

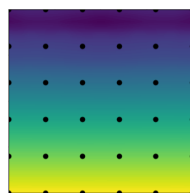
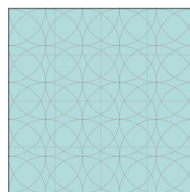
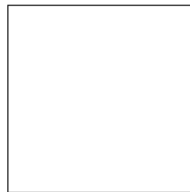


THE UNIVERSITY *of* EDINBURGH  
School of Engineering

---

## PyMFS: Developing a code for the method of finite spheres

---



Thomas **Aston**  
(Supervisor: Dr. Filipe Teixeira Dias)




## Personal Statement

Throughout this project I have worked on the development of an original Python framework (PyMFS) for implementation of the meshless numerical method: the method of finite spheres. The work carried out is not a continuation of any existing work carried out at the university or in industry, and the project idea was self-proposed following discussions with my supervisor. The early stages of the project were subject to delays as a result of my commitments representing the University of Edinburgh's rocketry team, endeavour, which culminated in my attendance at the European Rocketry Challenge in Portugal in mid-October. Following this, and before commencing with coding for the project, a significant portion of time was dedicated to gaining an understanding of the theory behind the method of finite spheres, a method proposed by Suvaranu De and Klaus-Jürgen Bathe. Time was also dedicated to improving my general Python programming skills, for which my previous experience was in the form of simple personal projects, and limited exposure during a six-month research placement in the first half of 2021.

Following this initial period of research, I began developing the code for PyMFS. Originally, the plan was to immediately develop the code with the intent of solving problems in two-dimensional linear elastic analysis of solids, before moving towards solving more complex impulsive dynamics problems. However, in hindsight this approach was overly ambitious, and further project delays were experienced due to the presence of bugs in my code of unknown origin, resulting in highly inaccurate solutions to problems. I therefore shifted the focus of the project in the early stages to initially solving less complicated differential equations defined over geometrically simpler domains. Once this was achieved, and a working code was developed, the project focus was moved back towards the analysis of solid mechanics problems, but there was no longer enough time available to extend the code for use in impulsive dynamics applications.

During the project I have had weekly meetings with my supervisor, where I would discuss the work that I had completed. Whilst my supervisor would offer general advice from his extensive experience with the programming and use of numerical methods, the source code for PyMFS is entirely my original work, and all existing libraries and codes which have been implemented in this framework (either directly, or indirectly by way of providing inspiration) are cited throughout this report where necessary. Overall, the project was undertaken without any further major setbacks aside from minor bugs presenting issues throughout. I have completed a number of the objectives that I set out to achieve at the beginning of the project, and have produced a novel, functioning Python implementation for the method of finite spheres. During this time my programming skills have improved greatly, and I have greatly enjoyed the process of drawing upon my understanding of numerical methods (from both university courses and wider reading) and directly applying them to develop a robust piece of software that can be applied to novel applications.

I declare that this thesis is my original work except where stated.

Signed: 

Thomas **Aston**, April 13, 2022



## Executive Summary

Meshless numerical methods for solving problems posed by differential equations defined over geometrically complex domains have seen significant developments in recent decades. These methods overcome the mesh-related issues faced by the traditional finite element method (FEM): the most widely applied numerical method in the field of solid mechanics.

A review of meshless methods identified the method of finite spheres (MFS) as a meshless method with particular promise: overcoming the mesh-related issues faced by the FEM, whilst avoiding problems specific to other meshless methods, such as the instability issues faced by smoothed-particle hydrodynamics. Existing software which utilises meshless numerical methods in solid mechanics is scarce, and in particular there does not exist a readily-available code which implements the MFS.

As a result of this, a Python-based framework for implementation of the method of finite spheres is developed with an initial focus on solving simple Poisson's equation problems, and problems involving the analysis of linear elastic solid bodies under equilibrium conditions. The PyMFS framework presents novel pre and post-processing modules which facilitate efficient generation and analysis of problems specifically tailored to the unique aspects of the MFS. A solver which implements the underlying theory behind the MFS is also developed, making PyMFS the first available open-source code capable of solving problems using the MFS. The framework is developed with a focus on modular programming, maximising the efficiency of future extensions or adaptations to the code.

To validate the accuracy of the solver, a series of example problems developed and solved using PyMFS are considered, and the results are compared with simple analytical models and results from the FEM. It is shown that PyMFS exhibits an order of convergence consistent with the observations of others, and is capable of achieving results with a similar level of accuracy to the FEM, but at the expense of computational times greater by up to four orders of magnitude.

Future work relating to PyMFS therefore includes improving its efficiency by utilising parallel processing, and implementing algorithms which reduce the number of required computations. Studies should also address the accuracy of solutions surrounding surfaces where essential boundary conditions are applied, before the application of PyMFS is extended to the field of impulsive dynamics, for which the MFS has shown particular promise.

*Keywords:* The Method of Finite Spheres, Meshless Numerical Methods, Solid Mechanics, Software, Programming, Python.



## Acknowledgements

I would firstly like to thank my project supervisor, Dr. Filipe Teixeira-Dias, not only for his ever-enthusiastic guidance throughout this project, but for the continuous support he has given me for the past three years. His words of encouragement kept me going throughout the most difficult times in this project, and his wealth of knowledge in relation to programming, numerical methods, and report writing has been invaluable.

I would also like to extend my gratitude to Dr. James Young for his programming help during the early stages of the project. Although our interactions were brief, his advice has stuck with me for the duration of the academic year, and his influence on this project is one that I am greatly thankful for.

Finally, I want to thank my close friends and family who I have been lucky enough to have by my side this year. Their moral support has been incredible, and they have been vital in helping me to unwind when I have needed it most.

Signed:



Thomas **Aston**, April 13, 2022





## Word count summary

Chapter	Word count	Running total
1. Introduction	624	624
2. Literature review	3,305	3,929
3. The method of finite spheres - theory	2,288	6,217
4. Programming the method of finite spheres - PyMFS	1,417	7,634
5. Validation cases	1,924	9,558
6. Conclusions	431	9,989



# Contents

<b>Notation</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Aims and objectives . . . . .	1
1.2 Project planning and management . . . . .	2
<b>2 Literature review</b>	<b>3</b>
2.1 Numerical methods in continuum mechanics . . . . .	3
2.2 The finite element method . . . . .	4
2.2.1 Formulation and implementation . . . . .	4
2.2.2 Limitations . . . . .	9
2.3 The method of finite spheres . . . . .	9
2.3.1 An overview of meshless methods . . . . .	9
2.3.2 Basic principles . . . . .	11
2.3.3 Existing developments and implementation of the MFS . . . . .	12
2.4 Moving forward - PyMFS . . . . .	14
<b>3 The method of finite spheres - theory</b>	<b>15</b>
3.1 Discretisation . . . . .	15
3.2 Interpolation scheme . . . . .	16
3.2.1 Partition of unity . . . . .	16
3.2.2 Approximation spaces . . . . .	17
3.3 Local weak form . . . . .	19
3.4 Boundary conditions . . . . .	20
3.5 Numerical integration . . . . .	22
<b>4 Programming the method of finite spheres - PyMFS</b>	<b>25</b>
4.1 Structure . . . . .	25
4.1.1 Overview . . . . .	25
4.1.2 Pre-processing . . . . .	27
4.1.3 Solve . . . . .	28
4.1.4 Post-processing . . . . .	28
4.2 Usage . . . . .	28
4.2.1 Pre-processing . . . . .	28
4.2.2 Solve . . . . .	34
4.2.3 Post-processing . . . . .	34

<b>5</b>	<b>Validation cases</b>	<b>35</b>
5.1	Poisson's equation . . . . .	35
5.1.1	1D . . . . .	35
5.1.2	2D . . . . .	35
5.2	2D elastostatics . . . . .	39
5.2.1	Case 1: tension . . . . .	39
5.2.2	Case 2: cantilever bending . . . . .	43
5.2.3	Case 3: plate with a hole . . . . .	47
<b>6</b>	<b>Conclusions</b>	<b>51</b>
6.1	Summary . . . . .	51
6.2	Future work . . . . .	51
<b>A</b>	<b>Project Gantt chart</b>	<b>57</b>
<b>B</b>	<b>Flowchart and programming diagram legend</b>	<b>59</b>
<b>C</b>	<b>MFS local weak form derivation</b>	<b>61</b>
<b>D</b>	<b>Selected MFS equations list</b>	<b>63</b>
<b>E</b>	<b>Gauss-Legendre product rule</b>	<b>67</b>
<b>F</b>	<b>MFS file format</b>	<b>69</b>

# Notation

## Variables and symbols

$\underline{\partial_\epsilon}$	– Matrix of differential operators
$\delta_I$	– Kronecker delta
$\underline{\epsilon}$	– Strain tensor
$\underline{\varphi}$	– Vector of nodal shape functions
$\varphi_I^0$	– Shepard partition of unity functions associated with node $I$
$\varphi_I$	– Shape function associated with node $I$
$\underline{\phi}$	– Matrix containing vector of nodal shape functions
$\underline{\sigma}$	– Stress tensor
$B_I$	– Sphere element associated with node $I$
$\underline{B}$	– Deformation matrix
$\underline{C}$	– Elasticity tensor
$\underline{f}$	– Global load tensor
$\underline{f}^B$	– Applied body forces
$\underline{f}^S$	– Applied surface forces
$f_{Im}$	– Forcing term associated with degree of freedom $m$ of node $I$
$\hat{f}_{Im}$	– Boundary forcing matrix term associated with degree of freedom $m$ of node $I$
$KU_{ImJn}$	– Additional forcing term required for Dirichlet boundaries
$\underline{K}$	– Global stiffness matrix
$K_{ImJn}$	– Stiffness matrix term associated with DoFs $m$ and $n$ of nodes $I$ and $J$
$KU_{ImJn}$	– Additional stiffness matrix term required for Dirichlet boundaries
$N$	– Total number of spheres within a given domain
$\underline{N}$	– Direction cosines
$p$	– Local polynomial basis
$q$	– Nodal degrees-of-freedom
$r_I$	– Radius associated with sphere element for node $I$
RMSE	– Root-mean-square error
$s$	– Localised radial coordinate
$S$	– Domain boundary
$S_f$	– Neumann boundary surface
$S_u$	– Dirichlet boundary surface
$\underline{u}$	– Displacement
$\underline{u}^S$	– Prescribed displacements on Dirichlet boundary surface
$v^\phi$	– General function within the solution space

---

$V_I^\phi$	–	Local approximation space associated with node $I$
$V$	–	Domain volume
$\underline{u}$	–	Displacement
$\underline{x}$	–	Position vector

## Acronyms and abbreviations

1D	–	One-dimensional
2D	–	Two-dimensional
3D	–	Three-dimensional
BC	–	Boundary condition
BVP	–	Boundary value problem
DEM	–	Diffuse element method
DoF	–	Degree of freedom
EFG	–	Element free Galerkin
FEM	–	Finite element method
FEA	–	Finite element analysis
MFS	–	The method of finite spheres
MLPG	–	Meshless local Petrov-Galerkin
MLS	–	Moving least squares
OOP	–	Object-oriented programming
PDE	–	Partial differential equation
PU	–	Partition of unity
SPH	–	Smoothed-particle hydrodynamics
UI	–	User-interace

# Chapter 1

## Introduction

### 1.1 Aims and objectives

The ability to simulate real-world, physical phenomena is an extremely powerful tool in the field of science and engineering. The 17<sup>th</sup> century saw the birth of many of the scientific laws and mathematical principles upon which engineering problems are being solved today, from Snell's law of light refraction to Bernoulli's work on the calculus of variations [1, 2]. The present study is specifically interested in the field of solid mechanics, in which the behaviour of solid materials is studied under the influence of forces, as well as temperature and phase changes. To the engineer, the tools within this field provide a platform upon which analysis of the behaviour of solid bodies can be carried out to evaluate the effectiveness and/or safety of a certain design.

In particular, this focus looks at the application of *meshless numerical methods* for solid mechanics problems. Meshless methods have received the attention of researchers in recent decades due to their promise in overcoming the mesh-related issues faced by the traditional finite element method (FEM) [3, 4], the most widely applied method for solving problems described by differential equations over geometrically complex domains. However, at present the implementation of meshless methods in software (available both commercially and open-source) is scarce. The method of finite spheres (MFS) is a meshless method which has shown particular promise in the field of solid mechanics [5, 6], for which there is no existing software implementation.

The aim of this project is therefore to develop, implement and validate a novel Python [7] framework for solving problems in two-dimensional (2D) solid mechanics using the MFS, upon which future developments can be built, facilitating further investigation of the method. This primary aim is achieved through completion of the following objectives:

- Development of a preliminary code capable of solving basic differential equations over geometrically simple domains using the MFS. For example, one-dimensional (1D) Poisson's equation problems.
- Extension of this basic code to 2D solid mechanics applications, namely 2D elastostatics (study of linear elastic materials under equilibrium conditions).
- Validation of code accuracy, comparing results with those from simple analytical models and

the FEM.

- Development of a novel user-interface (UI) for pre and post-processing models specifically tailored to the MFS.

## 1.2 Project planning and management

To achieve the objectives outlined in Section 1.1 within the given timeframe for the project (September 2021–April 2022), the project work was divided into four distinct phases, which are outlined briefly as follows:

- **Phase 1:** Planning and literature review.
  - **Timeframe:** September 2021–December 2021
  - **Description:** Review existing literature on mesh-based and meshless numerical methods. Develop a clear mathematical understanding of the underlying equations behind the MFS and outline coding strategy.
- **Phase 2:** Preliminary code development.
  - **Timeframe:** December 2021–January 2022
  - **Description:** Develop preliminary code which solves simple problems in 1D and 2D using the MFS. Analyse options for pre and post-processing of models.
- **Phase 3:** Final code development.
  - **Timeframe:** January 2022–March 2022
  - **Description:** Extend preliminary code to solve problems which are more general and complex (2D elastostatics). Code integration with pre and post-processing UI.
- **Phase 4:** Final analysis and reporting.
  - **Timeframe:** March 2022–April 2022
  - **Description:** Compile validation cases for developed code and perform additional required numerical analysis. Finalise and publish code.

A Gantt chart outlining the project schedule in more detail can be found in Appendix A. It should be noted that whilst initially the project had the aim of extending the use of the code to problems in impulsive dynamics, the time required to develop a comprehensive understanding of the theory behind the MFS was underestimated, and thus the focus of the project was shifted towards developing a more general software framework for solving simple problems using the MFS, focusing on modularity and user-friendliness to facilitate more efficient extension of the code to wider problem classes in the future.



## Chapter 2

# Literature review

### 2.1 Numerical methods in continuum mechanics

The simplest class of problem in solid mechanics is that which studies linear elastic materials under the conditions of equilibrium: elastostatics. In 1660, Robert Hooke observed that for small deformations of an object, the magnitude of its deformation is directly proportional to the deforming load [8]. This proportionality is described by Hooke's Law, which for the multi-dimensional case in continuous media is described by the constitutive equation:

$$\underline{\sigma} = \underline{C} \underline{\epsilon} \quad (2.1)$$

where  $\underline{\sigma}$  is the stress tensor (expression of load),  $\underline{\epsilon}$  is the strain tensor (measurement of deformation) and  $\underline{C}$  is the fourth order elasticity tensor which provides the linear mapping between  $\underline{\sigma}$  and  $\underline{\epsilon}$ . Note the use of underline notation to denote array representation of tensor quantities in multiple dimensions. In three dimensions the stress tensor is written as:

$$\underline{\sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \quad (2.2)$$

where  $\sigma_{ij}$  are the scalar components of stress in the reference directions  $i, j = 1, 2, 3$ .

Consider the arbitrary linear-elastic three-dimensional body shown in Figure 2.1. By writing the sum of forces equal to zero, the continuum is governed by the equilibrium equation:

$$\partial_{\epsilon}^T \underline{\sigma} + \underline{f}^B = 0 \quad (2.3)$$

where  $\underline{f}^B$  is the vector of applied body forces and  $\partial_{\epsilon}$  is a matrix of differential operators providing the mapping between  $\underline{\epsilon}$  and displacement,  $\underline{u}$ , such that:

$$\underline{\epsilon} = \partial_{\epsilon} \underline{u} \quad (2.4)$$

The body is subject to Neumann boundary conditions:

$$\underline{N} \underline{\sigma} = \underline{f}^S, \text{ on } S_f \quad (2.5)$$

where  $\underline{f}^S$  is a prescribed traction load vector on the Neumann boundary  $S_f$ , and  $\underline{N}$  is a matrix of the direction cosines for the unit normal vector on  $S_f$ , where the outward direction is positive. The body is also subject to Dirichlet boundary conditions:

$$\underline{u} = \underline{u}^S, \text{ on } S_u \quad (2.6)$$

where  $\underline{u}^S$  is the vector of prescribed displacements on Dirichlet boundary  $S_u$ .

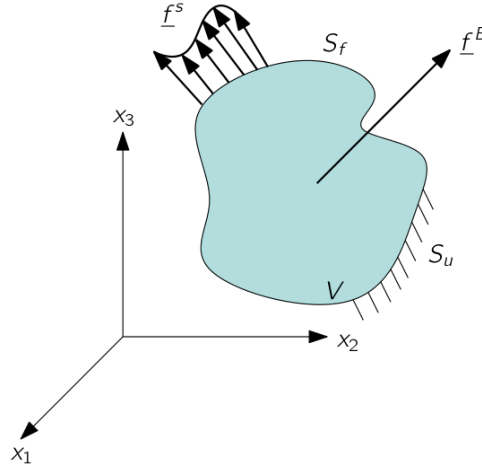


Figure 2.1: Arbitrary 3D body occupying a volume,  $V$ , with applied force  $F$  and boundary conditions  $S_f$  and  $S_u$ .

## 2.2 The finite element method

Unlike in the case of a simple geometry, such as a rectangular cross-sectioned beam, an analytical solution to Equation 2.3 for complex geometries, such as that shown in Figure 2.1, is not easily obtained. It is therefore common to adopt an approach to solving the problem in which the continuum is discretised: the governing equations of the problem are solved over simpler, well-understood discretised volumes, to obtain an approximate solution to the problem in question. Amongst the most widely-used and successful methods in this regard is the FEM, which solves complex boundary value problems (BVPs) by constructing a mesh of elements, such as the mesh of tetrahedral elements shown in Figure 2.2.

Whilst there is no definitive invention date for the method, its development can be attributed largely to work carried out by the likes of Hrennikoff and Courant in the early 1940s [9, 10], with the first appearance of the term '*finite element*' coming from Clough in 1960 [11]. Formalising a definition for the method is important not only because of what it represents conceptually, but also for its significance in moving towards the development of standard computational procedures that are capable of tackling a variety of problem classes. In perhaps the most widely referenced literature on the subject, Zienkiewicz et al. present the FEM as "*a general discretization procedure for continuum problems posed by mathematically defined statements*" [12].

### 2.2.1 Formulation and implementation

The steps involved in solving problems using the FEM are widely documented [12, 13], and are here outlined briefly due to their relevance to the methods analysed later in this study. The FEM

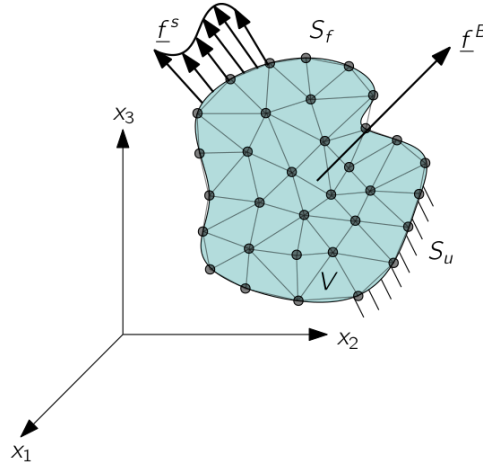


Figure 2.2: Discretisation of volume,  $V$ , with tetrahedral finite elements.

formulation for the case of two-dimensional elastostatic problems is here developed as a means of introducing key notation to be used throughout this report.

The formulation presented here is developed directly from *physical* principles due to their simplicity and significance in an engineering context. However, it is important to mention that the FEM can be developed using a more general approach, based on purely *mathematical* principles, where the problem is considered as a numerical approximation of a *strong form* BVP. Using the mathematical approach, the discretised system can be achieved by representing the system in a global *weak form*, where the continuous problem is formulated as an integral. The *strong form* refers to a solution space in which equations or conditions are required to hold absolutely. In the *weak form*, these equations or conditions are no longer required to hold absolutely, and there instead exist *weak* solutions with respect to specific "test functions".

The crucial point of note, however, is that any method which is developed from a *global* weak form requires a mesh for numerical integration to be performed. This concept is revisited in Section 2.3.

### The principle of virtual work

The FEM can be developed rather simply by considering the virtual work principle for the system shown in Figure 2.2. The principle of virtual work states: "*for the state of equilibrium the work of the impressed forces is zero for any infinitesimal variation of the configuration of the system which is in harmony with the given kinematical constraints*" [14]. This can be formally expressed in mathematical form as:

$$\int_V \delta \underline{\epsilon}^T \underline{\sigma} \, dV = \int_V \delta \underline{u}^T \underline{f}^B \, dV \quad (2.7)$$

which given the linear mappings for  $\underline{\sigma}$  and  $\underline{\epsilon}$  outlined in Equations 2.1 and 2.4 respectively, contains a single remaining unknown variable: displacement,  $\underline{u}$ . Thus far, the domain considered has been continuous, and it is therefore clear that the field variable  $\underline{u}$  (in two-dimensions  $\underline{u} = [u_x \ u_y]^T$ ) must be a continuous function of the position vector  $\underline{x}$  (in two-dimensions  $\underline{x} = [x \ y]^T$ ).

### Discretisation and interpolation

By considering a single linear triangular element in 2D (Figure 2.3a),  $\underline{u}$  can be written as a linear combination of known shape functions  $\underline{\phi}$  and the unknown nodal degrees-of-freedom (DoFs)  $\underline{q}$ , yielding the mapping  $\underline{u} \approx \underline{\phi} \underline{q}$ . In 2D,  $\underline{\phi}$  is given by the matrix:

$$\underline{\phi} = \begin{bmatrix} \underline{\varphi} & 0 \\ 0 & \underline{\varphi} \end{bmatrix}$$

where  $\underline{\varphi}$  is a vector of nodal shape functions of length  $n$  corresponding to the number of nodes per element,  $\underline{\varphi} = [\varphi_1 \ \varphi_2 \ \dots \ \varphi_n]$ . In the case of a linear triangular element, the shape functions are therefore given by three linear polynomials  $\underline{\varphi} = [\varphi_1 \ \varphi_2 \ \varphi_3]$ . A key characteristic of the shape functions in the finite element method is that they satisfy the Kronecker delta property illustrated in Figure 2.3b:

$$\varphi_I = \delta_I = \begin{cases} 1, & \text{at node } I \\ 0, & \text{at all other nodes.} \end{cases} \quad (2.8)$$

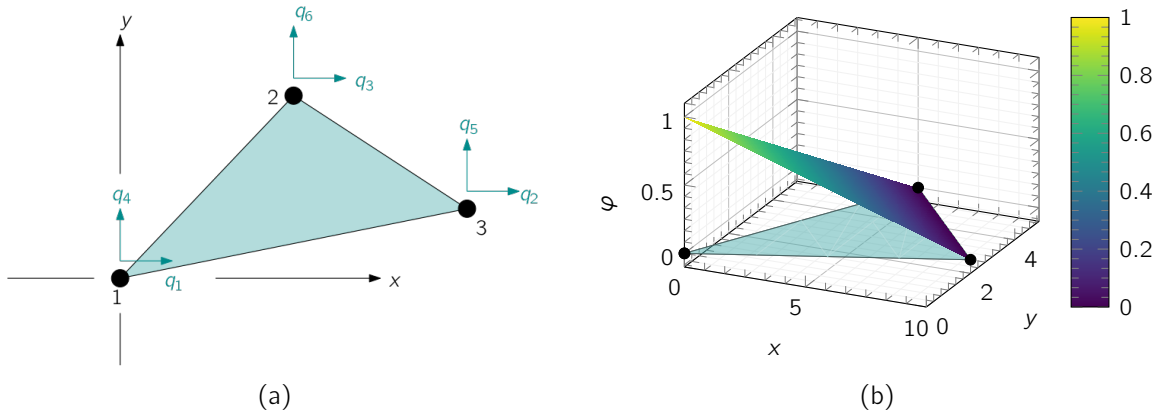


Figure 2.3: (a) Linear T3 triangle element, including nodal numbering and displacement degrees of freedom  $q_1$  to  $q_6$  and (b) illustration of the shape function  $\varphi_1$  of node 1.

### Global system connectivity

With a mapping between  $\underline{u}$  and  $\underline{q}$  established, it is possible to obtain  $\underline{\epsilon}$  as a linear combination of the DoFs and a mapping matrix,  $\underline{B} = \underline{\partial}_{\underline{\epsilon}} \underline{\phi}$  such that  $\underline{\epsilon} = \underline{B} \underline{q}$ . This expression can be substituted into Equation 2.7 and simplified to obtain the discretised virtual work equation:

$$\left( \int_V \underline{B}^T \underline{C} \underline{B} \, dV \right) \underline{q} \approx \int_V \underline{\phi}^T \underline{f}^B \, dV \quad (2.9)$$

which, ignoring the approximation, can be written concisely as the linear system:

$$\underline{K} \underline{q} = \underline{f} \equiv \begin{bmatrix} k_{11} & k_{12} & \dots & k_{1N} \\ k_{21} & k_{22} & & k_{2N} \\ \vdots & & \ddots & \vdots \\ k_{N1} & k_{N2} & \dots & k_{NN} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix} \quad (2.10)$$

where  $\underline{K}$  and  $\underline{f}$  are often referred to as the global system stiffness matrix and load vector, respectively. Stiffness matrix terms  $k_{mn}$  correspond to global system DoFs  $q_m$  and  $q_n$ , and load vector terms  $f_n$  correspond to global DoF  $q_n$ . It is clear from Figure 2.2 how a single node can be shared by multiple elements and, as a direct result of this, the stiffness matrix and load vector terms corresponding with the global system DoFs of a particular node will contain contributions from all of the elements for which this node is shared.

Figure 2.4 illustrates an example of this for a simple 2-element system. Here, node 2 is shared by elements  $e_1$  and  $e_2$ , and as such, the global DoF  $q_3$  receives contributions from local element DoF  $q_3^{e_1}$  and  $q_1^{e_2}$ . Therefore, the global stiffness matrix term  $k_{33}$ , for example, is obtained by summing the contributions of the individual element stiffness matrix terms  $k_{33}^{e_1}$  and  $k_{11}^{e_2}$  (i.e.  $k_{33} = k_{33}^{e_1} + k_{11}^{e_2}$ ). This illustrates the necessity for a connectivity array in the development of FEM codes, where the connectivity array provides the mapping from local DoFs to global assemblage DoFs.

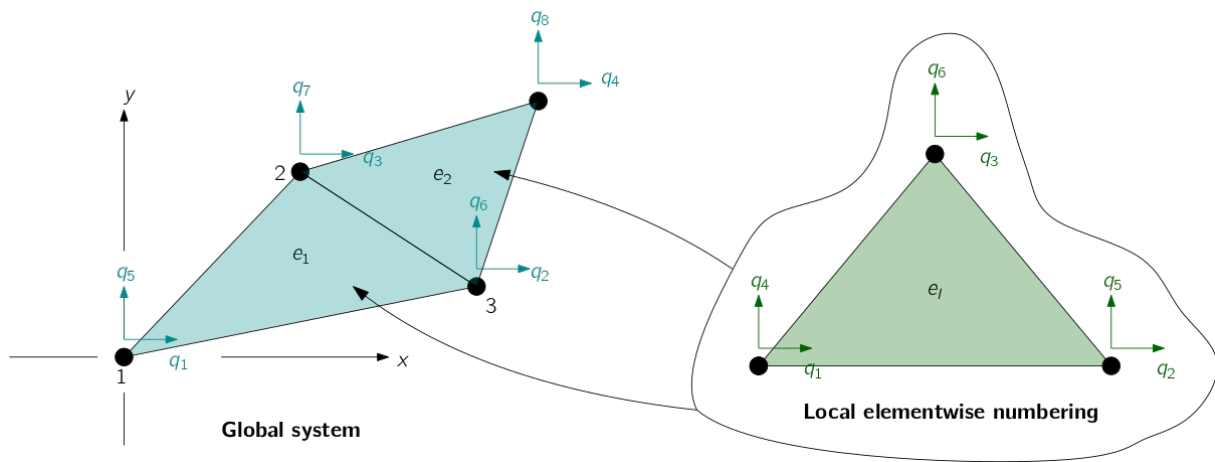


Figure 2.4: Example of two element finite element discretisation with local elementwise numbering system used for assembling global system matrices from local element matrices.

## Numerical integration and solution

The elementwise stiffness matrix and load vector terms require evaluation of the integrals from Equation 2.9, for which analytical integration is only possible in specific cases. Programs based on the FEM therefore adopt numerical integration schemes for performing integration, which can provide efficient evaluation of integrals by evaluating the integrand at a discrete set of points, known as integration points. This process is performed most efficiently with the FEM using Gauss-Legendre product rules, for which an  $n$ -point rule can yield exact results for polynomials of degree up to  $2n - 1$  [15].

Once the elementwise stiffness matrices have been evaluated, and the resulting global system matrices assembled, it is possible to solve the system of equations from Equation 2.10 to find the values of the DoFs  $\underline{q}$ . Whilst it is often possible to achieve this by simply inverting  $\underline{K}$ , for large systems of equations containing a larger number of DoFs this can become computationally expensive. The system of equations to be solved with the FEM is often sparse (containing many zeros), and thus it is common for FEM solvers to implement specific sparse equation solvers, such as that proposed by Irons [16].

### Procedure and software implementation

This section has introduced the fundamental principles behind the FEM: the most widely implemented method for solving partial differential equations (PDEs) problems in solid mechanics. Figure 2.5 provides a flowchart visualisation of how the general FEM procedure as discussed thus far is typically implemented in computer programs. Table B.1 in Appendix B should be used as a reference for all flowchart symbols used throughout this report.

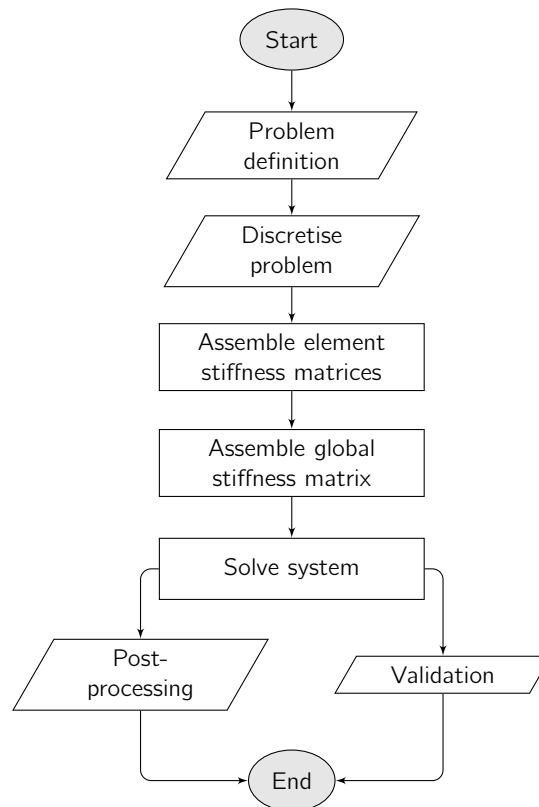


Figure 2.5: Flowchart outlining the general steps implemented in computer programs based on the FEM.

This representation of the overall FEM procedures includes all of the key ingredients of the numerical aspects of the FEM:

- *Interpolation scheme*: the fundamental principle which allows for the continuous domain to be discretised. Continuous field variables are approximated by a finite number of unknown DoFs, which are interpolated by a matrix of shape functions.
- *Integration scheme*: the method for evaluating the key parameters required to solve the algebraic system of equations.
- *Nodal connectivity*: the process of assembling global system matrices from the elementwise matrices obtained by numerical integration.
- *System solver*: the method for solving the resulting algebraic system of equations for the desired unknown variables.

It is these fundamental principles which have provided the basis for the implementation of the FEM in modern computer programs. The first program to implement the FEM for finite element analysis (FEA) of structures came in the late 1960s, when NASTRAN (NASA STRuctural ANalysis) was developed for NASA under U.S. government funding [17]. This program was the first of its kind, becoming the crest of a wave of implementation of computer-aided tools in engineering analysis. Since the 1960s, with developments in both science and technology, FEA tools have become increasingly advanced, with a wide range of commercial and open-source software packages offering FEA tools for studying a variety of engineering problems: from simple elastostatics to more complex problems such as transient wave propagation through inhomogeneous media.

### 2.2.2 Limitations

Despite its popularity, the FEM is not without its disadvantages. The FEM relies on a high-quality mesh in order to obtain accurate results, which can pose difficulties when the domain studied is geometrically complex (see Figure 2.1). Specific problems surrounding the mesh include:

- *Mesh generation*: production of a high-quality mesh on a complex domain is time consuming, and requires care to remove distorted elements [13].
- *Element distortion*: distorted elements, such as those shown in Figure 2.6, reduce the capability of an element to represent polynomials of the required order, causing inaccuracies in numerical integration and therefore a reduction in solution accuracy [18].
- *Mesh alignment and refinement*: problems containing discontinuities and singularities within the domain require special treatment [13, 19].

These mesh-related issues with the FEM can contribute to problems when modelling free surfaces, deformable boundaries, moving interfaces, large deformations and crack propagations [20]. There is therefore significant interest in the development of numerical methods which maintain the advantages of the FEM, whilst overcoming these mesh-related issues.

## 2.3 The method of finite spheres

### 2.3.1 An overview of meshless methods

The mesh-related issues faced by the FEM (as described above) cannot be overcome with a completely mesh-based method, and as a result a series of meshless numerical methods have been developed over recent decades. An overview of a number of promising meshless methods is given in [3, 4], including:

- *Smoothed-particle hydrodynamics* (SPH) [21];
- *The diffuse element method* (DEM) [22];
- *The element free Galerkin* (EFG) *method* [23];
- *The hp-clouds method* [24];
- *The meshless local Petrov-Galerkin* (MLPG) *method* [25].

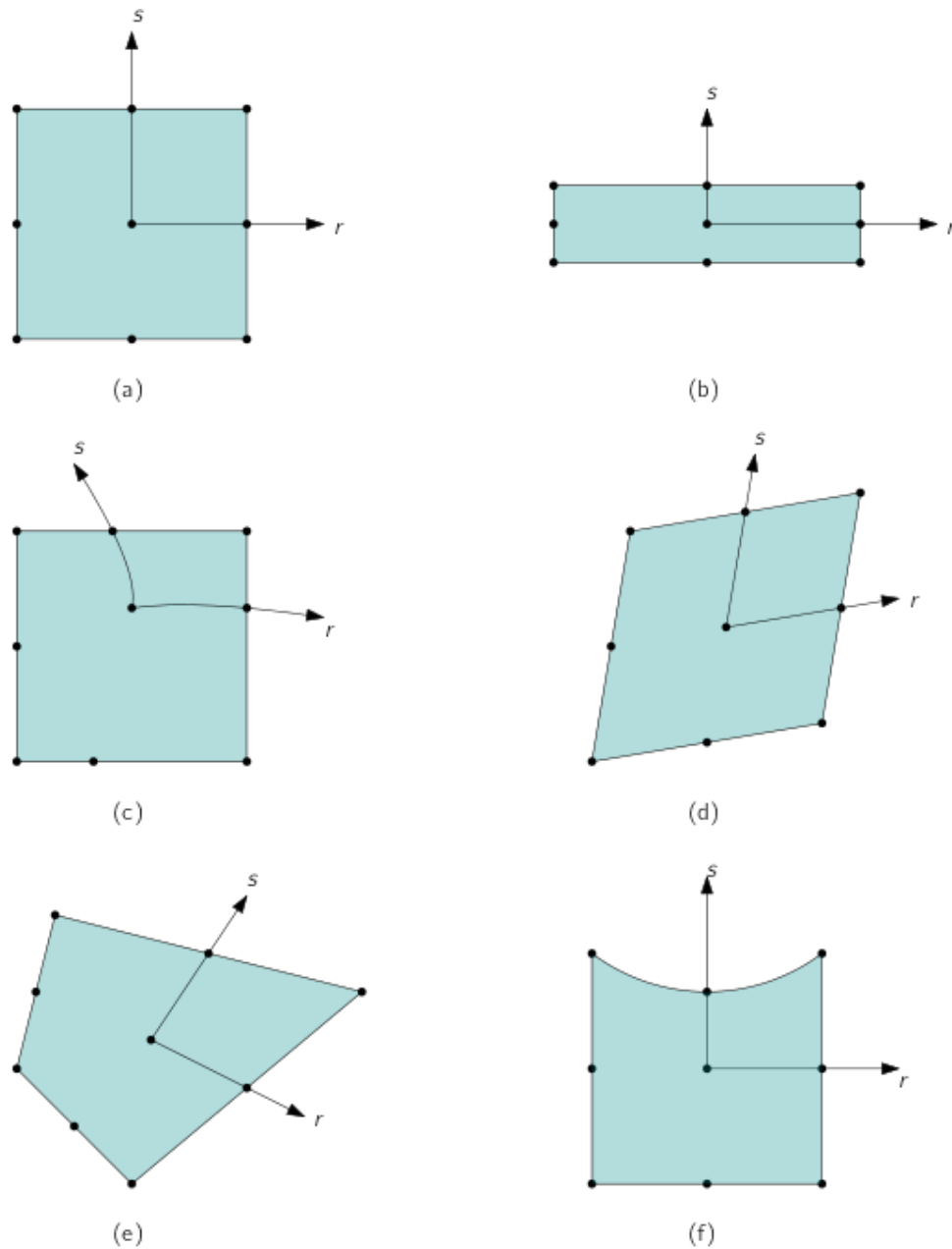


Figure 2.6: Classification of various distortions of a 9-node element: (a) undistorted configuration, (b) aspect-ratio distortion, (c) unevenly-spaced-nodes distortion, (d) parallelogram distortion, (e) angular distortion and (f) curved-edge distortion (adapted from [18]).



A "truly" meshless technique should operate with both an interpolation and integration scheme which are independent of any mesh (connectivity between a local and global system). Other than the MLPG method and SPH, the methods listed here should actually be considered "pseudo"-meshless [5]: the numerical integration techniques (and at times the imposition of boundary conditions) require the assembly of a background mesh. This is not the case for the MLPG method and SPH, which can be considered "truly" meshless. SPH is the oldest meshfree method, and offers a number of unique advantages as a result of being a meshless Lagrangian particle method in which the state of a system is represented by a set of particles. However, the method suffers from issues affecting solution accuracy, such as tensile instability and spurious boundary deformations [20, 21]. These inaccuracies, alongside the general requirement for a large number of particles and accurately tuned solution parameters, highlight the necessity for further work before the SPH can be considered a robust and efficient method.

As a "truly" meshless method, the MLPG method appears to be the most promising alternative. As is the case in the DEM and the EFG method, interpolation functions in the MLPG method are based on *moving least squares* (MLS), which allows for interpolation to be performed without a mesh. The EFG method and DEM, however, are global weak form methods. Therefore, similarly to in the case of the FEM, they require the construction of a mesh to perform numerical integration. The MLPG method, on the other hand, uses a *local weak form*. This allows for the numerical integration to be performed over local subdomains, removing the requirement for a mesh for *both* interpolation and integration [25].

With the MLPG method the local sub-domain may be any simple geometry, such as a sphere, cube or ellipsoid. For interpolation, any class of functions with compact support, and which satisfy certain approximation properties, such as MLS or *partition of unity* (PU) functions, can be used [26]. The method has been shown to be successful when applied to a range of problems, such as thin beams and elastostatics [27, 28], but its formulation remains general in nature. The *method of finite spheres* (MFS) is a particularly promising method, which seeks to build upon the advantages of the MLPG whilst moving towards a more tightly defined, yet still widely applicable, solution technique.

### 2.3.2 Basic principles

The MFS is a specialised case of the MLPG in which the choice of geometric sub-domains, test and trial function spaces, numerical integration technique, and a procedure for imposing the essential boundary conditions are all specified [5]. Figures 2.2 and 2.3 illustrated how the support of shape functions corresponding to nodes in the FEM are typically  $n$ -dimensional polytopes. The MFS, however, uses  $n$ -dimensional overlapping spheres as supports, as illustrated in Figure 2.7.

By introducing the influence of sphere overlap regions into the evaluation of integrated quantities, significant additional computational expense is incurred with the MFS [23]. There are also additional complications introduced by the MLS in the form of matrix invertability requirements. In the MFS, however, this can be bypassed by using the PU paradigm [5, 29]. A more detailed explanation of this, as well as other key ingredients unique to the MFS are provided in Chapter 3 where the formulation of the method is presented in full.

The important takeaway here is that the MFS possesses a number of the advantages of meshless methods (such as the MLPG method, DEM and EFG method) by eliminating the mesh-related issues of the FEM, but also inheriting the FEM's reliability and stability, which cannot be said for SPH, the most popular meshless method at present. Table 2.1 provides a brief qualitative overview

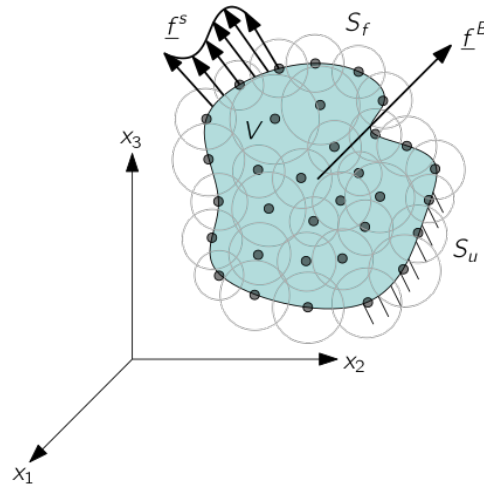


Figure 2.7: Discretisation of volume  $V$  with overlapping spheres centred around nodes. These spheres act as support for shape functions in the MFS.

of the key differences between the FEM, SPH and the MFS discussed here, from which the MFS can be identified as a particularly promising method.

Table 2.1: Comparison between the FEM, SPH and the MFS in terms of the accuracy and efficiency of each method.

Method	Meshless	Advantages	Disadvantages
FEM	×	<ul style="list-style-type: none"> <li>• More computationally efficient than current meshless methods.</li> </ul>	<ul style="list-style-type: none"> <li>• Requires generation of high-quality mesh to ensure accuracy.</li> <li>• Possibility of element distortions.</li> </ul>
SPH	✓	<ul style="list-style-type: none"> <li>• "Truly" meshless, eliminates mesh-related issues of the FEM.</li> <li>• Well-suited to high velocity and large deformation applications [21].</li> </ul>	<ul style="list-style-type: none"> <li>• Tensile instability.</li> <li>• Spurious boundary deformations.</li> <li>• Computationally inefficient compared to the FEM.</li> </ul>
MFS	✓	<ul style="list-style-type: none"> <li>• "Truly" meshless, eliminates mesh-related issues of the FEM.</li> <li>• More stable than SPH (essentially the FEM with overlapping elements).</li> </ul>	<ul style="list-style-type: none"> <li>• Computationally inefficient compared to the FEM.</li> <li>• Incorporation of element overlap regions adds complexity.</li> </ul>

### 2.3.3 Existing developments and implementation of the MFS

Due to the clear potential of the MFS as a relatively efficient and reliable meshless method, it has been the subject of various research works in recent years. Figure 2.8 gives a timeline of some of the research works which have presented key developments relating to the MFS and its

implementation, with a particular focus in more recent years on works which have been directly relevant to the study of solid mechanics problems.

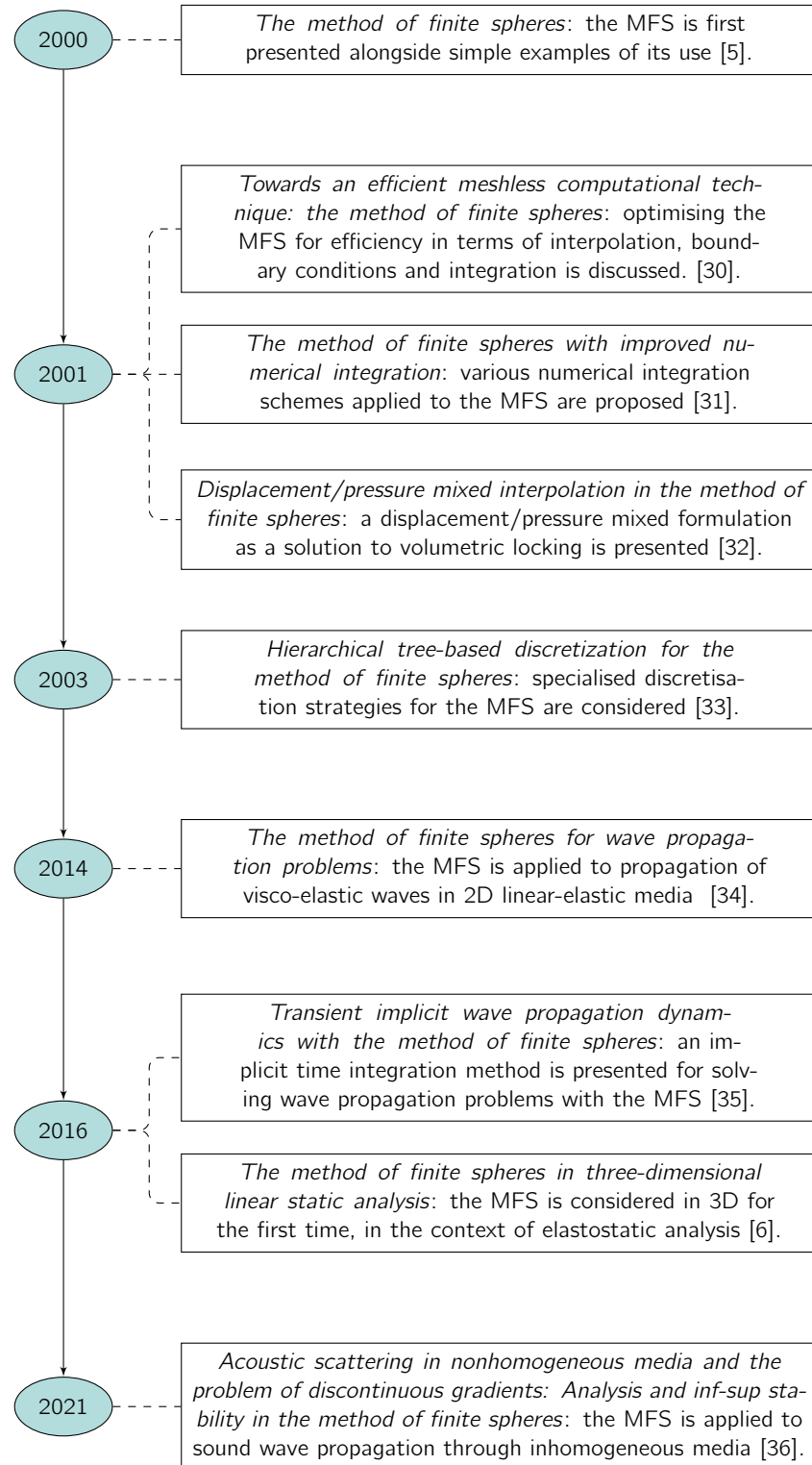


Figure 2.8: Timeline illustrating the main developments of the MFS in recent decades, with a focus on works directly relevant to the field of solid mechanics.

At the time of writing, a software implementation of the MFS has not been made publicly available through either open-source or commercial means, presenting a barrier to researchers and other interested parties from investigating the method further. Previous studies have implemented the MFS by programming custom interpolation and integration routines within existing commercial packages designed for FEA, namely ADINA, in order to solve specific problems relating to elastostatics and wave propagations [6, 34]. Whilst this presents a relatively efficient implementation to those already familiar with the method and the specific programming languages required for developing subroutines for individual FEA packages, there are a number of benefits to developing an open-source, easily adaptable framework which is specifically dedicated to the MFS.

In recent years, open-source implementations of the FEM and SPH in Python have received significant attention, notably through SfePy and PySPH [37, 38]. Python is a high-level programming language [7] featuring a number of readily available libraries for scientific computing. Studies have shown that higher-level programming languages can lead to greater programmer productivity [39, 40], and the essence of high-level languages (the use of more natural language elements than lower-level languages, such as C/C++ or FORTRAN) allows code to be more readily understood and adapted by a wider audience. The purpose of the present study is to therefore develop an open-source Python implementation of the MFS, which will provide a framework allowing for future research to further investigate the MFS and extend its use to a range of different applications.

## 2.4 Moving forward - PyMFS

The remainder of this report is therefore dedicated to the development, implementation and validation of PyMFS: a Python implementation of the method finite spheres, which is here developed to solve Poisson equation and elastostatic problems in one and two-dimensions, but by-design can be extended to be applied to a much wider range of problems. Chapter 3 introduces the theory behind the MFS in more detail, and presents the interpolation and integration schemes that are implemented in PyMFS. Chapter 4 then provides detail on the program structure of PyMFS, before giving an introduction on how to operate PyMFS interface to solve problems. This prefaces Chapter 5, which illustrates solutions to various example problems which have been solved using PyMFS, and discussed the obtained results, including present limitations of the developed code. Finally, Chapter 6 draws conclusions on the work presented, and offers suggestions for possible future work.

## Chapter 3

# The method of finite spheres - theory

A general formulation of the MFS is here presented as a means of introducing the underlying theory behind the PyMFS framework developed in Chapter 4, and is based on existing presentation of the required equations from literature [5, 6].

### 3.1 Discretisation

In Figure 3.1, the arbitrary three-dimensional domain considered thus far is again presented, with boundary  $S = S_u \cup S_f$ . The vector  $\underline{n}$  is also introduced, which is the unit normal to the boundary, where the outward direction is positive. The domain is discretised using the set of spheres  $\{B(\underline{x}_I, r_I); I = 1, \dots, N\}$ , where  $\underline{x}_I$  is the position vector of the central node of each sphere  $B_I$ , and  $r_I$  is the sphere radius. Here,  $N$  is the total number of spheres, where  $I$  is used as the nodal numbering label associated with each sphere. Figure 3.1 also illustrates the two sphere classifications in the MFS: interior and boundary spheres. Figure 3.2 provides an illustration of how sphere elements are represented in 1D, where the elements reduce to segments of a line.

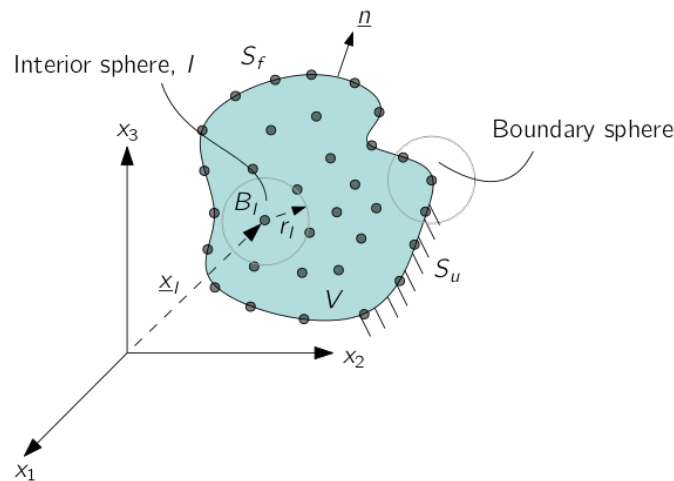


Figure 3.1: Illustration of the parameters which define each unique sphere in the MFS, adapted from [6].

For a valid domain discretisation the following three conditions must be met:

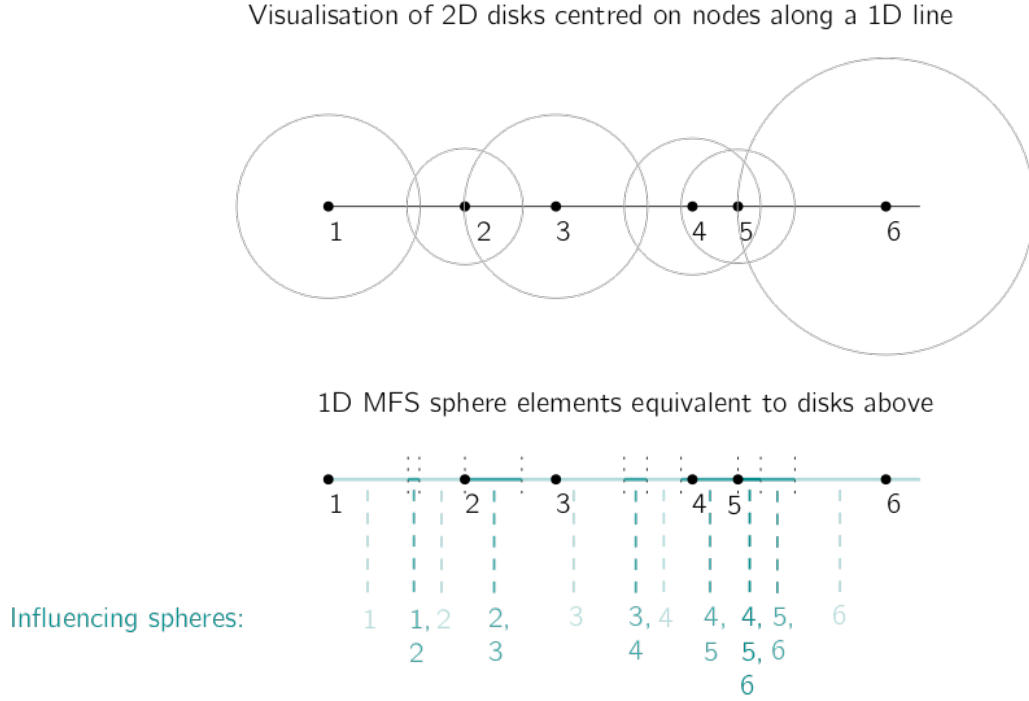


Figure 3.2: Illustration of the MFS sphere elements in 1D, where the "spheres" can be thought of as segments of a line.

- The centre of each sphere (which equates to the node associated with each sphere) must lie within the domain.
- The union of all spheres must form a complete covering of the domain.
- Spheres must not be completely contained inside other spheres.

## 3.2 Interpolation scheme

### 3.2.1 Partition of unity

As mentioned in Chapter 2, the MFS interpolation scheme is based on the PU paradigm [29]. In the MFS, the shape functions are given by the product of Shepard PU functions and local basis functions, where the Shepard PU functions are given by:

$$\varphi_I^0(\underline{x}) = \frac{W_I(\underline{x})}{\sum_{J=1}^N W_J(\underline{x})} \quad (3.1)$$

where  $W_I$  is a positive radial weighting function associated with node  $I$ . The Shepard PU functions can only reproduce constant functions exactly, and are therefore said to have *zeroth-order consistency*. It is also clear from Equation 3.1 that the Shepard functions are nonpolynomial in nature. A typical choice for the weighting function  $W_I$  following the work of [41] is the quartic spline, which can be written as:

$$W_I(s) = \begin{cases} 1 - 6s^2 + 8s^3 - 3s^4, & 0 \leq s \leq 1 \\ 0, & s > 1 \end{cases} \quad (3.2)$$

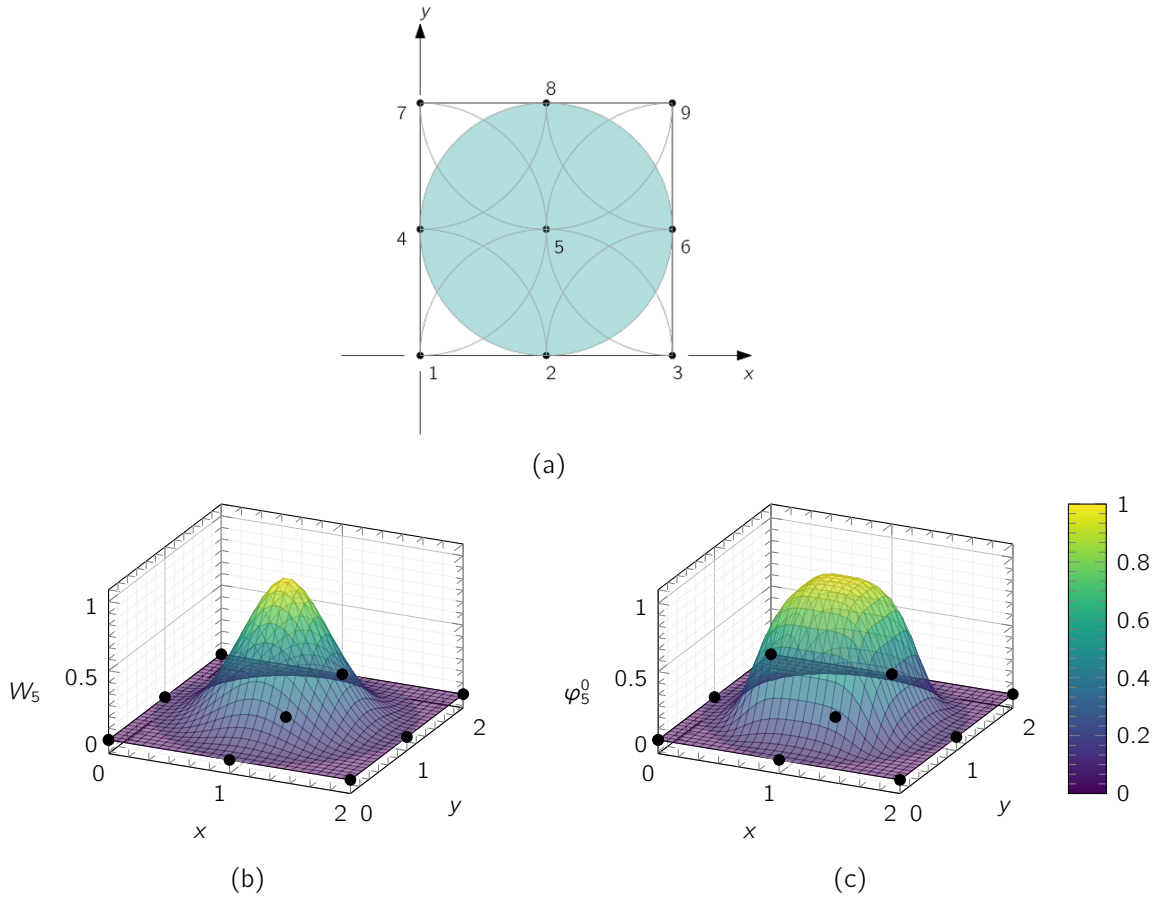


Figure 3.3: (a) MFS discretisation of square domain using 9 two dimensional sphere elements, (b) weighting function associated with node 5,  $W_5(\underline{x})$  and (c) the associated Shepard PU function  $\varphi_5^0$ .

where  $s$  can be thought of as a local radial coordinate of position  $\underline{x}$  with respect to nodal coordinate  $\underline{x}_I$ , given by:

$$s = \frac{\|\underline{x} - \underline{x}_I\|}{r_I}. \quad (3.3)$$

Figure 3.3a illustrates the example of a two dimensional square domain discretised with 9 sphere elements (which in 2D are disks) of equal radius. The weighting function associated with the central node is shown in Figure 3.3b and the resulting Shepard PU function is shown in Figure 3.3c.

### 3.2.2 Approximation spaces

To generate approximation spaces with greater than zeroth order consistency, it is necessary to introduce local approximation spaces at each node  $I$ :

$$V_I^\phi = \text{span}_{m \in \mathcal{J}} \{p_m(\underline{x})\} \quad (3.4)$$

where  $h$  is a measure of sphere size,  $\mathcal{J}$  is an index set and  $p_m(\underline{x})$  is a local basis member. The global approximation space,  $V^\phi$  can then be written as:

$$V^\phi = \sum_{I=1}^N \varphi_I^0 V_I^\phi. \quad (3.5)$$

Thus, any function within the solution space,  $v^\phi \in V^\phi$ , can be written as:

$$v^\phi(\underline{x}) = \sum_{l=1}^N \sum_{m \in \mathcal{J}} \phi_{lm}(\underline{x}) q_{lm} \quad (3.6)$$

where  $\phi_{lm}$  is the shape function associated with the  $m$ th degree of freedom at node  $l$  and  $q_{lm}$  is the scalar value of the  $m$ th degree of freedom at node  $l$ . The shape functions  $\phi_{lm}$  are given by:

$$\phi_{lm}(\underline{x}) = \varphi_l^0(\underline{x}) p_m(\underline{x}). \quad (3.7)$$

Here, a unique benefit of the MFS over the FEM arises: the local approximation space can be chosen depending on the specific class of problem being solved such that the accuracy and efficiency of the MFS are improved. For example, for problems in two-dimensional linear elasticity (which are elliptic in nature):

$$V_l^\phi = \text{span} \left\{ 1, \frac{x - x_l}{r_l}, \frac{y - y_l}{r_l}, \frac{x - x_l}{r_l} \cdot \frac{y - y_l}{r_l} \right\} \quad (3.8)$$

is a suitable local approximation space containing the terms of a complete polynomial of first degree, as prescribed by the Pascal triangle shown in Figure 3.4. However, it has been shown that for solving hyperbolic problems, such as those involving transient wave propagations, local bases containing trigonometric functions are suitable, and allow the MFS to compete with the FEM in terms of efficiency [34]. The local approximation space given in Equation 3.8 contains 3 terms, meaning that there are 3 unique shape functions for each translational direction. In two dimensions, there are two translational degrees of freedom, and therefore there are  $2 \times 4 = 8$  degrees of freedom associated with each node for the given approximation space. A visualisation of this concept is provided in Figure 3.4.

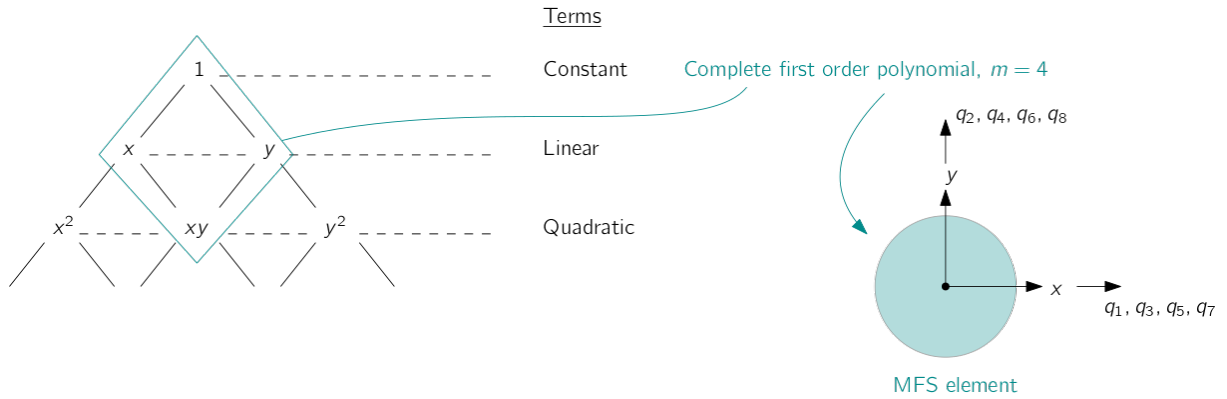


Figure 3.4: Visualisation of complete first-order polynomial using the Pascal triangle in two dimensions and illustration of how the choice of local basis terms determines the number of DoFs of the MFS element.

Again, using the approximation space from Equation 3.8, the nodal shape functions for a single translational direction can be developed by substituting the approximation space into Equation 3.7, yielding:

$$\phi_{lm \in \mathcal{O}}(\underline{x}) = \{\phi_{l1}, \phi_{l3}, \phi_{l5}, \phi_{l7}\} = \left\{ \varphi_l^0, \varphi_l^0 \frac{x - x_l}{r_l}, \varphi_l^0 \frac{y - y_l}{r_l}, \varphi_l^0 \frac{x - x_l}{r_l} \cdot \frac{y - y_l}{r_l} \right\} \quad (3.9)$$



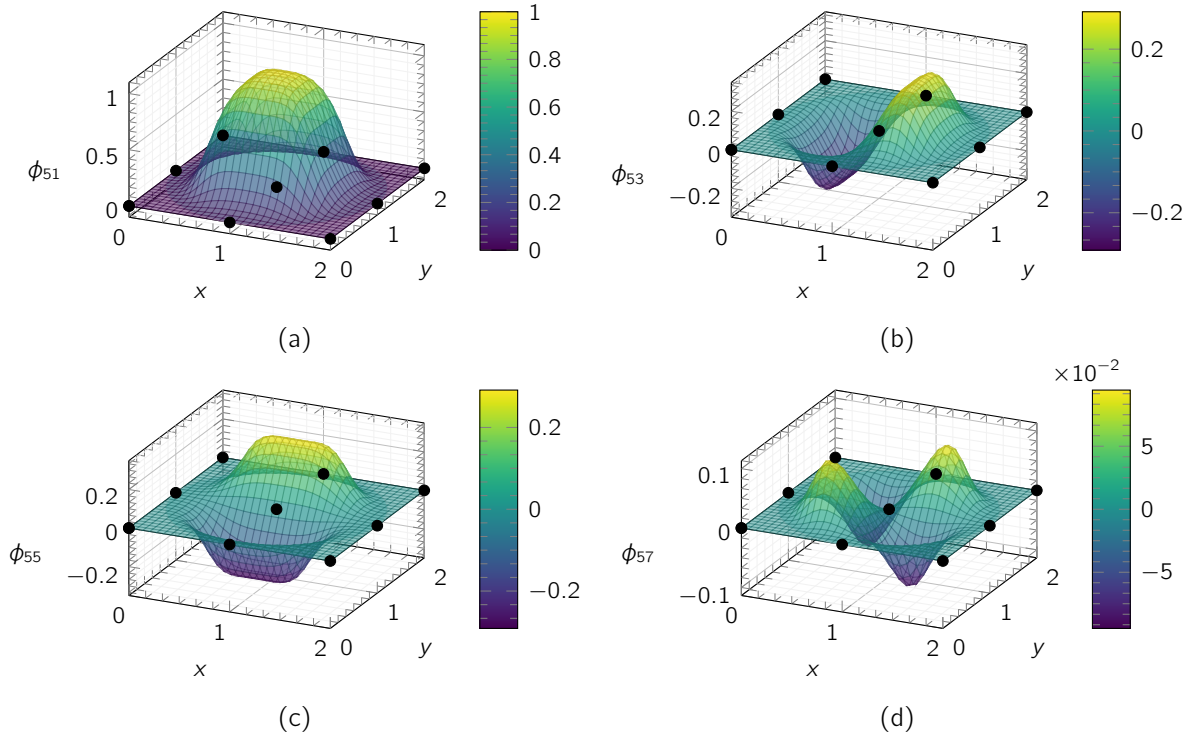


Figure 3.5: Shape functions associated with the  $x$ -direction of the (a) MFS discretisation of square domain using 9 two dimensional sphere elements, (b) weighting function associated with node 5,  $W_5(\underline{x})$  and (c) the associated Shepard PU function  $\phi_5^0$ .

where  $\mathcal{O}$  is the set of odd numbers, allowing for the shape functions for only a single translational direction to be written (in the case of the element from Figure 3.4 this would only consider the  $x$  DoFs). These shape functions are plotted (again for the central node in Figure 3.3a) in Figure 3.5.

From Figure 3.5 there are two key points to be addressed:

- The nonpolynomial nature of the MFS shape functions is clear, particularly when compared with the typical FEM shape function in Figure 2.3.
- Also unlike the FEM, these shape functions are not required to satisfy the Kronecker Delta property, and they are influenced by surrounding nodes and the subsequent overlap regions. It is therefore important to be aware that for a different arrangement of spheres in Figure 3.3a, the shape functions shown would also change.

### 3.3 Local weak form

The derivation of the local weak form of the MFS for solution of general PDEs is well documented [5]. In the interest of brevity, a full derivation is therefore here omitted, but provided in Appendix C for reference. It is here summarised that a general second-order PDE in a single variable  $u$  over the domain  $V$  in Figure 3.1, such as:

$$Au = f$$

where  $A$  is a second-order symmetric positive definite differential operator and  $f$  is a forcing function, can be written for the  $m$ th DoF of the  $l$ th node in Figure 3.1 as:

$$\sum_{J=1}^N \sum_{n \in \mathcal{J}} K_{lmJn} q_{Jn} = f_{lm} + \hat{f}_{lm} \quad (3.10)$$

where  $K_{lmJn}$  is a stiffness matrix term associated with DoFs  $m$  and  $n$  of nodes  $l$  and  $J$  respectively,  $f_{lm}$  is a term associated with internal domain forcing and  $\hat{f}_{lm}$  is a term associated with forcing on the domain boundaries. In their general form,  $K_{lmJn}$ ,  $f_{lm}$  and  $\hat{f}_{lm}$  can be written as:

$$K_{lmJn} = a(\phi_{lm}, \phi_{Jn}) = \int_{V_l \cap V} c(\underline{x}) \phi_{lm} \phi_{Jn} dV + \sum_{i,j=1}^d \int_{V_l \cap V} a_{ij}(\underline{x}) \frac{\partial \phi_{lm}}{\partial x_i} \frac{\partial \phi_{Jn}}{\partial x_j} dV, \quad (3.11)$$

$$f_{lm} = \int_{V_l \cap V} f \phi_{lm} dV, \quad (3.12)$$

$$\hat{f}_{lm} = \sum_{i,j=1}^d \int_{S_l} \phi_{lm} n_i a_{ij}(\underline{x}) \frac{\partial u^{\phi,p}}{\partial x_j} dS, \quad (3.13)$$

where  $c(\underline{x})$  and  $a_{ij}(\underline{x})$  are bounded coefficients. For interior spheres which have zero overlap with the boundary,  $\hat{f}_{lm} = 0$ , and therefore Equation 3.10 reduces to the system:

$$\sum_{J=1}^N \sum_{n \in \mathcal{J}} K_{lmJn} q_{Jn} = f_{lm}.$$

In Chapter 4, a solver is developed for obtaining solutions to specific problem classes using the MFS, and Chapter 5 presents selected results. Three distinct problems are considered:

- Poisson's equation in 1D :  $\frac{d^2 u(x)}{dx^2} + f(x) = 0$  (3.14)
- Poisson's equation in 2D :  $\frac{d^2 u(x, y)}{dx^2} + \frac{d^2 u(x, y)}{dy^2} + f(x, y) = 0$  (3.15)
- 2D elastostatics : See Equation 2.3.

Appendix D provides the complete set of equations required to solve the system in Equation 3.10 for each of these problem classes.

### 3.4 Boundary conditions

Since the MFS shape functions are not required to satisfy the Kronecker delta property, any sphere element which intercepts the domain boundary will contribute to the evaluation of the boundary forcing term  $\hat{f}_{lm}$ . The remainder of this section will present how this affects the incorporation of natural and essential boundary conditions (BCs) in the MFS, and the strategies adopted in PyMFS to deal with these BCs.

It is necessary to define the regions over which integration is required for boundary spheres. Figure 3.6 illustrates examples of two boundary spheres. Figure 3.6a shows a sphere which intersects

a natural boundary condition, for which integration over the sphere is performed over  $B_I \cap V$ , and integration over the Neumann boundary is performed on the surface segment  $S_{f_i}$ . Similarly, for spheres intersecting boundaries with Dirichlet conditions applied, integration is performed over  $B_I \cap V$  and boundary integration is now performed along  $S_{u_i}$ .

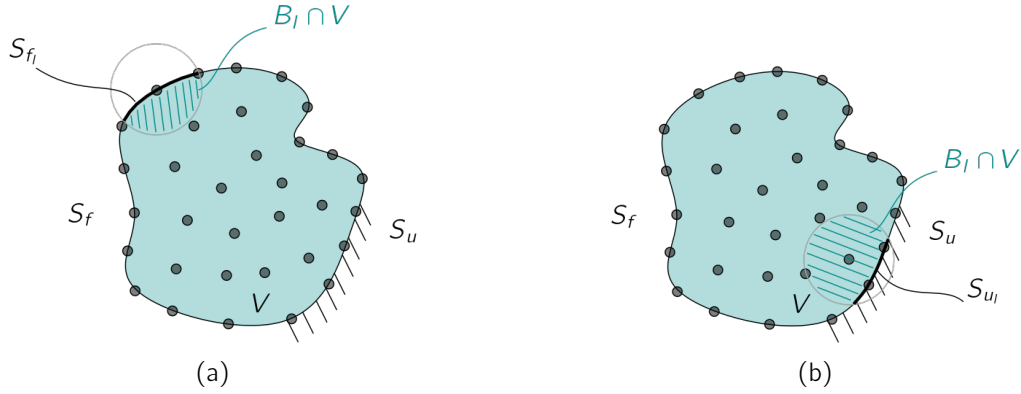


Figure 3.6: Illustration of regions of integration for spheres intersecting surfaces with (a) natural and (b) essential boundary conditions applied, adapted from [5].

### Natural boundary conditions

The imposition of natural boundary conditions in the MFS is relatively simple. For spheres with nonzero intersection with surfaces with prescribed natural BCs, Equation 3.10 applies, in which  $\hat{f}_{Im}$  is calculated as follow:

$$\hat{f}_{Im} = \int_{S_{f_i}} \phi_{Im} f^s \, dS. \quad (3.16)$$

### Essential boundary conditions

Due to the absence of the Kronecker delta property for the MFS shape functions, the imposition of essential boundary conditions is made considerably more complex. In [5] it is shown that for spheres which intersect surfaces with prescribed essential BCs,  $\hat{f}_{Im}$  can now be written as:

$$\hat{f}_{Im} = \sum_{J=1}^N \sum_{n \in \mathcal{J}} K U_{ImJn} q_{Jn} - f U_{Im} \quad (3.17)$$

where  $K U_{ImJn}$  and  $f U_{Im}$  can be written in general as:

$$K U_{ImJn} = \sum_{i,j=1}^d \int_{S_{u_i}} \frac{\partial}{\partial x_j} (a_{ij}(\underline{x}) \phi_{Im} \phi_{Jn} n_i) \, dS, \quad (3.18)$$

$$f U_{Im} = \sum_{i,j=1}^d \int_{S_{u_i}} u^s \frac{\partial}{\partial x_j} (a_{ij}(\underline{x}) \phi_{Im} n_i) \, dS. \quad (3.19)$$

By substituting Equation 3.17 into Equation 3.10 and rearranging, the linear system for spheres intersecting surfaces with essential BCs applied can be written as:

$$\sum_{J=1}^N \sum_{n \in \mathcal{J}} (K_{ImJn} - K U_{ImJn}) q_{Jn} = f_{Im} - f U_{Im}. \quad (3.20)$$

The strategy proposed here provides the most generalised approach to implementing Dirichlet BCs in the MFS. However, evaluation of the additional terms  $KU_{ImJn}$  and  $fU_{Im}$  adds computational expense and complexity, due to the requirement to calculate positions at which spheres intersect the surfaces upon which the BCs are applied. Therefore, [5] propose the special nodal arrangement on Dirichlet boundaries shown in Figure 3.7, which bypasses this requirement by forcing the Kronecker delta property.

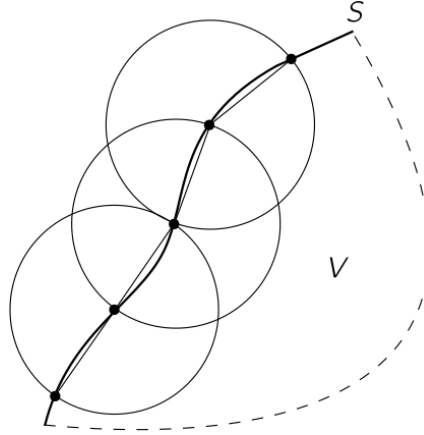


Figure 3.7: Special nodal arrangement for Dirichlet boundaries.

For the arrangement shown, in which nodes are equispaced along the surface and sphere radii are equal to the nodal spacing, the shape functions associated with the first DoF in each translational direction appease the Kronecker-delta property. It was not explicitly stated at the time, but this was highlighted for the regular arrangement of nodes in Figure 3.5. By adopting this arrangement, it is possible to delete the rows and columns from the global structure matrices associated with these DoFs, resulting in a reduced system of Equation 3.10. This arrangement is further investigated in Chapter 5.

### 3.5 Numerical integration

With the system of equations to be solved, and the required variables fully defined, it is necessary to implement a suitable numerical integration scheme for evaluation of the integrated quantities. As discussed in Chapter 2, various integration schemes have been proposed and evaluated for the MFS, such as in [31]. However, these schemes typically require coordinate transformations and subsets of rules depending on the sphere classification, which adds unnecessary complexity and computational expense.

The integration scheme implemented in PyMFS is a modified piecewise Gauss–Legendre quadrature rule [42] that was developed for solving wave propagation problems with the MFS [34] and has since been proven to be accurate in solving problems in 3D elastostatics [6]. This scheme appears to be the most simple meshless integration procedure used to-date, offering the advantage of a single universal integration rule applied to interior and boundary spheres, and sphere overlap regions, as defined by Figures 3.8a - 3.8c. Appendix E provides further detail on the underlying mathematical principles by which integrals are evaluated using Gauss–Legendre quadrature rules, whilst this section is limited to presenting the integration points adopted in PyMFS.

The adopted integration scheme is developed by dividing the subdomain of each sphere element into quadrants. Within each of these quadrants, it is then possible to use the same quadrature rules as adopted in FEA [13]. This provides two main advantages over other more complex integration rules developed for the MFS: the density of integration points is uniform, and integration of sphere overlaps is simple. Figures 3.8d - 3.8f show the included integration points for each sphere classification. Points which lie within the highlighted subdomains are included in the evaluation of integrals, whilst points which are within the integration quadrants, but outwith the highlighted subdomain, are not included in the evaluation of integrals. Note that due to the requirement to integrate complicated nonpolynomial functions over geometrically complex subdomains, a large number of integration points are required for the MFS [34]. Following the work of [34, 6], PyMFS therefore adopts  $6 \times 6 = 36$  integration points within each quadrant.

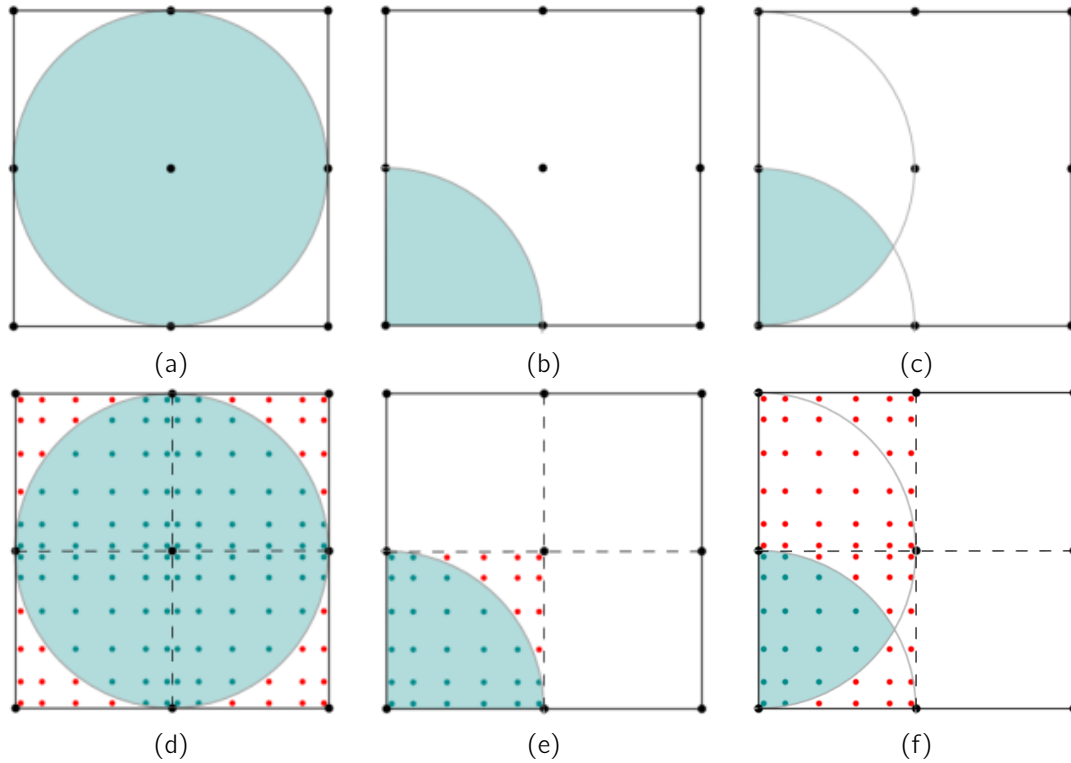


Figure 3.8: Illustration of sphere region definitions and their integration points: (a) interior sphere, (b) boundary sphere, (c) sphere overlap and the corresponding integration points associated with (d) interior sphere, (e) boundary sphere and (f) sphere overlap.



## Chapter 4

# Programming the method of finite spheres - PyMFS

This chapter details the implementation of the underlying theory behind the MFS (presented in Chapter 3) in a Python programming structure: PyMFS. PyMFS is a software application capable of developing, solving and post-processing Poisson equation problems in 1D and 2D, and elastostatics problems in 2D. The PyMFS project uses Git [43] for source code management and GitHub [44] for hosting, with the project currently hosted at the following repository:

### Link 1: PyMFS

<https://github.com/ThomasAston/PyMFS>

The software is developed with the aim of utilising Object-Oriented Programming (OOP) in Python [45] where possible, increasing modularity and subsequent efficiency of future extension of the code to a wider range of problems. The first official release of the code will be published following the publication of an accompanying journal article, for which a manuscript has been completed and is ready for submission to *Advances in Computational Mathematics* [46].

## 4.1 Structure

### 4.1.1 Overview

The broad structure of PyMFS is similar in nature to the typical structure of computer programs based on the FEM, illustrated in Figure 2.5. Figure 4.1 illustrates a flowchart of the general user workflow in PyMFS. With PyMFS the key stages of the solving process are handled using *classes*, between which *objects* containing *properties* are passed. It can be seen that the program is comprised of three core classes: `pre_process`, `solve` and `post_process`. A description of these classes and their primary purpose is given in Table 4.1.

Whilst similar to FEM solvers in a broad sense, PyMFS is built primarily for the purpose of solving problems with the MFS. It aims to make up for the current lack of efficiency of the solution

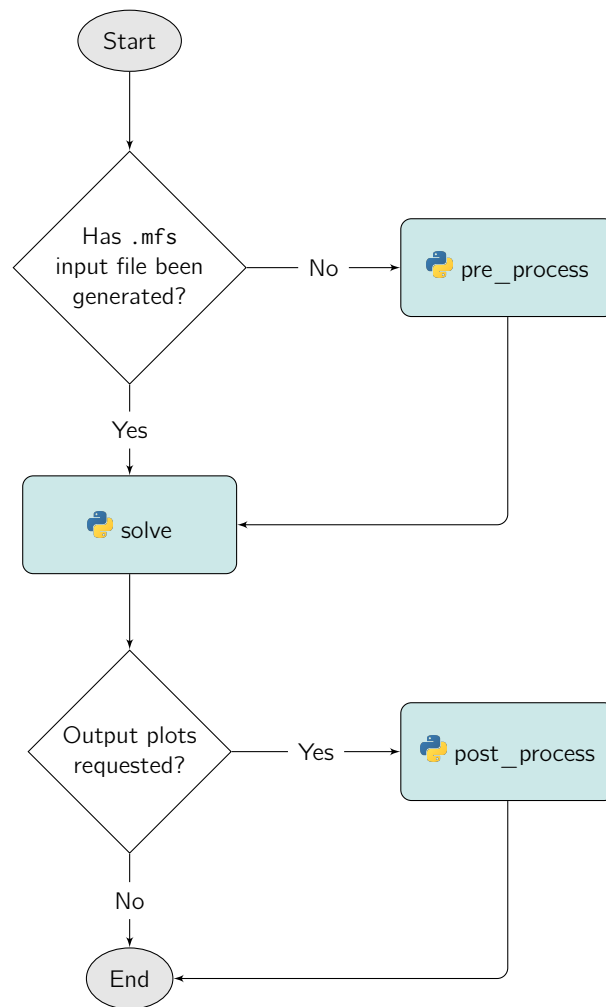


Figure 4.1: Flowchart outlining the user workflow for PyMFS.

Table 4.1: Description of PyMFS core classes.

Class	Description
pre_process	Optional class. Used to generate .mfs files to be sent to solve. Handles input geometry, generates user-interface, creates <code>input_data</code> object to be packaged in .mfs file.
solve	Mandatory class. Solves input .mfs files. Generates <code>solution</code> object which is inherited by <code>post_process</code> if included.
post_process	Optional class. Inherits <code>solution</code> from <code>solve</code> . Contains classes capable of post-processing <code>solution</code> object to produce physically meaningful results and output plots.

procedure (discussed in more detail in Chapter 5) of the method by focusing efforts on optimising the potential user benefits that a meshless method offers: removing time-consuming pre-processing stages and utilising the OOP approach to facilitate user-friendly customisation of process stages. The project also utilises a number of pre-existing Python libraries, which are listed alongside a



description of their purpose in PyMFS in Table 4.2.

Table 4.2: Description of existing Python libraries used in PyMFS.

Library	Description
numpy [47]	Provides rapid vectorised operations through the creation of NumPy arrays.
scipy [48]	For dealing with sparse matrices, solvers and algorithms.
sympy [49]	For handling basic symbolic maths operations.
matplotlib [50]	For producing high-quality, $\text{\LaTeX}$ compatible plots in pre and post-processing.
Shapely [51]	For basic geometry manipulation tasks.

### 4.1.2 Pre-processing

Preparation of domain geometry, definition of material properties and application of BCs and forces are handled by the `pre_process` class. The hierarchy of related classes is illustrated in Figure 4.2, which illustrates how objects containing user-defined properties related to domain geometry and nodal coordinates are passed to the `pre_process` class. A user-interface is then generated, allowing the user to apply BCs, loads and material properties, from which a file with the extension `.mfs` is compiled, containing all of the information that is required to be passed to the `solve` class. More detailed instruction on how this is performed by the user is provided in Section 4.2.

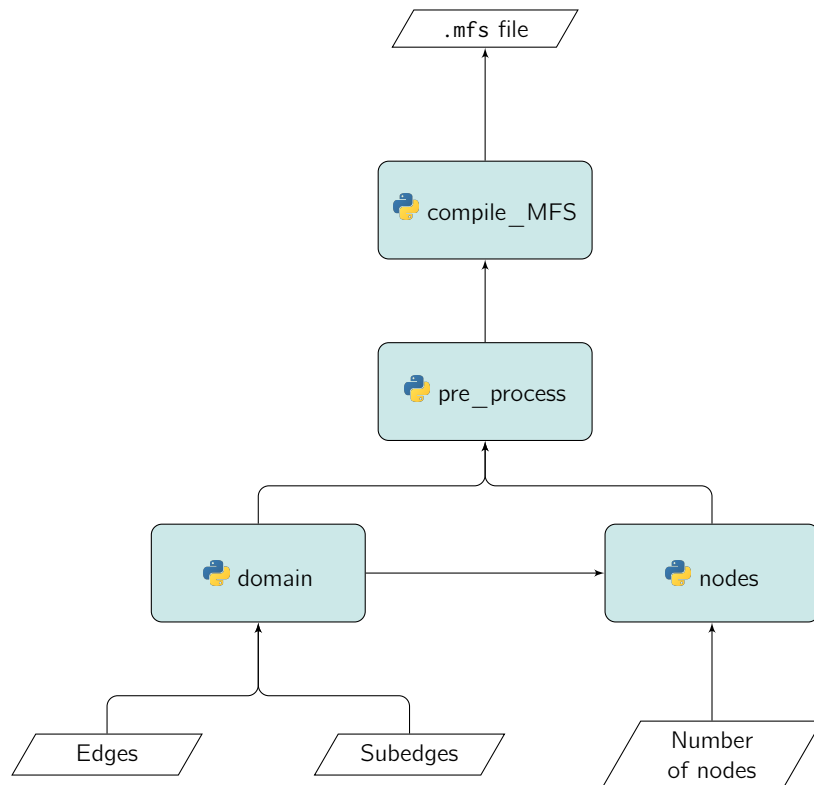


Figure 4.2: Hierarchy of classes related to `pre_process`.

### 4.1.3 Solve

Implementation of the core equations outlined in Chapter 3 is performed primarily using the `solve` class. A diagram illustrating the class hierarchy of related classes is again provided, here in Figure 4.3. It is shown here how the `.mfs solve` class inherits properties from the generation of the system variables to create an object possessing the properties which are inherited by the post-processing module. It is here that the OOP approach is of particular benefit: Python object serialisation can be performed using the `pickle` [52] module to rapidly save solution data *before* post-processing. More detail on this is provided in Section 4.2.

Figure 4.4 provides a flowchart visualisation of the broad structure of the algorithm which is implemented by the `system_variables` class for generation of the variables  $\underline{K}$  and  $\underline{f}$ , performed by looping over the list of nodes in the problem domain.

### 4.1.4 Post-processing

Post-processing of the results contained in the solution object produced by the `solve` class are handled by the `post_process` class. The hierarchy of related class is shown in Figure 4.5. It can be seen here that `post_process` inherits solution properties from `solution`, from which the user can then specify their desired output data and plots, which are produced by reassembling function fields from the obtained nodal DoFS. The current version of PyMFS allows the user to select from results for displacement, strain and stress. Again, further information on how this is carried out is provided in Section 4.2.

## 4.2 Usage

This section provides an introduction to the user-implementation of the components required to start solving problems with PyMFS: from an empty Python script to a full set of output results with minimal lines of code. The first stage of any analysis with PyMFS is to navigate to the directory containing the PyMFS installation, open a new Python script, and import the required components of PyMFS using:

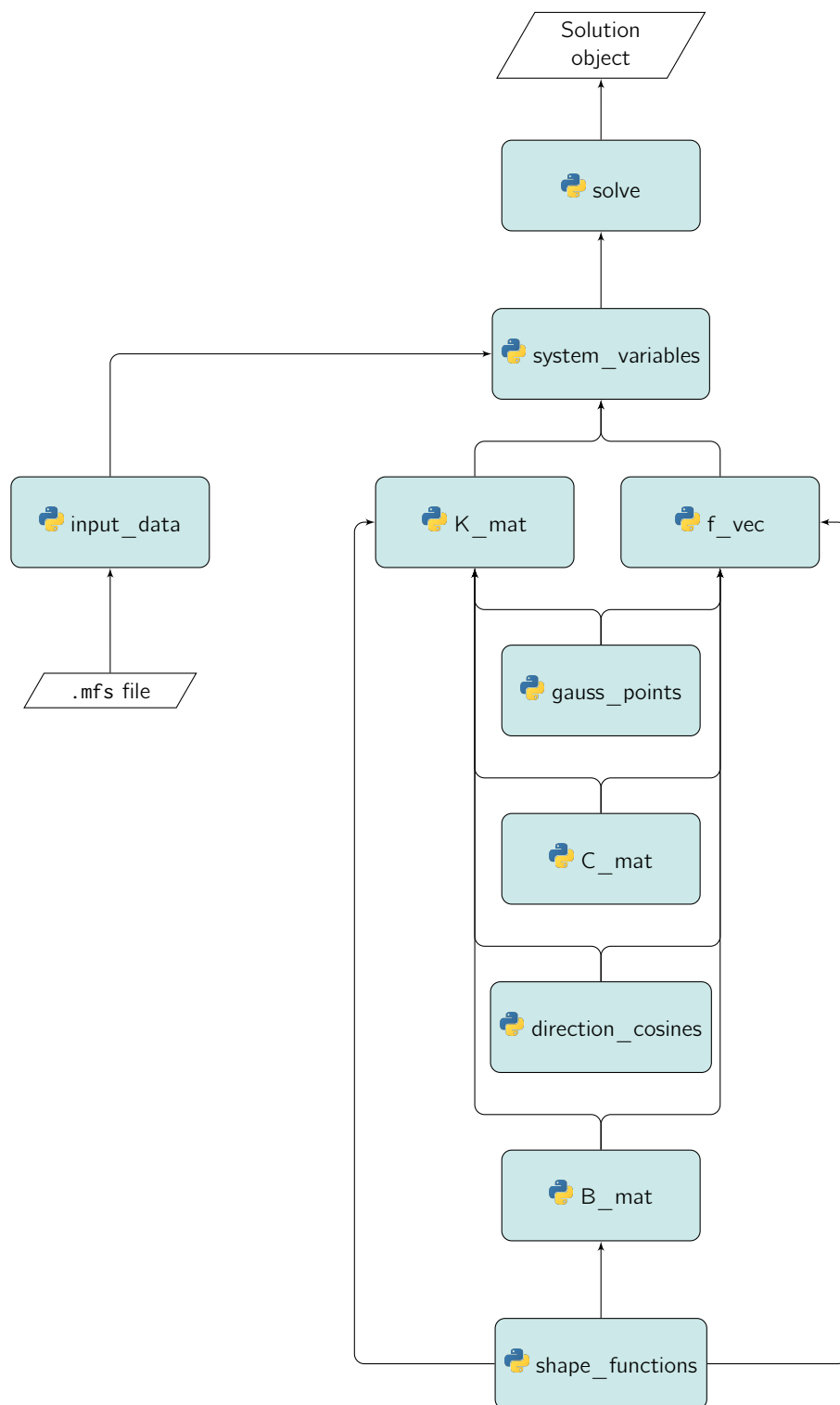
---

```
from PyMFS import *
```

---

### 4.2.1 Pre-processing

With PyMFS imported, it is possible to begin preparing the problem geometry. PyMFS is currently capable of handling geometry in 2D, which is prepared by compiling a list of external and internal surfaces with a set of functions, before compiling the surfaces in a single domain object with the `domain` class. For example, the code listing which follows produces the geometry in Figure 4.6.

Figure 4.3: Hierarchy of classes related to `solve`, including user input requirements.

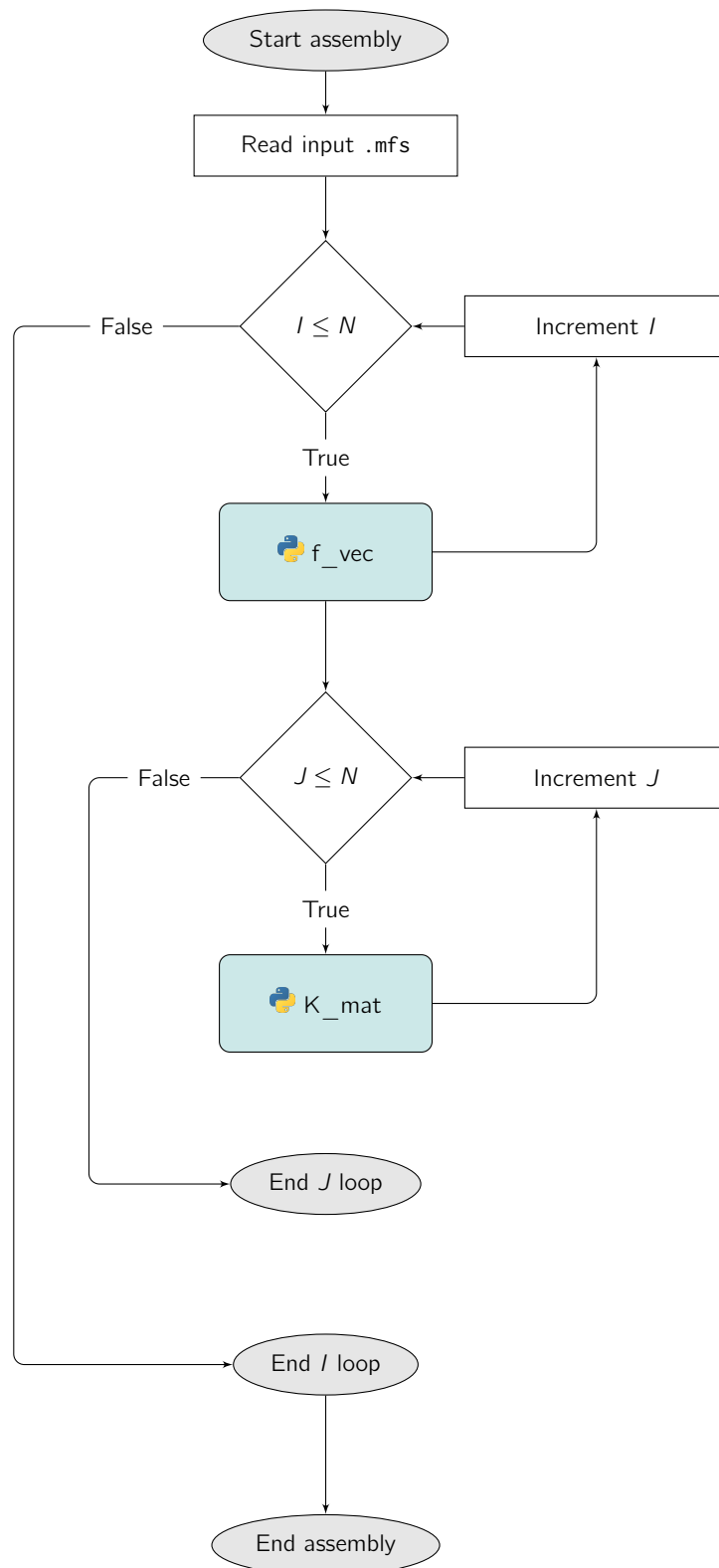


Figure 4.4: Flowchart illustrating system matrix assembly in PyMFS,  $N$  is the total number of nodes in the domain.  $\underline{f}$  is assembled in the `f_vec` class whilst looping over every node, whilst  $\underline{K}$  is assembled in the `K_mat` class whilst looping over every node *and* the influence of all nodes in the domain.

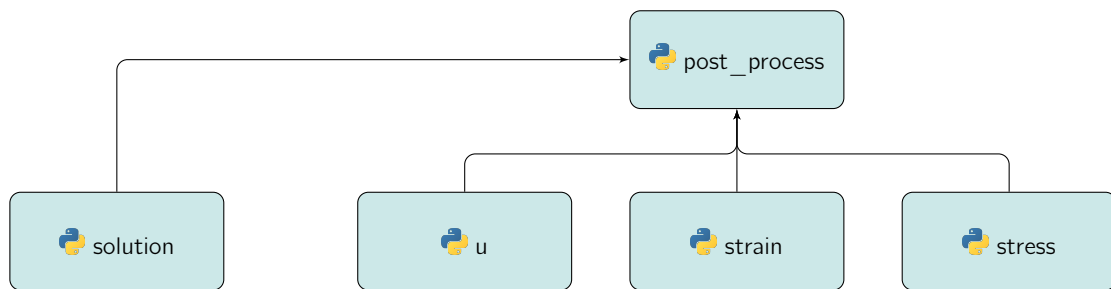


Figure 4.5: Hierarchy of classes related to `post_process`.

---

```

'''
External surfaces:
'''
ex_surface1 = straight_line(point1=[5, 0], point2=[10, 0])
ex_surface2 = circular_segment(center=[0,0], radius=10, start=0, end=np.pi/2)
ex_surface3 = straight_line(point1=[0, 10], point2=[0, 5])
ex_surface4 = circular_segment(center=np.array[0,0], radius=5, start=np.pi/2, end=0)
ex_surfaces = [ex_surface1, ex_surface2, ex_surface3, ex_surface4]
'''

Internal surfaces:
'''
in_surface1 = circle(center=[7,2],radius=1)
in_surface2 = circle(center=np.array[2,7],radius=1)
in_surfaces = [in_surface1, in_surface2]
'''

Generate a domain object from the surfaces:
'''
my_domain = domain(ex_surfaces, in_surfaces)

```

---

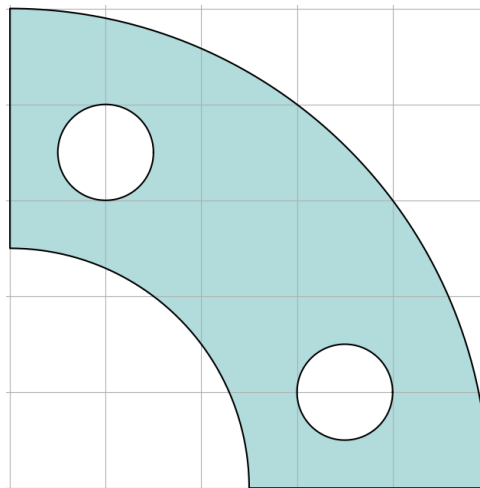


Figure 4.6: Example domain generated with PyMFS.

With the geometry defined, the domain object can then be passed to the `nodes` class to generate an object containing nodal coordinate properties around which sphere elements are centred, based on user inputs on the desired number of nodes along the  $x$  and  $y$  directions. The `nodes` class also contains the option to input the nodal distribution method. The below code generates the domain discretisation in Figure 4.7.

---

```

'''
Discretise the domain by selecting number of nodes.
'''
my_nodes = nodes(my_domain, nx=10, ny=10, method='Regular')

```

---

The final stage of pre-processing is to define BCs and forcing, which can be performed by passing the domain and node objects to the `pre_process` class. Here, the user should also define a 'job-ID', which will be used as the file name for the `.mfs` input file generated upon job submission. The following code snippet prompts the opening of the UI window shown in Figure 4.8.

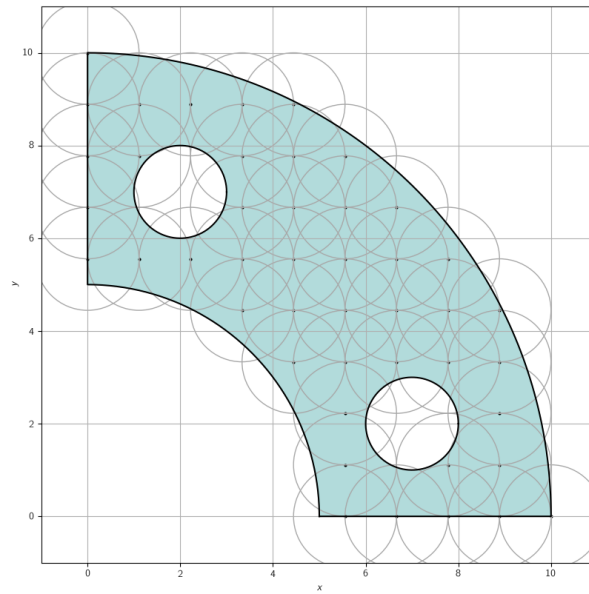


Figure 4.7: Example domain discretisation in PyMFS.

```
'''
Enter the pre-processing UI to view geometry, set boundary conditions and
submit the job for solving:
'''
pre_process(my_domain, my_nodes, job_ID='Example')
```

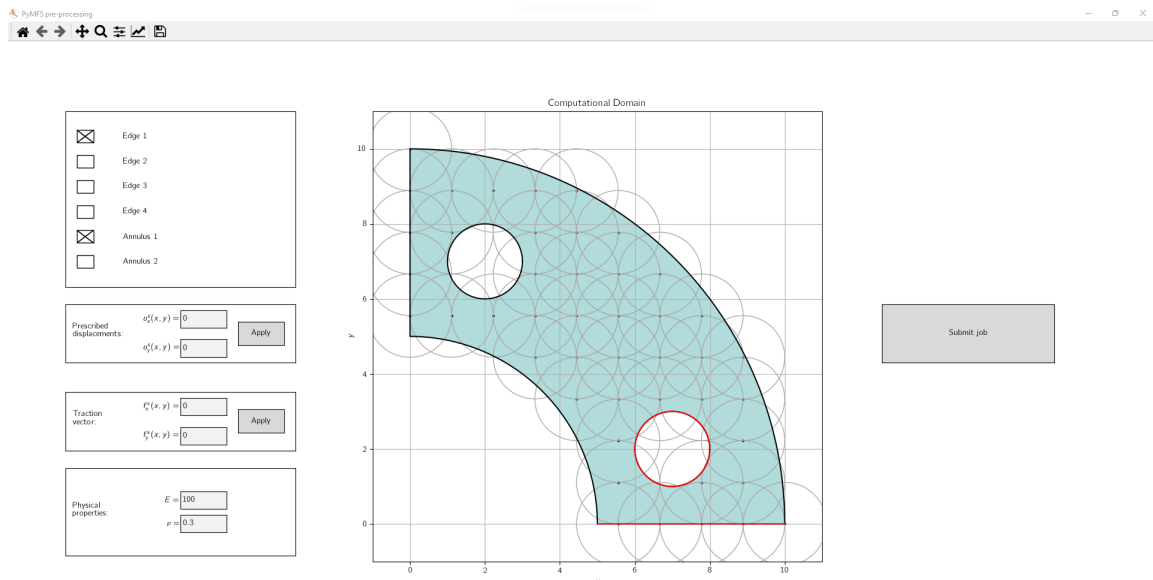


Figure 4.8: Example pre-processing UI in PyMFS.

After defining BCs and loads, the user selects the 'Submit job' button, upon which all pre-processing properties are compiled and saved in a `.mfs` file, which can be opened and edited in a text file editor

of choice. This allows efficient editing of jobs for which small changes are desired before solving. Appendix F provides more detail on the .mfs file format and editing procedure.

### 4.2.2 Solve

With a complete problem definition contained within the generated input file, it is possible to solve for the unknown DoFs. In PyMFS this is simply done by sending the .mfs file to the solve class as follows:

---

```
'''
Select the input file to be solved and send it to PyMFS solver:
'''
solution = solve(job_ID='Example.mfs')
```

---

It is recommended that the resulting solution object is saved to a file of its own using pickle, which can be performed with the following code:

---

```
'''
Recommended, dump solution object into .sol file for later use.:
'''
import pickle
filehandler = open("Example.sol", 'wb')
pickle.dump(solution, filehandler)
```

---

The saved file can then be loaded as a solution object using:

---

```
'''
Open existing .sol file and load solution object:
'''
import pickle
filehandler = open("Example.sol", 'rb')
solution = pickle.load(filehandler)
```

---

### 4.2.3 Post-processing

Once a solution object is obtained, it can be sent to the post\_process class for post-processing. The user can also specify desired output contour plots and data files as shown in the below code.

---

```
'''
Pass solution object to post-processing.
'''
post_process(solution)
post_process(solution).u()      # Displacement contours
post_process(solution).strain() # Strain contours
post_process(solution).stress() # Stress contours
```

---

Raw output data files are saved in .csv format, whilst contour plots of the data are generated using matplotlib, utilising its compatibility with L<sup>A</sup>T<sub>E</sub>X to produce high-quality plots suitable for direct use in scientific and technical reports.



## Chapter 5

# Validation cases

In this chapter, a series of example problems are considered for validation of the accuracy of the PyMFS solver. The PyMFS download available at [Link 1](#) contains all of the .mfs files required to run the problems presented here.

### 5.1 Poisson's equation

This section briefly presents the suitability of PyMFS for solving Poisson's equation problems for cases in 1D and 2D. The problems presented here are solved using the MFS in [5], and thus detailed analysis is reserved for when differences between the results presented here and those from [5] arise.

#### 5.1.1 1D

The first problem considered is Poisson's equation on a 1D domain (Equation 3.14), in which a bar is subjected to a distributed axial load. The load  $f(x)$ , and natural and essential boundary conditions  $f^s$  and  $u^s$  respectively, are selected as:

$$f(x) = x, \quad f^s = \left[ \frac{du}{dx} \right]_{x=1} = 2, \quad u^s = u(0) = 1,$$

such that the analytical solution to Equation 3.14 is:

$$\hat{u}(x) = \frac{1}{2} \left( x - \frac{x^3}{3} \right) + 2x + 1.$$

In Figure 5.1 it is illustrated that for regular arrangements of  $N = 3$  and  $N = 7$  nodes along the domain with radii (in 1D, line segments) equal to the distance between nodes, it is possible to exactly satisfy the analytical solution, which is consistent with the findings of [5].

#### 5.1.2 2D

The first example considered in 2D is another Poisson's equation problem, here posed over a square domain with mixed BCs, as illustrated in Figure 5.2.

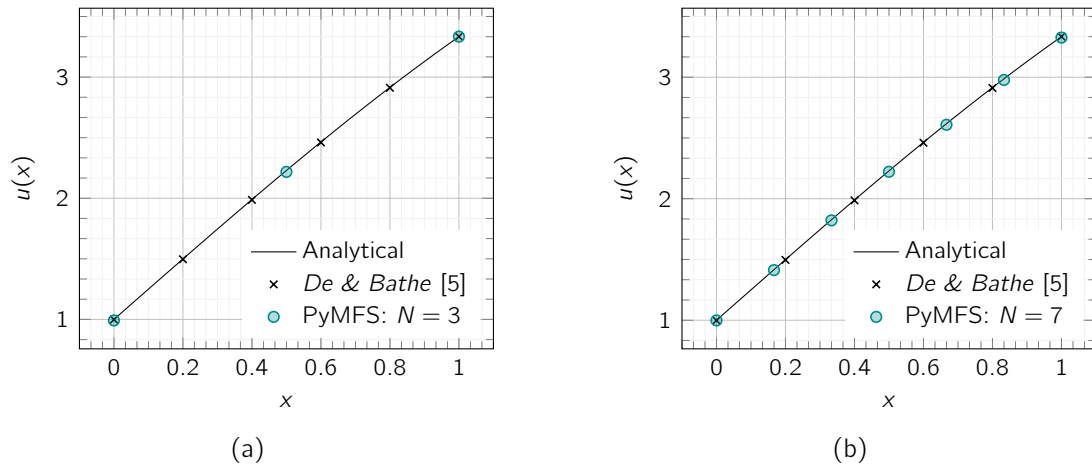


Figure 5.1: Analytical solution to Poisson's equation in 1D, compared with regular distribution of 6 1D sphere elements as presented in [5], as well as solutions obtained with PyMFS using (a) 3 sphere elements and (b) 7 sphere elements.

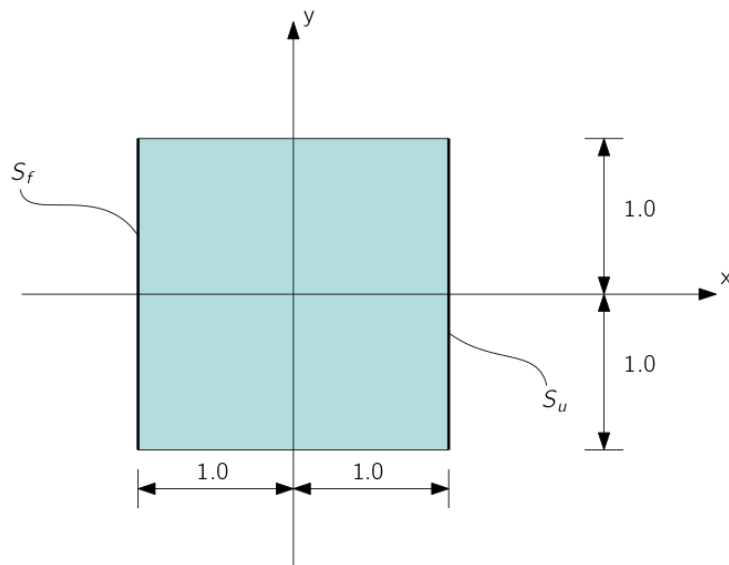


Figure 5.2: 2D square domain on which Poisson's equation is solved. Neumann BC applied to surface  $S_f$  at  $x = -1$  and Dirichlet BC applied to surface  $S_u$  at  $x = 1$ .

The domain forcing  $f(x, y)$ , and natural and essential boundary conditions  $f^s$  and  $u^s$  respectively, are selected as:

$$f(x, y) = [\pi^2 (7x + x^7) - 42x^5] \cos \pi y, \quad f^s(x = -1, y) = 14 \cos \pi y, \quad u^s(x = 1, y) = 8 \cos \pi y$$

such that the analytical solution  $\hat{u}(x, y)$  to Equation 3.15 is:

$$\hat{u}(x, y) = (7x + x^7) \cos \pi y.$$

Figure 5.3a provides a 3D surface representation of the analytical solution across the domain. The problem is solved using the PyMFS framework for two domain discretisations: regular arrangements of  $N = 3 \times 3 = 9$  nodes and  $N = 6 \times 6 = 36$  nodes over the domain. Figures 5.4a and 5.4b illustrate 3D surface representations of the solution  $u(x, y)$  obtained using the respective domain discretisations, for which Figure 5.4 presents contour plots of absolute error in the obtained solution,  $\hat{u}(x, y) - u(x, y)$ .

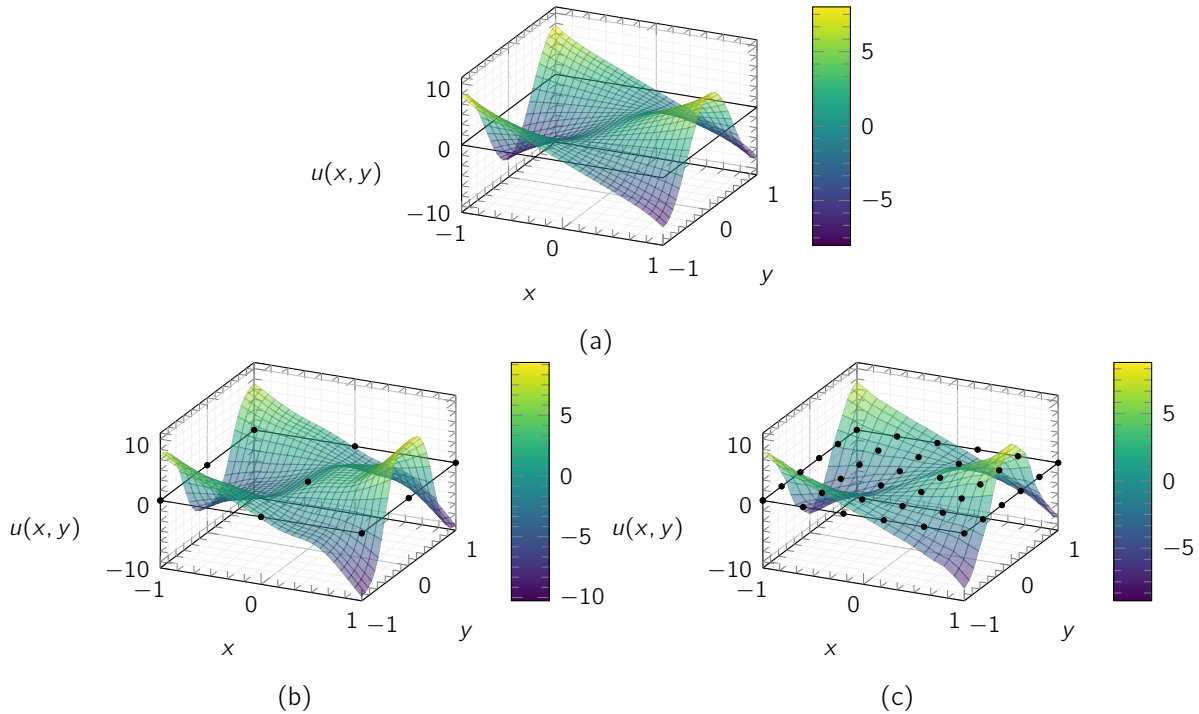


Figure 5.3: Contour plot of  $u(x, y)$  obtained by solving Poisson's equation over square domain: (a) analytically, (b) using  $3 \times 3 = 9$  sphere elements and (c) using  $6 \times 6 = 36$  sphere elements.

Figure 5.5 presents plots of the analytical solution and solution obtained using PyMFS for 36 nodes along a series of lines over the solution domain, including results from [5]. It is observed that the solution obtained using PyMFS is weakest at  $x = 1$ , which corresponds with surface upon which the Dirichlet BC is applied. Unlike in [5], PyMFS is here implemented without exploiting the special nodal arrangement presented in Section 3.4: the rows and columns of the global system matrices associated with the Kronecker-delta appeasing DoFs are not deleted, and thus there are additional numerical errors present in the solution at the Dirichlet boundary due to the inclusion of numerically integrated quantities  $KU_{ImJn}$  and  $fU_{Im}$ . The implications of adopting the specialised arrangement are discussed in more detail in the following sections.

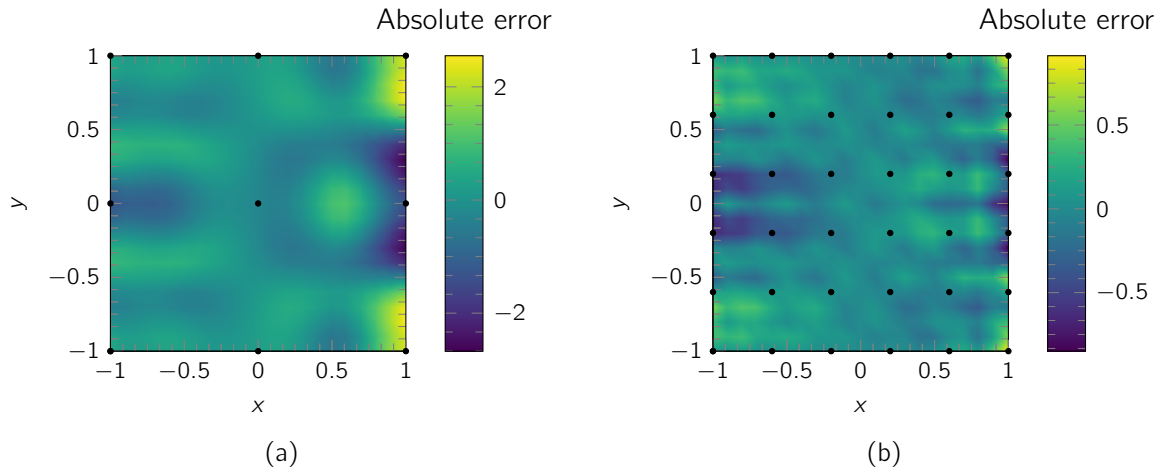


Figure 5.4: Contour plots illustrating the absolute error in the solutions to Poisson's equation  $u(x, y)$  obtained using (a) 9 and (b) 36 nodes.

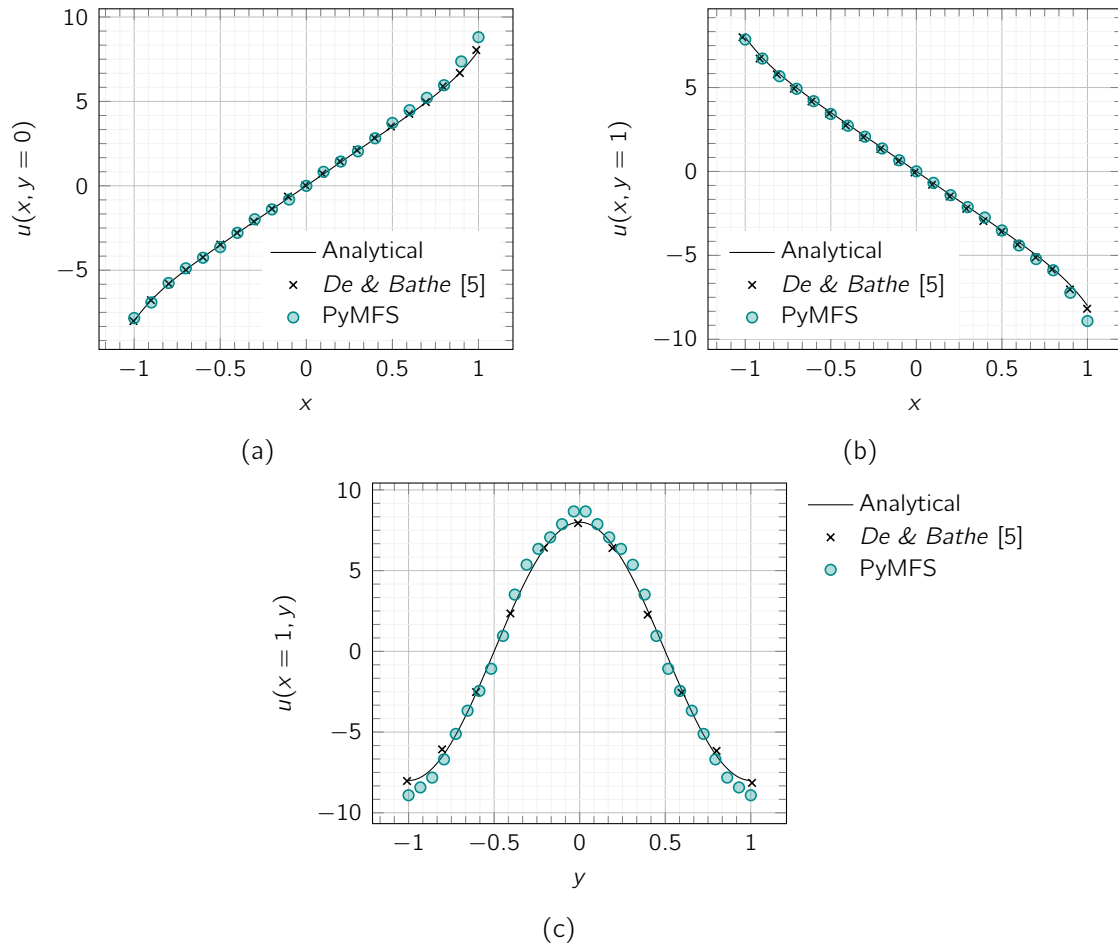


Figure 5.5: Comparison between analytical solution to 2D Poisson's equation with results from [5] and PyMFS using regular arrangement of 36 nodes along slice of domain at: (a)  $y = 0$  (b)  $y = 1$  and (c)  $x = 1$ .

## 5.2 2D elastostatics

The use of PyMFS for solving solid mechanics problems in 2D elastostatics is introduced. The special arrangement of nodes along Dirichlet boundaries is considered, and the results of PyMFS are compared with those obtained using the FEM.

### 5.2.1 Case 1: tension

The first problem considered in the category of 2D elastostatics is that of a square plate, fixed at one end and subjected to a tensile load, as illustrated by Figure 5.6. Relevant material and domain properties are given in Table 5.1. The domain discretisations used to solve the problem using PyMFS are shown in Figure 5.7, for which nodes are equispaced across the domain with equal sphere radius in all cases.

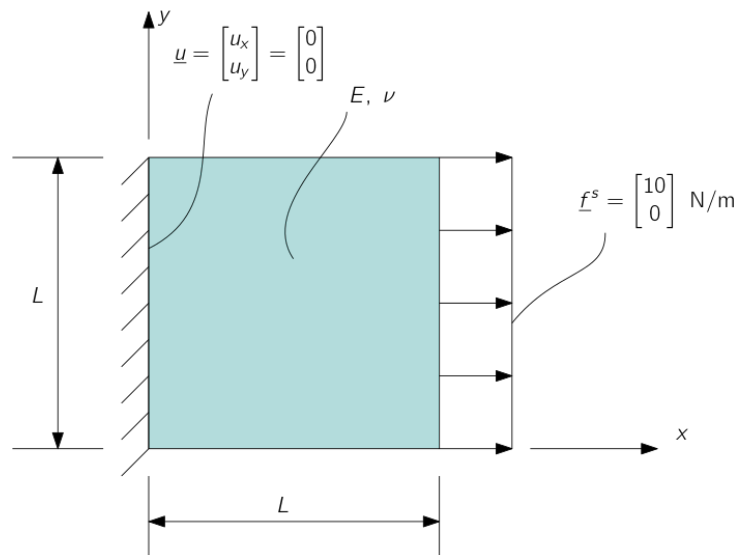


Figure 5.6: Problem definition for case 1.

Table 5.1: Parameters used in study of case 1.

Parameter	Symbol	Value	Unit
Young's modulus	$E$	100	$\text{N/m}^2$
Poisson's ratio	$\nu$	0	-
Length	$L$	2	m

Figure 5.8 shows contour plots of the assembled displacement field in the  $x$  direction,  $u_x$ , obtained with PyMFS discretisations D1-D4, whilst Figure 5.9 similarly shows contour plots for  $u_y$ . Figures 5.10a and 5.10b illustrate the analytical solution for  $u_x$  along two lines over the domain, at  $x = 0$  m and  $y = 2$  m, respectively. Plots of the solutions obtained using D1-D4 are also presented, in which the special nodal arrangement has once again been neglected. Figure 5.10b, however, includes plots of the obtained solution where the exploitation of the Kronecker-delta property has been applied for D2 and D3, signified with a \*.

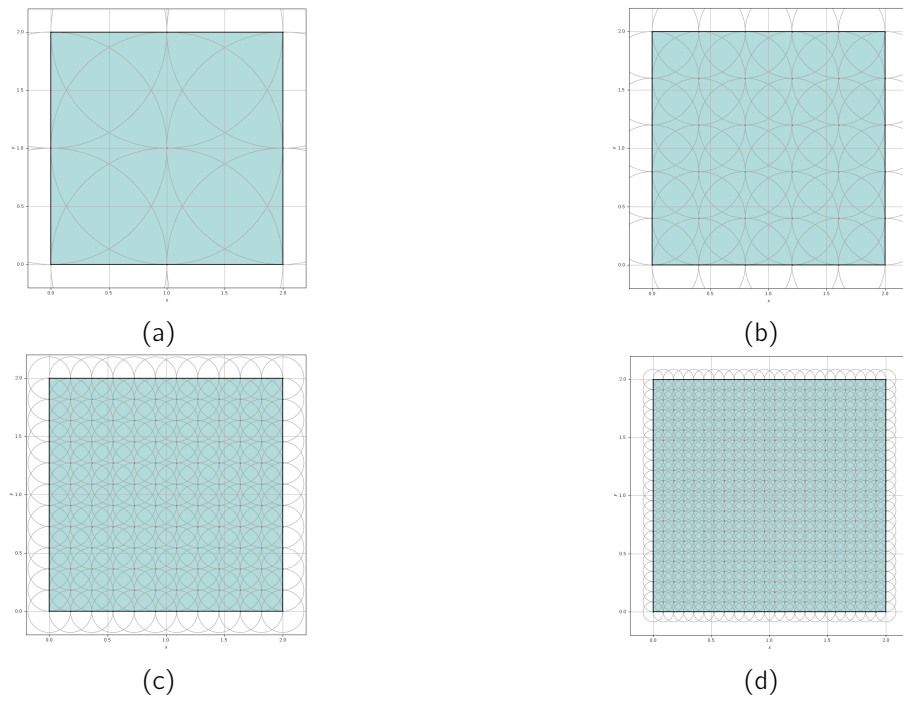


Figure 5.7: Sphere element discretisations used for case 1: (a) D1, (b) D2, (c) D3 and (d) D4.

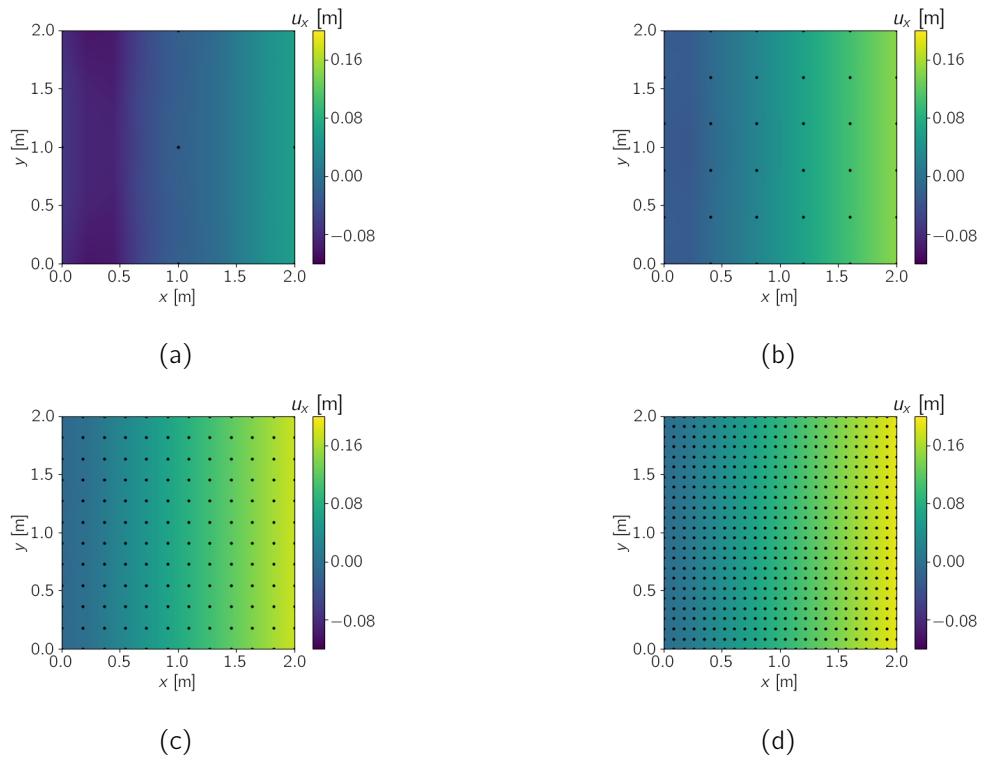


Figure 5.8: Contour plots of displacement field in the  $x$  direction for: (a) D1, (b) D2, (c) D3 and (d) D4.

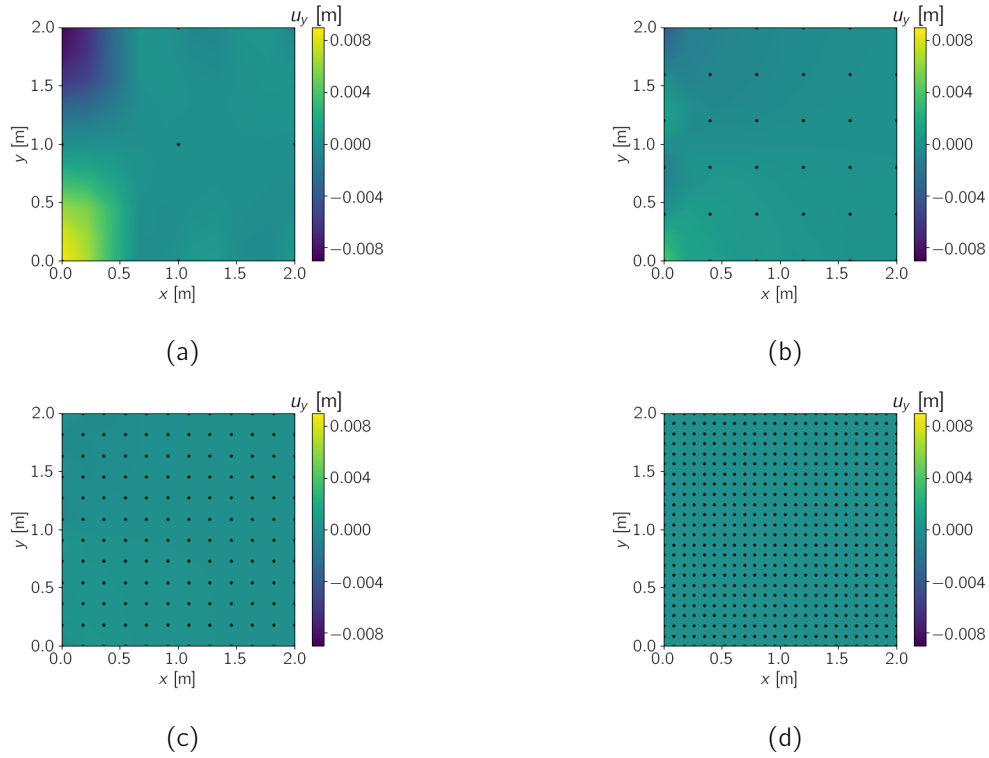


Figure 5.9: Contour plots of displacement field in the  $y$  direction for: (a) D1, (b) D2, (c) D3 and (d) D4.

From Figures 5.8, 5.9 and 5.10a, it is once again clear that whilst increasing the number of nodes results in convergence towards the analytical result, solutions obtained using PyMFS still fail to exactly satisfy the conditions prescribed by the Dirichlet BC,  $u_x(x=0, y)$ ,  $u_y(x=0, y) \neq 0$ . In Figure 5.10b, however, it can be seen that by exploiting the properties of the special nodal arrangement presented in Section 3.4, the prescribed displacements at the Dirichlet boundary are satisfied exactly. Table 5.2 presents the root-mean-square error,  $\text{RMSE}(u_x)$ , for each of the PyMFS solutions  $u_x$ , calculated with  $n = 10$  equispaced points along the line  $y = 2$  using the equation:

$$\text{RMSE}(u_x) = \sqrt{\frac{\sum_{i=1}^n (u_x - \hat{u}_x)^2}{n}} \quad (5.1)$$

where  $\hat{u}_x$  is the analytical value of the  $x$ -displacement at a given point. RMSE is a simple measure which has been shown to be effective in the accuracy analysis of meshless numerical methods [53].

Convergence of RMSE is presented in Figure 5.11, for which an order of convergence (slope) of approximately 1.05 is observed. This is considerably lower than [5], in which an order of convergence of 4 is observed (using a complete *second-order* polynomial approximation space, unlike the first-order approximation space used here), and [6] where an order of convergence of approximately 1.92 was observed.

From Table 5.2 it is observed that by exploiting the regular arrangement of nodes on the Dirichlet BC a greater solution accuracy is achieved. For example, for D3\*,  $\text{RMSE}(u_x) = 0.010$ , whilst for the equivalent discretisation D3,  $\text{RMSE}(u_x) = 0.025$ . For the remainder of this chapter the

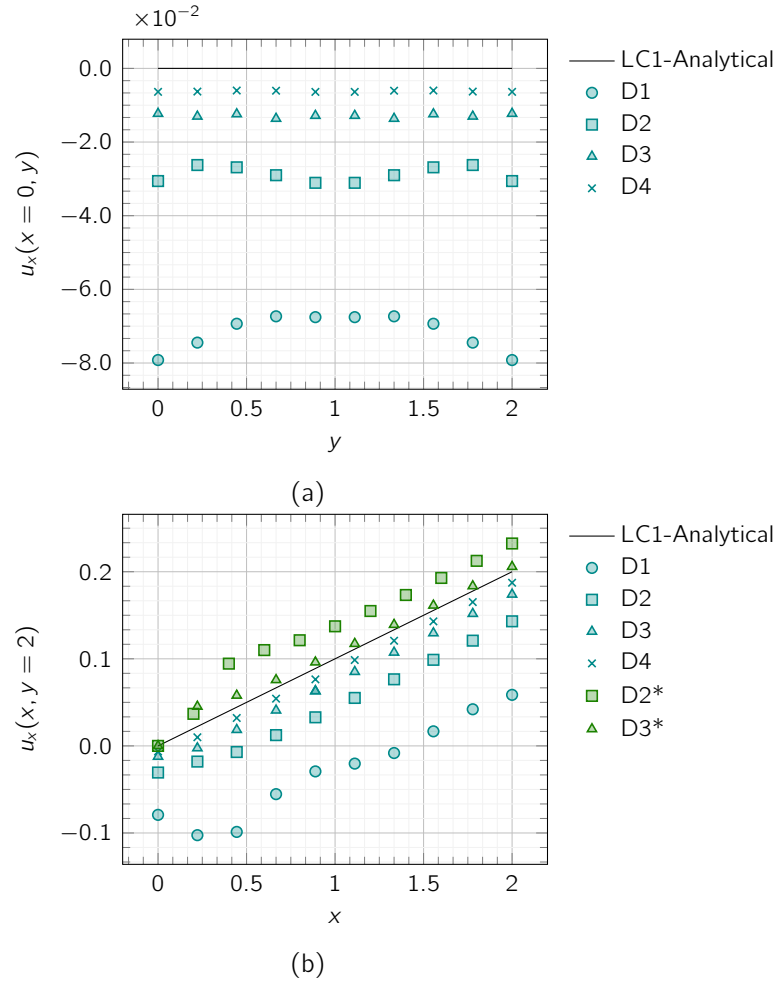


Figure 5.10: Case 1 analytical solution and solutions using PyMFS discretisations,  $u_x$ , along lines: (a)  $x = 0$  and (b)  $y = 2$  for case 1. Note, \* denotes instances where the special nodal arrangement is exploited.

Table 5.2: Case 1 RMSE of solutions obtained with discretisations D1-D4, and discretisations D2\* and D3\* which exploit a special nodal arrangement.

Discretisation	Nodes	DoFs	RMSE	Units
D1	9	72	0.129	m
D2	36	2,592	0.055	m
D3	144	1,152	0.025	m
D4	576	4,608	0.012	m
D2*	36	2,592	0.036	m
D3*	144	1,152	0.010	m

proposed special nodal arrangement along Dirichlet BCs is implemented in the `solve` class of PyMFS for greater accuracy.



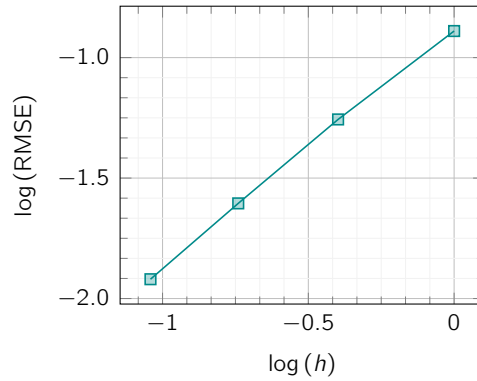


Figure 5.11: Case 1 convergence of RMSE for D1-D4.

### 5.2.2 Case 2: cantilever bending

The accuracy of PyMFS is now considered for the case of a cantilever bending problem under 2D plane stress conditions, subject to the conditions presented in Figure 5.12, where the relevant parameters are now listed in Table 5.3. The domain discretisations used with PyMFS remain those from Figure 5.7, but as discussed the `solve` class now deletes the rows and columns of the system matrices associated with DoFs which satisfy the Kronecker-delta property.

Results from PyMFS are here compared with those obtained using the FEM, for which the meshes FEM1-FEM4 used in the analysis contain an equivalent number of nodes to D1-D4, as shown in Figure 5.13. 4-node, bilinear plane stress quadrilateral elements with full numerical integration are used for the FEM solutions, such that each element has an equal number of DoFs as the MFS sphere elements. Monotonic convergence of results is expected due to the strategy of mesh subdivision [13]. The limit solution is obtained by solving the same problem with a FEM mesh of 100x100 elements.

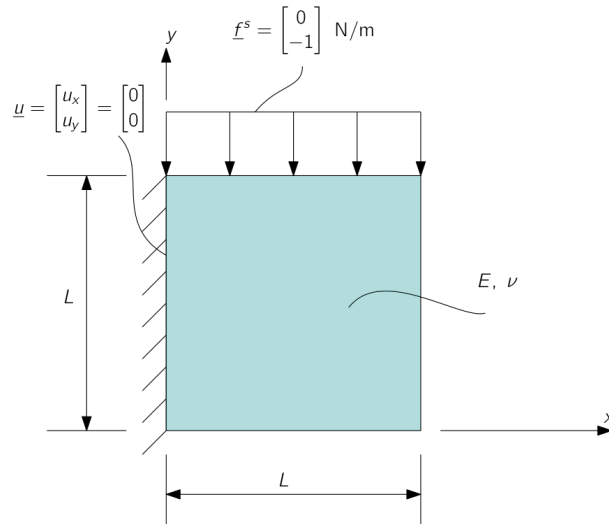


Figure 5.12: Problem definition for case 2.

Figure 5.14 shows the deformed shapes of the beam obtained using the MFS, superimposed on the deformed shape of the corresponding FEM solutions, and Figure 5.15 illustrates the contours

Table 5.3: Parameters used in study of case 2.

Parameter	Symbol	Value	Unit
Young's modulus	$E$	100	N/m <sup>2</sup>
Poisson's ratio	$\nu$	0	-
Length	$L$	2	m

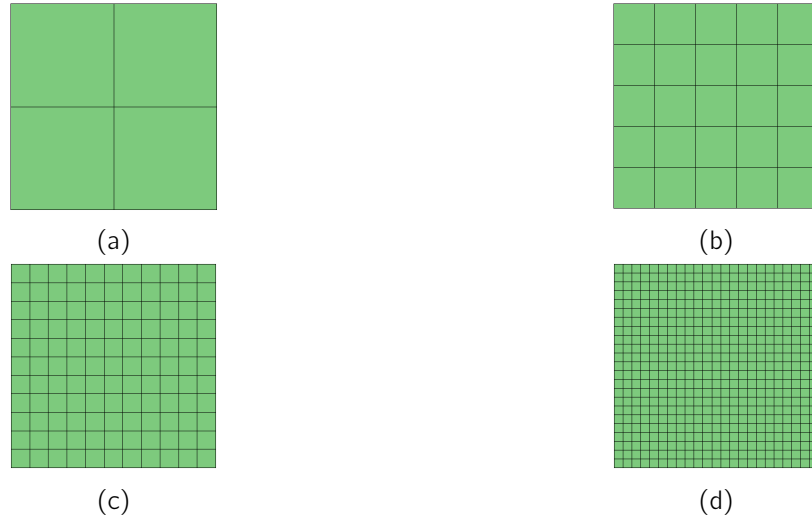


Figure 5.13: FEM discretisations used for case 2: (a) FEM1, (b) FEM2, (c) FEM3 and (d) FEM4.

of  $u_y$  for the solution using discretisation D4. Table 5.4 presents the values of  $\text{RMSE}(u_y)$  for each of the solutions, where the FEM-Limit solution is used as the zero-error reference. This table also gives the CPU time taken to obtain each solution, presented as multiples of the CPU time for the FEM limit solution.

Table 5.4: Case 2 RMSE and time multiplier, using the FEM-Limit solution as reference.

Discretisation	Nodes	DoFs	RMSE	Unit	Time multiplier
D1	9	72	0.5864	m	3.56
D2	36	2,592	0.0511	m	13.04
D3	144	1,152	0.0117	m	108.31
D4	576	4,608	0.0038	m	1650.50
FEM1	9	18	0.0070	m	0.10
FEM2	36	72	0.0023	m	0.10
FEM3	144	288	0.0007	m	0.11
FEM4	576	1,152	0.0005	m	0.21
FEM-Limit	10,000	20,000	-	m	- <sup>a</sup>

<sup>a</sup>FEM-Limit, CPU time = 0.8s

From Table 5.4, it can be seen that for PyMFS to achieve a level accuracy with the same order of magnitude as the FEM (where Abaqus/Standard has been used for all FEM models), it requires a CPU time which is four orders of magnitude greater. This performance in terms of efficiency

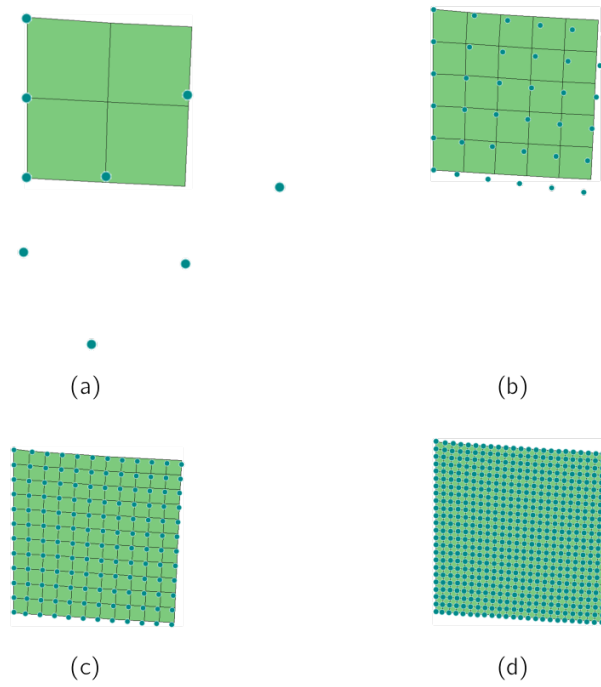


Figure 5.14: Deformed shapes obtained using: (a) D1 and FEM1, (b) D2 and FEM2, (c) D3 and FEM3 and (d) D4 and FEM4.

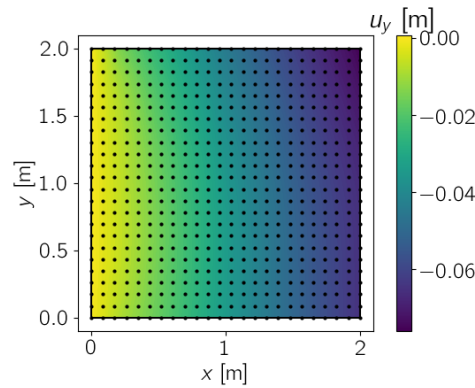


Figure 5.15: Contours of  $u_y$  for solution to case 2 using D4.

is significantly worse than that achieved by [6], who found the MFS to be around one order of magnitude slower for 3D elastostatics when implementing the MFS as a user-defined element subroutine within the ADINA program. This indicates that there are significant efficiency gains to be made in the PyMFS algorithm. As a result of the large number of integration points per element (see Section 3.5), the largest contribution to computational time comes from the system matrix assembly, due to the expense of looping over every integration point. Going forward, efficiency could therefore be improved by implementing a strategy in which calculations for identical sphere elements are only performed once, as applied by [6]. Another potential solution which requires investigation is to utilise parallel processing: calculation of the contributions from multiple sphere elements can be done in parallel by assigning individual calculations to different CPU cores.

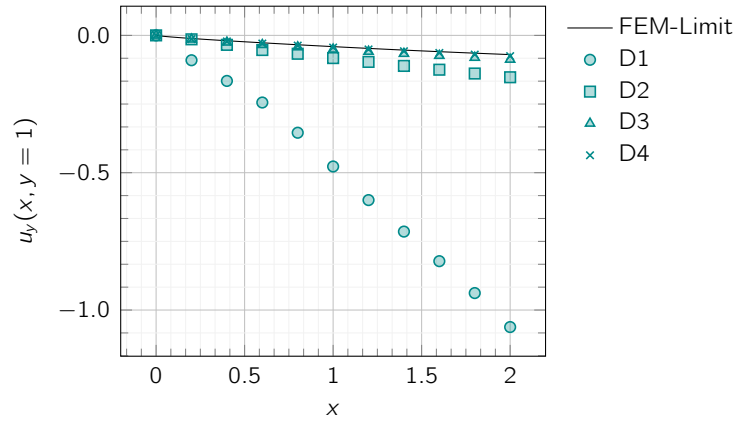


Figure 5.16: Case 2 FEM-Limit solution and solutions obtained using MFS discretisations along beam mid-span,  $u_y(x, y = 1)$ .

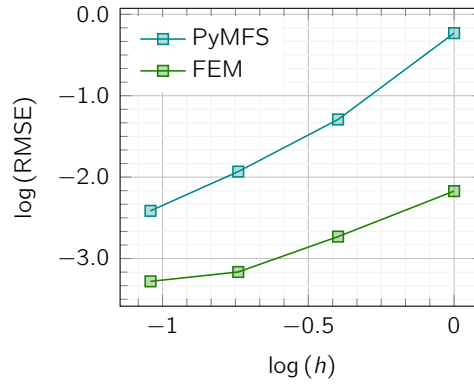


Figure 5.17: Case 2 convergence of RMSE for D1-D4 and FEM1-FEM4.

From Figure 5.17, it is determined that PyMFS now has an order of convergence of around 2.09, whilst the FEM shows, as expected, a lower order of convergence of 1.05. These results align more closely to those of [6], where an order of convergence of 1.92 was observed for the MFS in 3D elastostatics, highlighting the improved performance of the method when exploiting the Dirichlet BC nodal arrangement.

### 5.2.3 Case 3: plate with a hole

The final case deals with a rectangular plate with a hole subject to compressive loading, as shown in Figure 5.18, in order to briefly illustrate the suitability of PyMFS when dealing with problems where geometrically complex domains are considered. Geometry and material parameters used for case 3 are listed in Table 5.5. The PyMFS discretisations used are shown in Figure 5.19, whilst the FEM limit solution was obtained with a mesh of 4,355 elements following a suitable mesh convergence analysis.

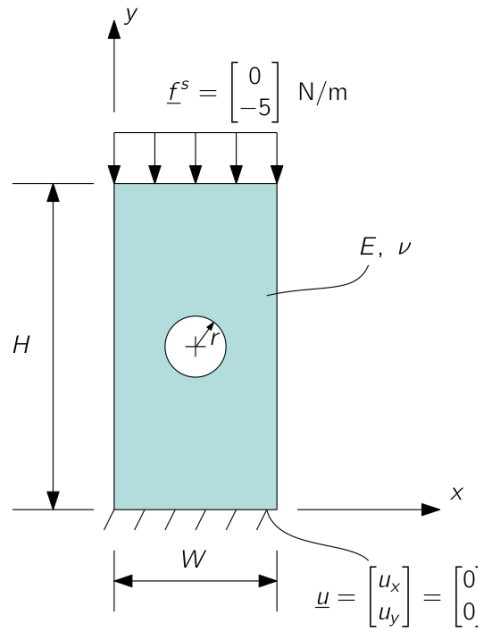


Figure 5.18: Problem definition for case 3.

Table 5.5: Parameters used in study of case 1.

Parameter	Symbol	Value	Unit
Young's modulus	$E$	70	N/m <sup>2</sup>
Poisson's ratio	$\nu$	0.3	-
Width	$W$	2	m
Height	$H$	4	m

Figure 5.20 shows the PyMFS deformed shapes superimposed on the deformed shape of the FEM limit solution, providing a visualisation of solution convergence. Figure 5.21 illustrates contours of  $u_x$  and  $u_y$  as obtained using D4. Plots of  $u_x(x, y = 2)$ ,  $u_y(x = 1, y)$  and principal stress component  $\sigma_{yy}(x = 1, y)$  are given in Figure 5.22, where dashed lines represent the hole edges. Table 5.6 presents the values of  $\sigma_{yy}$  at the top and bottom edges of the plate along the line  $x = 1$ ,  $\sigma_{yy}^t$  and  $\sigma_{yy}^b$  respectively, as well as the percentage difference between the values obtained using PyMFS and those from the FEM limit.

From Figure 5.22, convergence of quantities  $u_x(x, y = 2)$ ,  $u_y(x = 1, y)$  and  $\sigma_{yy}(x = 1, y)$  towards the FEM limit solution with each PyMFS discretisation is clear. In Table 5.6 it is shown that with less than half of the number of DoFs of the FEM limit solution, percentage errors of just 1.1% and -2.1% for  $\sigma_{yy}^t$  and  $\sigma_{yy}^b$  respectively are obtained with PyMFS D4. This indicates the suitability

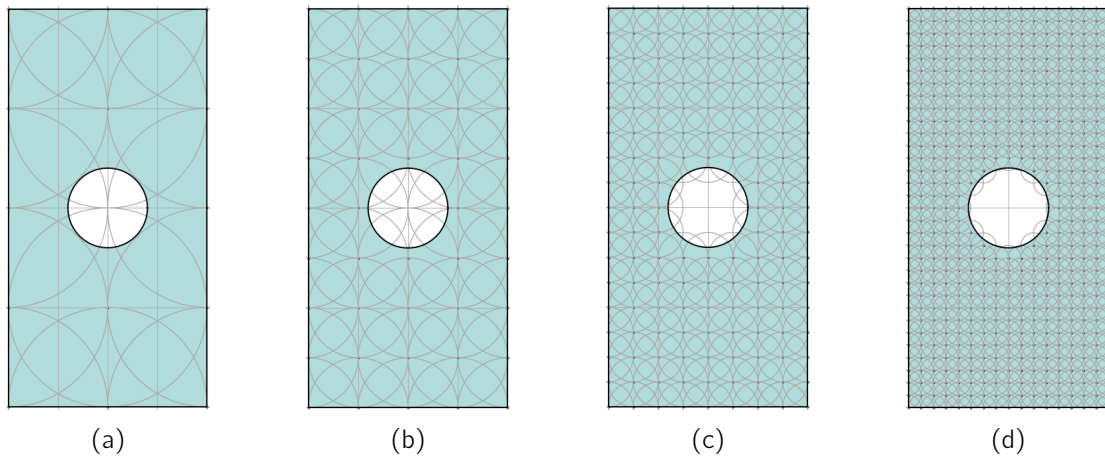


Figure 5.19: Sphere element discretisations used for case 3: (a) D1, (b) D2, (c) D3 and (d) D4.

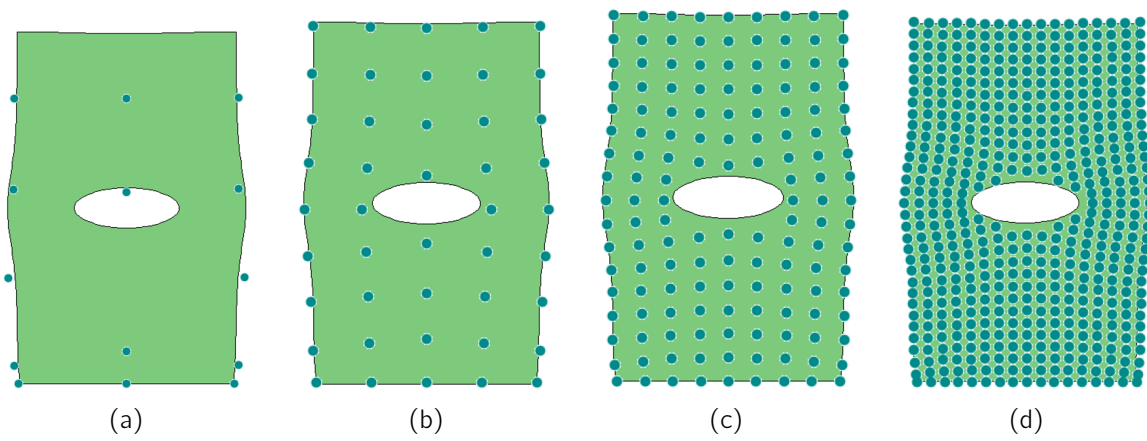


Figure 5.20: Deformed shapes obtained using the MFS for case 3 superimposed on FEM limit solution, shown for (a) D1, (b) D2, (c) D3 and (d) D4.

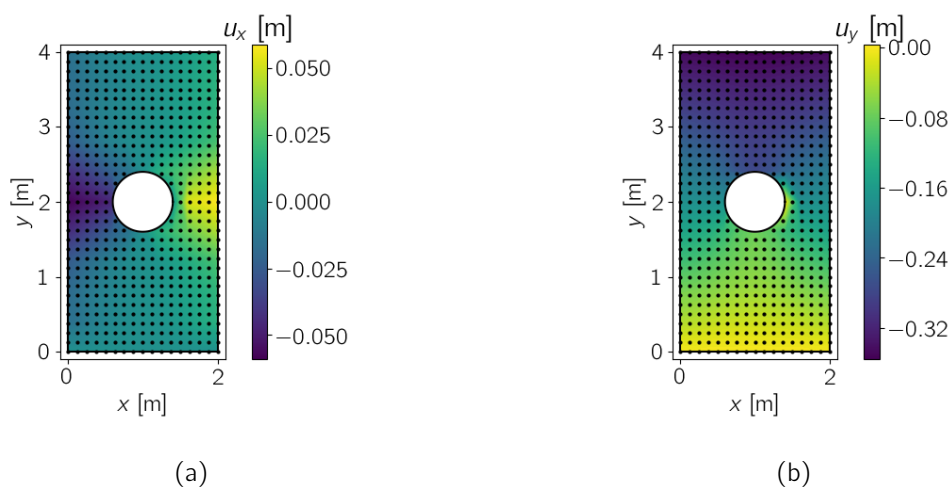


Figure 5.21: Contour plots for case 3, D4 of (a)  $u_x$  and (b)  $u_y$ .

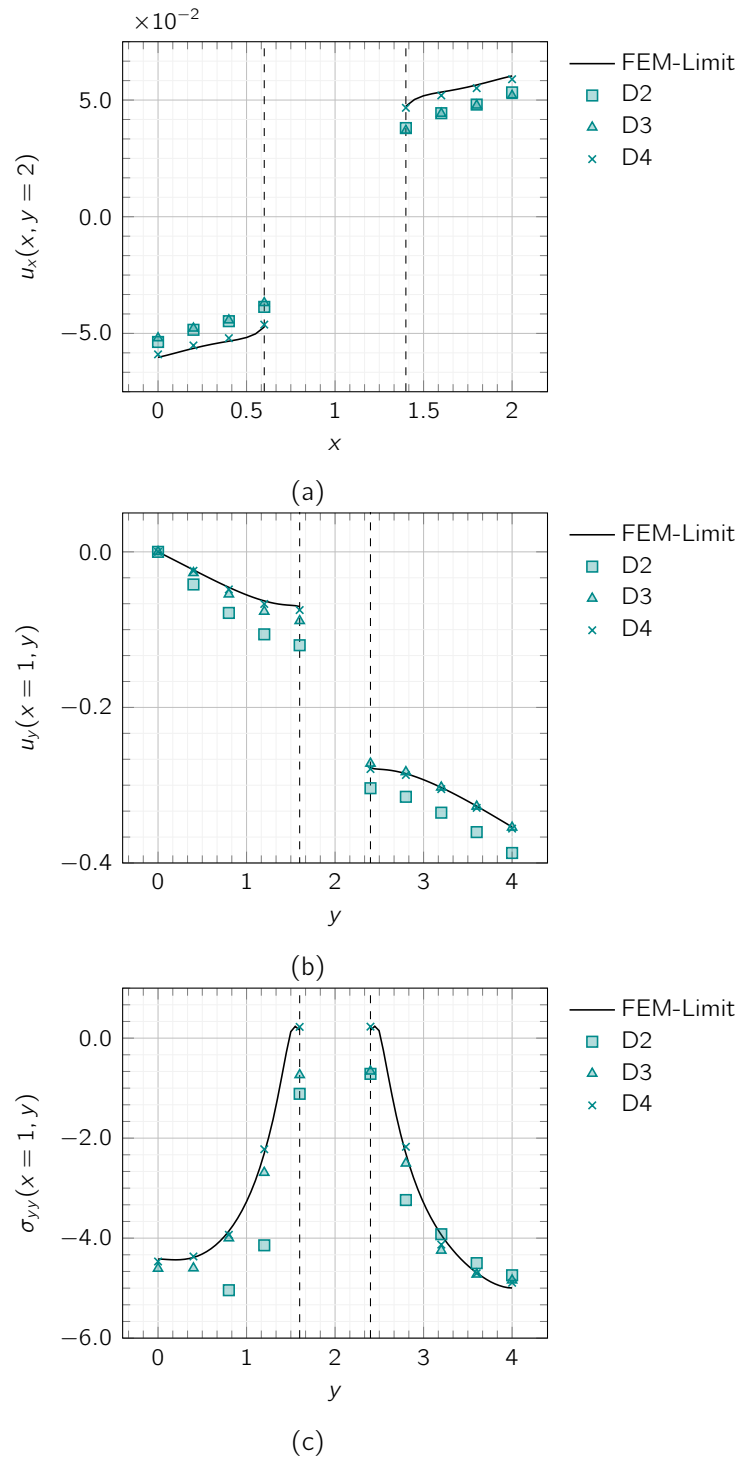


Figure 5.22: Case 3 FEM limit solution and solutions using PyMFS discretisations D1-D4 for: (a), (b) and (c). Dashed lines represent hole edges.

Table 5.6: Case 2 RMSE and time multiplier, using the FEM-Limit solution as reference.

Discretisation	Nodes	DoFs	$\sigma_{yy}^t$ (Pa)	Percentage error (%)	$\sigma_{yy}^b$ (Pa)	Percentage error (%)
FEM-Limit	4,355	8,710	-4.418	-	-4.997	-
D1	14	112	-51.530	1066.4	-4.734	-5.3
D2	44	352	-9.924	124.6	-4.743	-5.1
D3	144	1,152	-4.608	4.3	-4.842	-3.1
D4	524	4,192	-4.467	1.1	-4.892	-2.1

of the MFS where geometrically complex surfaces are considered, as there is no longer imperfect domain discretisation as a result of meshing: intersection of sphere elements with the boundary can be calculated analytically.



## Chapter 6

# Conclusions

### 6.1 Summary

Overall, the project outlined in this report has been successful in accomplishing its primary aim: a robust implementation of the MFS in a Python framework has been developed, and applied to a series of examples which illustrate its accuracy in solving problems in 2D elastostatics, amongst other simple problem classes.

Following a complete review of the literature on mesh-based and meshless numerical methods in Chapter 2, the MFS was identified as a particularly promising method, for which there does not exist a widely available software implementation. The underlying mathematical theory behind the method was then presented in detail, in order to outline the foundations upon which problems can be solved using the method. The overall structure and class hierarchy of PyMFS, a Python framework for solving problems using the MFS, was then presented in Chapter 4, including an introduction to its typical useage and workflow. A series of examples were then solved using this framework, validating the accuracy of the developed code, with an order of convergence consistent with literature observed.

There were, however, also a number of observations which prompt the requirement for future research works. Inaccuracies in solutions obtained with PyMFS along surfaces upon which essential boundary conditions are applied were observed in the absence of a special nodal arrangement, and the low computational efficiency of the PyMFS solver presents a barrier to its application to increasingly complex problems where a large number of system DoFs are required.

### 6.2 Future work

Based on the findings of this work, there are therefore three key areas of focus which have been identified for future research works, relating to both the PyMFS solver specifically, as well as the implementation of the MFS in general:

- *Accuracy*: the observed inaccuracy of PyMFS when implementing essential BCs should be further investigated, including a comprehensive analysis on the effect of sphere discretisation refinement in these areas.

- *Efficiency*: as a result of having computational times which are up to four orders of magnitude slower than the FEM, future work should be carried out which focuses on improving the efficiency of the PyMFS solver. In particular, efforts should focus on implementing algorithms which facilitate more efficient assembly of the global system matrices, by bypassing repeat calculations of identical spheres, and performing intensive computations in parallel by assigning calculations to multiple CPU cores.
- *Range*: Once the above works have been performed, efforts should then move towards the extension of the PyMFS framework to handle a wider class of problems (such as impulsive dynamics problems, where the MFS appears particularly promising [34]) applied to increasingly complex geometries (by increasing the pre and post-processing capabilities of PyMFS).

# Bibliography

- [1] C. Huygens, *Treatise on Light*, Jan. 2005. [Online]. Available: <https://www.gutenberg.org/ebooks/14725>
- [2] I. M. Gelfand, S. V. Fomin, and R. A. Silverman, *Calculus of Variations*. Courier Corporation, Jan. 2000, google-Books-ID: YkFLGQeGRw4C.
- [3] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl, "Meshless methods: An overview and recent developments," *Computer Methods in Applied Mechanics and Engineering*, vol. 139, no. 1-4, pp. 3–47, Dec. 1996. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S004578259601078X>
- [4] Y. Chen, J. D. Lee, and A. Eskandarian, *Meshless methods in solid mechanics*. New York, NY: Springer, 2006.
- [5] S. De and K. J. Bathe, "The method of finite spheres," *Computational Mechanics*, vol. 25, no. 4, pp. 329–345, Apr. 2000. [Online]. Available: <https://doi.org/10.1007/s004660050481>
- [6] B. Lai and K.-J. Bathe, "The method of finite spheres in three-dimensional linear static analysis," *Computers & Structures*, vol. 173, pp. 161–173, Sep. 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0045794916303820>
- [7] "Welcome to Python.org." [Online]. Available: <https://www.python.org/>
- [8] R. Hooke, *Lectures de Potentia Restitutiva, Or of Spring Explaining the Power of Springing Bodies*. John Martyn, 1678, google-Books-ID: LATPAAAAcAAJ.
- [9] A. Hrennikoff, "Solution of Problems of Elasticity by the Framework Method," *Journal of Applied Mechanics*, vol. 8, no. 4, pp. A169–A175, 1941. [Online]. Available: <https://doi.org/10.1115/1.4009129>
- [10] R. Courant, "Variational methods for the solution of problems of equilibrium and vibrations," *Bulletin of the American Mathematical Society*, vol. 49, no. 1, pp. 1–23, 1943. [Online]. Available: <https://www.ams.org/bull/1943-49-01/S0002-9904-1943-07818-4/>
- [11] R. W. Clough, "The finite element method in plane stress analysis," in *Conference papers 2nd Conference on Electronic Computation, Pittsburgh, Pa*, 1960.
- [12] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, *The finite element method: its basis and fundamentals*, seventh edition ed. Amsterdam: Elsevier, Butterworth-Heinemann, 2013, oCLC: ocn852808496.
- [13] K.-J. Bathe, *Finite Element Procedures*. Klaus-Jurgen Bathe, 2006, google-Books-ID: rWve-fGICfO8C.
- [14] C. Lanczos, *The Variational Principles of Mechanics*. Courier Corporation, Jan. 1986, google-Books-ID: ZWoYYr8wk2IC.
- [15] "METHODUS NOVA INTEGRALIU VALORES PER APPROXIMATIONEM INVENIENDI," in *Werke*, ser. Cambridge Library Collection - Mathematics, C. F. Gauss, Ed. Cambridge: Cambridge University Press, 2011, vol. 3, pp. 165–196. [Online]. Available: <https://www.cambridge.org/core/books/werke/methodus-nova-integralium-valores-per-approximationem-inveniendi/>

- 6812FCD62467D015CADC7776EDA290B8
- [16] B. M. Irons, "A frontal solution program for finite element analysis," *International Journal for Numerical Methods in Engineering*, vol. 2, no. 1, pp. 5–32, 1970, [\\_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620020104](https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620020104). [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620020104>
  - [17] "NASA STRuctural ANalysis (NASTRAN)(LAR-16804-GS) | NASA Software Catalog." [Online]. Available: <https://software.nasa.gov/software/LAR-16804-GS>
  - [18] N.-s. Lee and K.-J. Bathe, "Effects of Element Distortions on the Performance of Isoparametric Elements," *Int. Jour. Num. Meth. Eng*, vol. 36, pp. 3553–3576, 1993.
  - [19] T. Belytschko, N. Moës, S. Usui, and C. Parimi, "Arbitrary discontinuities in finite elements," *International Journal for Numerical Methods in Engineering*, vol. 50, no. 4, pp. 993–1013, 2001, publisher: Wiley. [Online]. Available: <https://hal.archives-ouvertes.fr/hal-01005275>
  - [20] M. B. Liu and G. R. Liu, "Smoothed Particle Hydrodynamics (SPH): an Overview and Recent Developments," *Archives of Computational Methods in Engineering*, vol. 17, no. 1, pp. 25–76, Mar. 2010. [Online]. Available: <https://doi.org/10.1007/s11831-010-9040-7>
  - [21] J. J. Monaghan, "An introduction to SPH," *Computer Physics Communications*, vol. 48, no. 1, pp. 89–96, Jan. 1988. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0010465588900264>
  - [22] B. Nayroles, G. Touzot, and P. Villon, "Generalizing the finite element method: Diffuse approximation and diffuse elements," *Computational Mechanics*, vol. 10, no. 5, pp. 307–318, Sep. 1992. [Online]. Available: <https://doi.org/10.1007/BF00364252>
  - [23] T. Belytschko, Y. Y. Lu, and L. Gu, "Element-free Galerkin methods," *International Journal for Numerical Methods in Engineering*, vol. 37, no. 2, pp. 229–256, 1994, [\\_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620370205](https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620370205). [Online]. Available: <http://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620370205>
  - [24] "An h-p adaptive method using clouds - ScienceDirect." [Online]. Available: <https://www-sciencedirect-com.ezproxy.is.ed.ac.uk/science/article/pii/S0045782596010857>
  - [25] S. N. Atluri and T. Zhu, "A new Meshless Local Petrov-Galerkin (MLPG) approach in computational mechanics," *Computational Mechanics*, vol. 22, no. 2, pp. 117–127, Aug. 1998. [Online]. Available: <http://link.springer.com/10.1007/s004660050346>
  - [26] S. Atluri, H.-G. Kim, and J. Cho, "Critical assessment of the truly Meshless Local Petrov-Galerkin (MLPG), and Local Boundary Integral Equation (LBIE) methods," *Computational Mechanics*, vol. 24, no. 5, pp. 348–372, 1999.
  - [27] S. N. Atluri, J. Y. Cho, and H.-G. Kim, "Analysis of thin beams, using the meshless local Petrov-Galerkin method, with generalized moving least squares interpolations," *Computational Mechanics*, vol. 24, no. 5, pp. 334–347, Nov. 1999. [Online]. Available: <https://doi.org/10.1007/s004660050456>
  - [28] Z. Han and S. Atluri, "Meshless local Petrov-Galerkin (MLPG) approaches for solving 3D problems in elasto-statics," *CMES - Computer Modeling in Engineering and Sciences*, vol. 6, no. 2, pp. 169–188, 2004.
  - [29] J. M. Melenk and I. Babuška, "The partition of unity finite element method: Basic theory and applications," *Computer Methods in Applied Mechanics and Engineering*, vol. 139, no. 1, pp. 289–314, Dec. 1996. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0045782596010870>
  - [30] S. De and K. Bathe, "Towards an efficient meshless computational technique: the method of finite spheres," *Engineering Computations*, vol. 18, no. 1/2, pp. 170–192, Jan. 2001, publisher: MCB UP Ltd. [Online]. Available: <https://doi.org/10.1108/02644400110365860>
  - [31] S. De and K.-J. Bathe, "The method of finite spheres with improved numerical integration,"

- Computers & Structures*, vol. 79, no. 22, pp. 2183–2196, Sep. 2001. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0045794901001249>
- [32] —, “Displacement/pressure mixed interpolation in the method of finite spheres,” *International Journal for Numerical Methods in Engineering*, vol. 51, no. 3, pp. 275–292, 2001, \_eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.168>. [Online]. Available: <http://onlinelibrary.wiley.com/doi/abs/10.1002/nme.168>
- [33] M. Macri, S. De, and M. S. Shephard, “Hierarchical tree-based discretization for the method of finite spheres,” *Computers & Structures*, vol. 81, no. 8–11, pp. 789–803, May 2003. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0045794902004753>
- [34] S. Ham, B. Lai, and K.-J. Bathe, “The method of finite spheres for wave propagation problems,” *Computers & Structures*, vol. 142, pp. 1–14, Sep. 2014. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0045794914001321>
- [35] K.-T. Kim and K.-J. Bathe, “Transient implicit wave propagation dynamics with the method of finite spheres,” *Computers & Structures*, vol. 173, pp. 50–60, Sep. 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0045794916303017>
- [36] W. L. Nicomedes, K.-J. Bathe, F. J. S. Moreira, and R. C. Mesquita, “Acoustic scattering in nonhomogeneous media and the problem of discontinuous gradients: Analysis and inf-sup stability in the method of finite spheres,” *International journal for numerical methods in engineering*, vol. 122, no. 13, pp. 3141–3170, 2021, place: Hoboken, USA Publisher: John Wiley & Sons, Inc.
- [37] R. Cimrman, V. Lukeš, and E. Rohan, “Multiscale finite element calculations in python using sfepy,” *Advances in Computational Mathematics*, 2019. [Online]. Available: <https://doi.org/10.1007/s10444-019-09666-0>
- [38] P. Ramachandran, A. Bhosale, K. Puri, P. Negi, A. Muta, A. Dinesh, D. Menon, R. Govind, S. Sanka, A. S. Sebastian, A. Sen, R. Kaushik, A. Kumar, V. Kurapati, M. Patil, D. Tavker, P. Pandey, C. Kaushik, A. Dutt, and A. Agarwal, “PySPH: A Python-based Framework for Smoothed Particle Hydrodynamics,” *ACM Transactions on Mathematical Software*, vol. 47, no. 4, pp. 1–38, Dec. 2021. [Online]. Available: <https://dl.acm.org/doi/10.1145/3460773>
- [39] L. Prechelt, “An empirical comparison of seven programming languages,” *Computer*, vol. 33, no. 10, pp. 23–29, Oct. 2000, conference Name: Computer.
- [40] J. Gmys, T. Carneiro, N. Melab, E.-G. Talbi, and D. Tuytens, “A comparative study of high-productivity high-performance programming languages for parallel metaheuristics,” *Swarm and Evolutionary Computation*, vol. 57, p. 100720, Sep. 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2210650220303734>
- [41] C. Armando Duarte and J. Tinsley Oden, “H-p Clouds - An h-p Meshless Method,” *Numerical Methods for Partial Differential Equations*, vol. 12, no. 6, pp. 673–705, Nov. 1996. [Online]. Available: <http://www.scopus.com/inward/record.url?scp=1542580610&partnerID=8YFLogxK>
- [42] G. H. Golub and J. H. Welsch, “Calculation of Gauss Quadrature Rules,” p. 10.
- [43] “Git.” [Online]. Available: <https://git-scm.com/>
- [44] “Build software better, together.” [Online]. Available: <https://github.com>
- [45] S. F. Lott, *Python object-oriented programming: build robust and maintainable object-oriented Python applications and libraries.*, 4th ed., ser. Expert Insight. Place of publication not identified: Packt Publishing, 2021.
- [46] “Advances in Computational Mathematics.” [Online]. Available: <https://www.springer.com/journal/10444>
- [47] C. R. Harris *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, number: 7825 Publisher: Nature Publishing Group. [Online]. Available:

- <https://www.nature.com/articles/s41586-020-2649-2>
- [48] P. Virtanen *et al.*, “SciPy 1.0: fundamental algorithms for scientific computing in Python,” *Nature Methods*, vol. 17, no. 3, pp. 261–272, Mar. 2020, number: 3 Publisher: Nature Publishing Group. [Online]. Available: <https://www.nature.com/articles/s41592-019-0686-2>
- [49] C. Smith *et al.*, “sympy/sympy: SymPy 1.10.1,” Mar. 2022. [Online]. Available: <https://zenodo.org/record/6371200>
- [50] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [51] S. Gillies *et al.*, “Shapely: manipulation and analysis of geometric objects,” 2007–. [Online]. Available: <https://github.com/shapely/shapely>
- [52] “pickle — Python object serialization — Python 3.10.4 documentation.” [Online]. Available: <https://docs.python.org/3/library/pickle.html#id7>
- [53] I. Karakan, C. Gürkan, and C. Avci, “Performance analyses of mesh-based local Finite Element Method and meshless global RBF Collocation Method for solving Poisson and Stokes equations,” *Mathematics and Computers in Simulation*, vol. 197, pp. 127–150, Feb. 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0378475422000684>

## **Appendix A**

# **Project Gantt chart**

For the purpose of clarity, the project Gantt chart is presented on the following page in landscape.

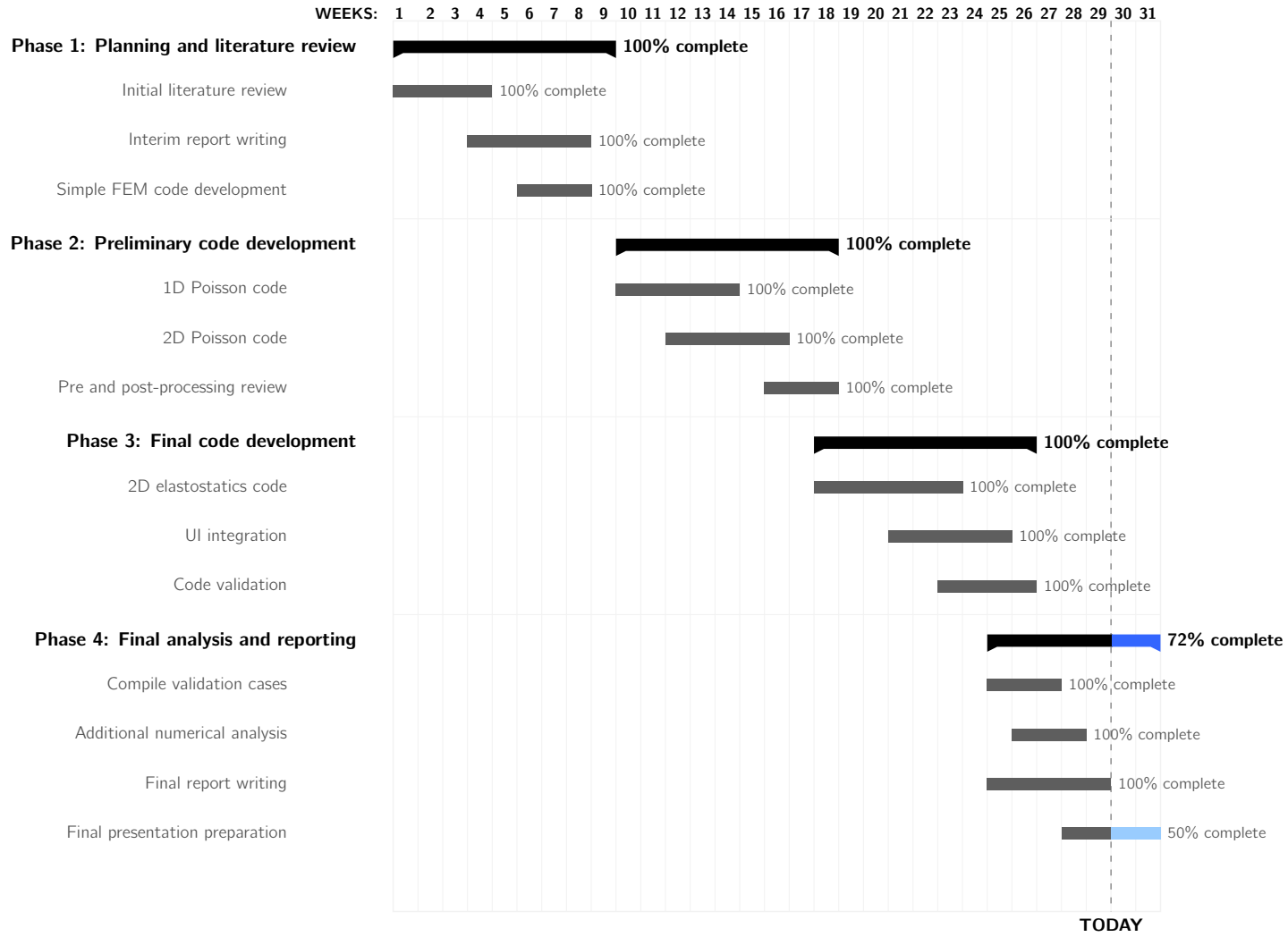









Figure A.1: Project Gantt chart.



## Appendix B

# Flowchart and programming diagram legend

Table B.1: Flowchart symbols used throughout report and their meaning.

Symbol	Name	Function
	Start/end	An oval represents a start or end point.
	Input/output	A parallelogram represents input or output.
	Process	A rectangle represents a process.
	Decision	A diamond represents a decision.
	Python class	A shaded, rounded rectangle represents a system class within a Python program.
	Primary arrow	A solid line acts as a connector between primary process steps and objects.
	Secondary arrow	A dashed line acts as a connector between secondary process steps and objects.



## Appendix C

# MFS local weak form derivation

A full derivation of the local weak form of the MFS is here provided to supplement Section 3.3. Considering again the arbitrary domain discretisation shown in Figure 3.1, the weak form is here derived for each sphere for a single variable second order PDE, but can be directly extended to equations in a greater number of variables with differential operators of higher order. The equation considered here is:

$$Au = f \quad (\text{C.1})$$

for which  $A$  is a second-order differential operator and  $f$  is a forcing function.  $A$  can be written as:

$$A = - \sum_{i,j=1}^d \frac{\partial}{\partial x_i} a_{ij}(\underline{x}) \frac{\partial}{\partial x_j} + c(\underline{x}) \quad (\text{C.2})$$

where  $d$  is the number of dimensions considered, and  $a_{ij}(\underline{x})$  and  $c(\underline{x})$  are measurable coefficients. By prescribing Neumann BCs on the surface  $S_f$  it is possible to write:

$$\sum_{i,j=1}^d a_{ij}(\underline{x}) \frac{\partial u}{\partial x_j} n_i = f^s \text{ on } S_f \quad (\text{C.3})$$

where  $n_i$  is the directional cosine for the  $i$ th direction of the surface under consideration. Similarly, by prescribing Dirichlet BCs on the surface  $S_u$ , it is possible to write:

$$u = u^s \text{ on } S_u. \quad (\text{C.4})$$

The MFS uses a Bubnov-Galerkin formulation as the weighted residual scheme, where the approximation ( $u^{\phi,p} \in V^{\phi,p}$ ) to the exact solution  $u$  is determined by setting the residual  $Au^{\phi,p} - f$  perpendicular to the shape functions  $\phi_{lm}$ . This yields:

$$(Au^{\phi,p} - f, \phi_{lm}) = 0, \quad m \in \mathcal{J} \quad (\text{C.5})$$

Then, using the discretisation from Equation 3.6 and applying Green's Theorem the local weak form can be written for the  $l$ th node in terms of its  $m$ th DoF as:

$$\sum_{J=1}^N \sum_{n \in \mathcal{J}} K_{lmJn} q_{Jn} = f_{lm} + \hat{f}_{lm} \quad (\text{C.6})$$

where:

$$K_{ImJn} = a(\phi_{Im}, \phi_{Jn}) = \int_{V_I \cap V} c(\underline{x}) \phi_{Im} \phi_{Jn} \, dV + \sum_{i,j=1}^d \int_{V_I \cap V} a_{ij}(\underline{x}) \frac{\partial \phi_{Im}}{\partial x_i} \frac{\partial \phi_{Jn}}{\partial x_j} \, dV, \quad (C.7)$$

$$f_{Im} = \int_{V_I \cap V} f \phi_{Im} \, dV, \quad (C.8)$$

$$\hat{f}_{Im} = \sum_{i,j=1}^d \int_{S_I} \phi_{Im} n_i a_{ij}(\underline{x}) \frac{\partial u^{\phi,p}}{\partial x_j} \, dS. \quad (C.9)$$

## Appendix D

### Selected MFS equations list

Here, the specific form of the MFS local weak form and its associated terms which must be evaluated to solve problems involving the chosen governing equations are presented in the tables below.

Table D.1: List of the key equations implemented by the PyMFS solver in Chapter 4.

Problem class	Governing equation
1D Poisson	$\frac{d^2 u(x)}{dx^2} + f(x) = 0$
2D Poisson	$\frac{d^2 u(x, y)}{dx^2} + \frac{d^2 u(x, y)}{dy^2} + f(x, y) = 0$
2D elastostatics	$\underline{\partial}_\epsilon^T \underline{\sigma} + \underline{f}^B = 0$
	Local weak form
1D Poisson	$\sum_{J=1}^N \sum_{n \in \mathcal{J}} K_{ImJn} q_{Jn} = f_{Im} + \hat{f}_{Im}$
2D Poisson	$\sum_{J=1}^N \sum_{n \in \mathcal{J}} K_{ImJn} q_{Jn} = f_{Im} + \hat{f}_{Im}$
2D elastostatics	$\sum_{J=1}^N \sum_{n \in \mathcal{J}} \underline{K}_{ImJn} \underline{q}_{Jn} = \underline{f}_{Im} + \underline{\hat{f}}_{Im} \quad (D.1)$
	Stiffness

1D Poisson	$K_{ImJn} = \int_{x_1}^{x_2} \frac{d\phi_{Im}}{dx} \frac{d\phi_{Jn}}{dx} dx \quad (D.2)$
2D Poisson	$K_{ImJn} = \int_{V_I \cap V} \left( \frac{\partial \phi_{Im}}{\partial x} \frac{\partial \phi_{Jn}}{\partial x} + \frac{\partial \phi_{Im}}{\partial y} \frac{\partial \phi_{Jn}}{\partial y} \right) dV \quad (D.3)$
2D elastostatics	$\underline{K}_{ImJn} = \int_{V_I \cap V} \underline{B}_{Im}^T \underline{C} \underline{B}_{Jn} dV \quad (D.4)$
<b>Forcing</b>	
1D Poisson	$f_{Im} = \int_{x_1}^{x_2} f(x) \phi_{Im} dx \quad (D.5)$
2D Poisson	$f_{Im} = \int_{V_I \cap V} f(x, y) \phi_{Im} dV \quad (D.6)$
2D elastostatics	$\underline{f}_{Im} = \int_{V_I \cap V} \underline{\phi}_{Im} \underline{f}^b dV \quad (D.7)$
<b>Boundary forcing</b>	
1D Poisson	$\hat{f}_{Im} = \begin{cases} 0, & \text{interior spheres} \\ f^s \phi_{Im}(x = x^f), & \text{Neumann spheres} \\ \sum_{J=1}^N \sum_{n \in \mathcal{J}} K U_{ImJn} q_{Jn} - f U_{Im}, & \text{Dirichlet spheres} \end{cases} \quad (D.8)$
2D Poisson	$\hat{f}_{Im} = \begin{cases} 0, & \text{interior spheres} \\ \int_{S_{f_I}} f^s \phi_{Im} dS, & \text{Neumann spheres} \\ \sum_{J=1}^N \sum_{n \in \mathcal{J}} K U_{ImJn} q_{Jn} - f U_{Im}, & \text{Dirichlet spheres} \end{cases} \quad (D.9)$
2D elastostatics	$\underline{\hat{f}}_{Im} = \begin{cases} 0, & \text{interior spheres} \\ \int_{S_{f_I}} \underline{\phi}_{Im} \underline{f}^s dV, & \text{Neumann spheres} \\ \sum_{J=1}^N \sum_{n \in \mathcal{J}} \underline{K} U_{ImJn} \underline{q}_{Jn} - \underline{f} U_{Im}, & \text{Dirichlet spheres} \end{cases} \quad (D.10)$
<b>Dirichlet stiffness</b>	
1D Poisson	$K U_{ImJn} = - \left[ \frac{d(\phi_{Im} \phi_{Jn})}{dx} \right]_{x=x^u} \quad (D.11)$
2D Poisson	$K U_{ImJn} = \int_{S_{u_I}} \frac{\partial}{\partial x} (\phi_{Im} \phi_{Jn}) dS \quad (D.12)$

2D elastostatics	$\underline{K} \underline{U}_{ImJn} = \int_{S_{uj}} \left( \underline{\phi}_{Im} \underline{N} \underline{C} \underline{B}_{Jn} + \underline{B}_{Im}^T \underline{C} \underline{N}^T \underline{\phi}_{Jn} \right) dS \quad (D.13)$
	<b>Dirichlet forcing</b>
1D Poisson	$f U_{Im} = -u^s \left[ \frac{d\phi_{Im}}{dx} \right]_{x=x^u} \quad (D.14)$
2D Poisson	$f U_{Im} = \int_{S_{uj}} u^s \frac{\partial \phi_{Im}}{\partial x} dS \quad (D.15)$
2D elastostatics	$\underline{f} U_{Im} = \int_{S_{uj}} \underline{B}_{Im}^T \underline{C} \underline{N}^T \underline{u}^s dS. \quad (D.16)$

Table D.2: Definitions of variables introduced in Table D.1. Note, empty entries indicate all variables of the corresponding entry from Table D.1 have been defined earlier in this report.

Problem class	Governing equation
1D Poisson	$u$ - sought variable, $x$ - position along length, $f$ - distributed loading.
2D Poisson	$x$ - Cartesian $x$ position, $y$ - Cartesian $y$ position.
2D elastostatics	$\underline{\partial}_\epsilon = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ \partial/\partial y & \partial/\partial x \end{bmatrix}$ , $\underline{\sigma} = [\sigma_{xx} \ \sigma_{yy} \ \sigma_{xy}]^T$ , $\underline{f}^B = [f_x^b(x, y) \ f_y^b(x, y)]^T$
	<b>Local weak form</b>
1D Poisson	-
2D Poisson	-
2D elastostatics	-
	<b>Stiffness</b>
1D Poisson	$x_1 = \max(x_{\min}, x_l - r_l)$ , $x_2 = \min(x_{\max}, x_l + r_l)$ where $x_{\min}$ and $x_{\max}$ are the minimum and maximum positions along the domain length respectively.
2D Poisson	-
2D elastostatics	$\underline{B}_{Jn} = \underline{\partial}_\epsilon \underline{\phi}_{Jn} = \begin{bmatrix} \partial \phi_{Jn}/\partial x & 0 \\ 0 & \partial \phi_{Jn}/\partial y \\ \partial \phi_{Jn}/\partial y & \partial \phi_{Jn}/\partial x \end{bmatrix}$ , $\underline{C} = \begin{bmatrix} c_{11} & c_{12} & 0 \\ c_{12} & c_{11} & 0 \\ 0 & 0 & c_{33} \end{bmatrix}$
	<b>Forcing</b>
1D Poisson	-
2D Poisson	-
2D elastostatics	-

	Boundary forcing
1D Poisson	$x^u$ - essential BC coordinate, $x^f$ - natural BC coordinate.
2D Poisson	-
2D elastostatics	$\underline{f}^s = [f_x^s(x, y) \ f_y^s(x, y)]^T$
	Dirichlet stiffness
1D Poisson	-
2D Poisson	-
2D elastostatics	$\underline{N} = \begin{bmatrix} n_x & 0 & n_y \\ 0 & n_y & n_x \end{bmatrix}$
	Dirichlet forcing
1D Poisson	-
2D Poisson	-
2D elastostatics	$\underline{u}^s = [u^s(x, y) \ v^s(x, y)]^T$



## Appendix E

### Gauss-Legendre product rule

The Gauss-Legendre product rule [42] is adopted within PyMFS, and is here presented for evaluation of integrals in 2D. A  $n$  point quadrature rule approximates the integral of a function  $f(\xi)$  as:

$$\int_{-1}^1 f(\xi) d\xi \approx \sum_{i=1}^n w_i f(\hat{\xi}_i) \quad (\text{E.1})$$

where  $w_i$  are the weighting coefficients at the integration points  $\hat{\xi}_i$ . This rule reproduces exact integrals for polynomials of degree  $d$  up to  $2n - 1$ . The integration weights and points for Gaussian quadrature rules up to  $n = 3$  are provided in Table E.1.

Table E.1: Gauss point coordinates and weights for quadrature rules up to  $n = 3$ .

$n$	$d$	$\hat{\xi}_i$	$w_i$
1	1	0	2
2	3	$-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$	1, 1
3	5	$-\sqrt{\frac{3}{5}}, 0, \sqrt{\frac{3}{5}}$	$\frac{5}{8}, \frac{8}{9}, \frac{5}{9}$



## Appendix F

# MFS file format

The PyMFS solve class accepts a .mfs file as input, the structure of which is outlined here, with a simple example. At present, the solve class in PyMFS requires the following properties generated during pre-processing:

- A list of the domain surfaces;
- A list of nodal coordinates;
- Sphere radius;
- Physical material properties;
- Prescribed essential boundary conditions;
- Prescribed natural boundary conditions.

The below listing illustrates a simple .mfs input file, which has been used in Chapter 5 in the study of case 1 and meets the above requirements.

---

```
# Job: examples\2d_elastostatics\tension\tension_N3.mfs

# External surfaces
[[[0, 0], [2, 0]], [[2, 0], [2, 2]], [[2, 2], [0, 2]], [[0, 2], [0, 0]]]

# Internal surfaces
[]

# Nodal coordinates
[[0, 0], [0, 1], [0, 2], [1, 0], [1, 1], [1, 2], [2, 0], [2, 1], [2, 2]]

# Sphere radius
1

# Physical properties
[100.0, 0.0, 1.0]
```

```
# Prescribed displacements [[u_sx], [u_sy], [u_sx_flag], [u_sy_flag], [surfaces]]  
[[0.0], [0.0], [0], [0], [[3]]]  
  
# Prescribed loads [[f_sx], [f_sy], [f_sx_flag], [f_sy_flag], [surfaces]]  
[[10.0], [0.0], [0], [0], [[1]]]
```

---

# List of Figures

2.1	Arbitrary 3D body occupying a volume, $V$ , with applied force $F$ and boundary conditions $S_f$ and $S_u$ . . . . .	4
2.2	Discretisation of volume, $V$ , with tetrahedral finite elements. . . . .	5
2.3	(a) Linear T3 triangle element, including nodal numbering and displacement degrees of freedom $q_1$ to $q_6$ and (b) illustration of the shape function $\varphi_1$ of node 1. . . . .	6
2.4	Example of two element finite element discretisation with local elementwise numbering system used for assembling global system matrices from local element matrices. . . . .	7
2.5	Flowchart outlining the general steps implemented in computer programs based on the FEM. . . . .	8
2.6	Classification of various distortions of a 9-node element: (a) undistorted configuration, (b) aspect-ratio distortion, (c) unevenly-spaced-nodes distortion, (d) parallelogram distortion, (e) angular distortion and (f) curved-edge distortion (adapted from [18]). . . . .	10
2.7	Discretisation of volume $V$ with overlapping spheres centred around nodes. These spheres act as support for shape functions in the MFS. . . . .	12
2.8	Timeline illustrating the main developments of the MFS in recent decades, with a focus on works directly relevant to the field of solid mechanics. . . . .	13
3.1	Illustration of the parameters which define each unique sphere in the MFS, adapted from [6]. . . . .	15
3.2	Illustration of the MFS sphere elements in 1D, where the "spheres" can be thought of as segments of a line. . . . .	16
3.3	(a) MFS discretisation of square domain using 9 two dimensional sphere elements, (b) weighting function associated with node 5, $W_5(\underline{x})$ and (c) the associated Shepard PU function $\varphi_5^0$ . . . . .	17
3.4	Visualisation of complete first-order polynomial using the Pascal triangle in two dimensions and illustration of how the choice of local basis terms determines the number of DoFs of the MFS element. . . . .	18
3.5	Shape functions associated with the $x$ -direction of the (a) MFS discretisation of square domain using 9 two dimensional sphere elements, (b) weighting function associated with node 5, $W_5(\underline{x})$ and (c) the associated Shepard PU function $\varphi_5^0$ . . . . .	19
3.6	Illustration of regions of integration for spheres intersecting surfaces with (a) natural and (b) essential boundary conditions applied, adapted from [5]. . . . .	21
3.7	Special nodal arrangement for Dirichlet boundaries. . . . .	22
3.8	Illustration of sphere region definitions and their integration points: (a) interior sphere, (b) boundary sphere, (c) sphere overlap and the corresponding integration points associated with (d) interior sphere, (e) boundary sphere and (f) sphere overlap. . . . .	23

4.1	Flowchart outlining the user workflow for PyMFS. . . . .	26
4.2	Hierarchy of classes related to <code>pre_process</code> . . . . .	27
4.3	Hierarchy of classes related to <code>solve</code> , including user input requirements. . . . .	29
4.4	Flowchart illustrating system matrix assembly in PyMFS, $N$ is the total number of nodes in the domain. $\underline{f}$ is assembled in the <code>f_vec</code> class whilst looping over every node, whilst $\underline{K}$ is assembled in the <code>K_mat</code> class whilst looping over every node <i>and</i> the influence of all nodes in the domain. . . . .	30
4.5	Hierarchy of classes related to <code>post_process</code> . . . . .	31
4.6	Example domain generated with PyMFS. . . . .	32
4.7	Example domain discretisation in PyMFS. . . . .	33
4.8	Example pre-processing UI in PyMFS. . . . .	33
5.1	Analytical solution to Poisson's equation in 1D, compared with regular distribution of 6 1D sphere elements as presented in [5], as well as solutions obtained with PyMFS using (a) 3 sphere elements and (b) 7 sphere elements. . . . .	36
5.2	2D square domain on which Poisson's equation is solved. Neumann BC applied to surface $S_f$ at $x = -1$ and Dirichlet BC applied to surface $S_u$ at $x = 1$ . . . . .	36
5.3	Contour plot of $u(x, y)$ obtained by solving Poisson's equation over square domain: (a) analytically, (b) using $3 \times 3 = 9$ sphere elements and (c) using $6 \times 6 = 36$ sphere elements. . . . .	37
5.4	Contour plots illustrating the absolute error in the solutions to Poisson's equation $u(x, y)$ obtained using (a) 9 and (b) 36 nodes. . . . .	38
5.5	Comparison between analytical solution to 2D Poisson's equation with results from [5] and PyMFS using regular arrangement of 36 nodes along slice of domain at: (a) $y = 0$ (b) $y = 1$ and (c) $x = 1$ . . . . .	38
5.6	Problem definition for case 1. . . . .	39
5.7	Sphere element discretisations used for case 1: (a) D1, (b) D2, (c) D3 and (d) D4. . . . .	40
5.8	Contour plots of displacement field in the $x$ direction for: (a) D1, (b) D2, (c) D3 and (d) D4. . . . .	40
5.9	Contour plots of displacement field in the $y$ direction for: (a) D1, (b) D2, (c) D3 and (d) D4. . . . .	41
5.10	Case 1 analytical solution and solutions using PyMFS discretisations, $u_x$ , along lines: (a) $x = 0$ and (b) $y = 2$ for case 1. Note, * denotes instances where the special nodal arrangement is exploited. . . . .	42
5.11	Case 1 convergence of RMSE for D1-D4. . . . .	43
5.12	Problem definition for case 2. . . . .	43
5.13	FEM discretisations used for case 2: (a) FEM1, (b) FEM2, (c) FEM3 and (d) FEM4. . . . .	44
5.14	Deformed shapes obtained using: (a) D1 and FEM1, (b) D2 and FEM2, (c) D3 and FEM3 and (d) D4 and FEM4. . . . .	45
5.15	Contours of $u_y$ for solution to case 2 using D4. . . . .	45
5.16	Case 2 FEM-Limit solution and solutions obtained using MFS discretisations along beam mid-span, $u_y(x, y = 1)$ . . . . .	46
5.17	Case 2 convergence of RMSE for D1-D4 and FEM1-FEM4. . . . .	46
5.18	Problem definition for case 3. . . . .	47
5.19	Sphere element discretisations used for case 3: (a) D1, (b) D2, (c) D3 and (d) D4. . . . .	48
5.20	Deformed shapes obtained using the MFS for case 3 superimposed on FEM limit solution, shown for (a) D1, (b) D2, (c) D3 and (d) D4. . . . .	48
5.21	Contour plots for case 3, D4 of (a) $u_x$ and (b) $u_y$ . . . . .	48

5.22 Case 3 FEM limit solution and solutions using PyMFS discretisations D1-D4 for: (a), (b) and (c). Dashed lines represent hole edges. . . . .	49
A.1 Project Gantt chart. . . . .	58





# List of Tables

2.1	Comparison between the FEM, SPH and the MFS in terms of the accuracy and efficiency of each method. . . . .	12
4.1	Description of PyMFS core classes. . . . .	26
4.2	Description of existing Python libraries used in PyMFS. . . . .	27
5.1	Parameters used in study of case 1. . . . .	39
5.2	Case 1 RMSE of solutions obtained with discretisations D1-D4, and discretisations D2* and D3* which exploit a special nodal arrangement. . . . .	42
5.3	Parameters used in study of case 2. . . . .	44
5.4	Case 2 RMSE and time multiplier, using the FEM-Limit solution as reference. . . .	44
5.5	Parameters used in study of case 1. . . . .	47
5.6	Case 2 RMSE and time multiplier, using the FEM-Limit solution as reference. . . .	50
B.1	Flowchart symbols used throughout report and their meaning. . . . .	59
D.1	List of the key equations implemented by the PyMFS solver in Chapter 4. . . . .	63
D.2	Definitions of variables introduced in Table D.1. Note, empty entries indicate all variables of the corresponding entry from Table D.1 have been defined earlier in this report. . . . .	65
E.1	Gauss point coordinates and weights for quadrature rules up to $n = 3$ . . . . .	67