisation of weight and base functions with any desired order of continuity which makes it particularly appropriate for irregular problems. Therefore in addition to the generation of highly accurate solutions for regular examples in each category of described applications the capabilities and the suitability of the developed scheme in these problems is demonstrated by:
 - i. Generation of super-convergent fit for discontinuous functions
 - ii. Obtaining very accurate mappings between geometrically complex source domains and regularly shaped target domains
 - iii. Obtaining accurate generalised solutions for non-smooth and singular Sturm-Liouville equations
 - iv. Obtaining accurate solutions for fourth order ordinary differential

equations with non-smooth right hand sides

- v. Obtaining accurate generalised solutions for Poisson equations with non-smooth right hand sides.
2. The application of the diffuse approximation to the mapping of boundary value problems shows an inherent difficulty associated with the use of mesh dependent methods such as the finite element technique. The underlying reason for this difficulty is the inability of piecewise polynomials, used in the mesh dependent methods, to generate smooth transformations for the unknown functions between source and target domains. The numerical tests show that the transformation errors for irregular functions can be orders of magnitude higher than the error of the geometrical mapping itself. Unlike the ordinary discretisation error, which can be reduced by mesh refinement, this problem cannot be resolved if a domain sub-division is used.
 3. Although the facility to use different combinations of weight and base functions and the quadrature method is the core reason for the power of the diffuse approximation technique, nevertheless, without the use of symbolic programming, in practical problems the full potential of the method cannot be realised.
 4. The detailed comparison of the present general diffuse approximation scheme with the previously reported meshless methods has showed the weaknesses of the latter methods. These weaknesses mainly stem from the use of specific techniques to formulate the meshless approximation which restrict the general applicability of the developed method. At a practical level, however, in cases where the mathematical modelling of a complicated problem is the immediate objective of the approximate

solution it may be more convenient to use a less general approach to generate solutions. It is envisaged that by further developments of the computational tools, such as the computer hardware and symbolic programming, this option may become totally redundant in near future.

Suggestions for Further Work .

In this project the main focus has been on the development of the general diffuse approximation technique and its immediate applications in different classes of problems of practical interest. However, there are important aspects of this techniques which need further investigations. The most important items amongst these aspects of the developed scheme which should be addressed in future are as follows.

In chapter two, the previous works related to the stability and convergence analyses of the meshless methods are mentioned. Obviously a more general approach is needed to provide such analyses for the present scheme. This will inevitably give a theoretically more rigorous error analysis for the generalised scheme. This is because that, as it is shown in chapter three, the process of generalisation itself provides a theoretical facility for these analyses.

The speed of computations required in general diffuse approximation solutions can be enhanced using following techniques:

- i. Development of computer programs based on using a combination of the symbolic language with other languages, such as extended FORTRAN or C++, in a way that purely numerical calculations can be done by specifically written efficient algorithms,
- ii. The utilisation of parallel processing algorithms in the development of computer programs

- iii. The application of numerical procedures such as the sparse matrix techniques to increase the speed of calculations.

A consequence of the generalisation of the diffuse approximation technique has been the possibility of developing a semi-discrete method in which the left hand side on the working equation of the approximation scheme is not discretised. The development of such a method will provide a scheme with inherently better accuracy. It is also self-evident that a semi-discretised scheme, properly implemented, will enhance the speed of computations.

In this project we have only considered the solution of elliptic boundary value problems, however, there is no theoretical reason to restrict the method only to this type of partial differential equations. The method can easily be extended to parabolic and hyperbolic equations and initial value problems.

References

- Babuska, I., Oden, J. T., and Lee, J. K., Mixed-hybrid finite element approximations of second-order elliptic boundary-value problems, *Comput. Methods Appl. Mech. Engrg.* 11 (1977) 175-206
- Barnhill, R. E., Representation and Approximation of Surfaces, in *Mathematical SOFTWARE III*, Academic Press, New York (1977) 69-120.
- Belytschko, T., Gu, L., and Lu, Y. Y., Fracture and crack growth by element free Galerkin methods, *Model. Simul. Sci. Engrg.* 115 (1994-b) 277-286.
- Belytschko, T., Krongauz, Y., Organ, D., Fleming, M., and Krysl, P., Meshless methods: an overview and recent developments, *Comput. Meth. Appl. Mech. Engrg.* 139 (1996) 3-47.
- Belytschko, T., Lu, Y.Y., and Gu, L., Element-Free Galerkin Methods, *Int. J. Numer. Methods Engg.* 37 (1994-a) 229-256.
- Berener, S.C., and Scott, L.R., (1994) *The Mathematical Theory of Finite Element Methods*, Springer-Verlag, New York.
- Ciarlet, P.G., (1978) *The Finite Element Method for Elliptic Problems*, North-Holland Publishing Company Amsterdam.
- Ciarlet, P.G., and Raviart, P.A., General Lagrange and Hermite interpolation in R^n with applications to finite element methods, *Arch. Rational Mech. Anal.* 46 (1972-a) 177-199.

Coddington, E.A., and Levinson, N., (1955) Theory of ordinary differential equations, McGraw-Hill, New York.

Duarte, C. A. and Oden, J. T., Hp Clouds- a meshless method to solve boundary value problems, Technical report 95-105, Texas Institute for computational and applied mathematics, University of Texas at Austin, (1995)

Glowinski, R., Approximation externes, par elements finis de Lagrange d' order un et deux, du probleme de Dirichlet pour l' operateur biharmonique, in Topics in Numerical Analysis (1973) (J. J. H. Miller Editor) Academic Press London.

Gordon, W.J., and Wixom, J.A., Shepard's method of 'metric interpolation' to bivariate and multivariate data, Math. Comput. 32 (1978) 253-264.

Han, C.T., Tsai, C.C., Yu, T.A., Liu, T.J., A finite-difference technique for solving the Newtonian jet swell problems, Int. J. Numer. Methods Engg. 15 (1992) 773-789.

Kesavan, S., (1989) Topics in Functional Analysis and Applications, John Wiley & Sons, New York.

Konderatev, V. A., Boundary Value problems for elliptic equation in domains with conical or angular points, Trudy Moscov. Mat. obsc 16 (1967) 209-292.

Lancaster, P., and Salkauskas, K., Surfaces generated by moving least squares

methods, Math. Comput. 37 (1981) 141-158.

Lapidus, L., and Pinder, G.F., (1982) Numerical Solution Of Partial Differential Equations In Science and Engineering, John Wiley & Sons, New York.

Linz, P., 1979 Theoretical Numerical Analysis, An introduction to advanced techniques, John Wiley & Sons, New York.

Liu, W. K., Jun, S., and Zhan, Y. F., Reproducing Kernel Particle Methods, Int. J. Numer. Methods Engrg. 20 (1995) 1081-1106.

Lu, Y.Y., Belytschko, T., and Gu, L., A new implementation of the element free Galerkin method, Comput. Methods Appl. Mech. Engrg. 113 (1994) 397-414.

Lucy, L.B., A numerical approach to the testing of the fission hypothesis, Astron. J. 82 (1977) 1013-1024.

McLain, D.H., Drawing contours from arbitrary data points, Comput. J. 17 (1974) 318-324.

Melenk, J. M., and Babuska, I., The partition of unity finite element method, Basic theory and application, Comput. Methods Appl. Mech. Engrg 139 (1996) 289-314.

Nayroles, B., Touzot, G., and Villon, P., Nuages De Points Et Approximation Diffuse. Seminaire D'analyse Convexe, Montpellier Expose' 16 (1991)

16-1 to 16-18.

Nayroles, B., Touzot, G., and Villon, P., Generalizing the finite element method: Diffuse approximation and diffuse elements, *Comput. Mech.* 10 (1992) 307-318.

Onate, E., Idelsohn, S., Zienkiewicz, O. C., Taylor, R. N., and Sacco, C., A finite point method for analysis of fluid mechanics problems. *Comput. Method Appl. Mech. Engrg.* 139 (1996) 315-346.

Petera, J., and Pittman, J.F.T., Isoparametric Hermite Elements, *Int J. Numer. Methods Engg.* 37 (1994) 3489-3519.

Renardy, M., and Rogers, R.C., (1993) *An Introduction to Partial differential equations*, Springer-Verlag, New York.

Rudin, W., (1964) *Principles of Mathematical Analysis*, 3rd ed. McGraw-Hill, New York.

Smith, R., Multi-mode models of flow and solute dispersion in shallow water, Part 1. General derivation, *J Fluid Mech.* 283 (1995) 231-248.

Smith, R., Multi-mode models of flow and solute dispersion in shallow water, Part 2. Logarithmic velocity profiles, *J. Fluid Mech.* 286 (1995) 277-290.

Smith, R., Multi-mode models of flow and solute dispersion in shallow water, Part 3. Horizontal dispersion tensor for velocity, *J. Fluid Mech.* 352 (1997)

331-340.

Stoer, J., and Bulirsch, R., (1980) Introduction to numerical analysis Springer-Verlag New York.

Thompson, J.T., and Warsi, Z.U.A., Boundary-Fitted Systems for Numerical Solution of Partial Differential Equations-A Review, J. Comput. Phys. 47 (1982) 1-108.

Zienkiewicz, O.C., and Taylor, R.L., (1994) The Finite Element Method, 4th ed. Vol 1, McGraw-Hill Book Company, London.

Appendix

Program Lists

APPLICATION OF DIFFUSE APPROXIMATION

DESIGNED AND WRITTEN BY:

M.R.MOKHTARZADEH

ON THE BASIS OF A RESEARCH

PROJECT SUGGESTED AND SUPERVISED BY:

DR.V.NASSEHI

JUNE 1998

VERSION 1

Program: adam

Application of the Diffuse Approximation Method

The running of this program depends on the following steps:

- i. Load Mathematica (version 3)
- ii. Enter adam
- iii. Choose an application by entering one of the following application codes:
 - bhb* = Solution of Biharmonic Equation Subject to Homogeneous Boundary Conditions
 - cfp* = Curve Fitting
 - evp* = Solution of Eigenvalue Problems
 - ode* = Solution of Fourth Order O.D.E.
 - phb* = Solution of Poisson Equation Subject to Homogeneous Boundary Conditions
 - phb3* = Solution of Three Dimensional Poisson Equation Subject to Homogeneous Boundary Conditions
 - pnh* = Solution of Poisson Equation Subject to Non-homogeneous Boundary Conditions
 - pmpb* = Plane Mapping Using Boundary Nodes
 - mpn* = Plane Mapping Using All Nodes
 - sfp* = Surface Fitting
 - slp* = Solution of Sturm-Liouville Equation
 - tap* = Approximation of Trace Operator

iv. Enter the required functions in each application

The following is the list of the available functions

1. *lib* = load library of the defined applications
2. *init* = define the problem and initialise problem parameters
(these parameters are given at the end of this section)
3. *das* = show the defined diffuse approximation space
4. *pdp* = show the defined problem
5. *obj* = show the defined object
6. *gsn* = show the source nodes (graphic)
7. *gsnn* = show the refined source nodes (graphic)
8. *gsnb* = show the nodes and the boundary of the source domain (graphic)
9. *gtm* = show the target nodes for mapping (graphic)
10. *gtmn* = show the refined target nodes for mapping (graphic)
11. *gwf* = show the selected weight functions (graphic)
12. *gda* = show the result of the diffuse approximation (graphic)
13. *eda* = show the error norms of the diffuse approximation
14. *dap* = run the loaded applications
15. *chk* = check the defined problems
16. *sys* = program algorithm information (Graphical representation of the program algorithm is given in section 3.7, and also in the program listing, which shows the number of options available within each application)
17. *txt* = description of the loaded application
18. *rpg* = random problem generator for testing

List of problem parameters required for the initialisation and running of the program

mode = Rational or Real

preci = Precision of the calculations

Domain description for 1-D problems

minx = e.g. 0

maxx = e.g. 1

dom = { *minx*, *maxx* }

Domain description for 2-D problems

minx1 =

minx2 =

maxx1 =

maxx2 =

dom = { {*minx1*, *maxx1*}, {*minx2*, *maxx2*} }

Domain description for 3-D problems

minx1 =

minx2 =

minx3 =

maxx1 =

maxx2 =

maxx3 =

dom = { {*minx1*, *maxx1*}, {*minx2*, *maxx2*}, {*minx3*, *maxx3*} }

In the mapping application the source domain is given as the problem domain. The target domain is defined as a unit square by default.

exp = function definitions (depends on the object of the loaded application)

tnn = total number of nodes

(sufficient for 1-D problems)

tnnx1 = total number of nodes in x_1 direction

tnnx2 = total number of nodes in x_2 direction

$tnn = (tnnx1) \times (tnnx2)$

(in 2-D problems)

$tnnx1 =$

$tnnx2 =$

$tnnx3 =$

$tnn = (tnnx1) \times (tnnx2) \times (tnnx3)$

(in 3-D problems)

dim = algebraic dimension of the selected base functions

(sufficient for 1-D problems)

$degx1$ = degree of the base functions w.r.t. x_1

$degx2$ = degree of the base functions w.r.t. x_2

$dim = (degx1+1) \times (degx2+1)$

(in 2-D problems)

$degx1 =$

$degx2 =$

$degx3 =$

$dim = (degx1+1) \times (degx2+1) \times (degx3+1)$

(in 3-D problems)

(base functions themselves are defined within the function 'lib')

$\$ pwf$ = array of parameters in the defined power weight functions generator

$\$ reff$ = refinement factor

(sufficient for 1-D problems)

(e.g. $\$ reff=2$ re-solve the problem using twice the number of nodes)

$\$ reffx1 =$

$\$ reffx2 =$

$\$ reff = reffx1, reffx2$

(in 2-D problems)

Similarly for 3-D problems

$$S_{\text{reff}} = \{ S_{\text{reffx1}}, S_{\text{reffx2}}, S_{\text{reffx3}} \}$$

■ Master Program List

Load Special Functions Created by Mathematica

```
<<showtime.m;  
<<gaussian.m;  
<<newtonco.m;  
<<NumericalMath`ListIntegrate`
```

Unix Commands

```
beep:=Print[FromCharacterCode[7]];  
af:=Print[FileNames[]];  
dd:=Print[Directory[]];  
del:=DeleteFile[FileNames["*%"]];  
delc:=DeleteFile["core"];  
lib:=(<<lib);  
swdh:=SetDirectory["/home"];  
swdc:=SetDirectory["/home/Dalek/cgmrn"];  
swda:=SetDirectory["/home/Dalek/cgmrn/adam"];  
swdal:=SetDirectory["/home/Dalek/cgmrn/all/alists"];  
swdgaw:=SetDirectory["/home/Dalek/cgmrn/adam/qaw"];  
swdc;
```

Main Program [adam][[List for Ph.D. Thesis]

```

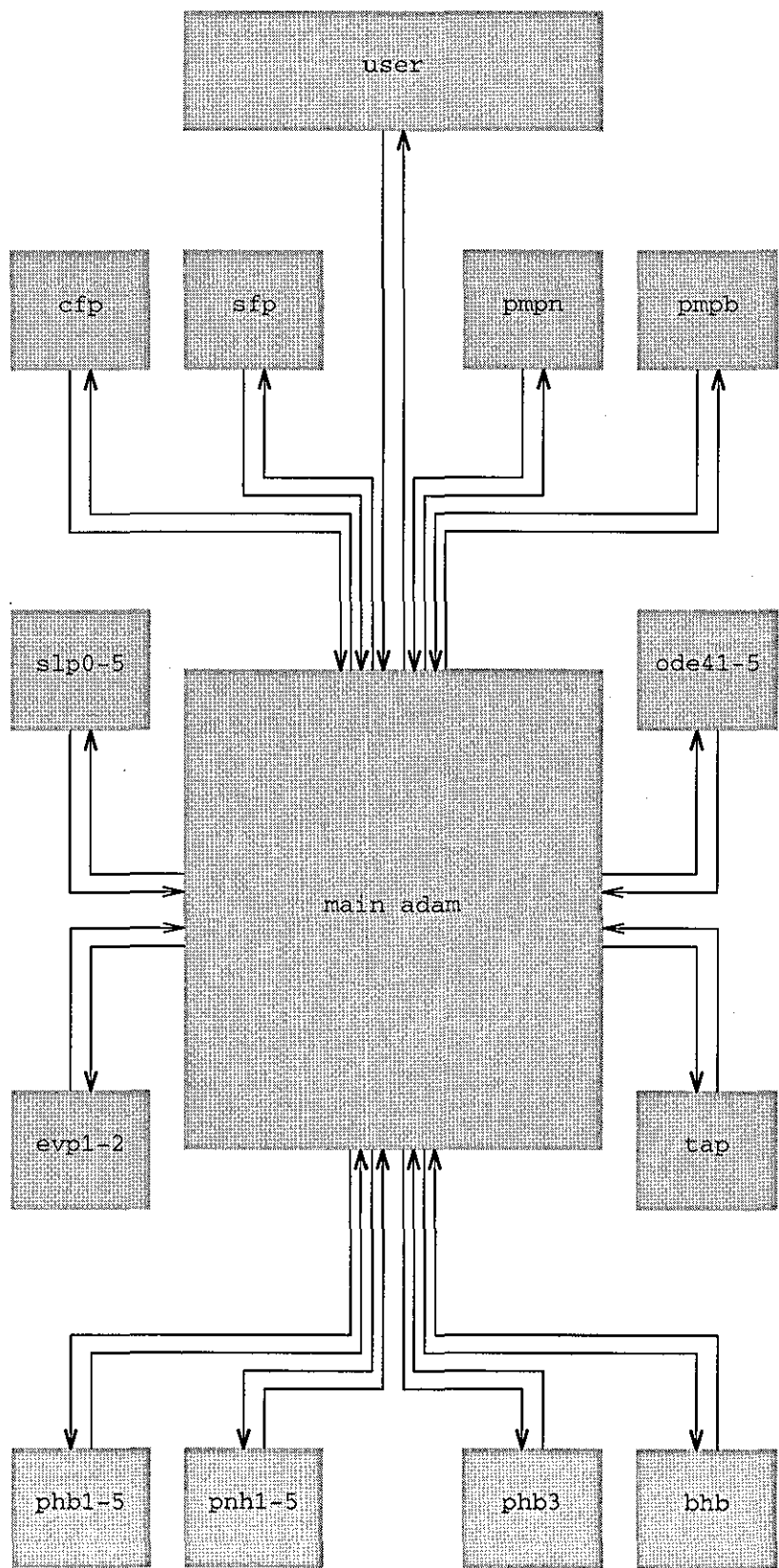
adam:=Module({},
swda;
swdcfp:=SetDirectory["/home/Dalek/cgmrn/adam/cfp"];
swdevp1:=SetDirectory["/home/Dalek/cgmrn/adam/evp1"];
swdevp2:=SetDirectory["/home/Dalek/cgmrn/adam/evp2"];
swdsfp:=SetDirectory["/home/Dalek/cgmrn/adam/sfp"];
swdtap:=SetDirectory["/home/Dalek/cgmrn/adam/tap"];
swdpdeb:=SetDirectory["/home/Dalek/cgmrn/adam/pdeb"];
swdpmpb:=SetDirectory["/home/Dalek/cgmrn/adam/pmpb"];
swdpmpn:=SetDirectory["/home/Dalek/cgmrn/adam/pmpn"];
swdpmpt1:=SetDirectory["/home/Dalek/cgmrn/adam/pmpt1"];
swdpmpt2:=SetDirectory["/home/Dalek/cgmrn/adam/pmpt2"];
swdpmpt3:=SetDirectory["/home/Dalek/cgmrn/adam/pmpt3"];
swdpmpt4:=SetDirectory["/home/Dalek/cgmrn/adam/pmpt4"];
swdslp0:=SetDirectory["/home/Dalek/cgmrn/adam/slp0"];
swdslp1:=SetDirectory["/home/Dalek/cgmrn/adam/slp1"];
swdslp2:=SetDirectory["/home/Dalek/cgmrn/adam/slp2"];
swdslp3:=SetDirectory["/home/Dalek/cgmrn/adam/slp3"];
swdslp4:=SetDirectory["/home/Dalek/cgmrn/adam/slp4"];
swdslp5:=SetDirectory["/home/Dalek/cgmrn/adam/slp5"];
swdslp6:=SetDirectory["/home/Dalek/cgmrn/adam/slp6"];
swdslp7:=SetDirectory["/home/Dalek/cgmrn/adam/slp7"];
swdslp8:=SetDirectory["/home/Dalek/cgmrn/adam/slp8"];
swdslp9:=SetDirectory["/home/Dalek/cgmrn/adam/slp9"];
swdode41:=SetDirectory["/home/Dalek/cgmrn/adam/ode41"];
swdode42:=SetDirectory["/home/Dalek/cgmrn/adam/ode42"];
swdode43:=SetDirectory["/home/Dalek/cgmrn/adam/ode43"];
swdode44:=SetDirectory["/home/Dalek/cgmrn/adam/ode44"];
swdode45:=SetDirectory["/home/Dalek/cgmrn/adam/ode45"];
swdphb1:=SetDirectory["/home/Dalek/cgmrn/adam/phb1"];
swdphb2:=SetDirectory["/home/Dalek/cgmrn/adam/phb2"];
swdphb3:=SetDirectory["/home/Dalek/cgmrn/adam/phb3"];
swdphb4:=SetDirectory["/home/Dalek/cgmrn/adam/phb4"];
swdphb5:=SetDirectory["/home/Dalek/cgmrn/adam/phb5"];
swdpnh:=SetDirectory["/home/Dalek/cgmrn/adam/pnh"];
swdpnh1:=SetDirectory["/home/Dalek/cgmrn/adam/pnh1"];
swdpnh2:=SetDirectory["/home/Dalek/cgmrn/adam/pnh2"];
swdpnh3:=SetDirectory["/home/Dalek/cgmrn/adam/pnh3"];
swdpnh4:=SetDirectory["/home/Dalek/cgmrn/adam/pnh4"];
swdpnh5:=SetDirectory["/home/Dalek/cgmrn/adam/pnh5"];
swdphb31:=SetDirectory["/home/Dalek/cgmrn/adam/phb31"];
swdphb32:=SetDirectory["/home/Dalek/cgmrn/adam/phb32"];
swdphb33:=SetDirectory["/home/Dalek/cgmrn/adam/phb33"];
swdphb34:=SetDirectory["/home/Dalek/cgmrn/adam/phb34"];
swdphb35:=SetDirectory["/home/Dalek/cgmrn/adam/phb35"];
swdbhb1:=SetDirectory["/home/Dalek/cgmrn/adam/bhb1"];
swdbhb2:=SetDirectory["/home/Dalek/cgmrn/adam/bhb2"];
swdbhb3:=SetDirectory["/home/Dalek/cgmrn/adam/bhb3"];
swdbhb4:=SetDirectory["/home/Dalek/cgmrn/adam/bhb4"];
swdbhb5:=SetDirectory["/home/Dalek/cgmrn/adam/bhb5"];
cfp:=Module({},swdcfp;capp;<<lib;init];
evp1:=Module({},swdevp1;capp;<<lib;init];
evp2:=Module({},swdevp2;capp;<<lib;init];
pde:=Module({},swdpde;capp;<<lib;init];
pmpb:=Module({},swdpmpb;capp;<<lib;init];
pmpn:=Module({},swdpmpn;capp;<<lib;init];
pmpt1:=Module({},swdpmpt1;capp;<<lib;init];
pmpt2:=Module({},swdpmpt2;capp;<<lib;init];
pmpt3:=Module({},swdpmpt3;capp;<<lib;init];
pmpt4:=Module({},swdpmpt4;capp;<<lib;init];
sfp:=Module({},swdsfp;capp;<<lib;init];
tap:=Module({},swdtap;capp;<<lib;init];
slp0:=Module({},swdslp0;capp;<<lib;init];
slp1:=Module({},swdslp1;capp;<<lib;init];
slp2:=Module({},swdslp2;capp;<<lib;init];
slp3:=Module({},swdslp3;capp;<<lib;init];
slp4:=Module({},swdslp4;capp;<<lib;init];
slp5:=Module({},swdslp5;capp;<<lib;init];
ode41:=Module({},swdode41;capp;<<lib;init];
ode42:=Module({},swdode42;capp;<<lib;init];
ode43:=Module({},swdode43;capp;<<lib;init];
ode44:=Module({},swdode44;capp;<<lib;init];
ode45:=Module({},swdode45;capp;<<lib;init];
phb1:=Module({},swdphb1;capp;<<lib;init];
phb2:=Module({},swdphb2;capp;<<lib;init];
phb3:=Module({},swdphb3;capp;<<lib;init];
phb4:=Module({},swdphb4;capp;<<lib;init];

```

```
phb5:=Module[{},swdphb5,capp;<<lib;init];
pnh:=Module[{},swdpnh,capp;<<lib;init];
pnh1:=Module[{},swdpnh1,capp;<<lib;init];
pnh2:=Module[{},swdpnh2,capp;<<lib;init];
pnh3:=Module[{},swdpnh3,capp;<<lib;init];
pnh4:=Module[{},swdpnh4,capp;<<lib;init];
pnh5:=Module[{},swdpnh5,capp;<<lib;init];
phb31:=Module[{},swdphb31,capp;<<lib;init];
phb32:=Module[{},swdphb32,capp;<<lib;init];
phb33:=Module[{},swdphb33,capp;<<lib;init];
phb34:=Module[{},swdphb34,capp;<<lib;init];
phb35:=Module[{},swdphb35,capp;<<lib;init];
bhb1:=Module[{},swdbhb1,capp;<<lib;init];
bhb2:=Module[{},swdbhb2,capp;<<lib;init];
bhb3:=Module[{},swdbhb3,capp;<<lib;init];
bhb4:=Module[{},swdbhb4,capp;<<lib;init];
bhb5:=Module[{},swdbhb5,capp;<<lib;init];
];
```

Optional Colours for Graphics

```
aquamarine=RGBColor[0.44, 0.86, 0.58];
black=RGBColor[0., 0., 0.];
blue=RGBColor[0., 0., 1.];
blueviolet=RGBColor[0.62, 0.37, 0.62];
brown=RGBColor[0.65, 0.16, 0.16];
cadetblue=RGBColor[0.37, 0.62, 0.62];
coral=RGBColor[1., 0.5, 0.];
cornflowerblue=RGBColor[0.26, 0.26, 0.44];
cyan=RGBColor[0., 1., 1.];
darkgreen=RGBColor[1-0.18, 1-0.31, 1-0.18];
darkolivegreen=RGBColor[0.31, 0.31, 0.18];
darkorchid=RGBColor[0.6, 0.2, 0.8];
darkslateblue=RGBColor[0.42, 0.14, 0.56];
darkslategray=RGBColor[0.18, 0.31, 0.31];
darkturquoise=RGBColor[0.44, 0.58, 0.86];
dimgray=RGBColor[0.33, 0.33, 0.33];
firebrick=RGBColor[0.56, 0.14, 0.14];
forestgreen=RGBColor[0.14, 0.56, 0.14];
gold=RGBColor[0.8, 0.5, 0.2];
goldenrod=RGBColor[0.86, 0.86, 0.44];
gray=RGBColor[0.75, 0.75, 0.75];
green=RGBColor[1-0., 1-1., 1-0.];
greenyellow=RGBColor[0.58, 0.86, 0.44];
indianred=RGBColor[0.31, 0.18, 0.18];
khaki=RGBColor[0.62, 0.62, 0.37];
lightblue=RGBColor[0.75, 0.85, 0.85];
lightgray=RGBColor[0.66, 0.66, 0.66];
lightsteelblue=RGBColor[0.56, 0.56, 0.74];
limegreen=RGBColor[0.2, 0.8, 0.2];
magenta=RGBColor[1., 0., 1.];
maroon=RGBColor[0.56, 0.14, 0.42];
midnightblue=RGBColor[0.18, 0.18, 0.31];
navy=RGBColor[0.14, 0.14, 0.56];
navyblue=RGBColor[0.14, 0.14, 0.56];
orange=RGBColor[0.8, 0.2, 0.2];
orangered=RGBColor[1., 0., 0.5];
orchid=RGBColor[0.86, 0.44, 0.86];
palegreen=RGBColor[0.56, 0.74, 0.56];
peach=RGBColor[0.44, 0.26, 0.26];
pink=RGBColor[0.74, 0.56, 0.56];
plum=RGBColor[0.92, 0.68, 0.92];
prussianblue=RGBColor[0.18, 0.18, 0.31];
purple=RGBColor[0.31, 0.18, 0.31];
red=RGBColor[1., 0., 0.];
salmon=RGBColor[0.44, 0.26, 0.26];
sandybrown=RGBColor[0.96, 0.64, 0.38];
seagreen=RGBColor[0.14, 0.56, 0.42];
sienna=RGBColor[0.56, 0.42, 0.14];
skyblue=RGBColor[0.2, 0.6, 0.8];
slateblue=RGBColor[0., 0.5, 1.];
springgreen=RGBColor[0., 1., 0.5];
steelblue=RGBColor[0.14, 0.42, 0.56];
thistle=RGBColor[0.85, 0.75, 0.85];
turquoise=RGBColor[0.68, 0.92, 0.92];
violet=RGBColor[0.31, 0.18, 0.31];
violetred=RGBColor[0.8, 0.2, 0.6];
wheat=RGBColor[0.85, 0.85, 0.75];
white=RGBColor[1., 1., 1.];
yellow=RGBColor[1., 1., 0.];
yellowbrown=RGBColor[0.86, 0.58, 0.44];
yellowgreen=RGBColor[0.6, 0.8, 0.2];
```

■ Solution of Biharmonic Equation with Homogeneous B.C.– Program List

```

bhb := Module[{}, lib :=
Module[{}, ri := Random[Integer, #1] & ; r := Random[Real, #1] & ;
rcol := RGBColor[r[{0, 1}], r[{0, 1}], r[{0, 1}]];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]];
condinp[cond_, def_, txt_] :=
Module[{inp}, inp = Input[txt];
If[cond[inp], Return[inp], Return[def], Return[def]]];
dap := Module[{}, init; swdqaw; data1 = Get[nquadx1];
data2 = Get[nquadx2]; swdbhb1;
qdata :=
Flatten[Table[{data1[[i]][[1]], data2[[j]][[1]],
data1[[i]][[2]]*data2[[j]][[2]]}, {i, 1, Length[data1]},
{j, 1, Length[data2]}], 1]; << "dapp";
udu =
(Append[res[#1[[1]], #1[[2]], anodes], #1[[3]]] & ) /@ qdata;
udu >> "udu"; bb[j_] :=
Sum[udu[[rr]][[8]]*udu[[rr]][[6]]*udu[[rr]][[6]][[j]] +
udu[[rr]][[8]]*udu[[rr]][[6]]*udu[[rr]][[7]][[j]],
udu[[rr]][[8]]*udu[[rr]][[7]]*udu[[rr]][[6]][[j]],
udu[[rr]][[8]]*udu[[rr]][[7]]*udu[[rr]][[7]][[j]],
{rr, 1, Length[udu]}];
CM = Table[bb[j], {j, 1, Length[inodes]}];
b[j_] :=
Sum[udu[[s]][[8]]*(exp /.
{x1 -> udu[[s]][[1]], x2 -> udu[[s]][[2]]})*
udu[[s]][[3]][[j]], {s, 1, Length[udu]}];
RHS = Table[b[j], {j, 1, Length[inodes]}];
sol = NN[LinearSolve[CM, RHS]]; sol >> "sol.out"; eda];
das := Module[{}, iel[5];
Print["THE CREATED DIFFUSE APPROXIMATION SPACE IS:"];
Print["MODE OF COMPUTATIONS :"]; Print["mode:= ", mode];
Print["PRECISION OF CALCULATIONS precision:= ", preci]; Print["TOTAL NUMBER OF NO
Print["DEGREE OF BASE FUNCTIONS WITH RESPECT TO x1 degx1:= ",
degx1]; Print["DEGREE OF BASE FUNCTIONS WITH RESPECT TO x2 degx2:= ", degx2]; Print["ALG
Print["THE PARAMETER [nquad] IS EITHER [ncaw3,ncaw4,...,ncaw101] OR "]; Print["
Print["nquad:= ", nquad]; iel[5];
dbp := Module[{}, base =
Union @@
Table[(1 - x1)^2*(1 - x2)^2*x1^(i + 1)*x2^(j + 1),
{i, 1, degx1}, {j, 1, degx2}];
dfp := Module[{}, uline1 = asol /. x2 -> 0;
uline2 = asol /. x1 -> 1; uline3 = asol /. x2 -> 1;
uline4 = asol /. x1 -> 0; dom = {{minx1, maxx1}, {minx2, maxx2}};
bcond =
{dom, {u0minx1, u1minx1, u0maxx1, u1maxx1, u0minx2, u1minx2,
u0maxx2, u1maxx2}}; difeq = {1, 2, 1, exp};
auval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) /@
aindex; buval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) /@

```

```

bindex; cuval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) / @
cindex; iuval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) / @
iindex]; dnp :=
Module[{}, aindex =
Sort[Union @@
Table[{k1, k2}, {k1, 0, tnnx1 - 1}, {k2, 0, tnnx2 - 1}]];
iindex =
Sort[Union @@
Table[{k1, k2}, {k1, 1, tnnx1 - 2}, {k2, 1, tnnx2 - 2}]];
cindex =
Sort[{{0, 0}, {tnnx1 - 1, 0}, {tnnx1 - 1, tnnx2 - 1},
{0, tnnx2 - 1}}];
bindex = Sort[Complement[aindex, Union[cindex, iindex]]];
sshx1 = NN[(maxx1 - minx1)/(tnnx1 - 1)];
sshx2 = NN[(maxx2 - minx2)/(tnnx2 - 1)];
anodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2} & ) / @ aindex;
inodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2} & ) / @ iindex;
cnodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2} & ) / @ cindex;
bnodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2} & ) / @ bindex;
ann = Length[anodes]; bnn = Length[bnodes]; cnn = Length[cnodes];
inn = Length[inodes]; iel[1];
init := Module[{}, capp; << "lib"; mode = "Real"; preci = 50;
NN := N[#1, preci] &; norm = norm2[#1] &;
norm2 := (#1.#1)^(1/2) &; minx1 = 0; maxx1 = 1; minx2 = 0;
maxx2 = 1; asol = x1^2*x2^2*(1 - x1)^2*(1 - x2)^2;
exp = D[asol, x1, x1, x1, x1] + 2*D[asol, x1, x1, x2, x2] +
D[asol, x2, x2, x2, x2]; u0minx1 = 0; u1minx1 = 0; u0maxx1 = 0;
u1maxx1 = 0; u0minx2 = 0; u1minx2 = 0; u0maxx2 = 0; u1maxx2 = 0;
uminx2 = 0; umaxx2 = 0; tnnx1 = 5; tnnx2 = 5; tnn = tnnx1*tnnx2;
degx1 = 2; degx2 = 2; dim = degx1*degx2; wfn = "pwf";
$pwf = {"k", "a"}, 4, 9/2; $reffx1 = 1; $reffx2 = 1;
nquadx1 = "gqaw20"; nquadx2 = "gqaw20"; nquad = {nquadx1, nquadx2};
dapp := Module[{}, res[z1_, z2_, nodes_] :=
Module[{}, iel[1]; Print["SOLVER PROGRAM FOR BIHARMONIC EQ WITH HBC IS RUNNING NOW"];
Sum[(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})\
, {ii, 1, Length[nodes]}]; cm :=
Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; det = Det[cm];
Print["det[cm] := ", N[det]]; icm = Inverse[cm];
beta[ss_, ii_] :=
(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
rhsm :=
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; rhsv := rhsm . auval; aa = icm . rhsv;
vuh = Simplify[(base /. {x1 -> z1, x2 -> z2}) . aa];
d1alpha[rr_, ss_] :=
Sum[(d1wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*

```

```

(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})\
, {ii, 1, Length[nodes]}; d1cm =
Table[Table[d1alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
d1beta[rr_, ii_] :=
(d1wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d1rhsm =
Table[Table[d1beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; d1rhsv = d1rhsm . auval;
d1base = D[base, x1]; rhsv11 = d1rhsv - d1cm . aa;
d1aa = icm . rhsv11; vd1uh =
Simplify[(d1base /. {x1 -> z1, x2 -> z2}) . aa +
(base /. {x1 -> z1, x2 -> z2}) . d1aa];
d2alpha[rr_, ss_] :=
Sum[(d2wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})\
, {ii, 1, Length[nodes]}; d2cm =
Table[Table[d2alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
d2beta[rr_, ii_] :=
(d2wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d2rhsm =
Table[Table[d2beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; d2rhsv = d2rhsm . auval;
d2base = D[base, x2]; rhsv21 = d2rhsv - d2cm . aa;
d2aa = icm . rhsv21; vd2uh =
Simplify[(d2base /. {x1 -> z1, x2 -> z2}) . aa +
(base /. {x1 -> z1, x2 -> z2}) . d2aa];
uh = (Coefficient[vuh, #1] & ) /@ iuval;
d1uh := (Coefficient[vd1uh, #1] & ) /@ iuval;
d2uh := (Coefficient[vd2uh, #1] & ) /@ iuval;
d11alpha[rr_, ss_] :=
Sum[(d11wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})\
, {ii, 1, Length[nodes]}; d11cm =
Table[Table[d11alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
d11beta[rr_, ii_] :=
(d11wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d11rhsm =
Table[Table[d11beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; d11rhsv = d11rhsm . uval;
d11base = D[d1base, x1];
rhsv2 = d11rhsv - 2*d1cm . d1aa - d11cm . aa; d11aa = icm . rhsv2;
vd11uh =
Simplify[(d11base /. {x1 -> z1, x2 -> z2}) . aa +
2*(d1base /. {x1 -> z1, x2 -> z2}) . d1aa +
(base /. {x1 -> z1, x2 -> z2}) . d11aa];
d22alpha[rr_, ss_] :=
Sum[(d22wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*

```

```

      (base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})\
, {ii, 1, Length[nodes]}; d22cm =
  Table[Table[d22alpha[rr, ss], {rr, 1, Length[base]}],
    {ss, 1, Length[base]}];
d22beta[rr_, ii_] :=
  (d22wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
  (base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d22rhsm =
  Table[Table[d22beta[rr, ii], {ii, 1, Length[nodes]}],
    {rr, 1, Length[base]}]; d22rhsv = ddrhsm . uval;
d22base = D[d2base, x2];
rhsv2 = d22rhsv - 2*d2cm . d2aa - d22cm . aa; d22aa = icm . rhsv2;
vd22uh =
  Simplify[(d22base /. {x1 -> z1, x2 -> z2}) . aa +
    2*(d2base /. {x1 -> z1, x2 -> z2}) . d2aa +
    (base /. {x1 -> z1, x2 -> z2}) . d22aa];
uh = (Coefficient[vuh, #1] &) /@ iuval;
d1uh := (Coefficient[vd1uh, #1] &) /@ iuval;
d2uh := (Coefficient[vd2uh, #1] &) /@ iuval;
d11uh = (Coefficient[vd11uh, #1] &) /@ iuval;
d22uh = (Coefficient[vd22uh, #1] &) /@ iuval;
Return[{z1, z2, uh, d1uh, d2uh, d11uh, d22uh}]]]

```

■ Curve Fitting–Program List

```

cfp := Module[{}, lib :=
Module[{}, usf[a_, x_] := Which[x < a, 0, x == a, 1/2, x > a, 1];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]];
condinp[cond_, def_, txt_] :=
Module[{inp}, inp = Input[txt];
If[cond[inp], Return[inp], Return[def], Return[def]]];
prompt :=
Module[{inputmain}, inputmain[countermain_] :=
inputmain[countermain] =
InputString[StringJoin["\n", "In<CFP>[",
ToString[countermain], "]="];
Module[{countermain = 1, conditionmain = "1"},
While[!conditionmain == "Xit",
conditionmain = inputmain[countermain];
ToExpression[conditionmain]; countermain = countermain + 1]]];
dap := Module[{}, << "dapp"; rda = (res1[#1, nodes] &) /@ nnodes;
rda >> "rda.out"; << "eda"];
das := Module[{}, iel[5];
Print["THE CREATED DIFFUSE APPROXIMATION SPACE IS:"];
Print["MODE OF COMPUTATIONS:"]; Print["mode:= ", mode];
Print["PRECISION OF CALCULATIONS precision:= ",
preci]; Print["TOTAL NUMBER OF NODES [tnn]"];
tnn := ", tnn]; Print["NODAL COORDINATES [dom] IS nodes:= "; Print[nodes];
Print["REFINEMENT FACTOR FOR THE GENERATION OF NEW NODES \
[nnodes]:"]; Print["$reff:= ", $reff];
Print["ALGEBRIAC DIMENSION OF BASE FUNCTIONS [dim] dim:= ",
dim]; Print["SELECTED BASE FUNCTIONS [base] ARE"];
Print["base:= ", base];
Print["SELECTED WEIGHT FUNCTION [wf] IS: ", wfn]; iel[5];
dbp := Module[{}, base = Table[x^i, {i, 0, dim - 1}];
pwf := Module[{}, k = $cwf[[2]]; a = $cwf[[3]]; dm = a*ssh;
wfn = "pwf"; cwexp := (1 - ((ni - x)/dm)^2)^k;
dcwexp := D[cwexp, x];
wfn[nodes_, i_, xx_] := cwexp /. {ni -> nodes[[i]], x -> xx};
dwfn[nodes_, i_, xx_] := dcwexp /. {ni -> nodes[[i]], x -> xx};
dcp := Module[{}, dexp = D[exp, x];
vf = (exp /. x -> #1 &) /@ nodes;
nvf = (exp /. x -> #1 &) /@ nnodes;
vdf = (dexp /. x -> #1 &) /@ nodes;
nvdf = (dexp /. x -> #1 &) /@ nnodes];
dnp := Module[{}, ssh = (maxx - minx)/(tnn - 1);
nodes = Table[minx + k*ssh, {k, 0, tnn - 1}];
nnodes = Table[minx + (k*ssh)/$reff, {k, 0, $reff*(tnn - 1)}];
pdp := Module[{}, iel[5];
Print["MINIMUM VALUE OF THE INDEPENDENT VARIABLE [x] minx:= ",
minx]; Print["MAXIMUM VALUE OF THE INDEPENDENT VARIABLE [x] \
maxx:= ", maxx]; Print["EXPLICIT FORM OF THE FUNCTION [f] IS:"];
Print[" f[x]:= ", exp];
Print["PROBLEM DOMAIN dom:= ", dom]; iel[5];
eda := Module[{}, daf = (#1[[2]] &) /@ << "rda.out";
dadf = (#1[[3]] &) /@ << "rda.out";
mindet = Min[Abs /@ (#1[[4]] &) /@ << "rda.out"];
error = Max[Abs /@ (daf - nvf)];
derror = Max[Abs /@ (dadf - nvdf)];
dl12e[dat_] :=
Module[{data1, int1, data2, int2},
data1 = ({#1[[1]], Abs[#1[[2]]]} &) /@ dat;
int1 = ListIntegrate[data1, 10];
data2 = ({#1[[1]], #1[[2]]^2} &) /@ dat;
int2 = ListIntegrate[data2, 10];
Print["DL1-error[f] := ", NN[int1];
Print["DL2-error[f] := ", NN[int2]]; iel[3];
Print["DU-Error[f] := ", error]; Print["DU-Error[df] := ", derror];

```

```

dl12e[<< "rda.out" - ({0, #1, 0, 0} & ) /@ nvf];
Print["min[det]:=", mindet]; iel[3];
gda := Module[{}, mplot[exp_, rda_] :=
Module[{data, ldata, pdata, txt1, txt2},
SetOptions[Graphics, AspectRatio -> 1, Axes -> True,
AxesLabel -> {"x", "f,daf"}, AxesOrigin -> Automatic,
AxesStyle -> Automatic, Background -> GrayLevel[1],
ColorOutput -> Automatic, DefaultColor -> GrayLevel[0],
Epilog -> {}, Frame -> False, FrameLabel -> None,
FrameStyle -> Automatic, FrameTicks -> Automatic,
GridLines -> None, ImageSize -> {400, 400},
PlotLabel -> " - f, . daf", PlotRange -> All,
PlotRegion -> Automatic, Prolog -> {}, RotateLabel -> True,
Ticks -> Automatic, DefaultFont -> $DefaultFont,
DisplayFunction -> $DisplayFunction,
FormatType -> $FormatType, TextStyle -> $TextStyle];
data :=
({#1, exp /. x -> #1} & ) /@
Table[minx + (k*(maxx - minx))/100, {k, 0, 100}];
ldata := Graphics[Line[data]];
pdata :=
Graphics[{{GrayLevel[0], AbsolutePointSize[3],
Point[#1]} & ) /@ ({#1[[1]], #1[[2]]} & ) /@ rda];
txt1 := Graphics[Text["- f", {maxx, maxx}]];
txt2 := Graphics[Text[" . daf", {maxx, maxx - 0.1}]];
Show[ldata]; Return[Show[ldata, pdata]];
mplot[exp, << "rda.out"]];
gwf := Module[{}, SetOptions[Plot, AspectRatio -> 1, Axes -> True,
AxesLabel -> {"x", "wf"}, AxesOrigin -> Automatic,
AxesStyle -> Automatic, Background -> GrayLevel[0],
ColorOutput -> Automatic, Compiled -> True,
DefaultColor -> rcol, Epilog -> {}, Frame -> False,
FrameLabel -> None, FrameStyle -> Automatic,
FrameTicks -> Automatic, GridLines -> None,
ImageSize -> Automatic, MaxBend -> 10., PlotDivision -> 30.,
PlotLabel -> "GRAPHICS OF [wf]", PlotPoints -> 50,
PlotRange -> All, PlotRegion -> Automatic,
PlotStyle -> Automatic, Prolog -> {}, RotateLabel -> True,
Ticks -> Automatic, DefaultFont -> $DefaultFont,
DisplayFunction -> $DisplayFunction, FormatType -> $FormatType,
TextStyle -> $TextStyle];
Plot[Evaluate[{wf[nodes, 1, xx], wf[nodes, 2, xx],
wf[nodes, 3, xx], wf[nodes, 4, xx]}, {xx, minx, maxx}]]; iel[1]
]; init := Module[{}, capp; << "lib"; preci = 20; NN := N[#1, preci] & ;
mode = Real; minx = 0; maxx = 1; dom = {minx, maxx};
exp = Sin[Pi*x^2]; tnn = 10; dim = 5;
$wff = ({"k", "a"}, 2, tnn + 1/2); $reff = 2; << "dnp"; << "dbp";
<< "dfp"; << "dcwp"; object =
{dom, nodes, base, wf, wfn, data, "rda.out", error, null1, null2,
null3, null4, null5, null6, exp}];
dapp := Module[{}, ClearAll[res1, alpha, cm, det, icm, beta, rhsm, rhsv,
vfh, aa, dalpha, dcm, dbeta, drhsm, drhsv, vdfh, daa];
res1[z_, nodes_] :=
Module[{alpha, beta, rhsm, rhsv, det, sol},
Print["OK.processor-1"];
alpha[rr_, ss_] :=
Sum[{wf[nodes, ii, xx] /. xx -> z}*
(base[[rr]] /. x -> nodes[[ii]])*
(base[[ss]] /. x -> nodes[[ii]]), {ii, 1, Length[nodes]}];
cm :=
Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; det = N @@ Det[cm];
Print["det[cm]:=", det]; icm = Inverse[cm];
beta[rr_, ii_] :=
(wf[nodes, ii, xx] /. xx -> z)*(base[[rr]] /. x -> nodes[[ii]]);
rhsm :=
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}];
rhsv := rhsm . NN[(exp /. x -> #1 & ) /@ nodes]; aa = icm . rhsv;
vfh = (base /. {x -> z}) . aa; Print["vfh:=", N[vfh]];
Print["nvf:=", Short[N[nvf]]]; iel[1]; Print["OK.processor-2"];
dalpha[rr_, ss_] :=
Sum[{dwf[nodes, ii, xx] /. {xx -> z}}*

```

```

(base[[rr]] /. {x -> nodes[[ii]]) *
(base[[ss]] /. {x -> nodes[[ii]]}), {ii, 1, Length[nodes]};
dcm =
Table[Table[dalpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
dbeta[rr_, ii_] :=
(dwff[nodes, ii, xx] /. {xx -> z}) *
(base[[rr]] /. {x -> nodes[[ii]]});
drhsm =
Table[Table[dbeta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}];
drhsv = drhsm . NN[(exp /. x -> #1 &) /@ nodes];
dbase = D[base, x]; daa = icm . (drhsv - dcm . aa);
vdfh = (dbase /. {x -> z}) . aa + (base /. {x -> z}) . daa;
Print["vdfh:= ", N[vdfh]]; Print["nvdf:= ", Short[N[nvdf]]];
Return[{z, vfh, vdfh, det}]]]

```


■ Evaluation of Eigenvalues—Program List

```

evp := Module[{}, lib :=
Module[{}, usf[a_, x_] := Which[x < a, 0, x == a, 1/2, x > a, 1];
refine[set_] :=
Module[{}, Return[Union[Table[(set[[i]] + set[[i + 1]])/2,
{i, 1, Length[set] - 1}], set]]];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]];
dap := Module[{}, init; swdqaw; data = Get[nquad]; swdevp1;
supp = (#1[[1]] & ) /@ data;
isupp = Complement[supp, {First[supp], Last[supp]}];
omega = (#1[[2]] & ) /@ data; << "dapp";
udu = (res2[#1, nodes] & ) /@ supp;
bb[j_] :=
Sum[(exp1 /. x -> supp[[r]])*omega[[r]]*udu[[r]][[2]]*
udu[[r]][[2]][[j]] +
(exp2 /. x -> supp[[r]])*omega[[r]]*udu[[r]][[1]]*
udu[[r]][[1]][[j]], {r, 1, Length[supp]}];
b[j_] :=
lambda*Sum[omega[[r]]*udu[[r]][[1]]*udu[[r]][[1]][[j]],
{r, 1, Length[supp]}]; BBBB = Table[bb[j], {j, 1, tnn - 2}];
BBB := BBBB /. lambda -> 0;
eval =
NN[Eigenvalues[NN[Inverse[BBBB - lambda*BBB /. lambda -> 1.] .
BBBB]]]; eval >> "eval"; eda];
eda := Module[{}, iel[5];
Print["COMPUTATIONAL ERRORS FOR SUCCESSIVE EIGENVALUES:"];
error =
Abs /@
(Sort[Table[NN[k^2*Pi^2], {k, 1, Length[<< "eval"]}]] -
Sort[<< "eval"]];
Do[Print["Error[lambda", ToString[i], "]:= ", error[[i]]];
iel[0], {i, 1, Length[error]}];
pwf := Module[{}, k = $pwf[[2]]; a = $pwf[[3]]; dm = a*ssh;
cwexp := (1 - ((ni - x)/dm)^2)^k;
dcwexp := D[(1 - ((ni - x)/dm)^2)^k, x];
wf[nodes_, i_, xx_] := cwexp /. {ni -> nodes[[i]], x -> xx};
dwf[nodes_, i_, xx_] := dcwexp /. {ni -> nodes[[i]], x -> xx};
pdp := Module[{}, init; iel[5];
Print["THE EIGENVALUE PROBLEM IS OF THE FORM:"]; iel[1];
Print["-d/dx((" , exp1, ")du/dx) + (" , exp2, ") u = " , exp3, ", "];
Print["u[" , minx, "] = " , uminx, ", u[" , maxx, "] = " , umaxx, ". "];
iel[5]; iel[1]; init :=
Module[{}, mode = "Real"; precision = 50; NN := N[#1, precision] & ;
minx = 0; maxx = 1; dom = {minx, maxx}; exp1 = 1; exp2 = lambda;
exp3 = 0; uminx = 0; umaxx = 0; tnn = 11; dim = 9; wfn = "pwf";
$pwf = {"nu1, nu2, mu1, mu2", 2, 21/2}; $reff = 2; nu1 = 2;
nquad = "gqaw10"; << "dfp"; << "dnp"; << "dbp"; Get[wfn];
object =
{difeq, bcond, trans, tdifeq, tbcond, nodes, base, wf, wfn, rdea,
fe1, fe2, fe3, fe4, fe5, fe6, fe7, fe8, fe9, asol}];
dapp := Module[{}, res1[z_, nodes_] :=
Module[{alpha, det, beta, rhsv1, a, da},
alpha[rr_, ss_] :=
Sum[(wf[nodes, ii, xx] /. xx -> z)*
(base[[rr]] /. x -> nodes[[iii]])*
(base[[ss]] /. x -> nodes[[iii]]), {ii, 1, Length[nodes]}];
cm = Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; det := Det[cm];
Print["det[cm]:= ", N[det]]; icm = Inverse[cm];
beta[rr_, ii_] :=
(wf[nodes, ii, xx] /. xx -> z)*(base[[rr]] /. x -> nodes[[iii]]);
rhsm =
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; rhsv = rhsm . uval;
vuh = Simplify[(base /. x -> z) . icm . rhsv];

```

```

Return[{cm, icm, rhsm, rhsv, uh}]; iel[1]];
res2[z_, nodes_] :=
Module[{dalpha, dbeta, rhsv1, dbase},
res1[z, nodes]; dalpha[rr_, ss_] :=
Sum[(dwf[nodes, ii, xx] /. xx -> z)*
(base[[rr]] /. x -> nodes[[ii]])*
(base[[ss]] /. x -> nodes[[ii]]), {ii, 1, Length[nodes]}];
dcm =
Table[Table[dalpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
dbeta[rr_, ii_] :=
(dwf[nodes, ii, xx] /. xx -> z)*(base[[rr]] /. x -> nodes[[ii]]);
drhsm =
Table[Table[dbeta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; drhsv = drhsm . uval; dbase = D[base, x];
vduh =
Simplify[(dbase /. x -> z) . icm . rhsv +
(base /. x -> z) . icm . (drhsv - dcm . icm . rhsv)];
uh := (Coefficient[vuh, #1] &) /@ iuval;
duh := (Coefficient[vduh, #1] &) /@ iuval; Return[{uh, duh}]]]

```

■ Solution of Fourth Order Ordinary Differential Equations— Program List

```

ode4 := Module[{}, init :=
Module[{}, capp; << "lib"; mode = "Real"; precision = 50;
NN := N[#1, precision] & ; minx = 0; maxx = 1; dom = {minx, maxx};
exp1 = 1; exp2 = 1; asol = x^6*(1 - x)^2;
exp3 = Simplify[D[exp1*D[asol, {x, 2}], {x, 2}] + exp2*asol];
u0minx = 0; u0maxx = 0; u1minx = 0; u1maxx = 0; tnn = 7; dim = 5;
wfn = "pwf"; $pwf = {"k,a", 4, 17/2}; $reff = 2; nu1 = 2;
nquad1 = "ncaw6"; nquadr = "ncaw7"; nquad = "gqaw10"; << "dnp";
<< "dbp"; << "dfp"; Get[wfn];
object =
{difeq, bcond, trans, tdifeq, tbcond, nodes, base, wf, wfn, rdea,
fe1, fe2, fe3, fe4, fe5, fe6, fe7, fe8, fe9, asol}};
lib := Module[{}, r := Random[]; ri[a_, b_] := Random[Integer, {a, b}];
rcol := RGBColor[r, r, r];
usf[a_, x_] := Which[x < a, 0, x == a, 1/2, x > a, 1];
refine[set_] :=
Module[{}, Return[Union[Table[{set[[i]] + set[[i + 1]]/2,
{i, 1, Length[set] - 1}], set]]];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]];
chk := Module[{}, text :=
(iel[2]; Print[" THE ANALYTIC SOLUTION OF THE PROBLEM IS \
NOT KNOWN OR "; Print[" THERE IS SOMETHING WRONG WITH THE PROBLEM."];
iel[2]); diff[p_, q_, r_, u_] :=
D[p*D[u, {x, 2}], {x, 2}] + q*u - r;
deq = Simplify[ExpandAll[diff[exp1, exp2, exp3, asol]]]; iel[2];
If[deq == 0, iel[1]; Print["OK. eq."], text, text];
If[{dom, {asol /. x -> minx, D[asol, x] /. x -> minx,
asol /. x -> maxx, D[asol, x] /. x -> maxx}} == bcond,
Print["OK. bc."], text; beep, text; beep]; iel[2];
das := Module[{}, iel[5];
Print["THE APPROXIMATION SCHEME IS BASED ON THE FULLY DISCRETISED \
TECHNIQUE"]; Print["USING THE FOLLOWING APPROXIMATION SPACE:"];
Print["mode:=", mode]; Print["precision:=", precision];
Print["tnn:=", tnn]; Print["nodes:=", N[Short[nodes]]];
Print["dim:=", dim]; Print["base:=", Short[base]];
Print["wf:=", wfn]; Print["REFINE FACTOR OF NODES [$reff] \
$reff:=", $reff]; Print["THE PARAMETER [nquad] IS EITHER \
[ncaw3,ncaw4,...,ncaw77] OR ";
Print[" [gqaw3,gqaw4,...,gqaw77] \
OR "; Print[" [rsaw3,rsaw4,...,rsaw101]. \
"]; Print["nquadl:=", nquadl]; Print["nquadr:=", nquadr];
Print["nquad:=", nquad]; iel[5];
base = Table[x^(i + 1)*(1 - x)^2, {i, 1, dim}];
dfp := Module[{}, dom = {minx, maxx}; difeq = {exp1, exp2, exp3};
bcond = {dom, {u0minx, u1minx, u0maxx, u1maxx}};
uval =
ToExpression /@ Table[StringJoin["u", ToString[i]], {i, 1, tnn}] \
; iuval = Complement[uval, {First[uval], Last[uval]}];
vval :=
ToExpression /@ Table[StringJoin["v", ToString[i]], {i, 1, tnn}]] \
; dnp := Module[{}, ssh = NN[(maxx - minx)/(tnn - 1)];
nodes = Table[minx + k*ssh, {k, 0, tnn - 1}];
bnodes = {First[nodes], Last[nodes]}; cnodes = bnodes;
inodes = Complement[nodes, bnodes]; nnodes = refine[nodes];
bnnodes = {First[nnodes], Last[nnodes]}; cnnodes = bnnodes;
innodes = Complement[nnodes, bnnodes];
eda := Module[{}, dasol = (asol /. x -> #1 &) /@ inodes;
error := Max[Abs /@ (<< "sol.out" - dasol)];
Print["THE ERROR OF THIS APPROXIMATION IS:"];
Print["DU-Error[f]:=", N[error]]];

```

```

gda := Module[{}, mplot[exp_, rda_] :=
Module[{data, ldata, pdata, txt1, txt2},
SetOptions[Graphics, AspectRatio -> 1, Axes -> True,
AxesLabel -> {"x", "u, dau"}, AxesOrigin -> Automatic,
AxesStyle -> Automatic, Background -> GrayLevel[0],
ColorOutput -> Automatic, DefaultColor -> rcol,
Epilog -> {}, Frame -> False, FrameLabel -> None,
FrameStyle -> Automatic, FrameTicks -> Automatic,
GridLines -> None, ImageSize -> {200, 200},
PlotLabel -> " - u, . dau", PlotRange -> All,
PlotRegion -> Automatic, Prolog -> {}, RotateLabel -> True,
Ticks -> Automatic, DefaultFont -> $DefaultFont,
DisplayFunction -> $DisplayFunction,
FormatType -> $FormatType, TextStyle -> $TextStyle];
data :=
({#1, exp /. x -> #1} &) /@
Table[minx + (k*(maxx - minx))/100, {k, 0, 100}];
ldata := Graphics[Line[data]];
pdata :=
Graphics[({yellow, AbsolutePointSize[2], Point[#1]} &) /@
Union[({#1[[1]], #1[[2]]} &) /@ rda, {{0, 0}, {1, 0}}]];
txt1 := Graphics[Text[" - u", {maxx, maxx}]];
txt2 := Graphics[Text[" . dau", {maxx, maxx - 0.1}]];
Return[Show[ldata, pdata]]; mplot[asol, << "rda.out"]];
gwf := Module[{}, SetOptions[Plot, AspectRatio -> GoldenRatio^(-1),
Axes -> Automatic, AxesLabel -> {"x", "wf"},
AxesOrigin -> Automatic, AxesStyle -> Automatic,
Background -> GrayLevel[0], ColorOutput -> Automatic,
Compiled -> True, DefaultColor -> rcol, Epilog -> {},
Frame -> False, FrameLabel -> None, FrameStyle -> Automatic,
FrameTicks -> Automatic, GridLines -> None, MaxBend -> 10.,
PlotDivision -> 30., PlotLabel -> None, PlotPoints -> 50,
PlotRange -> Automatic, PlotRegion -> Automatic,
PlotStyle -> Automatic, Prolog -> {}, RotateLabel -> True,
Ticks -> Automatic, DefaultFont -> $DefaultFont,
DisplayFunction -> $DisplayFunction];
Plot[Evaluate[Table[wf[nodes, k, xx], {k, 1, 3}]],
{xx, minx, maxx}];
pdp := Module[{}, iel[5];
Print["THE BOUNDARY VALUE PROBLEM IS OF THE FORM:"]; iel[1];
Print[d^2, "/", dx^2, "[('', exp1, ''), d^2, '/', dx^2, ''] + ('',
exp2, '') u = ''"]; Print[exp3]; iel[1];
Print["u['', 0, ''] = u['', 0, ''] = u['', 1, ''] = u['', 1, ''] = 0];
iel[5]; pwf :=
Module[{}, k = $pwf[[2]]; a = $pwf[[3]]; dm = a^ssh;
pwexp := (1 - ((ni - x)/dm)^2)^k;
dppwexp := D[(1 - ((ni - x)/dm)^2)^k, x];
ddppwexp := D[(1 - ((ni - x)/dm)^2)^k, {x, 2}];
wf[nodes_, i_, xx_] := pwexp /. {ni -> nodes[[i]], x -> xx};
dwf[nodes_, i_, xx_] := dppwexp /. {ni -> nodes[[i]], x -> xx};
ddwf[nodes_, i_, xx_] := ddppwexp /. {ni -> nodes[[i]], x -> xx};
rpg := Module[{}, capp; << "lib"; minx = 0; maxx = 1;
dom = {minx, maxx}; exp1 = 1 + x^2; exp2 = 2*x;
asol = Sum[x^ri[1, 5], {i, 0, ri[1, 5]}]*(1 - x);
exp2 = Simplify[-D[exp1*D[asol, x], x] + exp2*asol]; uminx = 0;
umaxx = 0; tnn = 7; dim = tnn - 2; $pwf = {"k", "a", 2, 11/2};
$rnf = 2; object =
{difeq, bcond, trans, tdifeq, tbcond, nodes, base, wf, wfn, rdea,
fe1, fe2, fe3, fe4, fe5, fe6, fe7, fe8, fe9, asol}; ];
dap := Module[{}, swdqaw; data = Get[nquad]; swdode41;
supp = (#1[[1]] &) /@ data;
isupp = Complement[supp, {First[supp], Last[supp]}];
omega = (#1[[2]] &) /@ data; << "dapp";
udu = (res[#1, nodes] &) /@ nodes; << "dapp";
bbu = (res[#1, nodes] &) /@ supp;
bb[j_] :=
Sum[(exp1 /. x -> supp[[r]])*omega[[r]]*udu[[r]][[2]]*
udu[[r]][[2]][[j]] +
(exp2 /. x -> supp[[r]])*omega[[r]]*udu[[r]][[1]]*
udu[[r]][[1]][[j]], {r, 1, Length[supp]}];
BBBB = Table[bb[j], {j, 1, tnn - 2}];
b[j_] :=
Sum[(exp3 /. x -> supp[[r]])*omega[[r]]*udu[[r]][[1]][[j]],

```

```

{r, 1, Length[supp]}; BB = Table[b[j], {j, 1, tnn - 2}];
sol = LinearSolve[BBBB, BB]; sol >> "sol.out";
MapThread[{#1, #2} &, {inodes, sol}] >> "rda.out";
Print["sol:=", sol]; << "eda";
dapp := Module[{}, ClearAll[res, res2, det];
res[z_, nodes_] :=
Module[{alpha, beta, dalpha, dbeta, ddalpha, ddbeta, rhsv, rhsv1,
rhsv2, aa, daa, ddaa, cm, dcm, ddc, rhsm, drhsm, ddrhsm, vuh,
vduh, vdduh}, alpha[rr_, ss_] :=
Sum[(wf[nodes, ii, xx] /. xx -> z)*
(base[[rr]] /. x -> nodes[[ii]])*
(base[[ss]] /. x -> nodes[[ii]]), {ii, 1, Length[nodes]}];
cm = Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; det := Det[cm];
Print["det[cm]:=", N[det]]; icm = Inverse[cm];
beta[rr_, ii_] :=
(wf[nodes, ii, xx] /. xx -> z)*(base[[rr]] /. x -> nodes[[ii]]);
rhsm =
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; rhsv = rhsm . uval; aa = icm . rhsv;
vuh = Simplify[(base /. x -> z) . aa];
dalpha[rr_, ss_] :=
Sum[(dwf[nodes, ii, xx] /. xx -> z)*
(base[[rr]] /. x -> nodes[[ii]])*
(base[[ss]] /. x -> nodes[[ii]]), {ii, 1, Length[nodes]}];
dcm =
Table[Table[dalpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
dbeta[rr_, ii_] :=
(dwf[nodes, ii, xx] /. xx -> z)*(base[[rr]] /. x -> nodes[[ii]]);
drhsm =
Table[Table[dbeta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; drhsv = drhsm . uval; dbase = D[base, x];
rhsv1 = drhsv - dcm . aa; daa = icm . rhsv1;
ddalpha[rr_, ss_] :=
Sum[(ddwf[nodes, ii, xx] /. xx -> z)*
(base[[rr]] /. x -> nodes[[ii]])*
(base[[ss]] /. x -> nodes[[ii]]), {ii, 1, Length[nodes]}];
ddcm =
Table[Table[ddalpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
ddbeta[rr_, ii_] :=
(ddwf[nodes, ii, xx] /. xx -> z)*(base[[rr]] /. x -> nodes[[ii]]);
ddrhsm =
Table[Table[ddbeta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; ddrhsv = ddrhsm . uval;
ddbbase = D[dbase, x]; rhsv2 = ddrhsv - 2*dcm . daa - ddc . aa;
ddaa = icm . rhsv2; vdduh =
Simplify[(ddbbase /. x -> z) . aa + 2*(dbase /. x -> z) . daa +
(base /. x -> z) . ddaa];
uh = (Coefficient[vuh, #1] &) /@ iuval;
dduh = (Coefficient[vdduh, #1] &) /@ iuval; Return[{uh, dduh}]]]

```

■ Solution of Poisson Equation with Homogeneous B.C. – Program List

```

phb := Module[{}, lib :=
Module[{}, ri := Random[Integer, #1] & ; r := Random[Real, #1] & ;
rcol := RGBColor[r[{0, 1}], r[{0, 1}], r[{0, 1}]];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]];
condinp[cond_, def_, txt_] :=
Module[{inp}, inp = Input[txt];
If[cond[inp], Return[inp], Return[def], Return[def]]];
dap := Module[{} < swdqaw; data1 = Get[nquadx1];
data2 = Get[nquadx2]; swdphb1;
qdata :=
Flatten[Table[{data1[[i]][[1]], data2[[j]][[1]],
data1[[i]][[2]]*data2[[j]][[2]]}, {i, 1, Length[data1]},
{j, 1, Length[data2]}], 1]; << "dapp";
udu =
(Append[res[#1[[1]], #1[[2]], anodes], #1[[3]]] & ) /@ qdata;
udu >> "udu.out"; bb[j_] :=
Sum[udu[[rr]][[6]]*udu[[rr]][[4]]*udu[[rr]][[4]][[j]]+
udu[[rr]][[6]]*udu[[rr]][[5]]*udu[[rr]][[5]][[j]],
{rr, 1, Length[udu]}];
CM = Table[bb[j], {j, 1, Length[inodes]}];
b[j_] :=
Sum[udu[[s]][[6]]*(exp /.
{x1 -> udu[[s]][[1]], x2 -> udu[[s]][[2]]})*
udu[[s]][[3]][[j]], {s, 1, Length[udu]}];
RHS = Table[b[j], {j, 1, Length[inodes]}];
sol = NN[LinearSolve[CM, RHS]]; sol >> "sol.out"; eda];
das := Module[{}, iel[5];
Print["THE CREATED DIFFUSE APPROXIMATION SPACE IS:"];
Print["MODE OF COMPUTATIONS:"]; Print["mode:= ", mode];
Print["PRECISION OF CALCULATIONS precision:= \
", preci]; Print["TOTAL NUMBER OF NODES ON x1 DIRECTION [tnnx1] tnnx1:= \
", tnnx1]; Print["TOTAL NUMBER OF NODES ON x2 DIRECTION [tnnx2] tnnx2:= \
", tnnx2]; Print["TOTAL NUMBER OF NODES IS tnn:= \
", tnn]; Print["REFINEMENT FACTOR FOR THE GENERATION OF NEW NODES \
[nnodes]"]; Print["$reff:={$reffx1,$reffx2} $reff:= ", $reff];
Print["DEGREE OF BASE FUNCTIONS WITH RESPECT TO x1 degx1:= ",
degx1]; Print["DEGREE OF BASE FUNCTIONS WITH RESPECT TO x2 \
degx2:= ", degx2]; Print["ALGEBRAIC DIMENSION OF BASE FUNCTIONS [base] IS \
dim:= ", dim]; Print["SELECTED WEIGHT FUNCTION [wf] IS: ", wfn];
Print["THE PARAMETER [nquad] IS EITHER [ncaw3,ncaw4,...,ncaw101]\
OR "; Print["
[gqaw3,gqaw4,...,gqaw101] OR \
"]; Print["
[rsaw3,rsaw4,...,rsaw101] OR ";
Print["nquad:=", nquad]; iel[5];
dbp := Module[{}, base =
Union @@
Table[(1 - x1)*(1 - x2)*x1^i*x2^j, {i, 1, degx1}, {j, 1, degx2}]]];
; dfp := Module[{}, uline1 = asol /. x2 -> 0; uline2 = asol /. x1 -> 1;
uline3 = asol /. x2 -> 1; uline4 = asol /. x1 -> 0;
becond =
{{{minx1, maxx1}, {minx2, maxx2}},
{uline1, uline2, uline3, uline4}}];
dom = {{minx1, maxx1}, {minx2, maxx2}}; difeq = {exp};
auval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) /@
aindex; buval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) /@
bindex; cuval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) /@
cindex; iuval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) /@
iindex]; dnp :=
Module[{}, aindex =

```

```

Sort[Union @@
  Table[{k1, k2}, {k1, 0, tnnx1 - 1}, {k2, 0, tnnx2 - 1}]];
iindex =
Sort[Union @@
  Table[{k1, k2}, {k1, 1, tnnx1 - 2}, {k2, 1, tnnx2 - 2}]];
cindex =
Sort[{{0, 0}, {tnnx1 - 1, 0}, {tnnx1 - 1, tnnx2 - 1},
  {0, tnnx2 - 1}}];
bindex = Sort[Complement[aindex, Union[cindex, iindex]]];
sshx1 = NN[(maxx1 - minx1)/(tnnx1 - 1)];
sshx2 = NN[(maxx2 - minx2)/(tnnx2 - 1)];
anodes =
  ((minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2) &) /@ aindex;
inodes =
  ((minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2) &) /@ iindex;
cnodes =
  ((minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2) &) /@ cindex;
bnodes =
  ((minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2) &) /@ bindex;
ann = Length[anodes]; bnn = Length[bnodes]; cnn = Length[cnodes];
inn = Length[inodes];
eda := Module[{}, dasol =
  (asol /. {x1 -> #1[[1]], x2 -> #1[[2]]} &) /@ inodes;
  error := Max[Abs /@ (<< "sol.out" - dasol)]; iel[2];
  Print["DU-Error[u] := ", N[error]]; iel[2];
gnp := Module[{}, SetOptions[Graphics, AspectRatio -> 1,
  Axes -> True, AxesLabel -> {"x1", "x2"},
  AxesOrigin -> Automatic, AxesStyle -> Automatic,
  Background -> GrayLevel[0], ColorOutput -> Automatic,
  DefaultColor -> rcol, Epilog -> {}, Frame -> False,
  FrameLabel -> None, FrameStyle -> Automatic,
  FrameTicks -> Automatic, GridLines -> None,
  ImageSize -> Automatic, PlotLabel -> "NODAL POINTS",
  PlotRange -> All, PlotRegion -> Automatic, Prolog -> {},
  RotateLabel -> True, Ticks -> True,
  DefaultFont -> $DefaultFont,
  DisplayFunction -> $DisplayFunction, FormatType -> $FormatType,
  TextStyle -> $TextStyle];
cpoly := {navyblue, Polygon[{{0, 0}, {1, 0}, {1, 1}, {0, 1}}]};
ccnod := ({yellow, Point[#1]} &) /@ cnodes;
cinod :=
  ((GrayLevel[1], AbsolutePointSize[0.1], Point[#1]) &) /@ inodes;
; cbnod := ({red, AbsolutePointSize[1], Point[#1]} &) /@ bnodes;
Show[Graphics[{cpoly, ccnod, cinod, cbnod}]];
pdp := Module[{}, iel[5];
  Print["THE POISSON EQUATION WITH HOMOGENEOUS DIRICHLET BOUNDARY \
CONDITION"]; Print["ON THE DOMAIN [dom] IS GIVEN AS:"]; iel[1];
  Print["-(", Subscripted[U[x1x1]], " + ", Subscripted[U[x2x2]],
  ") := ", Simplify[ExpandAll[exp]]]; iel[1];
  Print["IN dom := ", dom]; iel[1];
  Print[" u := ", uline1, " ON L1"];
  Print[" u := ", uline2, " ON L2"];
  Print[" u := ", uline3, " ON L3"];
  Print[" u := ", uline4, " ON L4"]; iel[2];
pwf := Module[{}, k = $pwf[[2]]; a = $pwf[[3]];
ssh = norm[{sshx1, sshx2}]; dm = a*ssh;
pwexp := (1 - (norm[{ni1, ni2} - {x1, x2}]/dm)^2)^k;
d1pwexp := D[pwexp, x1]; d2pwexp := D[pwexp, x2];
wf[nodes_, i_, xx1_, xx2_] :=
  pwexp /.
  {ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]], x1 -> xx1,
  x2 -> xx2}; d1wf[nodes_, i_, xx1_, xx2_] :=
  d1pwexp /.
  {ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]], x1 -> xx1,
  x2 -> xx2}; d2wf[nodes_, i_, xx1_, xx2_] :=
  d2pwexp /.
  {ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]], x1 -> xx1,
  x2 -> xx2}]; init :=
Module[{}, capp; << "lib"; mode = "Real"; preci = 10;
NN := N[#1, preci] &; norm = norm2[#1] &;
norm2 := (#1 . #1)^(1/2) &; minx1 = 0; maxx1 = 1; minx2 = 0;
maxx2 = 1; asol = x1*x2*(1 - x1)*(1 - x2);
exp = -D[asol, x1, x1] - D[asol, x2, x2]; uminx1 = 0; umaxx1 = 0;

```

```

uminx2 = 0; umaxx2 = 0; tnnx1 = 4; tnnx2 = 4; tnn = tnnx1*tnnx2;
degx1 = 2; degx2 = 2; dim = degx1*degx2;
wfn = "pwf"*$pwf = {"k", "a"}, 2, 9/2; $reffx1 = 1; $reffx2 = 1;
nquadx1 = "gqaw4"; nquadx2 = "gqaw4"; nquad = {nquadx1, nquadx2};
<< "dnp"; << "dfp"; << "dbp"; << "pwf";
object =
{nodes, cnodes, bnodes, inodes, base, wf, wfn, rdea, error, null1,
null2, null3, null4, null5, asol};
dapp := Module[{}, ClearAll[res, alpha, cm, det, icm, beta, rhsm, rhsv,
vfh, aa, d1alpha, d1cm, d1beta, d1rhsm, d1rhsv, vd1fh, d1aa,
d2alpha, d2cm, d2beta, d2rhsm, d2rhsv, vd2fh, d2aa];
res[z1_, z2_, nodes_] :=
Module[{}, iel[1]; Print["SOLVER PROGRAM FOR POISSON EQ WITH HBC IS \
RUNNING"]; alpha[rr_, ss_] :=
Sum[(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
{ii, 1, Length[nodes]}];
cm :=
Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; det = Det[cm];
Print["det[cm] := ", N[det]]; icm = Inverse[cm];
beta[ss_, ii_] :=
(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
rhsm :=
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; rhsv := rhsm . auval; aa = icm . rhsv;
vuh = Simplify[(base /. {x1 -> z1, x2 -> z2}) . aa];
d1alpha[rr_, ss_] :=
Sum[(d1wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
{ii, 1, Length[nodes]}];
d1cm =
Table[Table[d1alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
d1beta[rr_, ii_] :=
(d1wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d1rhsm =
Table[Table[d1beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; d1rhsv = d1rhsm . auval;
d1base = D[base, x1]; rhsv11 = d1rhsv - d1cm . aa;
d1aa = icm . rhsv11; vd1uh =
Simplify[(d1base /. {x1 -> z1, x2 -> z2}) . aa +
(base /. {x1 -> z1, x2 -> z2}) . d1aa];
d2alpha[rr_, ss_] :=
Sum[(d2wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
{ii, 1, Length[nodes]}];
d2cm =
Table[Table[d2alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
d2beta[rr_, ii_] :=
(d2wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d2rhsm =
Table[Table[d2beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; d2rhsv = d2rhsm . auval;
d2base = D[base, x2]; rhsv21 = d2rhsv - d2cm . aa;
d2aa = icm . rhsv21; vd2uh =
Simplify[(d2base /. {x1 -> z1, x2 -> z2}) . aa +
(base /. {x1 -> z1, x2 -> z2}) . d2aa];
uh = (Coefficient[vuh, #1] & ) /@ iuval;
d1uh := (Coefficient[vd1uh, #1] & ) /@ iuval;

```

```
d2uh := (Coefficient[vd2uh, #1] &) /@ iuval;  
Return[{z1, z2, uh, d1uh, d2uh}]]]
```

■ Solution of Three Dimensional Poisson Equation with Homogeneous B.C. – Program List

```

phb3 := Module[{}, lib :=
Module[{}, ri := Random[Integer, #1] & ; r := Random[Real, #1] & ;
rcol := RGBColor[r[{0, 1}], r[{0, 1}], r[{0, 1}]];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]];
dap := Module[{}, swdqwaw; data1 = Get[nquadx1];
data2 = Get[nquadx2]; data3 = Get[nquadx3]; swdphb31;
qdata := Flatten[Flatten[
Table[{data1[[i]][[1]], data2[[j]][[1]], data3[[k]][[1]],
data1[[i]][[2]]*data2[[j]][[2]]*data3[[k]][[2]]},
{i, 1, Length[data1]}, {j, 1, Length[data2]},
{k, 1, Length[data3]}], 1, 1]; << "dapp";
udu =
(Append[res[#1[[1]], #1[[2]], #1[[3]], anodes], #1[[4]]] & ) /@
qdata; udu >> "udu.out";
bb[j_] :=
Sum[udu[[rr]][[8]]*udu[[rr]][[5]]*udu[[rr]][[5]][[j]]+
udu[[rr]][[8]]*udu[[rr]][[6]]*udu[[rr]][[6]][[j]]+
udu[[rr]][[8]]*udu[[rr]][[7]]*udu[[rr]][[7]][[j]],
{rr, 1, Length[udu]}];
CM = Table[bb[j], {j, 1, Length[inodes]}];
b[j_] :=
Sum[udu[[s]][[8]]*(exp /.
{x1 -> udu[[s]][[1]], x2 -> udu[[s]][[2]],
x3 -> udu[[s]][[3]]})*udu[[s]][[4]][[j]],
{s, 1, Length[udu]}];
RHS = Table[b[j], {j, 1, Length[inodes]}];
sol = NN[LinearSolve[CM, RHS]]; sol >> "sol.out"; eda];
das := Module[{}, iel[5];
Print["THE CREATED DIFFUSE APPROXIMATION SPACE IS:"];
Print["MODE OF COMPUTATIONS:"]; Print["mode:= ", mode];
Print["PRECISION OF CALCULATIONS precision:= \
", prec]; Print["TOTAL NUMBER OF NODES IN x1 DIRECTION {tnnx1} tnnx1:= \
", tnnx1]; Print["TOTAL NUMBER OF NODES IN x2 DIRECTION {tnnx2} tnnx2:= \
", tnnx2]; Print["TOTAL NUMBER OF NODES IN x3 DIRECTION {tnnx3} tnnx3:= \
", tnnx3]; Print["TOTAL NUMBER OF NODES tnn:= \
", tnn]; Print["REFINEMENT FACTOR FOR THE GENERATION OF NEW NODES \
[nnodes]"]; Print["$reff:={ $reffx1, $reffx2 } $reff:= ", $reff];
Print["DEGREE OF BASE FUNCTIONS WITH RESPECT TO x1 degx1:= ",
degx1]; Print["DEGREE OF BASE FUNCTIONS WITH RESPECT TO x2 \
degx2:= ", degx2]; Print["DEGREE OF BASE FUNCTIONS WITH RESPECT TO x3 \
degx3:= ", degx3]; Print["ALGEBRAIC DIMENSION OF BASE FUNCTIONS [base] IS \
dim:= ", dim]; Print["SELECTED WEIGHT FUNCTION [wf] IS: ", wfn];
Print["THE PARAMETER [nquad] IS EITHER [ncaw3,ncaw4,...,ncaw91] \
OR "; Print[" [ggaw3,ggaw4,...,ggaw91] OR \
"; Print[" [rsaw3,rsaw4,...,rsaw91] OR ";
Print["nquad:= ", nquad]; iel[2]];
dbp := Module[{}, base =
Union @@
Union @@
Table[(1 - x1)*(1 - x2)*(1 - x3)*x1^i*x2^j*x3^k, {i, 1, degx1},
{j, 1, degx2}, {k, 1, degx3}];
dfp := Module[{}, bcond =
{{{minx1, maxx1}, {minx2, maxx2}, {minx3, maxx3}},
{0, 0, 0, 0, 0}};
dom = {{minx1, maxx1}, {minx2, maxx2}, {minx3, maxx3}};
difeq = {exp11, exp12, exp22, exp0};
bcond = {dom, {uminx1, umaxx1, uminx2, umaxx2, uminx3, umaxx3}};
auval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) /@
aindex; buval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] & ) /@
bindex; cuval =

```

```

(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] &) /@
cindex; iuval =
(StringJoin["u", ToString[#1[[1]]], ToString[#1[[2]]]] &) /@
iindex; dnp :=
Module[{}, aindex =
Sort[Union @@
Union @@
Table[{k1, k2, k3}, {k1, 0, tnnx1 - 1}, {k2, 0, tnnx2 - 1},
{k3, 0, tnnx3 - 1}]];
iindex =
Sort[Union @@
Union @@
Table[{k1, k2, k3}, {k1, 1, tnnx1 - 2}, {k2, 1, tnnx2 - 2},
{k3, 1, tnnx3 - 2}]];
cindex =
Sort[{(0, 0, 0), {tnnx1 - 1, 0, 0}, {tnnx1 - 1, tnnx2 - 1, 0},
{0, tnnx2 - 1, 0}, {0, 0, tnnx1 - 1},
{tnnx1 - 1, 0, tnnx1 - 1}, {tnnx1 - 1, tnnx2 - 1, tnnx1 - 1},
{0, tnnx2 - 1, tnnx1 - 1}}];
bindex = Sort[Complement[aindex, Union[cindex, iindex]]];
sshx1 = NN[(maxx1 - minx1)/(tnnx1 - 1)];
sshx2 = NN[(maxx2 - minx2)/(tnnx2 - 1)];
sshx3 = NN[(maxx3 - minx3)/(tnnx3 - 1)];
anodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2,
minx3 + #1[[3]]*sshx3} &) /@ aindex;
bnodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2,
minx3 + #1[[3]]*sshx3} &) /@ bindex;
cnodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2,
minx3 + #1[[3]]*sshx3} &) /@ cindex;
inodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2,
minx3 + #1[[3]]*sshx3} &) /@ iindex; ann = Length[anodes];
bnn = Length[bnodes]; cnn = Length[cnodes]; inn = Length[inodes];
eda := Module[{}, sol = << "sol.out";
dasol =
(asol /. {x1 -> #1[[1]], x2 -> #1[[2]], x3 -> #1[[3]]} &) /@
inodes; error := Max[Abs /@ (<< "sol.out" - dasol)]; iel[2];
Print["DU-Error[u]=", N[error]]; iel[2];
pwf := Module[{}, k = $pwf[[2]]; a = $pwf[[3]];
ssh = norm[{sshx1, sshx2, sshx3}]; dm = a*ssh;
pwexp := (1 - (norm2[{ni1, ni2, ni3} - {x1, x2, x3}]/dm)^2)^k;
d1pwexp := D[pwexp, x1]; d2pwexp := D[pwexp, x2];
d3pwexp := D[pwexp, x3];
wff[nodes_, i_, xx1_, xx2_, xx3_] :=
pwexp /.
{ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]],
ni3 -> nodes[[i]][[3]], x1 -> xx1, x2 -> xx2, x3 -> xx3};
d1wff[nodes_, i_, xx1_, xx2_, xx3_] :=
d1pwexp /.
{ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]],
ni3 -> nodes[[i]][[3]], x1 -> xx1, x2 -> xx2, x3 -> xx3};
d2wff[nodes_, i_, xx1_, xx2_, xx3_] :=
d2pwexp /.
{ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]],
ni3 -> nodes[[i]][[3]], x1 -> xx1, x2 -> xx2, x3 -> xx3};
d3wff[nodes_, i_, xx1_, xx2_, xx3_] :=
d3pwexp /.
{ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]],
ni3 -> nodes[[i]][[3]], x1 -> xx1, x2 -> xx2, x3 -> xx3}];
init := Module[{}, capp; << "lib"; mode = "Real"; preci = 50;
NN := N[#1, preci] &; norm = norm2[#1] &;
norm2 := (#1 . #1)^(1/2) &; minx1 = 0; maxx1 = 1; minx2 = 0;
maxx2 = 1; minx3 = 0; maxx3 = 1;
asol = x1*x2*x3*(1 - x1)*(1 - x2)*(1 - x3);
exp = -D[asol, x1, x1] - D[asol, x2, x2] - D[asol, x3, x3];
uminx1 = 0; umaxx1 = 0; uminx2 = 0; umaxx2 = 0; uminx3 = 0;
umaxx3 = 0; tnnx1 = 4; tnnx2 = 4; tnnx3 = 4; tnn = tnnx1*tnnx2*tnnx3;
degx1 = 2; degx2 = 2; degx3 = 2; dim = degx1*degx2*degx3;
wfn = "pwf"; $pwf = {"k", "a"}, 2, 5/2; $reffx1 = 1; $reffx2 = 1;
$reffx3 = 1; nquadx1 = "gqaw6"; nquadx2 = "gqaw6"; nquadx3 = "gqaw6";

```

```

nquad = {nquadx1, nquadx2, nquadx3}; << "dnp"; << "dfp"; << "dbp";
<< "pwf"; object =
{nodes, cnodes, bnodes, inodes, base, wf, wfn, rdea, error, null1,
null2, null3, null4, null5, asol};
ClearAll[res, alpha, cm, det, icm, beta, rhsm, rhsv, vfh, aa,
d1alpha, d1cm, d1beta, d1rhsm, d1rhsv, vd1fh, d1aa, d2alpha, d2cm,
d2beta, d2rhsm, d2rhsv, vd2fh, d2aa];
dapp := Module[{}, res[z1_, z2_, z3_, nodes_] :=
Module[{}, iel[1]; alpha[rr_, ss_] :=
Sum[{wf[nodes, ii, xx1, xx2, xx3] /.
{xx1 -> z1, xx2 -> z2, xx3 -> z3}}*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
x3 -> nodes[[ii]][[3]]})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
x3 -> nodes[[ii]][[3]]}), {ii, 1, Length[nodes]}];
cm = Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; det := Det[cm];
Print["det[cm] := ", N[det]]; icm = Inverse[cm];
beta[ss_, ii_] :=
(wf[nodes, ii, xx1, xx2, xx3] /.
{xx1 -> z1, xx2 -> z2, xx3 -> z3})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
x3 -> nodes[[ii]][[3]]});
rhsm :=
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}; rhsv := rhsm . auval; aa = icm . rhsv;
vuh = Simplify[(base /. {x1 -> z1, x2 -> z2, x3 -> z3}) . aa];
uh = (Coefficient[vuh, #1] &) /@ iuval;
d1alpha[rr_, ss_] :=
Sum[{d1wf[nodes, ii, xx1, xx2, xx3] /.
{xx1 -> z1, xx2 -> z2, xx3 -> z3}}*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
x3 -> nodes[[ii]][[3]]})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
x3 -> nodes[[ii]][[3]]}), {ii, 1, Length[nodes]}];
d1cm =
Table[Table[d1alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
d1beta[rr_, ii_] :=
(d1wf[nodes, ii, xx1, xx2, xx3] /.
{xx1 -> z1, xx2 -> z2, xx3 -> z3})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
x3 -> nodes[[ii]][[3]]});
d1rhsm =
Table[Table[d1beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}; d1rhsv = d1rhsm . auval;
d1base = D[base, x1]; rhsv11 = d1rhsv - d1cm . aa;
d1aa = icm . rhsv11; vd1uh =
Simplify[(d1base /. {x1 -> z1, x2 -> z2, x3 -> z3}) . aa +
(base /. {x1 -> z1, x2 -> z2, x3 -> z3}) . d1aa];
d1uh := (Coefficient[vd1uh, #1] &) /@ iuval;
d2alpha[rr_, ss_] :=
Sum[{d2wf[nodes, ii, xx1, xx2, xx3] /.
{xx1 -> z1, xx2 -> z2, xx3 -> z3}}*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
x3 -> nodes[[ii]][[3]]})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
x3 -> nodes[[ii]][[3]]}), {ii, 1, Length[nodes]}];
d2cm =
Table[Table[d2alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
d2beta[rr_, ii_] :=
(d2wf[nodes, ii, xx1, xx2, xx3] /.
{xx1 -> z1, xx2 -> z2, xx3 -> z3})*
(base[[rr]] /.

```

```

{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
 x3 -> nodes[[ii]][[3]]};
d2rhsm =
Table[Table[d2beta[rr, ii], {ii, 1, Length[nodes]}],
 {rr, 1, Length[base]}]; d2rhsv = d2rhsm . auval;
d2base = D[base, x2]; rhsv21 = d2rhsv - d2cm . aa;
d2aa = icm . rhsv21; vd2uh =
Simplify[(d2base /. {x1 -> z1, x2 -> z2, x3 -> z3}) . aa +
 (base /. {x1 -> z1, x2 -> z2, x3 -> z3}) . d2aa];
d2uh := (Coefficient[vd2uh, #1] & ) /@ iuval;
d3alpha[rr_, ss_] :=
Sum[(d3wf[nodes, ii, xx1, xx2, xx3] /.
 {xx1 -> z1, xx2 -> z2, xx3 -> z3}) *
 (base[[rr]] /.
 {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
 x3 -> nodes[[ii]][[3]]}) *
 (base[[ss]] /.
 {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
 x3 -> nodes[[ii]][[3]]}), {ii, 1, Length[nodes]}];
d3cm =
Table[Table[d2alpha[rr, ss], {rr, 1, Length[base]}],
 {ss, 1, Length[base]}];
d3beta[rr_, ii_] :=
(d3wf[nodes, ii, xx1, xx2, xx3] /.
 {xx1 -> z1, xx2 -> z2, xx3 -> z3}) *
 (base[[rr]] /.
 {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]],
 x3 -> nodes[[ii]][[3]]});
d3rhsm =
Table[Table[d3beta[rr, ii], {ii, 1, Length[nodes]}],
 {rr, 1, Length[base]}]; d3rhsv = d3rhsm . auval;
d3base = D[base, x3]; rhsv31 = d3rhsv - d3cm . aa;
d3aa = icm . rhsv31; vd3uh =
Simplify[(d3base /. {x1 -> z1, x2 -> z2, x3 -> z3}) . aa +
 (base /. {x1 -> z1, x2 -> z2, x3 -> z3}) . d3aa];
d3uh := (Coefficient[vd3uh, #1] & ) /@ iuval;
Return[{z1, z2, z3, uh, d1uh, d2uh, d3uh}]]]

```

■ Plane Mapping Using Boundary Nodes– Program List

```

pmb := Module[{}, lib :=
Module[{}, ri := Random[Integer, #1] & ; r := Random[Real, #1] & ;
rcol := RGBColor[r[{0, 1}], r[{0, 1}], r[{0, 1}]];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]]; norm1 := Plus @@ Abs /@ #1 & ;
norm2 := NN[{#1 . #1}^(1/2)] & ; normi := Max[Abs /@ #1] & ;
condinp[cond_, def_, txt_] :=
Module[{inp}, inp = Input[txt];
If[cond[inp], Return[inp], Return[def], Return[def]]];
prompt :=
Module[{inputmain}, inputmain[countermain_] :=
inputmain[countermain] =
InputString[StringJoin["\n", "In<PMP>[",
ToString[countermain], "]="];
Module[{countermain = 1, conditionmain = "1"},
While[!conditionmain == "Xit",
conditionmain = inputmain[countermain];
ToExpression[conditionmain]; countermain = countermain + 1]]];
dsn := Module[{}, node55 = NN[mapf /@ node55];
cnodes55 = NN[mapf /@ cnodes55]; bnodes55 = NN[mapf /@ bnodes55];
inodes55 = NN[mapf /@ inodes55]; nnodes55 = NN[mapf /@ nnodes55];
cnnodes55 = NN[mapf /@ cnnodes55]; bnnodes55 = NN[mapf /@ bnnodes55];
innodes55 = NN[mapf /@ innodes55];
dtn := Module[{}, sshx1 = (maxx1 - minx1)/(tnnx1 - 1);
sshx2 = (maxx2 - minx2)/(tnnx2 - 1); ssh = normi[{sshx1, sshx2}];
cnodes55 =
{{minx1, minx2}, {maxx1, minx2}, {maxx1, maxx2}, {minx1, maxx2}}\
; node55 = Union @@
Table[{minx1 + k1*sshx1, minx2 + k2*sshx2},
{k1, 0, tnnx1 - 1}, {k2, 0, tnnx2 - 1}];
inodes55 =
Union @@
Table[{minx1 + k1*sshx1, minx2 + k2*sshx2},
{k1, 1, tnnx1 - 2}, {k2, 1, tnnx2 - 2}];
bnodes55 = Complement[node55, inodes55];
nnodes55 =
Union @@
Table[{minx1 + (k1*sshx1)/$refff1,
minx2 + (k2*sshx2)/$refff2}, {k1, 0, $refff1*(tnnx1 - 1)},
{k2, 0, $refff2*(tnnx2 - 1)}];
innodes55 =
Union @@
Table[{minx1 + (k1*sshx1)/$refff1,
minx2 + (k2*sshx2)/$refff2},
{k1, 1, $refff1*(tnnx1 - 1) - 1},
{k2, 1, $refff2*(tnnx2 - 1) - 1}];
bnnodes55 = Complement[nnodes55, innodes55];
dap := Module[{}, << "dap";
rda := (res[#1[[1]], #1[[2]], bnodes55, bnodes55] & ) /@ node55;
rda >> "rda.out"; << "eda"];
init := Module[{}, capp; << "lib"; preci = 50; NN := N[#1, preci] & ;
norm = norm2[#1] & ; minx1 = 0; maxx1 = 1; minx2 = 0; maxx2 = 1;
z = x1 + I*x2; iexp = Exp[z]; u1exp = Re[iexp]; u2exp = Im[iexp];
mapf = {u1exp / . {x1 -> #1[[1]], x2 -> #1[[2]]},
u2exp / . {x1 -> #1[[1]], x2 -> #1[[2]]}} & ; tnnx1 = 4; tnnx2 = 3;
tnn = tnnx1*tnnx2; degx1 = 1; degx2 = 1;
dim = (degx1 + 1)*(degx2 + 1); $cwf = {"k", "a", 2, tnn - 0.5};
$refff1 = 1; $refff2 = 1;
object =
{nodes55, cnodes55, bnodes55, inodes55, node55, cnodes55, bnodes55,
inodes55, base, wf, wfn, rdea, error, null1, null2, null3, null4,
null5, u1exp, u2exp};
dap := Module[{}, ClearAll[res, alpha, det, icm, beta, rhsm, rhsv1,
rhsv2, aa1, aa2, vulh, vu2h];
res[z1_, z2_, nodes_, node55_] :=

```

```

Module[{}, iel[1]; Print["OK. PROC_1"];
alpha[rr_, ss_] :=
Sum[(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[rr]) /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[ss]) /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
{ii, 1, Length[nodes]};
cm :=
Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; icm := Inverse[cm];
beta[ss_, ii_] :=
(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[ss]) /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
rhsm :=
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; rhsv1 := rhsm . (#1[[1]] &) /@ nodes;
rhsv2 := rhsm . (#1[[2]] &) /@ nodes; aa1 := icm . rhsv1;
aa2 := icm . rhsv2; vu1h := (base /. {x1 -> z1, x2 -> z2}) . aa1;
vu2h := (base /. {x1 -> z1, x2 -> z2}) . aa2;
Return[{z1, z2, vu1h, vu2h, det}]]]

```

■ Plane Mapping Using All Nodes– Program List

```

pmn := Module[{}, lib :=
Module[{}, r := Random[Real, #1] &;
rcol := RGBColor[r[{0, 1}], r[{0, 1}], r[{0, 1}]];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]]; norm1 := Plus @@ Abs /@ #1 &;
norm2 := NN[(#1.#1)^(1/2)] &; normi := Max[Abs /@ #1] &;
condinp[cond_, def_, txt_] :=
Module[{inp}, inp = Input[txt];
If[cond[inp], Return[inp], Return[def], Return[def]]];
prompt :=
Module[{inputmain}, inputmain[countermain_] :=
inputmain[countermain] =
InputString[StringJoin["\n", "In<PMP>[",
ToString[countermain], "]="];
Module[{countermain = 1, conditionmain = "1"},
While[!conditionmain == "Xit",
conditionmain = inputmain[countermain];
ToExpression[conditionmain]; countermain = countermain + 1]]];
dap := Module[{}, << "dapp";
rda := (res[#1[[1]], #1[[2]], nodels, nodelst] &) /@ nodels;
rda >> "rda.out"; << "eda";
das := Module[{}, iel[5];
Print["THE CREATED DIFFUSE APPROXIMATION SPACE IS:"];
Print["MODE OF COMPUTATIONS:"]; Print["mode:= ", mode];
Print["PRECISION OF CALCULATIONS precision:= ",
prec]; Print["TOTAL NUMBER OF NODES IN x1 DIRECTION [tnnx1] \
tnnx1:= ", tnnx1]; Print["TOTAL NUMBER OF NODES IN x2 DIRECTION [tnnx2] \
tnnx2:= ", tnnx2]; Print["TOTAL NUMBER OF NODES [tnn] tnn:= ", tnn];
Print["DEGREE OF THE BASE FUNCTIONS IN x1 [degx1] degx1:= \
", degx1]; Print["DEGREE OF THE BASE FUNCTIONS IN x2 [degx2] degx2:= \
", degx2]; Print["ALGEBRAIC DIMENSION OF THE BASE FUNCTIONS [dim] dim:= \
", dim]; Print["SELECTED WEIGHT FUNCTION [wf] IS: ", wfn]; iel[5];
dbp := Module[{}, base =
Union @@ Table[x1^i*x2^j, {i, 0, degx1}, {j, 0, degx2}]];
dfp := Module[{}, eqtb[t_] :=
Which[0 <= t <= 1, (1 - t)*cnodelst[[1]] + t*cnodelst[[2]],
1 <= t <= 2, (2 - t)*cnodelst[[2]] + (t - 1)*cnodelst[[3]],
2 <= t <= 3, (3 - t)*cnodelst[[3]] + (t - 2)*cnodelst[[4]],
3 <= t <= 4, (4 - t)*cnodelst[[4]] + (t - 3)*cnodelst[[1]]];
eqsb[t_] := mapf[eqtb[t]]; vu1 = (#1[[1]] &) /@ nodelst;
vu2 = (#1[[2]] &) /@ nodelst; nvu1 = (#1[[1]] &) /@ nodelst;
nvu2 = (#1[[2]] &) /@ nodelst; nodels = NN[mapf /@ nodelst];
cnodels = NN[mapf /@ cnodelst]; bnodels = NN[mapf /@ bnodelst];
inodels = NN[mapf /@ inodelst]; nnodels = NN[mapf /@ nnodelst];
cnodels = NN[mapf /@ cnodelst]; bnodels = NN[mapf /@ bnodelst];
inodels = NN[mapf /@ inodelst];
sshx1 = (maxx1 - minx1)/(tnnx1 - 1);
sshx2 = (maxx2 - minx2)/(tnnx2 - 1); ssh = normi[{sshx1, sshx2}];
cnodelst =
{{minx1, minx2}, {maxx1, minx2}, {maxx1, maxx2}, {minx1, maxx2}}\
; nodelst = Union @@
Table[{minx1 + k1*sshx1, minx2 + k2*sshx2,
{k1, 0, tnnx1 - 1}, {k2, 0, tnnx2 - 1}}];
inodelst =
Union @@
Table[{minx1 + k1*sshx1, minx2 + k2*sshx2,
{k1, 1, tnnx1 - 2}, {k2, 1, tnnx2 - 2}}];
bnodelst = Complement[nodelst, inodelst];
nnodelst =
Union @@
Table[{minx1 + (k1*sshx1)/$reffx1,
minx2 + (k2*sshx2)/$reffx2, {k1, 0, $reffx1*(tnnx1 - 1)},
{k2, 0, $reffx2*(tnnx2 - 1)}}];
inodelst =
Union @@

```



```

Table[{minx1 + (k1*sshx1)/$reffx1,
  minx2 + (k2*sshx2)/$reffx2,
  {k1, 1, $reffx1*(tnnx1 - 1) - 1},
  {k2, 1, $reffx2*(tnnx2 - 1) - 1}];
bnnodest = Complement[nnodest, innodest];
eda := Module[{}, dau1 = (#1[[3]] & ) /@ << "rda.out";
dau2 = (#1[[4]] & ) /@ << "rda.out";
nodesth = MapThread[{#1, #2} & , {dau1, dau2}];
error1 = Max[Abs /@ (dau1 - (#1[[1]] & ) /@ nnodest)];
error2 = Max[Abs /@ (dau2 - (#1[[2]] & ) /@ nnodest)];
mindet = Min[Abs /@ (#1[[5]] & ) /@ << "rda.out"]; iel[2];
Print["DU-error1:= ", error1]; Print["DU-error2:= ", error2];
Print["DU-error:= ", Max[{error1, error2}]];
Print["mindet:= ", mindet]; iel[2];
gwf := Module[{}, SetOptions[Plot3D, AmbientLight -> GrayLevel[0],
  AspectRatio -> Automatic, Axes -> True, AxesEdge -> Automatic,
  AxesLabel -> None, AxesStyle -> Automatic,
  Background -> GrayLevel[0], Boxed -> True,
  BoxRatios -> {1, 1, 0.4}, BoxStyle -> Automatic,
  ClipFill -> Automatic, ColorFunction -> Automatic,
  ColorOutput -> Automatic, Compiled -> True,
  DefaultColor -> Automatic, Epilog -> {}, FaceGrids -> None,
  HiddenSurface -> True, ImageSize -> Automatic, Lighting -> True,
  LightSources ->
  {{1., 0., 1.}, RGBColor[1/2, 0, 0]},
  {{1., 1., 1.}, RGBColor[0, 1, 0]},
  {{0., 1., 1.}, RGBColor[0, 0, 1]}], Mesh -> False,
  MeshStyle -> Automatic, Plot3Matrix -> Automatic,
  PlotLabel -> None, PlotPoints -> 30, PlotRange -> All,
  PlotRegion -> Automatic, Prolog -> {}, Shading -> True,
  SphericalRegion -> False, Ticks -> Automatic,
  ViewCenter -> Automatic,
  ViewPoint -> {1.3, -2.3999999999999999, 2.},
  ViewVertical -> {0., 0., 1.}, DefaultFont -> $DefaultFont,
  DisplayFunction -> $DisplayFunction, FormatType -> $FormatType,
  TextStyle -> $TextStyle]; ClearAll[trnn];
trnn = Table[ri[{1, Length[nodesth]}], {i, 1, 10}];
Plot3D[gwf[nodesth, trnn[[1]], xx1, xx2], {xx1, minx1, maxx1},
  {xx2, minx2, maxx2}];
Plot3D[gwf[nodesth, trnn[[2]], xx1, xx2], {xx1, minx1, maxx1},
  {xx2, minx2, maxx2}];
Plot3D[gwf[nodesth, trnn[[3]], xx1, xx2], {xx1, minx1, maxx1},
  {xx2, minx2, maxx2}];
Plot3D[gwf[nodesth, trnn[[4]], xx1, xx2], {xx1, minx1, maxx1},
  {xx2, minx2, maxx2}];
Plot3D[gwf[nodesth, trnn[[5]], xx1, xx2], {xx1, minx1, maxx1},
  {xx2, minx2, maxx2}];
init := Module[{}, preci = 30; NN := N[#1, preci] & ;
norm = norm2[#1] & ; minx1 = 0; maxx1 = 1; minx2 = 0; maxx2 = 1;
z = x1 + I*x2; iexp = (1 + I)*z; u1exp = Re[iexp]; u2exp = Im[iexp];
mapf = {u1exp / . {x1 -> #1[[1]], x2 -> #1[[2]]},
  u2exp / . {x1 -> #1[[1]], x2 -> #1[[2]]}} & ; tnnx1 = 4; tnnx2 = 4;
tnn = tnnx1*tnnx2; degx1 = 1; degx2 = 1;
dim = (degx1 + 1)*(degx2 + 1); $cwf = {"k", "a", 2, tnn - 0.5};
$reffx1 = 1; $reffx2 = 1; << "dtn"; << "dsn"; << "dfp"; << "dbp";
<< "dcwp"; object =
{nodesth, cnodesth, bnodesth, innodest, nnodest, cnodest, bnodest,
  innodest, base, wf, wfn, rdea, error, null1, null2, null3, null4,
  null5, u1exp, u2exp};
dapp := Module[{}, ClearAll[res, alpha, det, icm, beta, rhsm, rhsv1,
  rhsv2, aa1, aa2, vulh, vu2h];
res[z1_, z2_, nodes_, node_] :=
Module[{}, alpha[rr_, ss_] :=
  Sum[(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
    (base[[rr]] /.
      {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
    (base[[ss]] /.
      {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
    {ii, 1, Length[nodes]}];
cm :=
  Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
    {ss, 1, Length[base]}]; det := Det[cm]; Print["det[cm] := ", det];
icm := Inverse[cm]; beta[ss_, ii_] :=

```

```

(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
rhsm :=
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}; rhsv1 := rhsm . (#1[[1]] &) /@ nodet;
rhsv2 := rhsm . (#1[[2]] &) /@ nodet; aa1 := icm . rhsv1;
aa2 := icm . rhsv2; vulh := (base /. {x1 -> z1, x2 -> z2}) . aa1;
vu2h := (base /. {x1 -> z1, x2 -> z2}) . aa2; iel[2];
Return[{z1, z2, vulh, vu2h, det}]]]]

```

■ Surface Fitting–Program List

```

sfp := Module[{}, lib :=
Module[{}, po := Print[object];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]]; r := Random[];
ri := Random[Integer, #1] & ; rcol := RGBColor[r, r, r];
usf[a_, x_] := Which[x < a, 0, x == a, 1/2, x > a, 1];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]]; norm1 := NN @@ Plus @@ Abs @@ #1 & ;
norm2 := NN @@ ((#1 . #1)^(1/2)) & ;
normi := NN @@ Max[Abs @@ #1] & ;
condinp[cond_, def_, txt_] :=
Module[{inp}, inp = Input[txt];
If[cond[inp], Return[inp], Return[def], Return[def]]];
sshx1 = (maxx1 - minx1)/(tnnx1 - 1);
sshx2 = (maxx2 - minx2)/(tnnx2 - 1);
nodes = Union @@
Table[{minx1 + k1*sshx1, minx2 + k2*sshx2}, {k1, 0, tnnx1 - 1},
{k2, 0, tnnx2 - 1}];
nnodes =
Union @@
Table[{minx1 + (k1*sshx1)/$reffx1, minx2 + (k2*sshx2)/$reffx2},
{k1, 0, $reffx1*(tnnx1 - 1)}, {k2, 0, $reffx2*(tnnx2 - 1)}];
base = Union @@ Table[x1^i*x2^j, {i, 0, degx1}, {j, 0, degx2}];
d1exp = D[exp, x1]; d2exp = D[exp, x2];
f = exp /. {x1 -> #1, x2 -> #2} & ;
d1f = d1exp /. {x1 -> #1, x2 -> #2} & ;
d2f = d2exp /. {x1 -> #1, x2 -> #2} & ; vf = (f @@ #1 & ) /@ nodes;
nvf = (f @@ #1 & ) /@ nnodes; vd1f = (d1f @@ #1 & ) /@ nodes;
vd2f = (d2f @@ #1 & ) /@ nodes; nvd1f = (d1f @@ #1 & ) /@ nnodes;
nvd2f = (d2f @@ #1 & ) /@ nnodes; k = $cwf[[2]]; a = $cwf[[3]];
ssh = norm[{sshx1, sshx2}]; dm = a*ssh; wfn = "pwf";
cwexp := (1 - (norm[{ni1, ni2} - {x1, x2}]/dm)^2)^k;
d1cwexp := D[cwexp, x1]; d2cwexp := D[cwexp, x2];
wf[nodes_, i_, xx1_, xx2_] :=
cwexp /.
{ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]], x1 -> xx1,
x2 -> xx2}; d1wf[nodes_, i_, xx1_, xx2_] :=
d1cwexp /.
{ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]], x1 -> xx1,
x2 -> xx2}; d2wf[nodes_, i_, xx1_, xx2_] :=
d2cwexp /.
{ni1 -> nodes[[i]][[1]], ni2 -> nodes[[i]][[2]], x1 -> xx1,
x2 -> xx2}; eda :=
Module[{}, daf = #1[[3]] & ) /@ << "rda.out";
dad1f = #1[[4]] & ) /@ << "rda.out";
dad2f = #1[[5]] & ) /@ << "rda.out";
mindet = Min[Abs @@ (#1[[6]] & ) /@ << "rda.out";
error = Max[Abs @@ (daf - nvf)];
d1error = Max[Abs @@ (dad1f - nvd1f)];
d2error = Max[Abs @@ (dad2f - nvd2f)]; iel[2];
Print["DU-error:=", error]; Print["DU-d1error:=", d1error];
Print["DU-d2error:=", d2error]; Print["mindet:=", mindet]; iel[2]]];
; init := Module[{}, capp; preci = 10; NN := N[#1, preci] & ; mode = Real;
norm = norm2[#1] & ; minx1 = 0; maxx1 = 1; minx2 = 0; maxx2 = 1;
dom = {{minx1, maxx1}, {minx2, maxx2}}; exp = x1^2 + x2^2 + x1*x2;
tnnx1 = 3; tnnx2 = 4; tnn = tnnx1*tnnx2; degx1 = 1; degx2 = 1;
dim = (degx1 + 1)*(degx2 + 1); $cwf = {"k", "a"}, 2, 1/2;
$reffx1 = 2; $reffx2 = 2; $reff = {$reffx1, $reffx2};
object =
{dom, nodes, base, wf, wfn, vf, "rda.out", error, null1, null2,
null3, null4, null5, null6, exp};
dapp := Module[{}, ClearAll[res, alpha, cm, det, icm, beta, rhsm, rhsv,
vfh, aa, d1alpha, d1cm, d1beta, d1rhsm, d1rhsv, vd1fh, d1aa,

```

```

d2alpha, d2cm, d2beta, d2rhsm, d2rhsv, vd2fh, d2aa];
res[z1_, z2_, nodes_] :=
Module[{}, iel[1]; Print["OK.processor-1"];
alpha[rr_, ss_] :=
Sum[(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
{ii, 1, Length[nodes]}];
cm :=
Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; det = N @@ Det[cm];
Print["det[cm] := ", det]; icm = Inverse[cm];
beta[ss_, ii_] :=
(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
rhsm :=
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; rhsv := rhsm . (f @@ #1 &) /@ nodes;
aa = icm . rhsv; vfh = (base /. {x1 -> z1, x2 -> z2}) . aa;
Print["vfh := ", vfh]; Print["nvf := ", Short[N[nvf]]]; iel[1];
Print["OK.processor-2"];
d1alpha[rr_, ss_] :=
Sum[(d1wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
{ii, 1, Length[nodes]}];
d1cm =
Table[Table[d1alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
d1beta[rr_, ii_] :=
(d1wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d1rhsm =
Table[Table[d1beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; d1rhsv = d1rhsm . (f @@ #1 &) /@ nodes;
d1base = D[base, x1]; d1aa = icm . (d1rhsv - d1cm . aa);
vd1fh =
(d1base /. {x1 -> z1, x2 -> z2}) . aa +
(base /. {x1 -> z1, x2 -> z2}) . d1aa; Print["vd1fh := ", vd1fh];
Print["nvd1f := ", Short[N[nvd1f]]];
d2alpha[rr_, ss_] :=
Sum[(d2wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]})*
(base[[ss]] /.
{x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
{ii, 1, Length[nodes]}];
d2cm =
Table[Table[d2alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
d2beta[rr_, ii_] :=
(d2wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2})*
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d2rhsm =
Table[Table[d2beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; d2rhsv = d2rhsm . (f @@ #1 &) /@ nodes;
d2base = D[base, x2]; d2aa = icm . (d2rhsv - d2cm . aa);
vd2fh =
(d2base /. {x1 -> z1, x2 -> z2}) . aa +
(base /. {x1 -> z1, x2 -> z2}) . d2aa; Print["vd2fh := ", vd2fh];
Print["nvd2f := ", Short[N[nvd2f]]];
Return[{z1, z2, vfh, vd1fh, vd2fh, det}]]]

```

■ Solution of Sturm–Liouville Equation with Homogeneous B.C.– Program List

```

slp := Module[{}, lib :=
Module[{}, r := Random[{}]; ri[a_, b_] := Random[Integer, {a, b}];
rcol := RGBColor[r, r, r];
usf[a_, x_] := Which[x < a, 0, x == a, 1/2, x > a, 1];
refine[set_] :=
Module[{}, Return[Union[Table[(set[[i]] + set[[i + 1]])/2,
{i, 1, Length[set] - 1}], set]]];
iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]];
dap := Module[{}, swdqw; data = Get[nquad]; swdslp4;
supp = (#1[[1]] & ) /@ data;
isupp = Complement[supp, {First[supp], Last[supp]}];
omega = (#1[[2]] & ) /@ data; << "dapp";
udu = (res2[#1, nodes] & ) /@ supp;
bb[j_] :=
Sum[(exp1 /. x -> supp[[r]])*omega[[r]]*udu[[r]][[2]]*
udu[[r]][[2]][[j]] +
(exp2 /. x -> supp[[r]])*omega[[r]]*udu[[r]][[1]]*
udu[[r]][[1]][[j]], {r, 1, Length[supp]}];
BBBB = Table[bb[j], {j, 1, tnn - 2}];
b[j_] :=
Sum[(exp3 /. x -> supp[[r]])*omega[[r]]*udu[[r]][[1]][[j]],
{r, 1, Length[supp]}]; BB = Table[b[j], {j, 1, tnn - 2}];
sol = LinearSolve[BBBB, BB]; sol >> "sol.out";
MapThread[{#1, #2} & , {inodes, sol}] >> "rda.out"; << "eda";
das := Module[{}, iel[5];
Print["THE CREATED DIFFUSE APPROXIMATION SPACE IS:"];
Print["mode:= ", mode]; Print["precision:= ", precision];
Print["tnn:= ", Length[nodes]]; Print["nodes:= ", N[Short[nodes]]];
Print["dim:= ", Length[base]]; Print["base:= ", Short[base]];
Print["wf:= ", wfn]; Print["REFINE FACTOR OF NODES {Sreff}\
Sreff:= ", Sreff]; Print["THE PARAMETER [nquad] IS EITHER \
[ncaw3,ncaw4,...,ncaw77] OR "];
Print[" [gqaw3,gqaw4,...,gqaw77].\
"]; Print["numq:= ", nquad]; iel[5];
dbp := Module[{}, base = Table[(1 - x)*x^(i + 1), {i, 0, dim - 1}];
dcp := Module[{}, dom = {minx, maxx}; difeq = {exp1, exp2, exp3};
bcond = {dom, {uminx, umaxx}};
uval =
ToExpression /@ Table[StringJoin["u", ToString[i]], {i, 1, tnn}]\
; iuval = Complement[uval, {First[uval], Last[uval]}];
dnp := Module[{}, ssh = NN[(maxx - minx)/(tnn - 1)];
nodes = Table[minx + k*ssh, {k, 0, tnn - 1}];
bnodes = {First[nodes], Last[nodes]}; cnodes = bnodes;
inodes = Complement[nodes, bnodes]; nnodes = refine[nodes];
bnnodes = {First[nnodes], Last[nnodes]}; cnnodes = bnnodes;
innodes = Complement[nnodes, bnnodes];
eda :=
Module[{}, dasol = (asol /. x -> #1 & ) /@ inodes;
error := Max[Abs /@ (<< "sol.out" - dasol)];
data1 =
MapThread[{#1, Abs[#2]} & , {inodes, << "sol.out" - dasol}];
int1 = ListIntegrate[data1, tnn - 3];
data2 =
({#1[[1]], #1[[2]]^2} & ) /@
MapThread[{#1, #2} & , {inodes, << "sol.out" - dasol}];
int2 = Sqrt[ListIntegrate[data2, tnn - 3]];
Print["THE RESULTS OF THIS APPROXIMATION ARE:"];
Print["DL1-error[f] := ", N[int1]];
Print["DL2-error[f] := ", N[int2]];
Print["DU-Error[f] := ", N[error]];
pwf :=
Module[{}, k = $pwf[[2]]; a = $pwf[[3]]; dm = a*ssh;

```

```

cwexp := (1 - ((ni - x)/dm)^2)^k;
dcwexp := D[(1 - ((ni - x)/dm)^2)^k, x];
wf[nodes_, i_, xx_] := cwexp /. {ni -> nodes[[i]], x -> xx};
dwf[nodes_, i_, xx_] := dcwexp /. {ni -> nodes[[i]], x -> xx}]]]
; init := Module[{}, capp; << "lib"; mode = "Real"; precision = 50;
NN := N[#1, precision] & ; minx = 0; maxx = 1; dom = {minx, maxx};
exp1 = -1; exp2 = 1; solf[t_] :=
If[Inequality[0, Less, t, LessEqual, 1], t^3*Log[t], 0];
asol = solf[x]; exp3 = Simplify[-D[exp1*D[asol, x], x] + exp2*asol];
uminx = 0; umaxx = 0; tnn = 10; dim = 7; wfn = "pwf";
$pwf = {"nu1, nu2, mu1, mu2", 2, 21/2}; $reff = 2; nu1 = 2;
nquad = "ncaw20"; << "dfp"; << "dnp"; << "dbp"; Get[wfn];
object =
{difeg, bcond, trans, tdifeg, tbcond, nodes, base, wf, wfn, rdea,
fe1, fe2, fe3, fe4, fe5, fe6, fe7, fe8, fe9, asol}];
dapp := Module[{}, res1[z_, nodes_] :=
Module[{alpha, det, beta, rhsv1, a, da},
alpha[rr_, ss_] :=
Sum[(wf[nodes, ii, xx] /. xx -> z)*
(base[[rr]] /. x -> nodes[[ii]])*
(base[[ss]] /. x -> nodes[[ii]]), {ii, 1, Length[nodes]}];
cm = Table[Table[alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; icm = Inverse[cm];
beta[rr_, ii_] :=
(wf[nodes, ii, xx] /. xx -> z)*(base[[rr]] /. x -> nodes[[ii]]);
rhsm =
Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; rhsv = rhsm . uval;
vuh = Simplify[(base /. x -> z) . icm . rhsv];
Return[{cm, icm, rhsm, rhsv, uh}]; iel[1]];
res2[z_, nodes_] :=
Module[{dalp, dbeta, rhsv1, dbase},
res1[z, nodes]; dalp[rr_, ss_] :=
Sum[(dwf[nodes, ii, xx] /. xx -> z)*
(base[[rr]] /. x -> nodes[[ii]])*
(base[[ss]] /. x -> nodes[[ii]]), {ii, 1, Length[nodes]}];
dcm =
Table[Table[dalp[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}];
dbeta[rr_, ii_] :=
(dwf[nodes, ii, xx] /. xx -> z)*(base[[rr]] /. x -> nodes[[ii]]);
drhsm =
Table[Table[dbeta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; drhsv = drhsm . uval; dbase = D[base, x];
vduh =
Simplify[(dbase /. x -> z) . icm . rhsv +
(base /. x -> z) . icm . (drhsv - dcm . icm . rhsv)];
uh := (Coefficient[vuh, #1] & ) /@ iuval;
duh := (Coefficient[vduh, #1] & ) /@ iuval; Return[{uh, duh}]]]

```

■ Approximation of Trace Operator– Program List

```

tap := Module[{}, lib :=
Module[{}, iel[n_] :=
Module[{iii}, iii[0] := "";
iii[j_] := iii[j] = StringJoin[iii[j - 1], "\n"];
Return[Print[iii[n]]];
dap := Module[{}, ClearAll[gva, gvb, gvc, gvi, rda];
g = exp /. {x1 -> #1, x2 -> #2} &;
gvb =
(NN[{#1[[1]], #1[[2]], g[#1[[1]], #1[[2]]]}] &) /@ NN[bnodes];
gvc =
(NN[{#1[[1]], #1[[2]], g[#1[[1]], #1[[2]]]}] &) /@ NN[cnodes];
gvi = (NN[{#1[[1]], #1[[2]], 1}] &) /@ NN[inodes];
gva = Union[gvb, gvc, gvi]; << "dapp";
rda = (res[#1[[1]], #1[[2]], anodes, gva] &) /@ anodes;
rda >> "rda.out"; das :=
Module[{}, iel[5]; Print["THE FOLLOWING DIFFUSE APPROXIMATION SPACE [das] IS \
CREATED FOR "; Print["THE GENERATION OF AN APPROXIMATE FOR THE TRACE OPERATOR THIS IS SI
iel[1]; Print["MODE OF COMPUTATIONS:
mode:= ", mode]; Print["PRECISION OF CALCULATIONS
precision:= ", prec]; Print["TOTAL NUMBER OF NODES IN x1 DIRECTION \
[tnnx1] tnnx1:= ", tnnx1]; Print["TOTAL NUMBER OF NODES IN x2 DIRECTION \
[tnnx2] tnnx2:= ", tnnx2]; Print["TOTAL NUMBER OF NODES
tnn:= ", tnn]; Print["DEGREE OF BASE FUNCTIONS WITH RESPECT TO \
x1 degx1:= ", degx1]; Print["DEGREE OF BASE FUNCTIONS WITH RESPECT TO \
x2 degx2:= ", degx2]; Print["ALGEBRIAC DIMENSION OF BASE FUNCTIONS \
[base] IS dim:= ", dim]; Print["SELECTED WEIGHT FUNCTION [wf] IS
wfn:= ", wfn]; iel[1];
dbp := Module[{} . base =
Union @@ Table[x1^i*x2^j, {i, 0, degx1}, {j, 0, degx2}];
dfp := Module[{}, uline1 = exp /. x2 -> 0; uline2 = exp /. x1 -> 1;
uline3 = exp /. x2 -> 1; uline4 = exp /. x1 -> 0;
g1 := uline1 /. {x1 -> #1} &; g2 := uline2 /. {x2 -> #2} &;
g3 := uline3 /. {x1 -> #1} &; g4 := uline4 /. {x2 -> #2} &;
aindex =
Sort[Union @@
Table[{k1, k2}, {k1, 0, tnnx1 - 1}, {k2, 0, tnnx2 - 1}]];
iindex =
Sort[Union @@
Table[{k1, k2}, {k1, 1, tnnx1 - 2}, {k2, 1, tnnx2 - 2}]];
cindex =
Sort[{0, 0}, {tnnx1 - 1, 0}, {tnnx1 - 1, tnnx2 - 1},
{0, tnnx2 - 1}]; bindex = Sort[Complement[aindex, iindex]];
sshx1 = NN[(maxx1 - minx1)/(tnnx1 - 1)];
sshx2 = NN[(maxx2 - minx2)/(tnnx2 - 1)];
anodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2} &) /@ aindex;
inodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2} &) /@ iindex;
cnodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2} &) /@ cindex;
bnodes =
({minx1 + #1[[1]]*sshx1, minx2 + #1[[2]]*sshx2} &) /@ bindex;
ann = Length[anodes]; bnn = Length[bnodes]; cnn = Length[cnodes];
inn = Length[inodes];
eda := Module[{}, error =
(NN[{#1[[1]], #1[[2]], #1[[3]]}] &) /@ << "rda.out" - gva;
iel[2]; Print["DU-Error[g]:=", Max[(norm[#1] &) /@ error]];
iel[2]; gnp :=
Module[{}, ClearAll[cpoly1, ccnod1, cinod1, cbnod1];
SetOptions[Graphics, AspectRatio -> 1, Axes -> True,
AxesLabel -> {"x1", "x2"}, AxesOrigin -> Automatic,
AxesStyle -> Automatic, Background -> gl0,
ColorOutput -> Automatic, DefaultColor -> rcol, Epilog -> {},
Frame -> False, FrameLabel -> None, FrameStyle -> Automatic,
FrameTicks -> Automatic, GridLines -> None,
ImageSize -> Automatic,
PlotLabel -> "NODAL POINTS USED FOR TRACE APPROXIMATION",

```

```

icm = Inverse[cm]; beta[ss_, ii_] :=
(wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2}) *
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
rhsm := Table[Table[beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; rhsv := rhsm . (#1[[3]] &) /@ gva;
aa = icm . rhsv; uh = (base /. {x1 -> z1, x2 -> z2}) . aa;
d1alpha[rr_, ss_] :=
Sum[(d1wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2}) *
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}) *
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
{ii, 1, Length[nodes]}]; d1cm =
Table[Table[d1alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; d1beta[rr_, ii_] :=
(d1wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2}) *
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d1rhsm = Table[Table[d1beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; d1rhsv = d1rhsm . (#1[[3]] &) /@ gva;
d1base = D[base, x1]; d1aa = icm . (d1rhsv - d1cm . aa);
d1uh = (d1base /. {x1 -> z1, x2 -> z2}) . aa +
(base /. {x1 -> z1, x2 -> z2}) . d1aa;
d2alpha[rr_, ss_] :=
Sum[(d2wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2}) *
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}) *
(base[[ss]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]}),
{ii, 1, Length[nodes]}]; d2cm =
Table[Table[d2alpha[rr, ss], {rr, 1, Length[base]}],
{ss, 1, Length[base]}]; d2beta[rr_, ii_] :=
(d2wf[nodes, ii, xx1, xx2] /. {xx1 -> z1, xx2 -> z2}) *
(base[[rr]] /. {x1 -> nodes[[ii]][[1]], x2 -> nodes[[ii]][[2]]});
d2rhsm = Table[Table[d2beta[rr, ii], {ii, 1, Length[nodes]}],
{rr, 1, Length[base]}]; d2rhsv = d2rhsm . (#1[[3]] &) /@ gva;
d2base = D[base, x2]; d2aa = icm . (d2rhsv - d2cm . aa);
d2uh = (d2base /. {x1 -> z1, x2 -> z2}) . aa +
(base /. {x1 -> z1, x2 -> z2}) . d2aa; Return[{z1, z2, uh, d1uh, d2uh}]

```