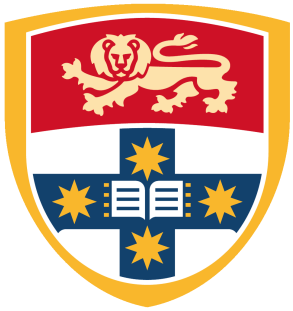


The University Of Sydney

School of Aerospace, Mechanical
and Mechatronic Engineering

Doctoral Thesis

Radial Basis Function Methods in Fluid-Structure Interaction

Author:
Adam James Murray

Supervisors:
Dr. Gareth A. Vio
Prof. Benjamin Thornber

2023

Statement of Originality

I certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work, and that all the assistance received in preparing this thesis and sources have been acknowledged.

Adam James Murray, June 2023

Authorship Attribution Statement

Appendix B of this thesis includes the conference paper *Catastrophe Theoretic Modelling of Hysteresis in Transonic Shock Buffet*, delivered at the AIAA Scitech Forum, 2020.

I completed the modelling of the buffet hysteresis phenomenon from the 2D buffet simulations by Nicholas Giannelis. I wrote sections I, II, IV, and V, while Nicholas Giannelis wrote section III. The paper was presented at the 2020 AIAA SciTech Forum by Nicholas Giannelis.

No other material in this thesis has been previously published.

Adam James Murray, June 2023

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Gareth A. Vio, June 2023

I wish to thank Dr. Gareth Vio for his dedication and guidance over the course of this work. His calm and thoughtful supervision was a much-needed antidote to the various internal and external challenges faced during candidature.

I would also like to thank Prof. Ben Thornber, whose guidance and wisdom at some critical moments always provided great insight, Dr. Markus Flaig, whose help with the various aspects of the cavity-store simulations was greatly appreciated, and my colleagues Dr. Nicholas Giannelis and Dr. David Munk, whose support and good humour over the years has pulled me through some difficult times.

Lastly I wish to thank my wife Carol for her unending patience and support over these past years. It simply would not have been possible without her.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATION & NOMENCLATURE

CFD
: Computational Fluid Dynamics

CSD
: Computational Structural Dynamics

FEA
: Finite Element Analysis

FSI
: Fluid-Structure Interface

FSC
: Fluid-Structure Coupling

FtS
: Fluid-to-Structure

IIM
: Inverse Isoparametric Mapping

PHS
: Polyharmonic Spline

StF
: Structure-to-Fluid

RBF
: Radial Basis Function(s)

RBF-FD
: Radial Basis Function Finite Difference

VLM
: Vortex Lattice Method

c
: Control node in RBF interpolation

v
: Volume node in RBF interpolation

n_c
: Number of control nodes

n_v
: Number of volume nodes

t_0
: Initial time, taken to be before any mesh deformation has occured, i.e. during the preprocessing phase

$\mathbf{s}_t$, $\mathbf{a}_t$
: Position of structural/aerodynamic nodes at time t

d
: Normalised distance, a measure of the difference between two point clouds

c_{rad}
: Coefficient of radius, a measure of the RBF radius required for a wing

$\deg(p)$
: Degree of the polynomial p

$=\!|$
: End of proof

F_i
: General function $\mathbb{R}^n \to \mathbb{R}$

p^i
: Polynomial $\mathbb{R}^n \to \mathbb{R}$

q
: Polynomial $\mathbb{R}^n \to \mathbb{R}$ of restricted degree, or mesh quality measure (orthogonal)

γ^i_j
: Polynomial coefficient

α^i_j
: Polynomial coefficient

H
: RBF coupling matrix

ψ
: Radially symmetric basis function (RBF)

A_{cc}
: Control-Control coupling matrix

A_{cv}
: Control-Volume coupling matrix

A_{bb}
: Base-Base coupling matrix (Kedward's method)

A_{br}
: Base-Refinement coupling matrix (Kedward's method)

A_{rr}
: Refinement-Refinement coupling matrix (Kedward's method)

N_b, N_v
: Number of base/volume nodes (Kedward's method)

α
: Proportion of volume nodes captured within the radii of the base set of control nodes

$\boldsymbol{S}^i$
: Control node vector with additional zeros for polynomial solution

b^i
: Control node coefficients augmented with polynomial coefficients

P
: Sub-block matrix of A_{cc}

M
: Sub-block matrix of A_{cc}

I
: Identity matrix, or integral operator on radial basis functions

$C1 - C6$
: Wendland-type basis functions

V
: Volume for a control surface to act upon

∂V
: Boundary of V

C
: Control surface, or cost of multistage motion algorithm

S
: Collection of vertices of graph

$s_i, s_{i,j}$
: Stage of multistage mesh motion algorithm (vertex in graph)

$(\alpha_i, \beta_i, \gamma_i)$
: Tuple describing single stage of a multistage mesh motion algorithm

M, C, K
: Mass, damping, stiffness matrices

U_i
: Modeshape of structural system

δ_{ij}
: Kronecker delta

Φ
: Modal matrix

η_i
: Modal amplitudes

$\mathcal{L}$
: General linear operator

ψ_i
: Cardinal basis function

w_i
: Linear operator applied to cardinal basis function

D
: Differentiation matrix

r_s
: Support radius for RBF-FD method

S
: Stencil

n_s
: Number of nodes in stencil for RBF-FD method

v_0
: Freestream velocity

ν
: Kinematic viscosity

ρ
: Density

ω
: Vorticity

ψ
: Streamfunction

u, v
: Components of 2D streamfunction

h
: Perpendicular distance from boundary/wall

v_t, v_n
: Tangential/normal velocity to boundary/wall

Δ
: Laplacian operator, or general difference symbol

$A_{Poisson}$
: Differentiation matrix for Poisson equation

A_x, A_y
: Partial differentiation matrices in x and y directions

k_i
: Coefficients in Runge-Kutta method

St
: Strouhal number

C_D
: Drag coefficient

C_L
: Lift coefficient

β
: Blockage ratio for object in flow domain

ζ
: Damping ratio

ω_n
: Natural frequency

1. INTRODUCTION

This work is broadly split into two areas of focus within the application of radial basis function (RBF) methods to fluid-structure interaction problems:

1. A multistage method to augment existing mesh motion methods for the problem of smoothly interpolating boundary movement to surrounding discretised domains.
2. A meshless, partitioned fluid-structure solver using RBF finite difference (RBF-FD) approximations coupled with structural dynamics via RBF mesh motion methods.

1.1 Novelty & Contributions

The primary novel contributions of this thesis are:

1. A multistage mesh motion framework for further reducing costs of a broad range of existing mesh motion algorithms;
2. Adaptation of traditional mesh-based Thom/Jensen style boundary conditions to the meshless case, for the streamline-vorticity formulation of the incompressible Navier Stokes equations;
3. A method of coupling structural and fluid dynamics via moving boundary conditions for the streamline-vorticity formulation;
4. Development of a partitioned FSI solver based on the RBF-FD method, able to handle arbitrary body motions.

1.2 Background

1.2.1 Mesh Motion

As an introductory note on terminology, it should be observed that while the areas of study in this work are often referred to as 'mesh motion', the RBF methods presented here require no adjacency information on nodes, hence are inherently meshless. However, as the work is equally applicable to traditional mesh-based methods, and in keeping with most of the existing of literature, we will use the term mesh motion.

In numerical simulation of dynamical systems, the computational domain itself is often also required to be dynamic to account for changes in the physical system being modelled. Most methods of numerical simulation require the domain of interest to be discretised in some fashion[1] - as physical quantities can only be calculated at some discrete set of points within the (physically) continuous domain. Once the domain has been discretised, there are various methods to dynamically treat these discrete points with a changing domain of interest.

In the realm of the simulation of fluid flow, and fluid-structure interactions, we are most often interested in how to change this set of discretised points in response to a boundary that is moving. In some cases, it may be possible to model the changes of a boundary through changes in the flow itself - e.g. a vertically oscillating cylinder can be described by changing the angle of the incoming flow, rather than moving the cylinder itself, as shown in fig. 1.1a and fig. 1.1b. Similarly, a horizontally oscillating cylinder can be described by increasing/decreasing the flow speed. However in cases where flow boundaries are physical, and are in motion relative to each other (or themselves), we must deform the points of the computational domain to account for this motion. Imagine now that the cylinder is oscillating vertically between a horizontal walls as shown in fig. 1.1c - the distance between the cylinder and the walls is not something that can be described purely by the properties of the flow itself. In these cases, methods that transfer motions of domain boundaries to the discretised flow domain are of great importance.

In these cases where the motion of boundaries cannot be accounted for by modifying flow properties, we must deform the discretised points in the domain in a sensible way, one that maintains any internal coherence. In the case of a classical mesh, either structured or unstructured, this coherence is explicit - the adjacency of points is described by the edges in the mesh that make up the cells, and the local structure of the points is straightforward, as in fig. 1.2a. In cases of more general point clouds, i.e. those that don't necessarily have any explicit adjacency

[1] Not *all* numerical methods require discretisation of the domain, at least not in space. The nature of digital computation, however, requires discretisation *somewhere* in the solution process.

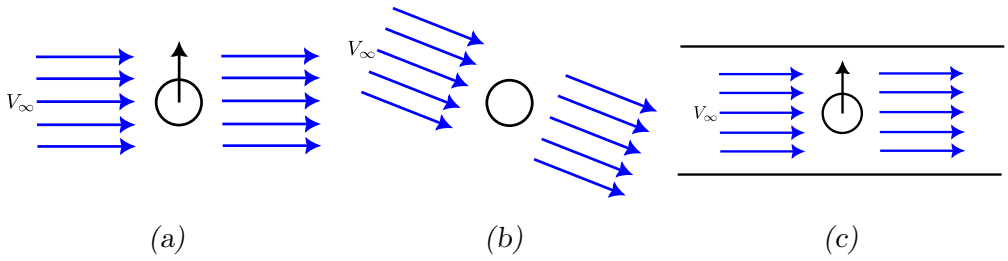


Fig. 1.1: Flows over an oscillating cylinder. Here the upwards motion of the cylinder in fig. 1.1a can be described by modifying the angle of the incoming flow, and leaving the cylinder stationary, as in fig. 1.1b. However the addition of walls means that the cylinder now moves relative to another fixed boundary, hence the upwards motion of the cylinder cannot be effectively described by modifying the properties of the flow alone, and some form of domain deformation is required.

information, as in fig. 1.2b, the maintaining of this coherence is somewhat more difficult to measure, hence the mathematical underpinnings the methods of any point motion must be carefully considered and tested. Often methods that are applicable in the latter case are equally applicable to the former, while methods requiring explicit structural information are often more specific in the use-cases.

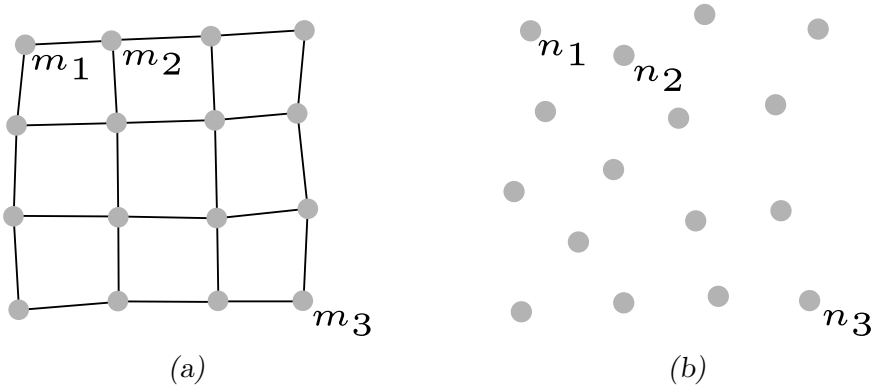


Fig. 1.2: Explicit vs. implicit adjacency. In fig. 1.2a, a classical mesh is shown, where m_1 is next to m_2 in an explicit sense - they are connected via the mesh edge, while m_1 is not next to m_3. In fig. 1.2b, n_1 is next to n_2 in an implicit sense - they are close to each other, while n_1 is far from n_3, but maintaining and quantifying these relationships can be difficult, as distances change when deforming the mesh.

In this work we present methods in RBF mesh motion (see note on the term 'mesh motion' at the beginning of this section), which is a class of techniques using the mathematical framework of radial basis functions, and is applicable to any point cloud - either the points contained within a classical mesh, or the

more general case. Specifically, we present a method by which to improve their efficiency.

A primary limiting factor in the application of numerical methods to physical problems is computational cost. With Moore's Law in decline, the optimisation and simplification of existing algorithms, as well as the development of faster, more efficient methods is of significant interest in engineering and scientific computing.

During the early development of RBF-based mesh motion algorithms, much attention was given to their computational cost, which at first was prohibitive for all but the simplest of cases. In the intervening years, numerous solutions have been developed to improve the scalability of the methods, unlocking the ability to simulate more and more complex problems. As such, the broad applicability of the method of cost reduction presented here will hopefully provide a further boost to a great variety of existing methods.

A more detailed discussion on mesh motion, its various methods, their history, and the current state of the art can be found in section 2.1.1.

1.2.2 Radial Basis Functions

Radial basis functions are a powerful framework for interpolation and approximation. In the context of mesh motion, RBFs allow displacements at a boundary to be interpolated to the rest of the domain.

A radial basis function is any function $\mathbb{R}^n \to \mathbb{R}$ that is radially symmetric, that is, its value only depends on the distance of its input from some fixed point. In a manner similar to a Taylor series that expresses a function in terms of a sum of its derivatives, or a Fourier series that expresses a function in terms of simple trigonometric functions, a function can be expressed via a weighted sum of radial basis functions:

$$f(x) \approx \sum_{i=1}^{n} \alpha_i \phi_i(x). \tag{1.1}$$

Here ψ_i represents a set of n basis functions, with their fixed points at n known values of f, while α_i represents their corresponding weights. The weights can be determined by using the known values of f and requiring that these values be reconstructed exactly, resulting in a simple linear system that can be solved. Once the weights are determined, f can be approximated at any x, by using the representation in eq. (1.1).

In the mesh motion case, we often want to approximate a function describing a smooth displacement field. In this case we have a number of known displacements, usually at the boundaries, which can be used to obtain the weights in eq. (1.1). Once obtained, those weights can be used to determine the displacements at the remaining points in the domain, and the mesh can be deformed appropriately.

For a more detailed description of how RBFs are used in mesh motion, see section 2.2.2.

1.2.3 RBF Finite Difference Methods (RBF-FD)

The importance of finite difference methods in numerical analysis and simulation cannot be understated. The give the ability to transform general continuous, non-linear differential equations into a discrete linear system that can be solved using standard techniques in linear algebra. One of the simplest forms of a finite difference approximation can be seen in the definition of a derivative:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}.$$

If we remove the limit, this gives a finite difference expression that approximates f' at x_0 in terms of f itself:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}. \tag{1.2}$$

Note that we must know the value of f at both x_0 and $x_0 + h$ (note the similarity of this requirement to the discussion of RBF approximation in section 1.2.2 when calculating the weights). In practice, it is more common to know the value of f at a discrete set of points $x_0, x_1, ...$ that are a distance h apart, and eq. (1.2) is expressed as

$$f'(x_n) \approx \frac{f(x_{n+1}) - f(x_n)}{h}.$$

However the difficulty in this traditional approach is that we introduce a hidden adjacency requirement - given a point x_n, we now need to know which point is x_{n+1}. In the 1D case, this is rather unconcerning - we can simply determine x_{n+1} by moving to the right along the numberline. But in a 2D domain, which may not necessarily have this adjacency information, it is not clear how x_{n+1} may be determined from the point set itself.

RBF finite differences alleviate this problem, by having the approximation rely purely on distances between points, rather than any adjacency information. From

the basic RBF approximation expression of eq. (1.1), we can take the derivative of each side to obtain

$$f'(x) \approx \sum_{i=1}^{n} \alpha_i \phi_i'(x).$$

Hence we now have an approximation of f' expressed in terms of the derivatives of radial basis functions, which we often have an explicit expression for, and can calculate ahead of time.

Further background on the RBF-FD method is given in section 4.1.1, with technical details and more examples given in section 4.2.

In the current work, we develop a fluid-structure coupling method for an RBF-FD based fluid flow solver to enable its use in the simulation of FSI problems. The application of RBF-FD methods to FSI problems appears to be limited in the current literature, but due to the strengths of the RBF-FD method, the development of such solvers is desirable, as it provides a simple framework for meshless simulation of FSI problems, with one of the primary advantages of RBF-FD methods being their simplicity in implementation [1].

Until now, most meshless FSI solvers have focused on particle based methods. Approaches to moving boundaries in particle methods are often tightly coupled, often making the FSI aspect complex from an implementation point of view [2]. Section 4.1.1 provides an overview of the history and current state of meshless solvers and FSI, while section 5.1.1. In this work we demonstrate that the RBF-FD method can be coupled with structural dynamics to develop a novel, partitioned FSI solver.

1.3 Thesis Outline

This thesis comprises four main chapters. To keep the chapters as self-contained as possible, each contains its own detailed introduction and background.

1.3.1 Multistage Mesh Motion (Chapter 2)

This chapter develops a framework for reducing computational costs when interpolating the motion of a fluid-structure interface into a large fluid domain point cloud. This framework can be applied to a number of existing methods. The framework involves partitioning of the point cloud of the domain into a hierarchy of subdomains and their boundaries, a structure often inherited from the subdomains or blocks of the chosen CFD solver. In this way, an interpolation

can be applied iteratively through the hierarchy, treating each stage as a new interpolation problem, allowing existing methods to be applied at each individual stage. This results in a modest but non-negligible reduction in the computational cost (given some mild restrictions) of large scale matrix multiplications when updating the volume mesh, both as constant proportional reduction, as well as a reduction in asymptotic complexity, leading to the best savings in dense domains with large deformations and complex interfaces. Examples are given using the existing mesh motion methods of Rendall & Allen [3], and Kedward [4], and some more generally applicable estimates on the maximal reductions are developed for 2D and 3D cases.

1.3.2 Cavity-Store Simulation (Chapter 3)

The multistage point motion of the previous chapter is applied to the simulation of a real-world FSI problem of a vibrating store in an aircraft cavity. The method is implemented in the University of Sydney's in-house CFD solver FLAMENCO coupled with a modal structural solver. The underlying interpolation used is that of Kedward [4]. The simulation is a recreation of wind tunnel experiments conducted at Sandia National Labs, and was funded by the Australian Defence Science and Technology Group (DSTG). The simulation matched well with the experiments, both in the static (non-FSI) case and the coupled (FSI) cases, with the coupled case predicting additional higher order modes seen in the experimental data. The results show that the coupling of fluid and structural solvers is important for the simulation of high fidelity vibrational behaviours. Estimates of the cost of the multistage method over the standard mesh motion show a saving of approximately 64.57% on the volume update step.

1.3.3 Meshless RBF-FD Fluid Solver (Chapter 4)

An RBF-FD solver is developed in MATLAB to simulate unsteady, incompressible, viscous flow using the streamfunction-vorticity formulation of the Navier-Stokes equations, using methods adapted from Flyer et al. [5]. A somewhat novel approach to Dirichlet boundary conditions for vorticity is presented, relying only on boundary normals and using the formulas of Thom and Jensen from traditional (meshed) finite difference methods. This reduces complexity compared to existing methods, e.g. where additional ghost cells are used outside the domain, or structure/adjacency information is required in the near-wall point cloud. A number of test cases are presented to validate the simulation against basic theory as well as cases from the literature, including a lid-driven cavity and square cylinder.

1.3.4 Meshless RBF-FD FSI Solver (Chapter 5)

The meshless RBF-FD fluid solver of the previous chapter is augmented with moving boundaries and structural dynamics to produce a partitioned FSI solver. The solver uses dynamic Dirichlet boundary conditions on the streamfunction coupled with RBF point motion to trace moving boundaries through the domain. Expensive recalculation of RBF-FD coefficients at each time step is required, however with the naturally parallel nature of these computations, the simulation is shown to be able to be reasonably run on modest desktop hardware for domains of sizes comparable to many static cases in the current literature. The square cylinder case is recreated with motion enabled.

1.4 Appendices

In addition to the main chapters, this thesis contains two appendices.

The first (appendix A) contains a brief user guide and some sample inputs for the RBF-FD FSI solver developed in chapters 4 and 5, as well as a full listing of the source code[2].

The second (appendix B) contains a conference paper, *Catastrophe Theoretic Modelling of Hysteresis in Transonic Shock Buffet*, delivered at SciTech2020 [6], wherein RBF neural nets are used to predict the onset and offset of buffet behaviour in aircraft wings. It is provided merely for interest, as it is unrelated to the current work, but highlights the use of RBFs in an alternate context.

[2] Also available at `https://gitlab.com/ajmurra/rbf-fd-fsi`

2. MULTISTAGE MESH MOTION

2.1 Introduction

In this chapter, we present a framework to provide savings when applying mesh motion methods to a moving point mesh, coined Multistage Mesh Motion[1]. The chapter is structured as follows:

Section 2.1 gives some background on the mesh motion problem with previous and current solutions.

Section 2.2 introduces a general description of interpolation-based mesh motion, and describes the RBF method of Rendall and Allen [3], as well as the work of Kedward [4], which was significant in removing difficult scalability barriers traditionally associated with RBF methods.

Section 2.3 gives a framework for applying the methods of section 2.2 in a recursive manner by subdividing domains, which is then shown to have further runtime cost savings, especially for dense meshes and large motions. Some estimates for approximate cost reductions are given, which also provide a guide for subdivisioning technique. A simple toy example to inform the estimates is given.

Section 2.4 gives two more realistic examples, a 2D aerofoil mesh, and a 3D wing mesh, described in [4], and uses them to benchmark the multistage approach against the single stage. Resulting mesh quality is investigated, and some additional caveats are discussed.

2.1.1 Mesh Motion

In many areas of science and engineering, the simulations of dynamic domains and/or boundaries are of great importance. Dynamic problems in fluids, structures, and their combination (fluid-structure interaction), are present in almost every major field of engineering, and broader dynamic PDE problems permeate the physical sciences near completely. As such, the study of techniques for simulating dynamic domains has received much attention in recent decades.

[1] An early description of the Multistage Mesh Motion method was presented at SciTech2019 [7], though the method has progressed significantly since.

Various approaches have looked to leverage mesh generation techniques, with the view to regenerate new meshes as the boundaries moved etc., however in applications where the regeneration must happen frequently (e.g. at every timestep of an FSI problem), it was recognised that the cost, and difficulty of automation, of mesh regeneration was often prohibitive. As such mesh deformation techniques were found to be more promising in these dynamic cases [8].

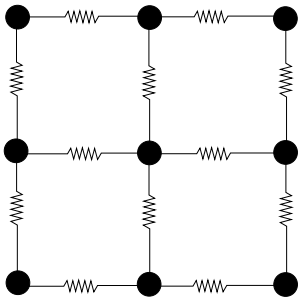
Although generally cheaper than mesh generation, the issue of computational efficiency is still a priority in mesh deformation techniques, and has been a major driving factor in the development of such methods [9]. Other significant challenges in mesh deformation techniques have been related to robustness - how the quality of the mesh is maintained under various deformation regimes ranging from basic rigid-body motions, to highly complex non-linear deformations [9].

In much of the existing literature, mesh deformation techniques are separated into two types - physical analogy, and interpolation. Physical analogy methods rely on interpreting the domain dynamics as some kind of physical phenomenon, which is then able to be solved by techniques specific to the approach (numerical or otherwise), such as regarding the entire domain as a soft-body structure. Interpolation methods apply displacements across the domain by approximating them from known displacements, e.g. taking displacement inversely proportional to distance. The line between the approaches is often not particularly distinct, both at a conceptual level, where the specific physical analogy used can be somewhat removed from the physical presentation of the domain, and at a practical level, since both can often be reduced to similar computational steps (e.g. solving a linear system).

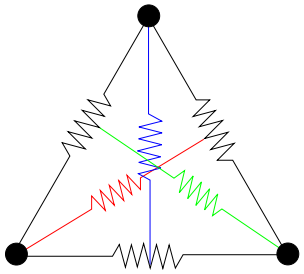
Physical Analogy Methods

An enduring and popular technique firmly within the physical analogy category is that of linear springs, originally developed by Batina [10]. In this method, mesh edges are taken to be linear springs with stiffness inversely proportional to edge length, and connected at the corner points of the mesh elements as shown in fig. 2.1a. Deformation at a boundary corresponds to initial displacement of the springs, and the entire mesh can be deformed by solving the connected system. The conceptual elegance of this approach is quite appealing, however it comes with two major issues. Firstly scalability, since the solution of the connected system will generally have cubic complexity in the number of edges, which becomes quickly intractable. Secondly, robustness or mesh quality, since the solution in 3D and higher dimensions can produce overlapping or inverted elements. Much work has been done to overcome these limitations, particularly the issues around mesh quality, including: adding torsional springs between elements themselves [11]; a semi-torsional method in which the original linear springs are used but with a different method of computing the stiffness [12] (extended to 3D in [13]);

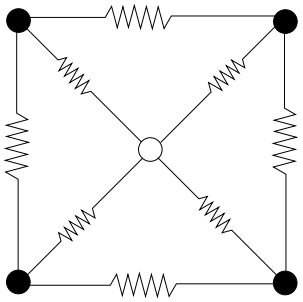
a ball-vertex method (fig. 2.1b) where additional springs are added between each vertex and the opposite face [14]; an ortho-semi-torsional approach where additional springs are added temporarily to inform the stiffness calculation on the original springs [15]; a cell-centre method (fig. 2.1c) where additional springs are added from the centre of each element to its vertices [16].



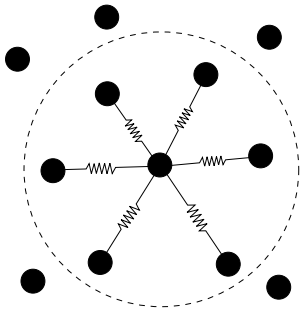
(a) Basic spring analogy - each node is connected to its neighbours as defined by the mesh structure.



(b) Ball-vertex method - each vertex of the cell is connected to an opposing face.



(c) Cell-center method - within each cell, a virtual node is added with springs connected to each vertex.



(d) Mesh-free method - each vertex is connected to all others within a defined radius.

Fig. 2.1: Various spring analogy techniques for mesh motion.

A common theme in modern spring analogy methods is augmenting the original method via additional springs, adjusting the stiffness definition in the original springs, or a combination of the two. Although the significant work done since the original paper has addressed many of the quality and scalability issues of the approach, they often require connectivity information, or are only pertinent to a particular type of mesh or domain discretisation (tets/quads, structured/unstructured etc.), and hence not all methods are universally applicable. More recently a spring analogy method has been applied for adaptive meshless domain refinement by Lashkariani and Firoozjaee [17] wherein they add springs between each element and its neighbours within some predefined radius (see fig. 2.1d), and calculate stiffnesses depending on a test solution. Modifications of this approach may prove useful for motion in both meshed and meshless cases.

Another (possibly even more elemental, at least conceptually) approach is to consider the domain as an elastic material, with potentially varying elasticity

properties throughout. Again using appropriate forcing and/or displacement terms, the elasticity equations can be discretised (usually using an FEA model) and solved to determine the overall displacements. A physical demonstration of the concept can be seen in fig. 2.2, where a dish sponge is marked with a cartesian grid and deformed in different ways. These approaches have the advantage of being able to leverage the massive body of knowledge surrounding structural FEA methods. The methods have been in use in various forms since the early 1990s [18], with a large focus of work since then concerned with the selection of the material properties E (modulus of elasticity) and ν (Poisson's ratio) appearing in the elasticity equations. One approach is to choose a physically valid constant ν and set E inversely proportional to wall distance, or element volume [19]. An alternate approach includes holding E constant and choosing ν as a function of the aspect ratio of the cell [20]. Other authors have proposed constrained optimisation be performed on either E or ν, though this often results in higher computational costs [21]. One significant issue is that of mesh distortion accumulation introduced by the history and path dependent nature of solving the FEA problem using only information at the latest timesteps. This has been studied and addressed more recently by Takizawa [22] via the so-called back-cycle-based mesh moving (BCBMM) method.

(a) Undeformed sponge with cartesian grid marked.

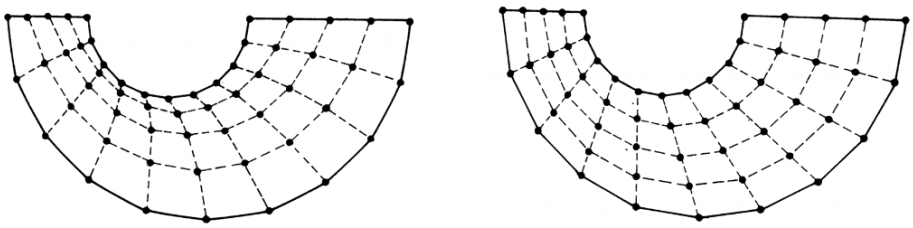
(b) Deforming the top and bottom boundaries of the sponge. Note that the internal grid deforms accordingly.

(c) Even for more complex deformations of the sponge boundaries, the internal grid is still relatively smooth and coherent.

Fig. 2.2: Deforming a rectangular sponge to demonstrate mesh deformation through an elastic material. A standard dish sponge is marked with a cartesian grid, then deformed at the boundaries. Deformations at the sponge boundaries are carried into the internal grid via the material properties of the sponge. Treating a computational domain as an elastic material allows mesh deformation techniques to leverage well-studied structural simulation methods.

A somewhat less studied physical analogy is the use of Laplacian smoothing, in which a modified Laplacian $\nabla(\gamma^p \nabla)$ is used, where γ is a diffusion coefficient (compare this to the hyperviscosity damping term of section 4.3.5). Laplacian smoothing was generalised for mesh applications in a paper by Herrmann [23], who originally used it for grid generation rather than mesh motion (see fig. 2.3), but it has now been applied in various geometric smoothing applications over a

long period of time. Roughly speaking, solutions of this equation seek to minimise the distance of a point from the average position of its neighbours. Much of the study into this method with regard to mesh motion concerns the selection of γ, though the method appears to have seen little use due to practical limitations with the dissipation of higher order modes of deformation, i.e. beyond rigid body motions [9].



(a) (b)

Fig. 2.3: Laplacian-isoparametric method for grid generation, from [23]. In fig. 2.3a, a basic Laplacian method is applied to generate an internal grid with points on the boundary specified. Here we see that the cells close to the boundary are poorly formed, and are heavily skewed. By generalising the equations and introducing an additional parameter, a more desirable result can be seen in fig. 2.3b, where the cells have a much more uniform shape. This particular instance of the general method is referred to as the isoparametric scheme. Such grid generation techniques can be applied to mesh motion by regenerating the mesh at each time step.

Interpolation Methods

The other type of mesh motion methods are interpolation methods. One of the main advantages of interpolation methods is that they are broadly applicable across different domain discretisation strategies, as they generally do not rely on any explicit connectivity information between points in the domain. In addition, interpolation strategies are also often computationally cheaper than physical analogy methods, making them very appealing for many applications. However, the lack of connectivity information means that it can be difficult to enforce direct constraints on mesh quality, and as such the methods often require some level of case-by-case parameter tuning.

An early interpolation method was transfinite interpolation (TFI), originally applied to geometric problems in computer aided design, and first introduced in relation to mesh generation by Gordon and Hall [24]. The method constructs functions on a domain that match a given function on a boundary. In this way, interpolation of boundary displacements can be carried into the interior of the domain, however as no restraints are placed on element integrity, the original method has problems with mesh quality, especially near boundaries and for large deformations. The method is generally only applicable to structured meshes,

and more recently TFI has been applied in combination with other interpolation methods to alleviate some of the mesh quality difficulties while leveraging the lower costs [25, 26]. The work by Ding [25] is of particular interest with regard to the current work, as it is a specific case that can be reinterpreted into the more general multistage mesh motion framework, though the use of TFI limits it to structured Cartesian block meshes.

A number of interpolation methods calculate the displacement at an interior point as some function of distance to the boundary. One such method, known as inverse distance weighting (IDW), was developed by Barier and Keller [27], based on previous work in the geosciences [28]. The method sets the displacement at each interior domain node as a weighted average of displacements at boundary control nodes using a given weighting function (often an inverse squared distance). In this way, the mesh is more rigid at the boundary and more flexible further away. Another boundary distance method (referred to as algebraic damping in [9]) was introduced by Zhao and Forhad [29]. In this method each interior node is assigned the closest boundary node using a modified distance function, based on exponential damping functions, which is bound between 1 at the boundary and 0 far from the boundary. This again gives rigid mesh behaviour close to boundaries, becoming more flexible towards the interior of the mesh. Skewing and overlap is addressed by using what amounts to a Laplacian smoothing method, where node positions are averaged to that of their neighbours [23]. Optimisations to distance based methods were proposed by Luke [30], whereby using a specific form of the distance function similar to those used in n-body problems, tree-code algorithms could be applied to significantly reduce the costs of the method to $\mathcal{O}(n \log n)$ or even $\mathcal{O}(n)$.

An approach using Delaunay triangulation was developed by Liu et al. [31] in which an intermediate mesh can be constructed from the original boundaries and used to interpolate boundary displacements to interior nodes. In this method, a Delaunay triangulation of the boundary nodes is constructed, with each interior node being assigned to a specific element in the triangulation, and its triangular coordinates calculated. The boundaries are then moved, and the interior points are moved in relation to their containing Delaunay element. The method was shown to have reasonable robustness characteristics, since the interior mesh points will not overlap as long as the triangulation remains intact. In cases where the triangulation becomes degenerate, the stage can be divided into smaller and smaller motions, and the triangulation regenerated at each step, to avoid the degeneracy. This splitting and regeneration has the obvious downside of additional runtime costs, as well as requiring some care to apply the splitting part of the algorithm efficiently.

The final and perhaps most used interpolation method in recent years is that of radial basis functions. RBFs have been an important construct for generalising interpolation schemes since they were first developed in the 1970s, and were first applied to the mesh motion problem in 2007 by de Boer and van der Schoot

[32]. The approach was extended by Rendall and Allen [3] to unify mesh motion with the reverse problem of fluid-to-structure interpolation. One of the main drawbacks of RBFs in the original method was the prohibitive cost of calculating the matrix connecting the surface to the volume, which included the requirement to invert an $n_s \times n_s$ matrix where n_s was the number of surface points. Not only was this matrix often large in many cases, but also poorly conditioned due to the scales of node separation vs the global surface geometry. This meant that the interpolations in complex cases could suffer from numerical artifacting. This problem was addressed by a number of works using reduced subsets of surface nodes and localised implementations [33, 34] with the work of Kedward, Allen, and Rendall [4] providing a highly scalable approach where the radii on the surface mesh were modified to give the offending matrix structural properties so that solving the system was reduced to a small inversion plus a back substitution. As further cost savings to RBF methods are the focus of the current work, the methods are described in more detail in section 2.2.

Meta Methods

A number of other developments in mesh motion are not necessarily mechanisms for computing mesh deformations directly, rather they use the physical analogy and interpolation methods in a novel way to improve performance or robustness, which here we will refer to as *meta methods*. As the current work is essentially a meta method - a general multistage scheme that could feasibly be applied to a good proportion of the mesh motion methods discussed here - we distinguish these techniques from the direct methods.

In a 2002 NASA technical note [35], Samareh noted the lack of use of boundary rotation information in existing deformation schemes. To address this, he proposed the addition of quaternions that could trace boundary rotation in addition to displacement. These quaternions could be propagated in a similar fashion to displacements, and hence either interpolated (e.g. TFI) or physical analogy methods (e.g. springs) could be used to apply the rotations. The cost of propagating the quaternions is of course dependent on the chosen method. Further work on the calculation of the quaternions was done by Maruyama [36], where a Laplacian smoothing was used to propagate both translation and rotational information into the domain. It is still not common to use boundary rotations in mesh motion schemes, however for high fidelity cases where mesh quality is of crucial importance, it may improve the results of various other methods using translation alone. An example of complex mesh deformation is given by Maruyama [36], in which the mesh around a standard NACA23012 aerofoil is mapped to a complex surface representing ice-accretion at the leading edge. In the example, the quaternion method is seen to be superior in maintaining orthogonality near the boundary compared to translation only methods, as seen in fig. 2.4 when compared to an integral method. Mesh orthogonality is critical

for the accurate calculation of near-wall effects in CFD, hence any method that provides advantage in this area is of great interest.

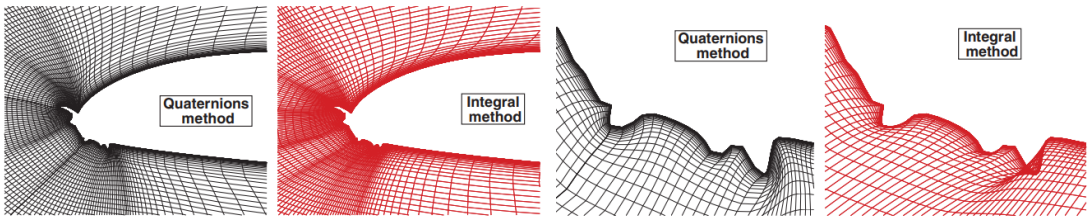


Fig. 2.4: Quaternion vs. integral method for mesh motion/deformation taken from [36]. This shows global and detail of ice-accretion on the leading edge of a NACA23012 aerofoil. Note that the quaternion method is superior in maintaining orthogonality in the near-boundary cells, with the integral method even producing an overlapping mesh (i.e. inverted cells) in one portion of the surface.

To maintain mesh quality in detailed viscous boundary meshes, where aspect ratios may be on the order of 10^6, McDaniel and Morton [37] proposed a two phased approach where the boundary layer mesh was deformed rigidly with the moving surface, and the displacements carried to the outer regions using any chosen mesh motion scheme. In their study, they investigated the use of the Delaunay triangulation method of Liu [31], as well as a custom hybrid of works by Melville [38] and Allen [39], although the method could reasonably be applied using a number of other mesh motion schemes.

2.2 Single Stage

In this section we give a brief, general example of mesh motion, and introduce the RBF mesh motion schemes that we will use for the remainder of the thesis.

2.2.1 General Mesh Motion

In their most general form, point-cloud based mesh motion interpolation schemes operate on two subsets of the domain - a set of points with known displacements prescribed by the simulation (e.g. static boundary points or points on a structural model), and a set of points at which the displacement is to be approximated. Specific mesh motion algorithms may call for further subdivisions of these sets, but for clarity when describing the method, we work in the more general framework.

We refer to the points with prescribed displacement as the *control points*, denoted x_i, and the points at which the displacement is to be calculated as the

interpolation points, denoted y_i. A single stage of the mesh motion algorithm consists of

1. calculating or gathering the displacements Δx_i at the control points
2. moving the control points, i.e. $x_i \to x_i + \Delta x_i$,
3. calculating the displacements Δy_i at the interpolation points,
4. moving the interpolated points, i.e. $y_i \to y_i + \Delta y_i$.

In practice, steps 2 and 4 are often trivial, and step 1 is usually the result of a structural solver. Each step may have a number of sub-steps, most notably the calculation of the displacements Δy_i in step 3. The calculation of Δy_i is usually the main concern of any given interpolation method, and will be the focus here.

Interpolation schemes operate on the known quantities x_i, y_i and Δx_i to produce a function (implicit or explicit) $f : \mathbb{R}^n \to \mathbb{R}^n$, usually with the restriction $f(x_i) = \Delta x_i$, that generates interpolated displacements at each y_i, i.e. $f(y_i) = \Delta y_i$. Clearly f is not unique without additional restrictions, which will vary depending on the interpolation scheme used and the problem at hand.

As an example, consider the case of an oscillating 2D cylinder with a surrounding mesh shown in fig. 2.5. Assume that the boundary points of the cylinder are being moved as a single rigid body in a predictable sinusoidal motion, and that the points at the exterior boundary of the mesh are to be fixed in space. In this case, the boundary points of the cylinder, as well as those at the exterior boundary of the mesh, are the control points x_i, as we can calculate their displacements directly. The remaining points are the interpolation points y_i at which we wish to calculate the corresponding displacement, given the motion of the control points. The exact requirements of this interpolation will usually be problem specific, but for the case of this example, assume that we have the intuitive (but surprisingly illusive and at times ill-defined) idea that the overall structure of the point cloud should not change in any significant way - points above the cylinder should remain above it, points fore and aft should stay that way etc.

We begin by moving each point x_i of the cylinder by a corresponding Δx_i, as per fig. 2.6a. That is, we set $x_i \to x_i + \Delta x_i$. After this is complete, we move the internal domain nodes y_i in a corresponding manner, as shown in fig. 2.6b. As we can see, the domain nodes have maintained an intuitively 'similar' structure to the original domain.

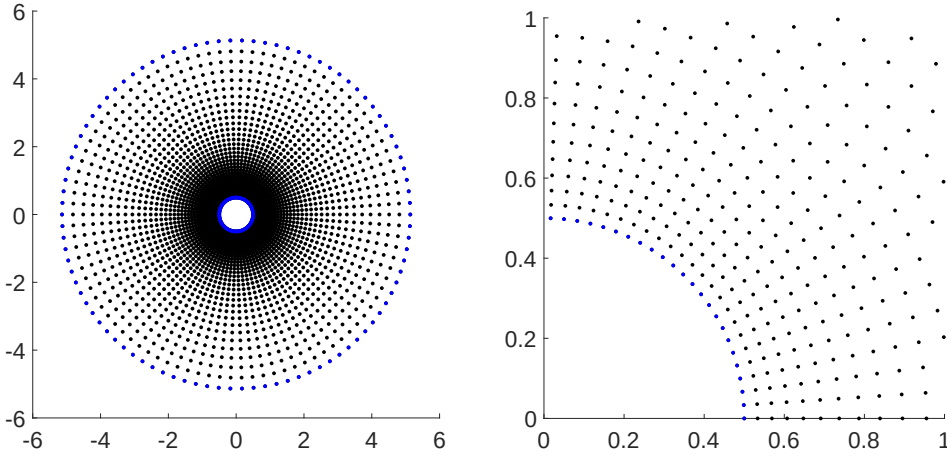
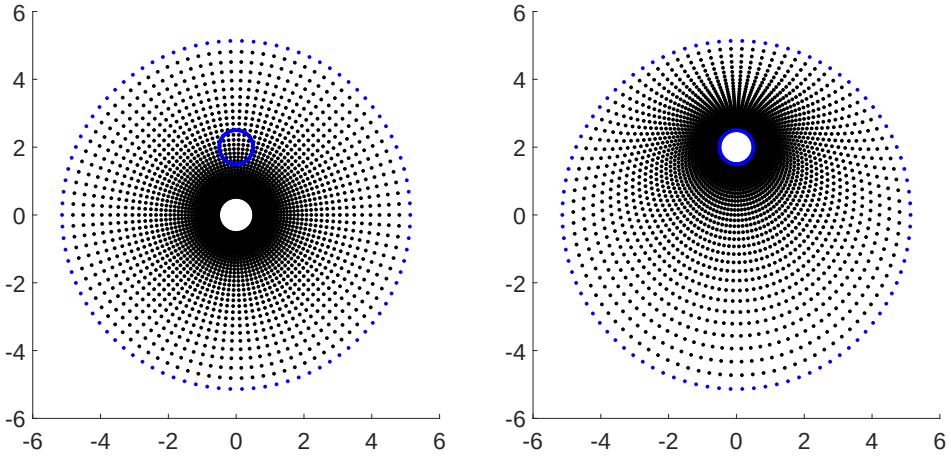


Fig. 2.5: Point cloud around cylinder, with detail. Control points (x_i) are highlighted blue, interpolation points (y_i) are in black.



(a) Movement of cylinder. *(b)* Movement of domain nodes.

Fig. 2.6: Control point and interpolation point motion.

2.2.2 *RBF Mesh Motion (de Boer et al. [40], Rendall & Allen [3])*

General RBF mesh motion was first described by de Boer et al. in [40], with significant work on the topic done by Rendall and Allen [3], who extended the method to include fluid-to-structure interpolation with no additional complexity. It should be noted that as many splining methods can be framed in terms of RBFs, there does exist a number of works that can be considered much earlier examples of RBF mesh motion, e.g. Hounjet and Meijer [41] or Spekreijse et al. [42].

Here we follow the detailed derivation of [3], which includes some additional use of the matrix structure to aid in solving the resulting system.

The problem is to find a function $F : \mathbb{R}^3 \to \mathbb{R}^3$ that maps displacements at the

control nodes to displacements at the volume nodes. In this case, we assume that each direction can be handled individually, and that it can be represented as a sum of radial basis functions, i.e.

$$F(\mathbf{x}) = \begin{pmatrix} F_x(\mathbf{x}) \\ F_y(\mathbf{x}) \\ F_z(\mathbf{x}) \end{pmatrix}$$

where each of F_x, F_y and F_z are given by

$$F_i(\mathbf{x}) = \sum_{j=1}^{n_c} \alpha_j^i \phi(\|\mathbf{x} - \mathbf{x}_j\|) + p^i(\mathbf{x}) \tag{2.1}$$

where n_c is the number of control nodes. Here $\|\cdot\|$ is the chosen norm on $\mathbb{R}^3$ and $p^i : \mathbb{R}^3 \to \mathbb{R}$ is a polynomial which allows for exact reconstruction of rigid body motions and guarantees uniqueness of the interpolation. For purposes of clear exposition, we consider the p^i to be linear (degree 1, or $\deg(p) = 1$), i.e.

$$p^i(\mathbf{x}) = \gamma_0^i + \gamma_1^i x + \gamma_2^i y + \gamma_3^i z,$$

but the same derivation can be used with polynomials of arbitrary degree.

From above, we see that the coupling problem involves three individual interpolation problems - one in each of x, y and z coordinates.

Given the structure of eq. (2.1), the problem is now to determine the coefficients α_j^i as well as the coefficients of the polynomial p^i. Hence we have $n_c + \deg(p) + 1$ unknowns. We obtain the first n_c equations by requiring the exact reconstruction of the control node positions, i.e.

$$F(\mathbf{c}) = \mathbf{c} \tag{2.2}$$

for every control node $\mathbf{c}$. The remaining $\deg(p) + 1$ equations are obtained by the condition

$$\sum_{j=1}^{n_c} \alpha_j^i q(\mathbf{x}_j) = 0 \tag{2.3}$$

for all polynomials q with $\deg(q) \leq \deg(p)$. This condition is discussed in more detail in section 2.2.3. Looking more closely at eq. (2.2), we have

$$\mathbf{c}_k^i = \alpha_1^i \phi(\|\mathbf{c}_k - \mathbf{c}_1\|) + p^i(\mathbf{c}_1) + \cdots + \alpha_{n_c}^i \phi(\|\mathbf{c}_k - \mathbf{c}_{n_c}\|) + p^i(\mathbf{c}_{n_c})$$

for $1 \leq k \leq n_c$. Hence

$$\begin{pmatrix} \mathbf{c}_1^i \\ \vdots \\ \mathbf{c}_{n_c}^i \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{c}_1^x & \mathbf{c}_1^y & \mathbf{c}_1^z & \phi_{1,1} & \cdots & \phi_{1,n_c} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{c}_{n_c}^x & \mathbf{c}_{n_c}^y & \mathbf{c}_{n_c}^z & \phi_{n_c,1} & \cdots & \phi_{n_c,n_c} \end{pmatrix} \begin{pmatrix} \gamma_0^i \\ \vdots \\ \gamma_3^i \\ \alpha_1^i \\ \vdots \\ \alpha_{n_c}^i \end{pmatrix} \tag{2.4}$$

where $\phi_{m,n} = \phi(\|\mathbf{c}_m - \mathbf{c}_n\|)$. Turning then to the condition of eq. (2.3), we have

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 1 \\ \mathbf{c}_1^x & \cdots & \mathbf{c}_{n_c}^x \\ \mathbf{c}_1^y & \cdots & \mathbf{c}_{n_c}^y \\ \mathbf{c}_1^z & \cdots & \mathbf{c}_{n_c}^z \end{pmatrix} \begin{pmatrix} \alpha_1^i \\ \vdots \\ \alpha_{n_c}^i \end{pmatrix}. \tag{2.5}$$

Combining eqs. (2.4) and (2.5) into a single equation gives

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \mathbf{c}_1^i \\ \vdots \\ \mathbf{c}_{n_c}^i \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & \cdots & 1 \\ 0 & 0 & 0 & 0 & \mathbf{c}_1^x & \cdots & \mathbf{c}_{n_c}^x \\ 0 & 0 & 0 & 0 & \mathbf{c}_1^y & \cdots & \mathbf{c}_{n_c}^y \\ 0 & 0 & 0 & 0 & \mathbf{c}_1^z & \cdots & \mathbf{c}_{n_c}^z \\ 1 & \mathbf{c}_1^x & \mathbf{c}_1^y & \mathbf{c}_1^z & \phi_{1,1} & \cdots & \phi_{1,n_c} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{c}_{n_c}^x & \mathbf{c}_{n_c}^y & \mathbf{c}_{n_c}^z & \phi_{n_c,1} & \cdots & \phi_{n_c,n_c} \end{pmatrix} \begin{pmatrix} \gamma_0^i \\ \vdots \\ \gamma_3^i \\ \alpha_1^i \\ \vdots \\ \alpha_{n_c}^i \end{pmatrix}. \tag{2.6}$$

Using similar notation to existing literature [3, 43], we will write eq. (2.6) as

$$\mathbf{S}^i = A_{cc} b^i.$$

The subscript in A_{cc} indicates a control-to-control matrix. Since the augmented control node vector $\mathbf{S}^i$ is known, we can now solve for the unknown coefficient vector b^i, which is given by

$$b^i = A_{cc}^{-1} \mathbf{S}^i.$$

We have now determined all the unknown coefficients of eq. (2.1), and have an effective interpolation function F. However the computation of F for every node at runtime is not ideal. It is also not clear exactly how to interpolate the displacements of n_v volume nodes from the displacements of n_c control nodes. As such, we now wish to develop a structure to fluid coupling matrix H, such that we can map from the displacements at control nodes to displacements at

volume nodes by

$$\begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{n_v} \end{pmatrix} = H \begin{pmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_{n_c} \end{pmatrix}. \tag{2.7}$$

To determine H, we first calculate the matrix of the positions of the volume points as given by eq. (2.1), i.e.

$$F_i(\mathbf{v}) = \alpha_1^i \phi(\|\mathbf{v}_k - \mathbf{c}_1\|) + p^i(\mathbf{c}_1) + \cdots + \alpha_{n_c}^i \phi(\|\mathbf{v}_k - \mathbf{c}_{n_c}\|) + p^i(\mathbf{c}_{n_c})$$

which we can write as

$$\begin{pmatrix} \mathbf{v}_1^i \\ \vdots \\ \mathbf{v}_{n_a}^i \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{c}_1^x & \mathbf{c}_1^y & \mathbf{c}_1^z & \phi_{1,1} & \cdots & \phi_{1,n_c} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{c}_{n_a}^x & \mathbf{c}_{n_a}^y & \mathbf{c}_{n_a}^z & \phi_{n_a,1} & \cdots & \phi_{n_a,n_c} \end{pmatrix} \begin{pmatrix} \gamma_0^i \\ \vdots \\ \gamma_3^i \\ \alpha_1^i \\ \vdots \\ \alpha_{n_c}^i \end{pmatrix}$$

where here $\phi_{m,n} = \phi(\|\mathbf{v}_m - \mathbf{c}_n\|)$. We call this matrix A_{vc}. We see then that

$$\mathbf{v}^i = A_{vc} b^i = A_{vc} A_{cc}^{-1} \mathbf{S}^i = H \mathbf{S}^i.$$

At this point, we have completely determined the coupling matrix H, however since A_{cc} has a zero block in the top left, we can reduce computations involved in calculating the inverse of A_{cc}. We subdivide A_{cc} into the block matrices P and M such that

$$A_{cc} = \begin{pmatrix} 0 & P \\ P^T & M \end{pmatrix}.$$

Then using a Schur complement method [44], we find

$$A_{cc}^{-1} = \begin{pmatrix} (PM^{-1}P^T)^{-1} PM^{-1} \\ M^{-1} - M^{-1}P^T(PM^{-1}P^T)^{-1}PM^{-1} \end{pmatrix}.$$

In general, the Schur complement reduces complexity of inverting a block matrix to the complexity of inverting the major diagonal blocks individually. As A_{cc} contains a zero block on the major diagonal, the complexity is reduced to inverting M.

We can then write the coupling matrix H as

$$H = A_{vc} A_{cc}^{-1} = A_{vc} \begin{pmatrix} (PM^{-1}P^T)^{-1} PM^{-1} \\ M^{-1} - M^{-1}P^T(PM^{-1}P^T)^{-1}PM^{-1} \end{pmatrix}$$

with the interpolation from structural displacements to aerodynamic displacements given by eq. (2.7). At this stage, the construction of H has made it clear that it is the correct coupling matrix in the control to volume direction.

2.2.3 Polynomial Term

The role of polynomial terms in RBF interpolation techniques is a complex topic (e.g. an excellent 3-part series of papers by Flyer et al. [45, 46, 47] is available on their role with regards to RBF finite difference methods), and differs somewhat depending on application. For the case of mesh motion, the purpose of the polynomial term in eq. (2.1) is twofold.

Firstly, the additional requirement of eq. (2.3) ensures that simple transformations such as translations and rotations can be reconstructed exactly. This is especially important in cases with large displacements. Equation (2.3) also ensures that total forces and moments are conserved when applying the method in the fluid-to-structure direction in an FSI solver [3]. To see this, consider the thin plate spline of Harder and Desmarais [48], which has its roots in aeroelasticity. In their interpretation, the interpolation is found by fitting a flat plate to a given set of points by applying a number of point loads to gain the appropriate deflections. Then eq. (2.3) is simply an equilibrium condition relative to the fitted points, i.e. any balanced (unbalanced) forces applied to the known points will continue to be balanced (unbalanced) over the entire interpolation.

Secondly, in the case that our basis function ϕ has the additional property of conditional positive definiteness (definition 2.1), it serves to ensure uniqueness of the interpolation. This result is not obvious, and comes from a result of Madych and Nelson [49]. To begin, we give a definition and subsequent result related to the structure of our basis functions ϕ.

Definition 2.1 (Conditionally Positive Definite Function)

A radially symmetric function $\phi : \mathbb{R}^n \to \mathbb{R}$ is called conditionally positive definite of order m if for all $N \in \mathbb{N}$, sets of points $\{x_1, \ldots, x_N\} \subset \mathbb{R}^n$, and $\alpha \in \mathbb{R}^N \setminus \{0\}$ that satisfy eq. (2.3) for all polynomials q of degree $\leq m$, we have

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \phi(x_i - x_j) > 0. \tag{2.8}$$

The theorem of Madych and Nelson is then stated as follows.

Theorem 2.1 (Uniqueness of RBF Interpolation)

Let $\phi : \mathbb{R}^n \to \mathbb{R}$ be conditionally positive definite of order m and $X = \{x_1, \ldots, x_N\} \subset \mathbb{R}^N$ be such that the only polynomial of degree $< m$ that vanishes on X is the zero polynomial. Then there exists a unique function F_i of the form eq. (2.1) that satisfies eqs. (2.2) and (2.3).

A more gentle proof of this result can be found in Chapter 8 of Wendland's text *Scattered Data Interpolation*[50].

An important property of conditionally positive definite functions is that a conditionally positive definite function of degree m is also conditionally positive definite of degree n for any $n > m$. This fact follows almost immediately from definition 2.1.

Proposition 2.1 (Increased Degree of Conditional Positive Definite Functions)

Suppose ϕ is conditionally positive definite of order m. Then ϕ is conditionally positive definite of order n for all $n > m$.

Proof. Since ϕ is already conditionally positive definite of order m, let the set of $\alpha \in \mathbb{R}^N$ that uphold eq. (2.3) be given by A_m. Increasing the allowable degree of the polynomials in eq. (2.3) reduces the number of permissible α to a set $A_n \subset A_m$. Then since eq. (2.8) is true for all $\alpha \in A_m$, then it clearly continues to hold for all $\alpha \in A_n$. Hence ϕ is conditionally positive definite of degree n. ⫤

In particular, given theorem 2.1, this means that we can increase the degree of the polynomial term in eq. (2.1) to arbitrary degree so as to exactly capture higher order polynomial transformations. The only restriction on this increase is the number of structural points we are interpolating from, which must be sufficient to ensure we can solve for the appropriate coefficients. For the applications presented here, we consider the space of polynomials in $\mathbb{R}^3$ with degree at most m. This space has dimension $\binom{m+3}{m}$, hence we need at least this many structural points to ensure our interpolation is solvable.

Conditional positive definiteness has been verified for Wendland's basis functions, which were generated by repeated applications of an integral operator I to the truncated power function (x^n set to zero for $x < 0$) [50], where I is defined on radial functions by

$$I(\phi)(r) := \int_r^\infty \phi(x)\, dx.$$

Wendland shows in [51] that these basis functions are in fact of minimal degree, hence are in a some sense amongst the best candidates for the RBF mesh motion method. As such, for mesh motion in the current work, we prefer to use

Wendland's functions (usually C2), as do many other works on the subject. In later chapters using RBF methods for approximating derivatives, we will prefer polyharmonic spline bases, which are also (strictly) conditionally positive definite [45].

2.2.4 Multiscale Mesh Motion (Kedward [4])

The primary limiting factor of the method of section 2.2.2 has been the inversion of the A_{cc} matrix. The problem can be slightly reduced using the Schur complement method as described, however the size of the M block is still usually considerable, with dimension on the order of the number of control nodes. The conditioning of this matrix is often poor, and even when conditioned correctly via preconditioning methods or otherwise, the computation of the inverse is often prohibitively expensive, and/or numerical limitations produced poor results. A number of techniques were developed using reduced sets of control nodes to shrink the M matrix, however the methods still required significant preprocessing and at each timestep, and an additional corrective step to reduce error from the displacements from the reduced set of control nodes.

Kedward addresses the issues of both conditioning of the matrix and the inexactness of using a reduced set approximation by beginning with a significantly reduced set of control nodes, then adjusting the radii of each of the remaining control nodes to give a lower diagonal structure to the remainder of the matrix. This then allows the system to be solved first by a small matrix inversion for the base set (as few as 1% of the original control set), followed by forward substitution for the remainder of the control nodes. The algorithm for doing so is as follows:

1. Add the sparse base set of control nodes to an *active set*, with a predetermined radius, and all remaining control nodes to an *inactive set*.
2. Determine inactive node with greatest distance to active set.
3. Remove node from inactive set and add to active set, setting its radius to the distance calculated in step 2.
4. If inactive set is non-empty, return to step 2, otherwise, stop.

Once this procedure has been completed, the matrix A_{cc} from section 2.2.2 now has the structure

$$A_{cc} = \begin{pmatrix} A_{bb} & 0 \\ A_{br} & A_{rr} \end{pmatrix}$$

where subscript b refers to base points and r refers to the refinement points added from the inactive set. Here A_{bb} is typically a full, small, symmetric matrix, A_{br} is a full rectangular matrix, and A_{rr} is lower triangular. The inverse of A_{bb} can then be calculated ahead of time, and the relatively cheap forward substitution can be completed per timestep to solve the remainder of the system. The ill-conditioning and prohibitive size of the matrix inversion is avoided by selecting an appropriate base set, and the approximation error of using a reduced set of control nodes is avoided by including all remaining points included in the system, hence recovering the exact displacements at *all* control nodes.

From [4], the computational cost for updating the volume mesh using this method is

$$\alpha N_v N_b,$$

where α is the proportion of volume points that are captured within the radii of the base set of control nodes. This cost is indicative of interpolation methods in general, as the update step will usually involve the calculation of displacement at each volume point as a function of the control nodes. For large deformations, or those contained within more complex geometry (e.g. the cavity store of chapter 3), αN_v can still be large, and although the surface-to-volume update step is perfectly parallelisable in the RBF case, the costs can still be significant, especially over the course of long simulations.

As such, the next section will present a general framework for applying mesh motion methods in a multistage fashion to reduce this cost further, without sacrificing the ability to parallelise, or having any adverse effects on mesh quality. We will use both the full RBF method and the multiscale method described here as exemplary methods to benchmark the multistage concept.

2.3 Multistage

We now turn to combining multiple single stages into a mesh motion framework. This method can yield savings in both the storage and computational costs of applying mesh displacements. The focus of this work is to reduce the costs of applying *existing* mesh motion methods outright, rather than to address other limiting factors, e.g. parallelisation (often already a built-in feature of modern interpolation methods), or mesh quality. We use the RBF methods described in section 2.2 as our techniques at each individual stage, as these methods are state-of-the art, and are the most widely applicable methods currently available.

Previous work by Wong et al. [52] has alluded to a similar procedure for parallel mesh motion on multiblock grids, where the motion of a control set is translated firstly to block corners, then to block surfaces, and finally to block volumes. This work was completed prior to the advent of modern interpolation (RBF and

otherwise) methods, and used a combination of spring analogy and transfinite interpolation methods to complete the grid displacement, leveraging the relative advantages of each. Although it presented an effective means by which to parallelise a structured, multiblock mesh, it required a number of blending and smoothing operations to deal with arbitrary motions at block interfaces, irregular surface deformations, as well as requiring adjacency information on both the point cloud and block structure. As such, the method was not readily generalisable to arbitrary mesh motion applications. Similar early work was done by Jones et al. [53]. The current method alleviates such issues.

The method presented here relies on recursively dividing the domain into a hierarchy of subdomains, with the motion of points in each subdomain determined by a reduced set of points above it in the hierarchy. In practice, the reduced set of points will generally consist of the boundary of the subdomain, and in this way, the method is conceptually similar to the Craig-Bampton method [54] for reducing costs of solving FEA systems by treating a structure as a collection of substructures, whose behaviours are completely determined by the substructure boundaries. Figure 2.7 demonstrates this concept adapted to the mesh motion case - the effect of the motion of a control surface C on a volume V can be determined entirely by the effect of the control surface on the boundary ∂V of V.

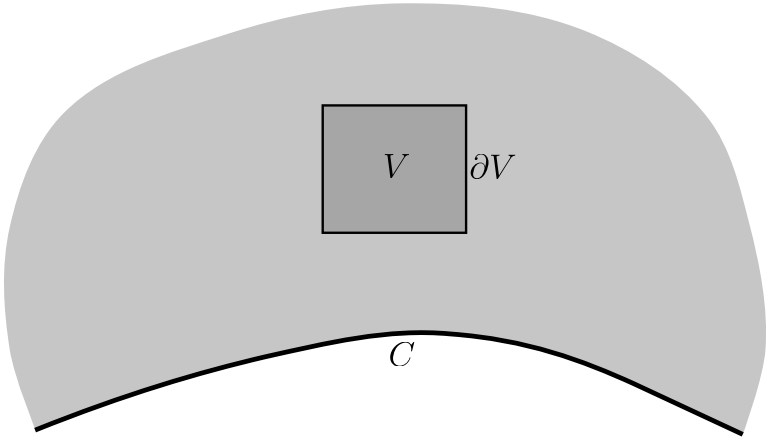


Fig. 2.7: The effect of motion of the control surface C on the domain volume V can be determined by the effect of C on the boundary ∂V of V.

2.3.1 Partitioned Domain Hierarchy

In this method, each single stage (see section 2.2) of the mesh motion algorithm now exists within an acyclic digraph structure, describing the dependencies of the stages. Each vertex represents a single stage pass, and each edge represents a dependency, with the direction showing the program flow. A stage cannot be computed until all the stages above it are complete. Figure 2.8 shows an

example structure with 13 stages. In this example, stage 1 must be completed before stages 3 and 4, which can be computed in parallel; stage 5 must wait for both stage 1 and 2 to complete, and so on.

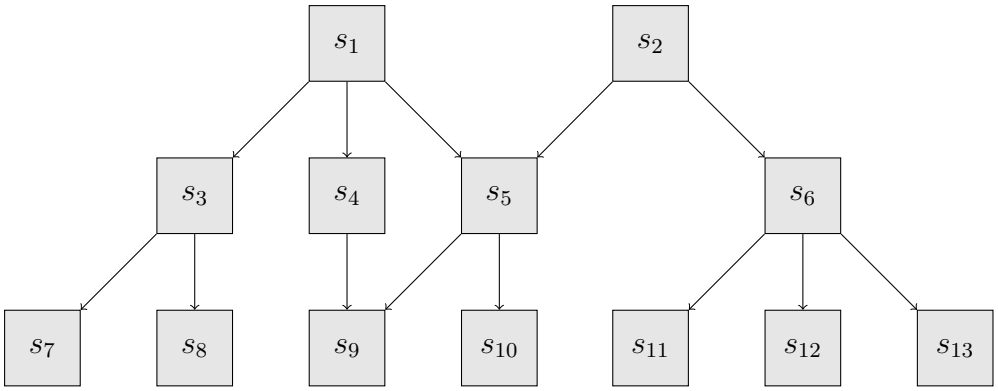


Fig. 2.8: Example of the acyclic digraph dependency structure.

Let this digraph structure be denoted in a somewhat non-standard way by using a single set S of vertices $\{s_1, ..., s_n\}$. Since the vertices here represent stages, we prefer S over the standard V. Building some additional structure into the graph S, define a vertex/stage s_i to be a 3-tuple $s_i := (\alpha_i, \beta_i, \gamma_i)$, with α_i and β_i the sets of points in the domain that we are interpolating values from/to[2], and $\gamma_i := \{s_j | \alpha_i \cap \beta_j \neq \emptyset\}$, i.e. γ_i is the set of stages whose output contains points in the input of s_i. This defines γ_i as the set of parent stages of s_i, e.g. $\gamma_5 = \{s_1, s_2\}$ in fig. 2.8.

This construction is clearly useful for exposition, but is also useful in a practical sense - ordering the domain in this way presents a straightforward method to determine the order of the stages by traversing the digraph structure from root nodes (stages without parents), and in particular, it determines which stages can be completed in parallel. As such, a further practical requirement is

$$\bigcap_{s_i \in S} \beta_i = \emptyset$$

which is the property that no two stages should output an interpolation at the same point. This avoids the potential for conflicting parallel calculations.

In the simplest case where S consists of a single stage, i.e. $S = \{s\}$, it would be usual that α is the set of all points with known displacements, and β is the volume points at which we wish to interpolate the displacements to.

[2]Note that α_i and β_i are not the input and output displacements of the mesh motion stage, rather the points at which these displacements occur.

2.3.2 Elementary Example

To give an initial demonstration of how the graph of section 2.3.1 is constructed, and how a multistage approach can provide savings on the storage and calculation of displacements within a domain, we present the following toy example as a building block towards more general cases.

Consider a square, 2D domain consisting of 4 subdomains and their boundaries as shown in fig. 2.9.

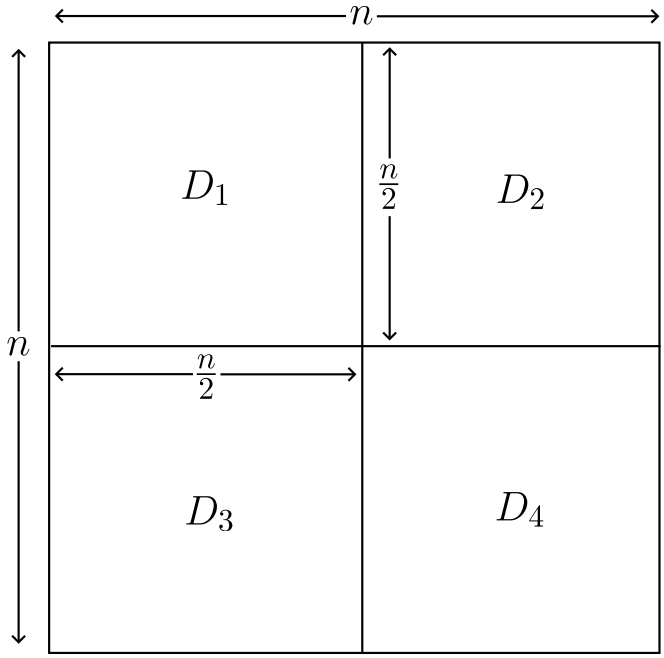


Fig. 2.9: Basic domain with 4 subdomains.

Let n be the number of points along one side of the domain, thus each subdomain has a perimeter and volume approximately $2n$ and $n^2/4$ points respectively, with the total domain having perimeter and volume of $4n$ and n^2.

For ease of computation, we take a geometric approach and ignore zeroth and first order terms, e.g. a domain with n points along each side will in reality have closer to $n^2 - 4n + 4$ points on the interior rather than simply n^2. However as we will see, cost savings will generally only appear as factors on the highest order terms, so it is reasonable to ignore lower orders for n above some sensible threshold, and indeed the estimates obtained by doing so are conservative.

In the single stage approach, using a simple RBF interpolation method for simplicity, the interpolation of a value on the exterior boundary of the domain to the entire volume will require the storage of an $n^2 \times 4n = 4n^3$ element matrix linking the boundary to the interior. Assuming the value we want to interpolate is a single scalar (e.g. deformation in a single direction), then the computational

cost of applying the interpolation will require the multiplication of this matrix by a 1D vector, which requires $n^2 \times 4n \times 1 = 4n^3$ each of multiplication and addition floating point operations[3]. Since this aligns with the storage costs, we can consider the two as direct proxies of each other.

If we instead apply the interpolation in stages we find the cost to be reduced.

Beginning by constructing the domain hierarchy, the various α and β of section 2.3.1 are shown in fig. 2.10. Constructing the γ sets, this results in the

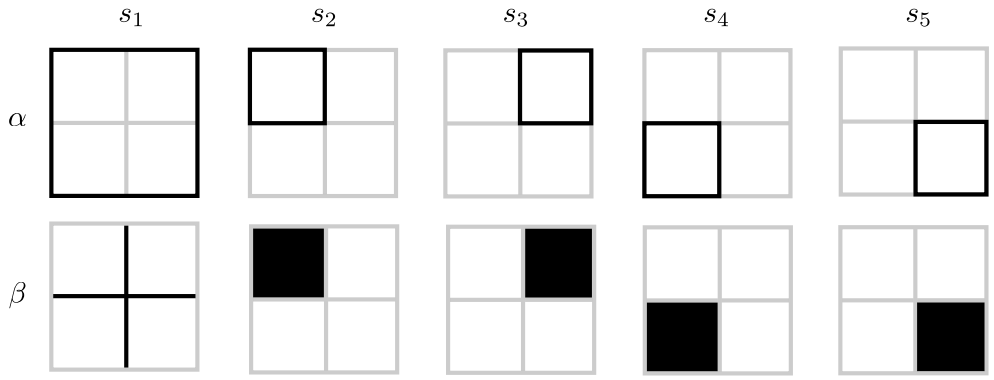


Fig. 2.10: α and β definitions for the 5 stages in 4-subdomain example.

straightforward hierarchy as shown in fig. 2.11. We see that we must perform 5 individual stages of mesh motion, with stages 2-5 able to be performed in parallel after the completion of stage 1.

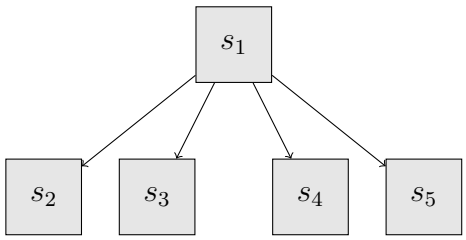


Fig. 2.11: Acyclic digraph dependency structure for 4-subdomain example.

Stage s_1: The interpolation is performed from the $4n$ points on the exterior perimeter to the $2n$ points on the remaining interior perimeters, which has a cost of $2n \times 4n = 8n^2$.

Stages $s_2 - s_5$: For each subdomain, the interpolation is performed from the $2n$ points on the perimeter to the $n^2/4$ points on the interior, with a cost of $n^2/4 \times 2n = n^3/2$. This is repeated 4 times for a cost of $4 \times n^3/2 = 2n^3$.

[3] Again some computational liberty is taken here with lower order terms, in reality only $n^2 \times (4n - 1) \times 1$ additions are required.

Hence the total cost of performing the multistage interpolation is $8n^2+2n^3$. This is a halving of the cost in the dominant term compared to the single stage ($4n^3$), hence although the asymptotic complexity remains unchanged, i.e. $\mathcal{O}(n^3)$, the cost is reduced for practical n. Of course this case serves only to provide example, and is not necessarily indicative of more complex cases, but as is demonstrated in the next section, it will be possible to choose subdivisioning in such a way that we not only will see a constant factor saving, but a reduction in order of complexity as well.

2.3.3 General Estimates

In studying savings in both storage and computation more generally, it is useful to look at two modes of application of the multistage interpolation described in section 2.3:

1. Widening the graph by splitting the domain into additional independent subdomains

2. Deepening the graph by recursively splitting subdomains into further subdomains.

Here we extend the example of the previous section to provide some rough indications of expected cost savings more generally. Of course each practical application is unique and is difficult to estimate exactly, but for initial estimates, these will suffice.

Condition for Cost Reductions

As a first step, we establish a condition on when a multistage approach will yield any savings, before proceeding to give estimates on the actual savings themselves. Assume we have a two stage graph as shown in fig. 2.12. This model will give us generality, as any graph (beyond the trivial single stage graph) can be considered two stages at a time for the purposes of cost estimates.

For the two stage model to be more cost-efficient than the single stage, we have the condition

$$|\alpha_1||\beta_1| + |\alpha_2||\beta_2| < |\alpha_1||\beta_2|. \tag{2.9}$$

This uses the assumption that the update cost of a single stage s_i is proportional to $|\alpha_i||\beta_i|$, which is the case for both the base method and the multiscale (does not include initial costs, see section 2.3.3).

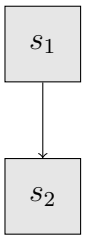


Fig. 2.12: Two stages of the multistage method.

For most cases, a simplifying assumption can be made that $|\alpha_2| < |\beta_1|$, i.e. that the output of a parent stage will be larger than the inputs of its child stages. In this case, eq. (2.9) has the weaker form

$$|\beta_1| < \frac{|\alpha_1||\beta_2|}{|\alpha_1| + |\beta_2|}.$$

Independent Subdivisions (Widening the Graph)

Here we adapt the domain from section 2.3.2 to include a greater number of subdivisions as shown in fig. 2.13. Again taking n to be the number of points along one side of the whole domain, let s be a splitting parameter denoting the number of divisions along a boundary (length n), i.e. we split the domain into approximately s^2 subdomains. Assuming the domain and therefore its subdomains are approximately square, the subdomains have perimeter $4n/s$ and volume n^2/s^2 (see note on lower order terms in section 2.3.2).

The length of all internal boundaries is $2ns$, hence interpolating from the exterior boundary to the subdomain boundaries requires a $2sn \times 4n = 8sn^2$ element matrix. The interpolation on each subdomain requires a $4n/s \times n^2/s^2 = 4n^3/s^3$ element matrix, and this is repeated s^2 times for a total of $4n^3/s$ elements. Therefore the total cost is approximately

$$C = 8n^2 s + \frac{4}{s} n^3. \tag{2.10}$$

Minimising this with respect to s gives

$$\frac{\partial}{\partial s}\left(8n^2 s + \frac{4}{s} n^3\right) = 8n^2 - \frac{4}{s^2} n^3 = 0 \implies s^2 = \frac{n}{2}. \tag{2.11}$$

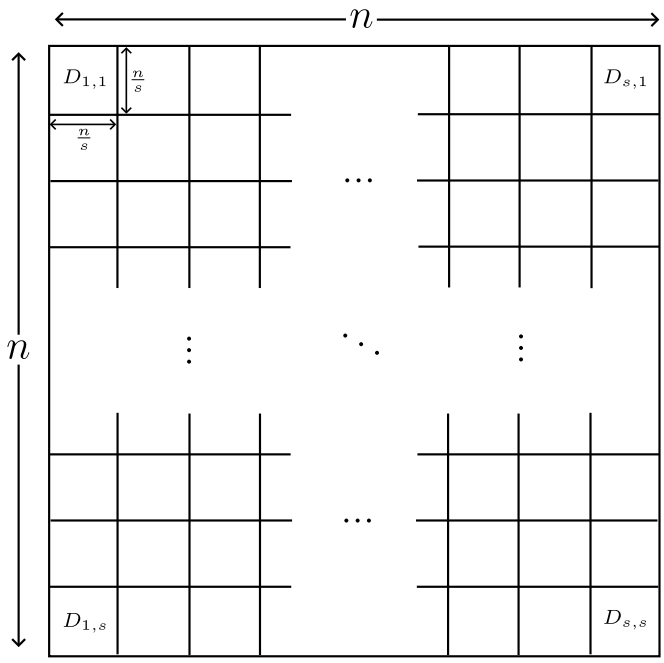


Fig. 2.13: Domain with additional subdomains.

Given the liberties with lower order terms as discussed in section 2.3.2, and the assumptions of a square domain and that $n, s \in \mathbb{R}$, this estimate is clearly not exact, but gives a good starting guide as to the optimal number of splits to minimise storage and computational requirements when splitting a domain into subdomains for a single stage.

Applying this to our example in section 2.3.2, this would indicate we should split the domain into approximately $100/2 = 50$ subdomains, i.e. 7 along each side would appear to be the optimum and would give an estimated total saving of approximately 73%. Due to the relatively low resolution of the example ($n = 100$), a real world implementation shows a saving closer to 78% (since lower order terms are dropped for the estimate), however for larger domains (increased n), the error in the estimate is reduced.

Substituting the optimal splitting parameter s from eq. (2.11) into eq. (2.10), we get

$$C_{\min} = \left(\frac{8}{\sqrt{2}} + 4\sqrt{2}\right) n^{5/2}.$$

Here we see that we have actually reduced the asymptotic complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^{2.5})$ by choosing an optimal s for each n, which implies that continuously greater benefits are seen as n increases. Figure 2.14 shows the estimated and actual as-implemented cost savings (as compared to the max cost $4n^3$) for varying side lengths n, as well as the optimal splitting parameter s used in each case.

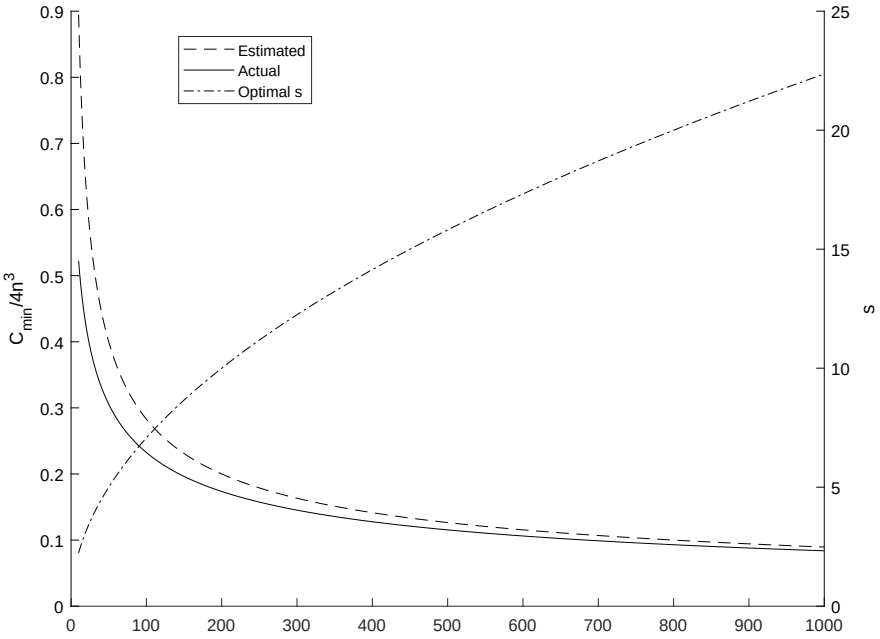


Fig. 2.14: Cost of interpolation after splitting a 2D domain as a proportion of the base cost $4n^3$. The optimal s of eq. (2.11) is also displayed.

Recursive Subdivisions (Deepening the Graph)

Let I be the total number of stages (i.e. levels in the digraph), with n as before, and s the number of subdivisions made at *each stage*, rather than in total. After some careful calculation we find that the total cost is approximately

$$\sum_{i=0}^{I} s^{2i}\left(8s\frac{n^2}{s^{2i}}\right) + 4\frac{n^3}{s^I} = 8Isn^2 + 4\frac{n^3}{s^I}$$

Comparing this to eq. (2.10), we find the factor on the n^3 term is equivalent, which is to be expected, since splitting the side of each domain by s at each of I stages is the same as splitting it into s^I in the first instance. The only saving here is the factor on the lower order n^2 term, which now scales as Is compared to s^I in eq. (2.10), however this saving is caveated by the fact that the recursive approach introduces longer dependency chains in the dependency graph, and as such puts limitations on parallelism. As such, the choice between using either a pure domain splitting approach, or recursive application of the interpolation method should likely be dictated by other factors including ease of implementation, as the cost savings are approximately equivalent. However as we will see in section 2.4 it appears that a combination of the two may be the optimal solution from a pure cost point of view.

3D Case

So far we have only treated the cost reduction in 2 dimensions. Of course in practical applications in FSI problems, the 3D case is also of interest. If we again take n as the number of points along a single edge of our domain, in this case a 3D cubic volume, then the total cost of interpolating a value from the external surface of the domain to its interior without domain splitting is approximately $6n^5$. By similar calculation to the 2D case, we find that the cost when each edge is split into s parts is given by

$$C = 18sn^4 + \frac{6}{s^2}n^5.$$

Again minimising and applying the optimal s we find

$$C_{\min} = 18\sqrt[3]{\frac{2}{3}}n^{13/3} + \frac{6}{\sqrt[3]{\frac{2}{3}}}n^{14/3} \approx 22.68n^{4.33} + 4.76n^{4.67}.$$

We see that in the 3D case, we do not get the same asymptotic reduction of the highest order term, with the power reduction only 1/3, instead of 1/2 as in the 2D case. This lower reduction is reflected in the 3D cases presented in section 2.4 and section 3.3. Higher dimensional (> 3) cases continue to show reduced power reductions, however 2D and 3D are the primary dimensions of interest for FSI applications.

Non-Runtime Costs

Implementation Complexity

An important aspect of any new method is of course the complexity of its implementation, an issue that is particularly pervasive yet difficult to quantify exactly. The major task of the method is in the dividing of a given domain into a sensible array of subdomains and placing them in the digraph hierarchy as described. In many modern applications, it is certainly usual for a basic single layer hierarchy to exist already due to the splitting of a domain into individual elements to be processed in parallel, and leveraging this existing structure is obviously desirable. With some care, additional divisions are then able to be automated reasonably well given a few parameters based on the estimates given in previous sections.

The other aspect of implementation is applying the method at each node of the hierarchy, though this is hopefully more straightforward, as it is largely a case of recursively applying the chosen interpolation method to each domain/subdomain.

Preprocessing Costs

RBF interpolation techniques usually require at least one matrix inversion in the initialisation phase when constructing the interpolation matrix. Although some reductions can be made given the structure of the matrix (generally symmetric positive definite), the most significant reduction thus far has been the multiscale technique of [4] as described in section 2.2.4. When using the multistage method, an additional matrix must be inverted for each vertex in the digraph fig. 2.8. However two factors serve to mitigate these costs. Firstly, for each layer in the digraph, the size of the matrices decreases exponentially, and secondly, as these matrices are essentially a property of the domain itself (a function of chosen RBF parameters and geometry), they can be calculated once ahead of time and reused across multiple applications. The size of these matrices is also small compared to the volume matrices, hence the additional storage costs do not adversely affect the stated savings.

2.4 Examples

Here we demonstrate savings on a number of more complex cases. We use the same benchmark cases as [4] so as to be able to compare mesh quality metrics and ensure that the multiscale approach does not adversely affect the mesh quality.

NACA0012 Aerofoil

Even 2D meshes can have sizes significant enough to warrant the multistage method - simulations run at NASA Ames for the 4th AIAA CFD High Lift Prediction Workshop included a number of refinement levels for 2D meshes of a CRM-HL aerofoil, up to a mesh containing a boundary layer of 4.5 million points, and a total domain of approximately 64 million points. Due to hardware limitations, we do not attempt that case here, but instead opt for a 2D NACA0012 O-grid with 1025 surface points and a total of 58,425 points (1025×257) as in [4].

As the domain is 1025 points along the surface of the aerofoil, and 257 deep in the normal direction, we opt for a multistage approach where we first split the domain into 4 subdomains consisting of approx 256×257 points each. From fig. 2.14 for $n = 256$, we find that the optimal splitting parameter is approximately 11, hence we further split each of the 4 subdomains into a further $11^2 = 121$ subdomains of dimensions approximately 24×24 points each, as shown in fig. 2.15. This splitting yields the hierarchy shown in fig. 2.16.

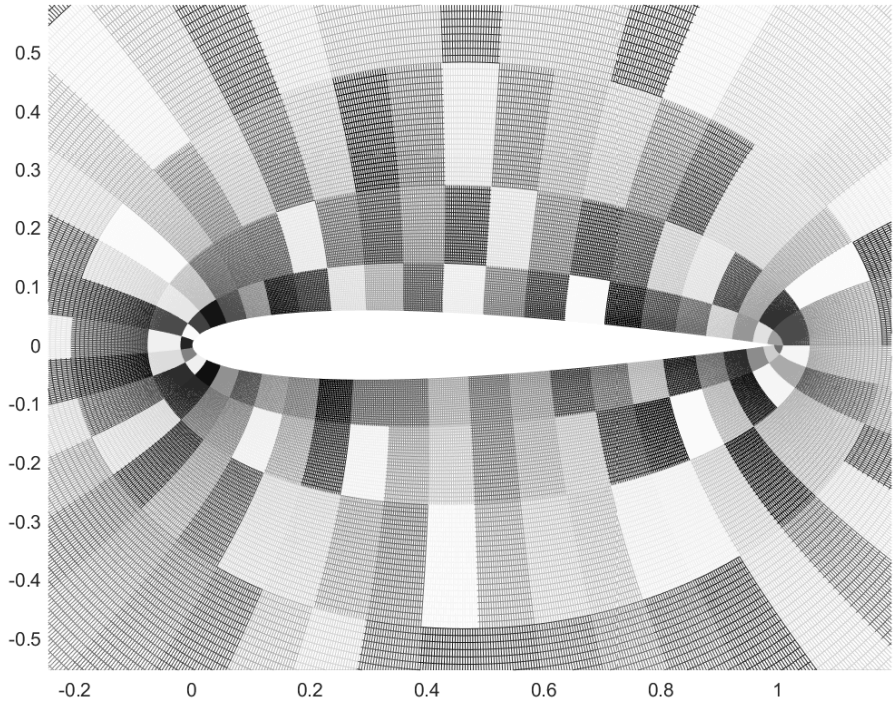


Fig. 2.15: Detail of final subdomain splits for NACA0012 mesh.

In this instance we have combined both the independent and recursive subdivision methods to maintain approximately square subdomains, though a purely independent subdivisioning approach is also possible. Table 2.1 shows the surface-to-volume cost savings compared with a single stage method for full and multiscale methods. As we can see, savings range between 80-90%, with the combined recursive and independent subdivisioning approach markedly cheaper, suggesting that a combined approach may be optimal.

	Full	Multiscale 10%	Multiscale 5%	Multiscale 2.5%
I	0.1793	0.1778	0.1896	0.2096
R + I	0.0975	0.0960	0.1077	0.1277

Tab. 2.1: Surface-to-volume cost ratios of multistage vs. single stage for full and multiscale mesh motion methods. Both purely independent subdivisioning (I) and combined recursive and independent subdivision (R + I) methods are compared. Multiscale methods are listed with % of surface points used as base points.

To compare grid quality to single stage methods, the aerofoil was pitched at 30 degrees angle of attack. As in [4], we use an orthogonality measure and compare it against the original grid. For a structured grid, the orthogonality measure for a point takes the 4 direction vectors $v_1, ..., v_4$ from the point to its 4 neighbouring

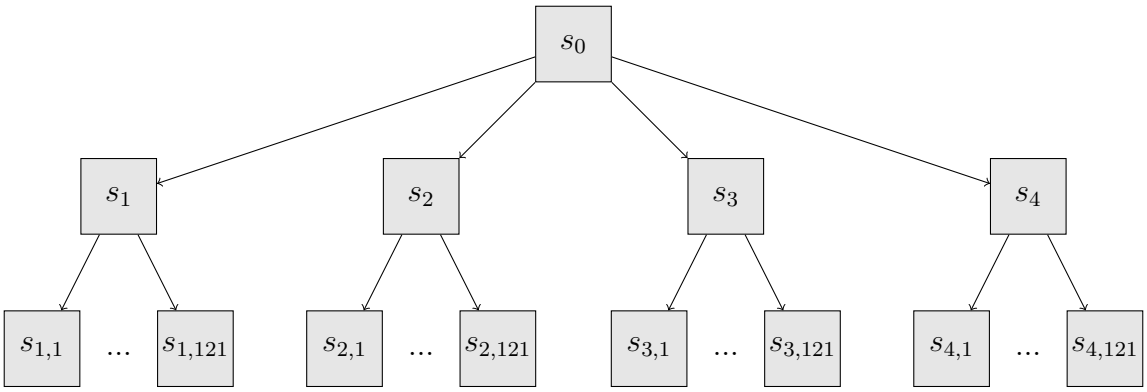


Fig. 2.16: Hierarchy for NACA0012 mesh.

points (in a clockwise or anticlockwise fashion), then calculates the quality of the grid at that point as

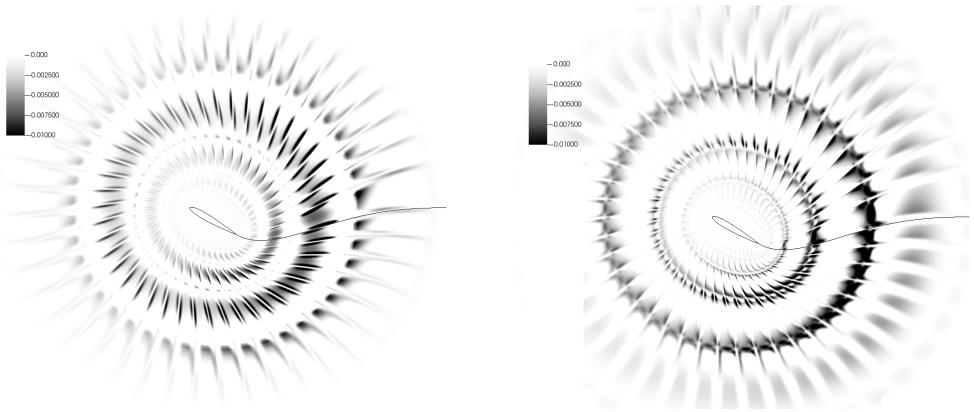
$$q = \frac{1}{4}\left(\frac{v_1 \cdot v_2}{|v_1|^2|v_2|^2} + \frac{v_2 \cdot v_3}{|v_2|^2|v_3|^2} + \frac{v_3 \cdot v_4}{|v_3|^2|v_4|^2} + \frac{v_4 \cdot v_1}{|v_4|^2|v_1|^2}\right). \tag{2.12}$$

This tracks any skewing of the grid at each point via the angles between neighbouring points. In this way measuring changes in orthogonality is akin to a discrete measure of how close to conformal the motion transform is, or how much it changes angles locally. From eq. (2.12), a measure of $q = 0$ is perfectly orthogonal (each angle 90 degrees), while 1 is fully degenerate (each angle 0 or 180 degrees).

Comparing mesh quality differences between the single and multistage approaches, it was found that mesh quality was largely static, with some very minor variations ($\pm 1\%$) between a single stage approach (with the same underlying method - full or multiscale). These differences are likely to do with the RBF parameter selection for the individual subdomains, and could be further reduced. Figure 2.17 shows an exaggerated view of the differences between the single and multistage approach using a full RBF mesh motion scheme. As can be seen, the differences are primarily clustered around subdomain boundaries, with fig. 2.17a and fig. 2.17b together producing a clear outline of the subdomains.

MDO Wing

A series of meshes developed by Allen [55] for the Brite-Euram MDO (multidisciplinary optimisation) wing has featured as a standard test case in a number of works related to RBF mesh motion. Here we use the 8.62 million point mesh as described in [4]. The mesh consists of 8 blocks/domains as shown in fig. 2.18, with the density increasing greatly towards the surface of the wing itself, as shown in fig. 2.19. The surface of the wing contains 61,601 points.



(a) Quality decrease *(b)* Quality increase

Fig. 2.17: Exaggerated differences in near-field mesh quality between multistage and single stage approach with full RBF mesh motion. White and black represent minimum/maximum differences of 0 and 1% respectively. Differences are primarily clustered around subdomain boundaries.

We begin by only taking the first 40 layers of the mesh from the surface, which corresponds approximately to 1 root chord, as indicative of a situation in which a limited deflection is expected. We then split the 8 blocks up into subdomains as close to cubes as possible, resulting in 84 cubic subdomains approximately $40 \times 40 \times 40 = 64,000$ points each. Table 2.2 gives additional details.

Block #	Dimensions	Cubic Subdomains	Total Points
1	$81 \times 40 \times 193$	10	625,320
2	$161 \times 40 \times 193$	20	1,242,920
3	$161 \times 40 \times 193$	20	1,242,920
4	$81 \times 40 \times 193$	10	625,320
5	$81 \times 40 \times 81$	4	262,440
6	$161 \times 40 \times 81$	8	521,640
7	$161 \times 40 \times 81$	8	521,640
8	$81 \times 40 \times 81$	4	262,440
	Total	84	5,304,640

Tab. 2.2: Details of splitting of first 40 layers of MDO mesh from wing surface.

Using the multiscale mesh motion method of section 2.2.4 with approximately 600 or 1% of the 61,601 surface mesh as base points, as recommended in [4]. Hence the base method requires on the order of approximately $600 \times 5,304,640 = 3.2 \times 10^9$ operations/storage to complete the interpolation/store the matrix. We now apply a multistage approach:

1. Interpolate motion on the surface of the wing by interpolating from the 600 base points to the surfaces of each of the 8 main blocks.

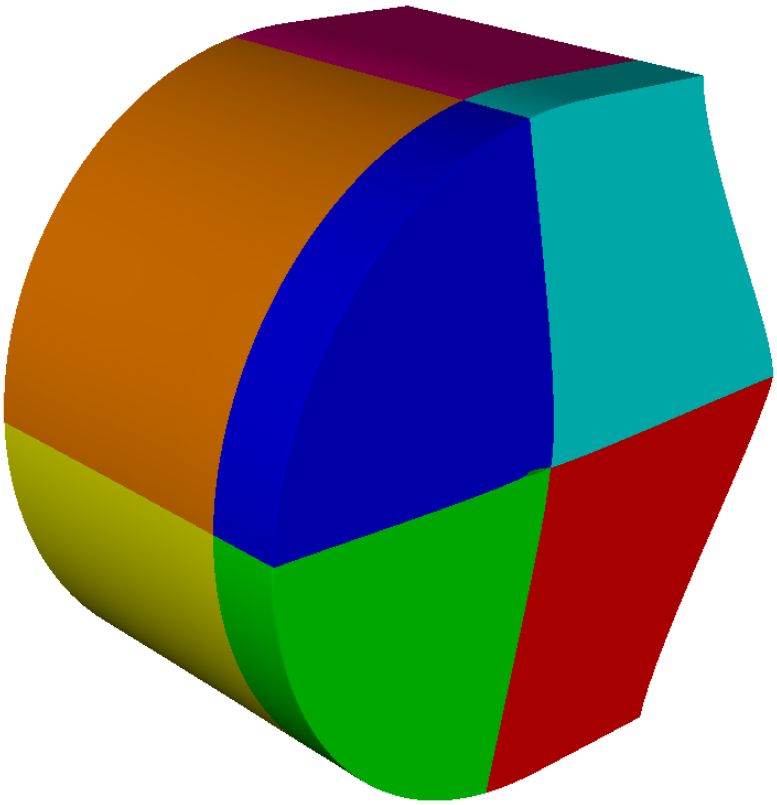


Fig. 2.18: 8 blocks of MDO wing mesh

2. Again apply the multiscale approach by taking 1% of the surface point of each main block and interpolating it to the surfaces of the interior cubic subdomains.

3. Interpolate from 1% of the surface points of each cubic subdomain to the 40^3 volume points on the interior.

Completing these computations, we find the total update cost to be 8.51×10^8, for a total saving of approximately 67.8%. This is a significantly smaller saving than section 2.4 due to the non-optimal subdomain splitting, as well as being a 3D problem (see section 2.3.3). It is expected that a more optimal subdivision scheme would yield considerably better results. Mesh quality results were similar to that of section 2.4, with no significant differences being found between the multistage and single stage approaches.

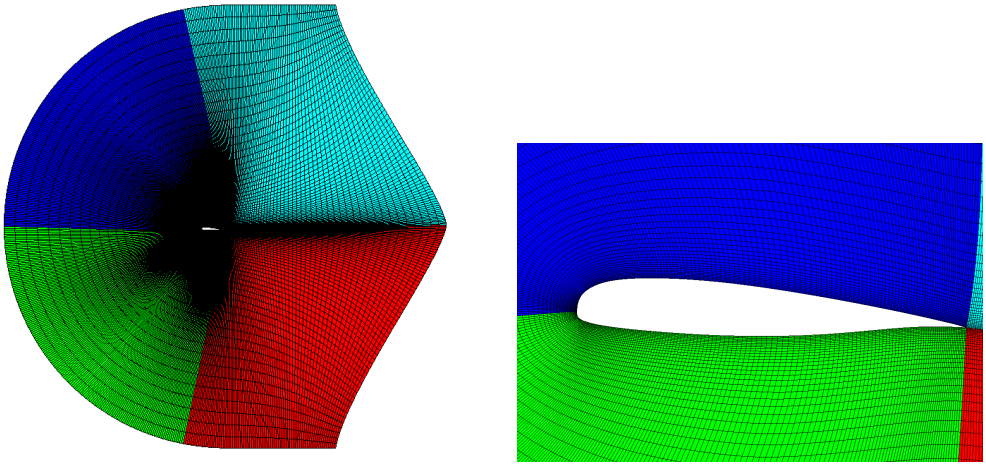


Fig. 2.19: Mesh detail of MDO wing.

2.5 Conclusions

In this chapter we have described a multistage structure in the context of RBF mesh motion methods, although it is possible that the multistage approach can be applied to somewhat arbitrary mesh motion methods, since it does not rely on any particular aspect of the methods used in the examples here, and makes very conservative assumptions about the costs of volume update steps. It is shown that the surface-to-volume computations of mesh motion can be greatly reduced, with not only constant factor improvements, but also modest gains in asymptotic complexity. Some estimates were given on expected cost savings in scenarios with square/cubic domains, with examples using real world CFD meshes provided. The examples showed that with the correct subdivisioning approach, costs of surface-to-volume computations and storage can potentially be reduced by over 90%. Cost savings for coarse grids were found to be negligible, but for finer grids the costs savings were seen outweigh the additional startup costs and implementation complexity. A multistage approach was not seen to have significantly different mesh quality characteristics from single stage methods, with differences less than 1%.

After a number of significant advances in the field in recent years, a multistage approach further reduces traditional costs associated with RBF mesh motion, removing another potential barrier for full-scale mesh motion schemes in FSI applications.

3. CAVITY-STORE SIMULATION

3.1 Introduction

Here we present a case study using the methods of chapter 2 in a large scale mesh to simulate flow induced vibrations of a cylindrical store within a box cavity[1]. The simulations mirror experiments performed at Sandia National Laboratories to better understand complex fluid-structure interactions that occur in aircraft bays during flight [56]. Previous simulations have been completed by Sandia using a rigid store, as well as 1-way coupling (fluid to structure), and qualitative agreement with the experiments was found for low frequency structural modes, however simulations showed poor agreement at higher frequency modes [56]. The main purpose of the work in this chapter is to demonstrate that FSI-enabled flow simulation could better match the experiments, proving that the modeling of a dynamic structure within the flow is necessary to capture detailed physical phenomena. A secondary goal was to show the use of the multistage mesh motion methods of chapter 2 in practical numerical simulations, with large domains and complex geometry. As no baseline mesh motion implementation was available in FLAMENCO (i.e. non-multistage), the gains in efficiency can only be estimated, however the results of this chapter demonstrate that the multistage mesh motion methods perform excellently in a high fidelity simulation, and are computationally feasible.

This study is a demonstration of the applicability of the method of chapter 2 to practical FSI cases, and utilises the University of Sydney in-house FLAMENCO (Flow Analysis and Modelling Environment with Combustion) CFD tool[2], which contains algorithms up to 5th/3rd-order accurate in space and time respectively. FLAMENCO is a massively parallel, fully compressible, structured multi-block solver with RANS, LES, and DES models. The underlying algorithms have been validated extensively in complex flows such as turbulent mixing problems, scramjet intakes, aero-acoustics of cavity flows, shock-boundary layer interaction, rocket nozzles, ship airwakes, and combustion [57, 58, 59, 60, 61].

For the FSI implementation in this chapter to remain largely independent of the FLAMENCO algorithms, a partitioned approach was necessary, a strength of

[1] Initial cavity-store simulation results were presented by the author at SciTech2019 [7].

[2] FLAMENCO is maintained by Professor Ben Thorner with contributions from a number of staff and students, see https://fluids.eng.sydney.edu.au/computer-codes/ for more detail.

RBF-based FSI methods. A previous implementation of FSI within the FLAMENCO code base, based on finite-element 'super blocks', although successful in initial tests (and especially so with large deformations), was identified as having future roadblocks. These included requirements for complex and case-specific grid generation code to integrate the structural components with the existing code base, and a limit of a single finite-element per CFD block. It was determined that overcoming these limitations would include a monolithic approach and would be against the requirement of method independence.

Section 3.2 gives details on the configuration of the FLAMENCO fluid solver, the FSI implementation, as well as the development of a modal structural solver to perform structural calculations, with accompanying benchmarks for validation. Section 3.3 presents the results and conclusions of the cavity-store FSI simulation, comparing them to the Sandia experiments and rigid/1-way simulations.

The calculations of this chapter were completed primarily on the National Computational Infrastructure (NCI) and the University of Sydney's *Artemis* HPC

3.2 Simulation Setup & Tests

3.2.1 FLAMENCO Fluid Solver

FLAMENCO is a CFD solver that has been adapted for various research applications since its development, written in FORTRAN and using MPI for parallel computations. It has been used extensively on various HPC clusters. The two-way RBF FSI coupling was implemented within the same codebase, but as a modular component that relied as little as possible on the surrounding code, so that the implementation was able to be adapted easily in the future. FLAMENCO is written in FORTRAN, and uses MPI for parallel computations, making it particularly well-suited to running on HPCs.

3.2.2 Modal Structural Solver

For this work, a modal structural solver was also implemented to perform the structural solve at each time step. This was implemented within the FLAMENCO code base for efficiency, but was functionally independent, retaining the required partitioned approach.

To begin we take the equations of motion to be a coupled system of damped oscillators

$$M\ddot{x} + C\dot{x} + Kx = F \tag{3.1}$$

where M, C, and K are the mass, damping, and stiffness matrices respectively, and F is the applied force. To obtain the natural frequencies ω_i and mode shapes U_i of the system, we set $C = 0$ and apply no load then solve the eigenproblem

$$KU_i = \omega_i^2 MU_i. \tag{3.2}$$

Once the mode shapes/eigenvectors are found, we can define an inner-product using the mass matrix M and use our preferred algorithm to convert the eigenvectors U_i to an orthonormal basis ϕ_i with respect to this inner product, i.e.

$$\phi_i^T M \phi_i = \delta_{ij} \tag{3.3}$$

where δ_{ij} is the Kronecker delta. This gives a modal matrix Φ consisting of mode shapes such that $\Phi^T M \Phi = I$. We can now describe the system using linear combinations of these mode shapes combined with amplitudes η_i known as generalised coordinates, i.e.

$$x = \Phi\eta.$$

Applying this to eq. (3.1) it becomes

$$M\Phi\ddot{\eta} + C\Phi\dot{\eta} + K\Phi\eta = F.$$

Multiplying through by Φ^T, from eq. (3.2) and eq. (3.3), we can see that the system in generalised coordinates becomes

$$\ddot{\eta} + \Phi^T C \Phi \dot{\eta} + \omega^2 \eta = \Phi^T F$$

where ω^2 is the diagonal matrix containing the eigenvalues ω_i^2 along the diagonal. Unfortunately in general we have few guarantees on the structure of C, in particular, it is not necessarily diagonal, which introduces coupling to an otherwise uncoupled system. There are a number of methods for dealing with non-diagonal C, including diagonalisation, modal projection, and educated guessing - all of which we conveniently avoid by assuming the system is undamped - a reasonable enough assumption given the relatively light damping of the rigid store. Hence we have the final equation of motion in modal coordinates

$$\ddot{\eta} + \omega^2 \eta = \Phi^T F.$$

Boundary conditions in spatial coordinates are related to generalised modal coordinates via $x(0) = \Phi\eta(0)$. To obtain boundary conditions in modal coordinates directly, we can avoid inverting Φ by exploiting eq. (3.3) to find

$$\eta(0) = \Phi^T M x(0)$$
$$\dot{\eta}(0) = \Phi^T M \dot{x}(0).$$

Since Φ will ordinarily have the n columns (mode shapes) equal to the number

of points in the original (spatial) system, there is no immediate advantage in converting between the two. The benefits appear in cases where the structural motion can be sufficiently described via selection of a reduced set of m modes, i.e.

$$x \approx \sum_{i=1}^{m} \phi_i \eta_i$$

where $m < n$. This can reduce the computational complexity significantly. Next we perform a benchmark using 3 modes of a 10-mode system and find minimal loss of accuracy.

A benchmark of the modal solver was performed on a 10-point, 3-mode, cantilevered beam and compared to a 10-element FEA model run in the Strand7 software. Figure 3.1 shows the comparison of the cantilevered beam with a sinusoidal driving force applied at the tip. Each wave represents one of the 10 points along the beam. The displacement at the end of the beam match exactly, with the models differing through the body of the beam by a maximum of approx. 3% of maximum deflection. This error was found to be a global maximum across all frequencies tested (1Hz-300kHz), with the error completely disappearing at the resonant (modal) frequencies of the beam.

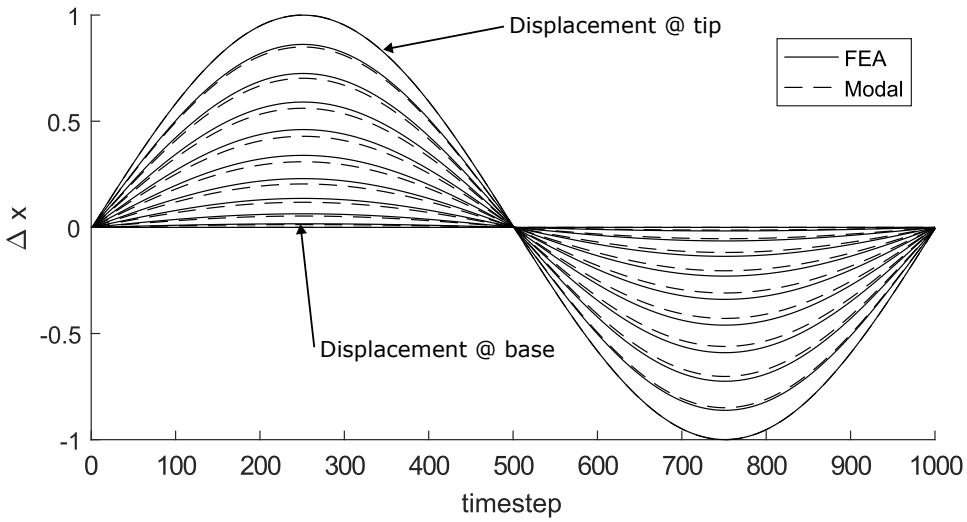
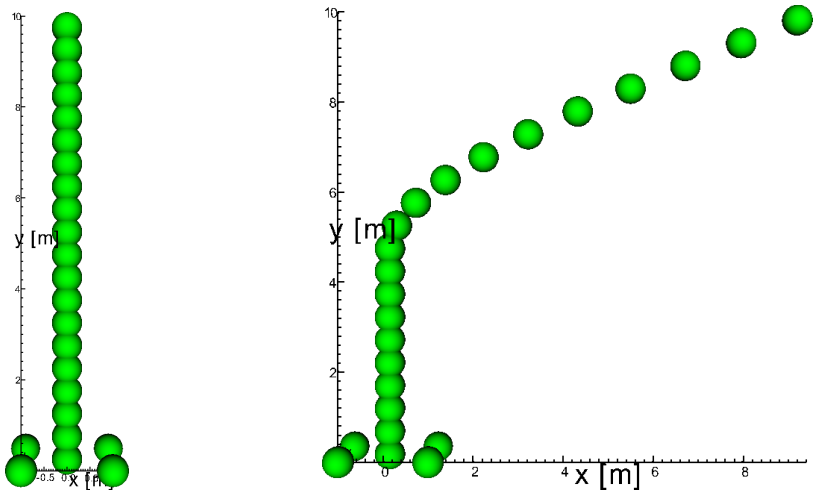


Fig. 3.1: Comparison of response to single period forcing of FEA and modal models. Each wave represents an individual point along the beam.

3.2.3 Bending Beam Test

The first coupled test completed was a 2D cantilevered beam. A 10×5 computational domain was constructed with a 0.32×2.5 beam anchored at $x = 5$

on the bottom wall. The beam had a single mode with frequency $\omega = 106.8Hz$, with a mode shape as shown in fig. 3.2b. As FLAMENCO is a 3D solver, the implementation of the RBF coupling is 3D, and hence the structure was given a baseplate to eliminate singularities caused by a reduced dimension model. The initial fluid conditions were set to a constant density $\rho_0 = 1.2kg/m^3$, pressure $p_0 = 101325Pa$, and constant horizontal velocity $v_{x0} = 22.7m/s$.



(a) Undeformed *(b)* First mode shape

Fig. 3.2: Undeformed beam with 4 additional supporting points at the base, and the (exaggerated) first mode shape used in the benchmark of section 3.2.3.

The simulation was run over $0.3s$ to test the coupling behaviour of the implementation. Figure 3.3 shows a snapshot of the pressure and velocity fields at $t = 0.27$. The complex force history at the tip of the beam shown in fig. 3.4 shows the transient behaviour as the flow is developing. These transient fluctuations are generated in part by non-perfect initial conditions of the flow. As the flow develops, these small, non-periodic effects would be expected to largely disappear, but as this test was only designed to test the *coupling* between the structural and fluid solvers, it was not necessary to develop the flow fully. The modal structure is seen to be interacting with the flow appropriately - reacting predictably with the calculated forces, with a small lag behind the forcing, as shown in fig. 3.4.

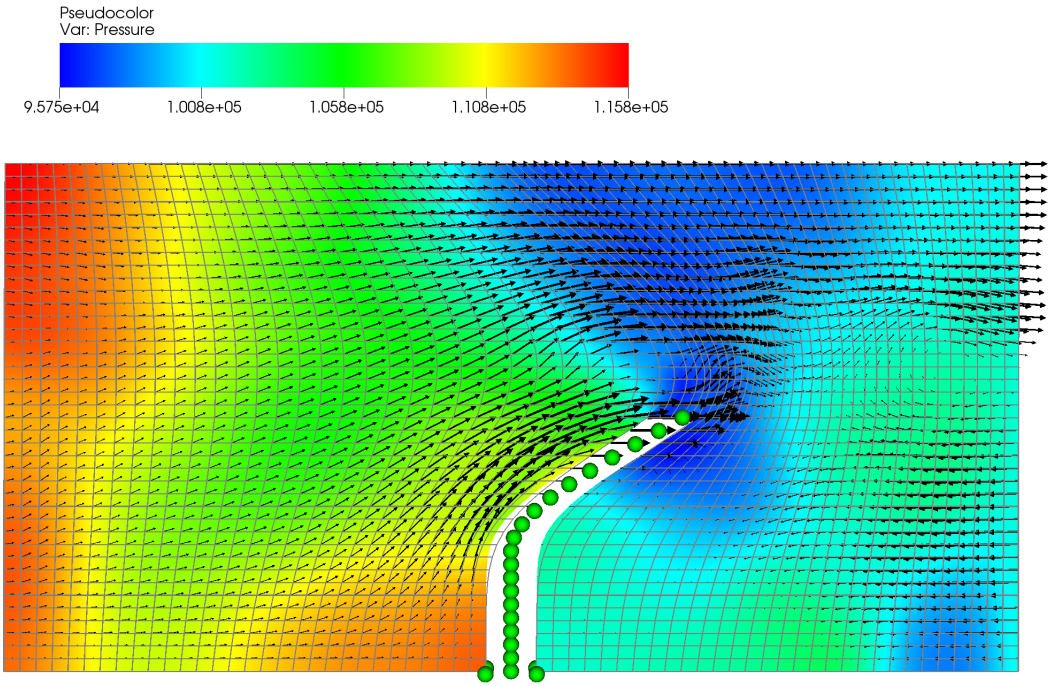


Fig. 3.3: Pressure distribution (in Pa) for $t = 0.27$. The velocity field, indicated by the black arrows, ranges from a maximum of 126.7 m/s near the beam tip, to a minimum of 0 at the downstream vortices.

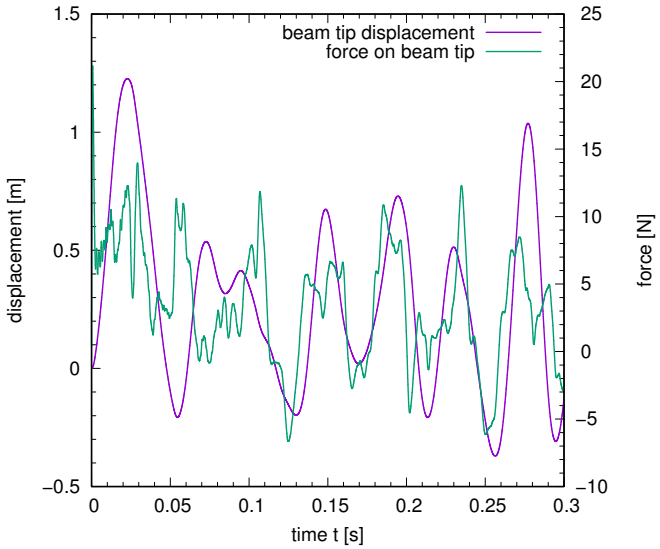


Fig. 3.4: Displacement vs. force at beam tip. Here the complex, high frequency transient forces of the flow field are damped out by the structure, resulting in a relatively smooth motion of the beam tip.

3.2.4 Turek-Hron Benchmark

Following the basic bending beam test, the Turek-Hron test case [62] was implemented for further validation. This test considers a laminar, viscous flow over an elastic bar. This test has been used successfully to benchmark various discretisation and solution methods. The overview given by Turek et al. [63], highlights the efficacy of the benchmark in establishing grid independent results, regardless of the solution scheme.

The computational mesh is shown in fig. 3.5. The domain measures 2.5 m in length and 0.41 m in height. A cylinder of radius $r = 0.05$ m is positioned at point $(0.2, 0.2)$ measured from the left bottom corner of the channel. The position of the cylinder is kept fixed. An elastic bar of length $l = 0.35$ m and height $h = 0.02$ m is positioned with the right bottom corner at $(0.6, 0.19)$. The left end is fully attached to the fixed cylinder. The mesh is intentionally non-symmetric (see fig. 3.5) to prevent the dependence of the onset of oscillation on the precision of the computation.

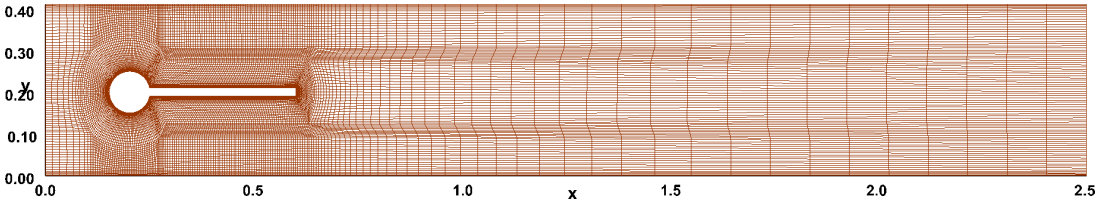


Fig. 3.5: Structured domain mesh for Turek-Hron benchmark problem. Vertical asymmetry has been included to encourage oscillation.

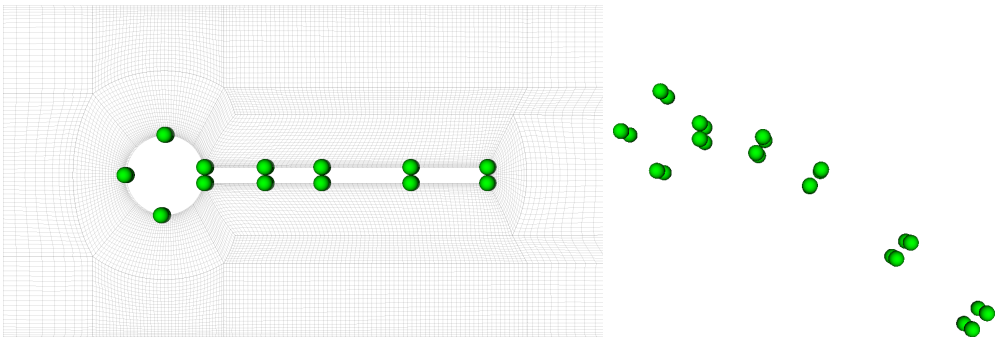
A parabolic velocity profile of the form

$$v = 35.7y(0.41 - y)v_0$$

is prescribed at the left channel inflow. At the right end, outflow boundary conditions are prescribed with a gauge pressure of zero. At all other boundaries, the no-slip condition is applied.

The initial conditions are constant density $\rho = 1000$ kg/m^3, zero velocity and pressure 1 bar. The dynamic viscosity is assumed to be constant and set to $\mu = 1$ kg/ms. The velocity profile at the inlet is increased smoothly from $v_0 = 0$ until $v_0 = 2$ m/s. The Reynolds number, defined as $Re = 2rv_{\text{mean}}/\nu$ is set to 200 once v_0 has been increased to its maximum value. These settings correspond the case labelled FSI3 in [62]. For simplicity, a very simple structural model with 26 structural points and one mode is used. The position of the structural points and the mode shape is shown in fig. 3.6. The beam (referred to as a flag in [62]) is divided into 4 sections, with the sections closer to the base shorter than those closer to the tip. This is to capture to the greater expected bending closer to the

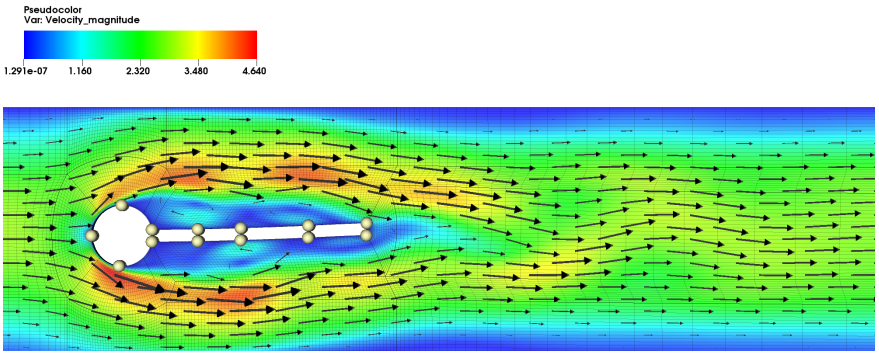
base, especially in the low order modes (e.g. first mode shown in fig. 3.6b). The mode shapes of the points were generated using modal beam analysis in MSC NASTRAN.



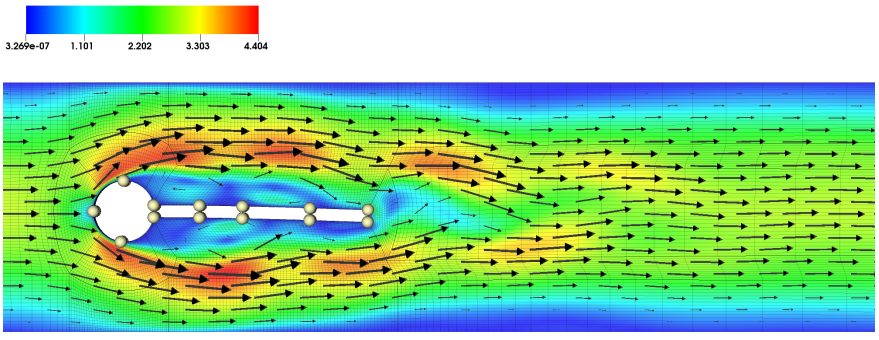
(a) Structural nodes *(b)* Structural mode shape.

Fig. 3.6: Structural nodes and mode for Turek-Hron benchmark.

Figure 3.7 shows flow visualisations at $t = 6.2$ s and $t = 6.8$ s, from a fully coupled simulation, after the onset of flow oscillations. The structure responds with the flow oscillations, and carries the deformations into the mesh, with the flag of the test oscillating at approximately 8.5% of the total channel width.



(a) $t = 6.2s$



(b) $t = 6.8s$

Fig. 3.7: Velocity magnitude plots for Turek-Hron test case. Velocities are given in m/s.

3.2.5 Cavity Store Setup

The cavity store case mirrors wind tunnel experiments conducted at Sandia National Laboratories. In these experiments, a square cavity of 127 mm length (L) and 38.1 mm depth was fitted with a two-legged cylindrical store model measuring 108 mm in length ($0.85L$) with a diameter of 19.1 mm. The two cylindrical supporting legs had a diameter of 12.7 mm. The geometry of the cavity with store is shown in fig. 3.8. The volume of the store geometry was approximately 6% of the cavity.

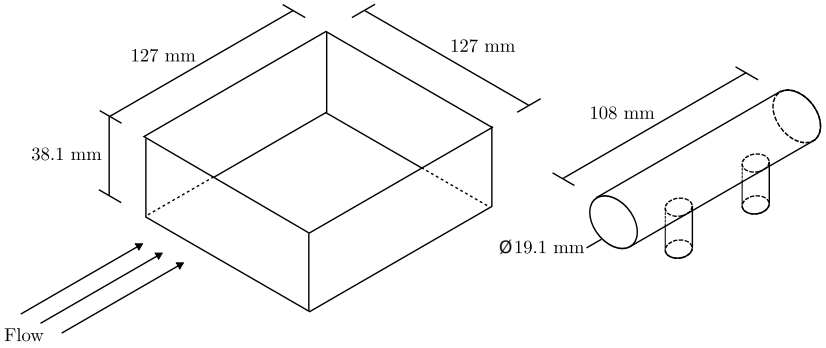


Fig. 3.8: Geometry details for cavity and store (not to scale).

The computational mesh for the cavity store case was made up of 534 subdomains, ranging in size from 1,430 to 117,649 nodes, with the total mesh size approximately 18 million nodes. Figure 3.9 shows the mesh, with detail of the store and surrounding cavity. As can be seen, the nodes are primarily clustered within the cavity and near the upper wall of the tunnel. The remaining tunnel walls are located 0.7 m away from the cavity in their respective directions. The grid size in the regions furthest away from the cavity at the entrance, exit and other walls of the wind tunnel is sufficiently large that numerical dissipation significantly damps out reflections. Symmetry boundary conditions are used at all tunnel boundaries except for the upper wall, which uses a wall boundary condition, and the inlet and outlet, which use inflow and outflow boundary conditions respectively. The initial flow conditions are given in table 3.1.

Mach Number (M)	0.79
Stagnation Pressure (P_0)	111 kPa
Dynamic pressure (q)	32.1 kPa
Temperature (T_0)	324 K
Velocity (u_∞)	270.02 m/s

Tab. 3.1: Flow conditions for cavity store case as detailed in [56].

For the store structural model, a simple (locally) 1D stick model was constructed with 10 modes. The model is shown in fig. 3.10. Again a baseplate has been

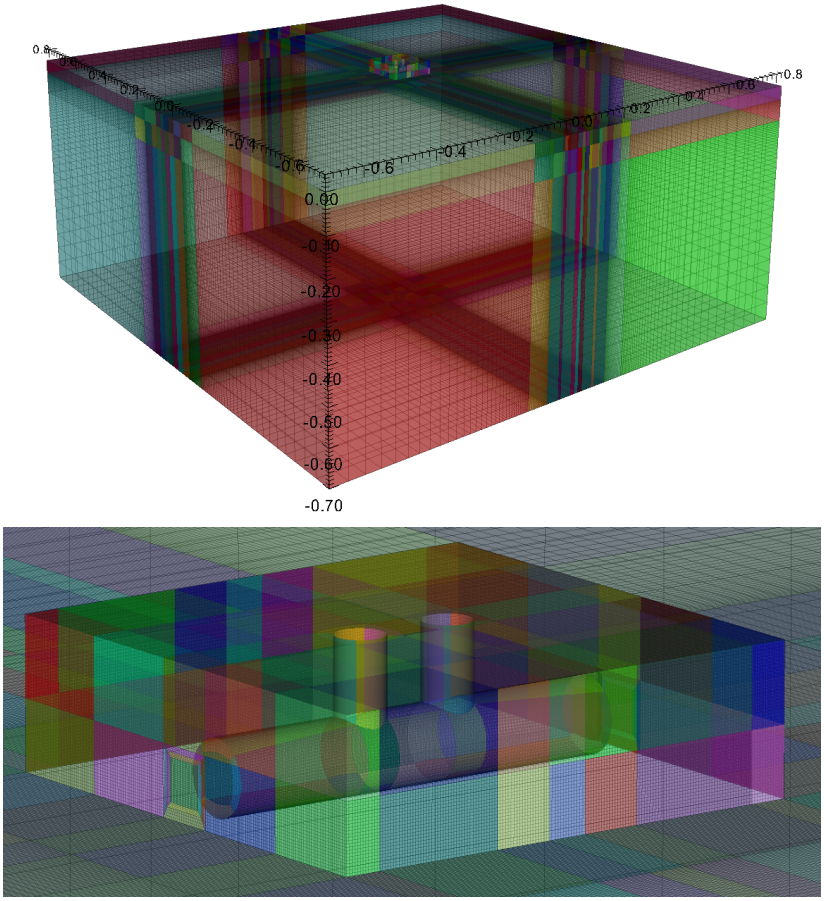


Fig. 3.9: 18 million node mesh (with detail) for cavity store case. Random colouring of the individual subdomains is used for ease of visualisation, and has no bearing on the structure of the mesh overall.

used to make the model 3D and avoid any of the singularities when using the 3D mesh motion routines on a lower dimensional point set. The baseplate consists of 4 points in the horizontal plane, perpendicular to the remaining points of the structural model, as can be seen in fig. 3.10.

3.3 Results

The simulation was run in both static and FSI configurations[3]. Pressure measurements were taken from a point directly behind the store in the streamwise direction (referred to as AWP1 in [56]). The power spectrum of the pressure in the cavity is shown in fig. 3.11 for both static and FSI simulations, as well as the original data from the Sandia experiments. Given the vibrations of the

[3] A video showing exaggerated motion (x5000) of the store can be found at `https://youtu.be/4mmX_MBq3fE`

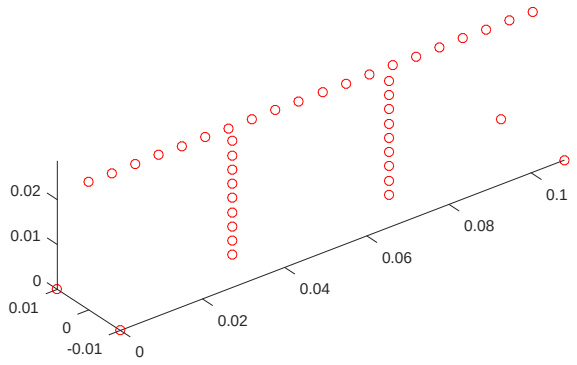


Fig. 3.10: Structural model (vertically inverted) of store in cavity store case. Note the 4 baseplate points below the model on the horizontal plane to ensure the resulting linear system is of correct rank.

store are small (approximately 10^{-6} m), the static case shows clearly the first two major aeroacoustic modes from the experiment. The FSI case gives similar results for the first two modes, but additionally reveals the 3rd, 4th, 5th modes, matching closely with those found in the experimental data. This shows that the interactions the vibrating store with the flow is important in predicting resonant behaviour, and cannot be faithfully recreated with the same fidelity with a static simulation.

The primary difference in the simulations compared to the experimental data is the increased noise floor, particularly above the second mode. This is seen in fig. 3.11 in both the static and FSI simulations where the power floor is above the experiments for frequencies above approximately 2000Hz. This is likely due to the short simulation time (0.17 s) compared to the time of the Sandia experiments (5 s). This shorter timescale means that the smoothing effect of sampling over a longer period is not as pronounced, and various transient flow features from the beginning of the simulation may still be in effect. In order to partially mitigate these effects, the sampling frequency used in the calculation of the power spectrum was increased from 10 Hz in the experiments, to 50 Hz in the simulations. While this increase in sampling frequency aided in the resolution of the modal frequencies, it was not able to fully eliminate the increased noise floor at the higher ($>$ 2000 Hz) frequencies. It is expected that a longer simulation time would further lower the high frequency noise floor, as well as smooth the power spectrum at those frequencies, helping to eliminate discrepancies between the simulation and experiment.

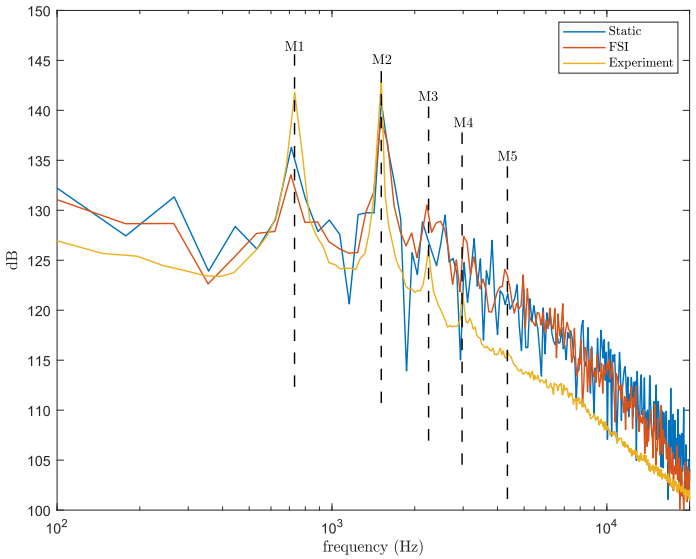


Fig. 3.11: Power spectrum for pressure measured on wall of cavity directly behind store. The first 5 modes are indicated. Both static and FSI simulations are able to accurately reproduce modes 1 and 2, while the FSI simulation additionally reproduces modes 3, 4 and 5 from the simulation, showing the advantages of the FSI simulation over the static.

3.3.1 Estimated Savings From Multistage Mesh Motion

Multistage mesh motion was implemented using the existing subdivisioning of the domain mesh (534 subdomains) to construct the acyclic digraph structure of section 2.3. The subdivisions were not all cubes, hence were not necessarily optimal, however the majority had low ratio of surface nodes to volume nodes (i.e. not too 'thin' in any direction), ensuring reasonable savings in the volume update step. The implementation used the multiscale method of Kedward [4] for the mesh motion at the individual stages, with approximately 10% of the control nodes at each stage being used as base points. As the goal of this work is to show the efficacy of FSI methods more generally to the problem of reproducing the experimental frequencies, the multistage mesh motion method was implemented directly, hence there exists no reference implementation of the mesh motion algorithm within FLAMENCO. This means it we can only approximate the reduction in cost of the volume update using the geometry of the mesh and its subdivisioning, using the methods of section 2.3.3.

For the set of control points, the structural nodes and the walls of the cavity are used. The walls of the cavity are used as control points, as we must ensure that the walls of the cavity do not flex with the motion of the store. The motion of the control nodes is then transferred to the individual faces of the subdomains (in parallel), then from the subdomain faces to the corresponding volume nodes,

using the principle shown in fig. 2.7. The resulting digraph is shown in fig. 3.12.

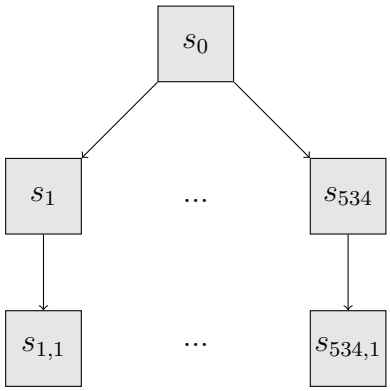


Fig. 3.12: Digraph structure for multiscale mesh motion in cavity store simulation. Motion is transferred from the base points to the subdomain faces ($s_0 \rightarrow s_1 \dots s_{534}$), then from the subdomain faces to the subdomain volumes ($s_1 \dots s_{534} \rightarrow s_{1,1} \dots s_{534,1}$).

Overall, the estimated savings for the volume update step (in both storage/memory and operations) from use of the multistage mesh motion algorithm in this case is 64.57%. This is similar to the results of section 2.4, and like that case, the savings are reduced by the suboptimal (non-cube) shape of the domains, but are still significant. A proxy for mesh quality, the cell Jacobians, is monitored during the running of FLAMENCO, which triggers warnings when cells begin to become degenerate. As such, it was found that no adverse decrease in absolute mesh quality was caused by either the multistage or multiscale mesh motion algorithms.

3.4 Conclusions & Future Work

In this chapter we have described the implementation of FSI functionality into the FLAMENCO fluid solver, using a modal structural solver, as well as the multistage mesh motion method of chapter 2. The solver was then used to replicate physical experiments conducted by Sandia National Labs. The results of the solver with FSI enabled out-performed both the static and one-way coupling simulations, and better matched the experimental data, accurately predicting all five modes seen in the experimental data, compared to the two modes predicted by the other simulations. This demonstrates that even for vibrational problems where motion is exceedingly small, FSI is necessary to accurately predict detailed aeroacoustic phenomena.

The size of the mesh (18×10^6 nodes) also shows that the multistage mesh motion method has favourable computational properties, and that it is able to be

successfully employed in large 'real world' FSI simulations. Unfortunately the lack of a baseline implementation meant that it was difficult to quantify the exact efficiency gains by using the multistage mesh motion method beyond general estimates, though the estimates line up well with the theory and experiments described in chapter 2.

4. RBF-BASED MESHLESS FLUID SOLVER

4.1 Introduction

The previous two chapters have dealt with methods and applications in the motion of arbitrary point clouds. Although these methods are themselves meshfree, they are equally applicable to almost any method in the solution to PDEs in which the domain has been discretised, independent of the actual mode of discretisation. In particular, it can be applied to structured, unstructured, meshed, or meshless domains. In this chapter we turn to a meshless method of solving flow equations on the point cloud alone, with no reliance on additional information about connectivity of points. We develop a solver based on the RBF finite difference (RBF-FD) method and apply it to the solution of unsteady, viscous flow expressed via the streamfunction-vorticity formulation of the incompressible Navier-Stokes equations. We describe a technique for easily establishing boundary conditions at non-slip walls on vorticity in the meshfree case. A number of studies are conducted on some common benchmark cases.

The chapter is structured as follows:

Section 4.1 gives some background of meshless vs. meshless solvers for incompressible flow, with particular attention to the RBF-FD method and the streamfunction-vorticity formulation.

Section 4.2 introduces two equivalent formulations for approximating derivatives of functions approximated by RBFs that will be used going forward in the remaining chapters, and provides a simple example of the method for 1D functions.

Section 4.3 then introduces the streamfunction-vorticity formulation for the incompressible Navier-Stokes equations. The derivation of the governing equations is gone through in detail, and various boundary conditions are described, including an adaptation of boundary conditions from meshed solvers to the meshless case. The solver is then benchmarked against the cases of freestream, flow in a pipe, lid driven cavity, and flow past a square cylinder.

4.1.1 Background

Meshfree or meshless methods for the solution of PDE are currently an area of great interest. One of the primary benefits of meshless methods is that they reduce or eliminate many of the complex mesh generation tasks involved in more traditional solution methods such as finite difference/volume/element or spectral methods. These issues are particularly prevalent when modelling complex, irregular, or very high resolution domains, especially when a structured mesh is necessary for the given solver. Even in cases where a meshless method requires, or performs better, on a 'structured' point cloud (e.g. when points are arranged on a Cartesian grid), the lack of a need to generate adjacency or grid connectivity information will often greatly reduce the complexity (manual or computational) of generation, as well as having benefits for processing and memory requirements, which can be significant in the meshed case. However these simplifications and considerable benefits in the setup phase of a solution have often come at the cost of increased runtime computational complexity as well as implementation complexity. Historically, these additional complexities have been found to be significant and difficult to overcome, and as such have limited the applicability of meshless methods in many cases. Other practical issues with meshless methods have traditionally been in relation to the application of boundary conditions, and the need for higher order numerical integration.

More recently, great advances have been made in meshless methods to reduce or eliminate many of these issues [64], and meshless methods are more and more being applied to model complex real world phenomena [5].

Some of the earliest meshfree methods came in the form of smoothed-particle hydrodynamics (SPH), which was developed by Lucy [65], and Gingold and Monaghan [66], to solve problems in astrophysics, and has since found significant applications in solid mechanics [67] and computer graphics [68]. Difficulties arise in imposing boundary conditions with SPH [69], as the boundary is in some sense virtual when compared to the very physical simulation of the particles. A number of methods have been proposed to deal with boundaries in SPH, yet it remains an active and significant area of research within the field. Another early form of meshless PDE solution was the so called generalised finite difference (GFD) method pioneered by Jensen [70]. GFD methods originally had many difficulties including selection of neighbouring points in such a way that avoided singular systems when solving. Many of these issues have since been resolved by introducing algorithms for neighbour selection, and using a least-squares techniques to determine the solution to the resulting over-constrained system.

Meshfree methods were then developed from the Galerkin framework, and were notably some of the first to address the difficulties with the application of boundary conditions. The diffuse-element method (DEM) in which derivatives of the

basis functions are approximated directly, rather than by differentiating the approximation of the primary variable, was one of the early successes of Galerkin formulated meshless methods [71]. Improvements were made to the DEM method to address accuracy at boundaries and discontinuities in the domain, namely element-free Galerkin (EFG) [72] and reproducing kernel particle (RKP) [73] methods. EFG introduced Lagrange multipliers and higher order quadrature based on a background mesh to improve accuracy, while RKP was a discretisation of existing kernel methods and could also serve as a correction to SPH methods. Even with these improvements, numerical integration in Galerkin methods proved to be difficult, with both direct integration and quadrature methods producing significant errors. These issues were addressed by the development of stabilised conforming nodal integration (SCNI) by Chen et al. [64] which employs a gradient smoothing to satisfy a divergence constraint on the numerical integration. This avoided the errors of the quadrature method, and was also able to serve as a correction to the direct method.

The foundational theory for RBF based methods for solving PDEs was established by Franke and Schaback [74], although RBFs of course had almost 3 decades of already established work as a framework for interpolation beginning with Hardy in 1971 [75], and practical examples of RBFs for numerically solving PDEs can be seen as early as 1990, e.g. Kansa [76]. RBFs are a popular method in the broad class known as collocation methods, in which an approximating finite basis for the typically infinite-dimensional solution space is chosen, along with a discrete set of points at which the relevant equations are to be satisfied. Much of the work in RBF collocation methods has been related to the conditioning of the resulting systems, with various methods proposed to alleviate the difficulties. Domain decomposition methods such as Wong et al. [77] and Kansa & Hon [78] were shown to improve condition numbers for the relevant systems, while Hon and Schaback [79] investigated the influence from the choice of basis function and related parameters. Localised methods for RBF interpolation more generally were introduced by Wendland [51], using compactly supported basis functions, which have been incredibly important in all areas of RBF interpolation. These ideas were also adapted by a number of authors by truncating (i.e. artificially enforcing compact support) multiquadric basis functions, among others. Partitions of unity were also introduced in conjunction with compact support to improve conditioning while retaining certain desirable convergence properties. More recently, significant development of RBF methods for PDEs, namely RBF-FD methods, especially in the context of fluid flows, have been completed by Flyer and Fornberg et al.[80]. We present more detail on these methods later in this chapter.

There are also examples of mixed methods, in which meshless and meshfree solvers are coupled in an attempt to leverage the advantages of each. Once such example is from Javed [81] in which flow over a cylinder is simulated using an RBF-FD solver close to the cylinder, with a conventional finite difference

solver used in the rest of the domain. Particular attention must be paid to the interface between the meshed and meshfree solvers, where the points must align, so boundary conditions can be more readily transferred between the meshed and meshfree solvers. As this method has been used for FSI applications, more detail is available in section 5.1.1.

4.2 Approximation of Derivatives Using RBFs

The goal of this section is to detail methods of generating derivatives of an RBF-approximated function. This is most commonly done by generating a differentiation matrix D such that a derivative can be approximated through a single matrix multiplication.

There are two main methods of approximating derivatives with RBF interpolation: a standard method in which the interpolation equation is differentiated directly, and a so-called Lagrange method, in which RBF interpolation is reformulated in a form using cardinal basis functions. Both methods are equivalent, although one may be preferred over the other depending on application and implementation. Although in the current work differentiation is the focus, these methods apply equally to any linear operator, hence we prefer a more general notation for the time being.

4.2.1 Standard Method

Given the form of standard RBF interpolation, we can approximate the application of a linear operator $\mathcal{L}$ by simply applying it directly to the interpolation expression. That is, given

$$f(x) = \sum_i \alpha_i \phi(r_i) \tag{4.1}$$

where $r_i = \|x - x_i\|$, we apply $\mathcal{L}$ to each side and exploit linearity to obtain

$$\mathcal{L}f(x) = \sum_i \alpha_i \mathcal{L}\phi(r_i). \tag{4.2}$$

Hence calculating the result of applying the operator to f is reduced to applying it to the basis functions ϕ, which for most practical cases will have a simple closed form expression. Note also that the interpolation coefficients α_i remain unchanged between eq. (4.1) and eq. (4.2), and so when applying the approximation, we need only use the appropriate ϕ_{ij} matrix, and a single solve for α_i from eq. (4.1) suffices for *any* linear operator. For practical purposes, this means

that coefficients can be calculated once and reused for derivatives of all orders and variables.

Since eq. (4.2) is determined for a specific point x, solving the equation over all desired points of interpolation yields a matrix which can be applied to the vectors of the basis functions and their derivatives. In practice, it can be more convenient to develop a *differentiation matrix* that can be applied directly to f itself. As such, the Lagrange formulation described in section 4.2.2 is more often seen, and forms the basis of RBF-FD methods.

4.2.2 Lagrange/Variational Method

The Lagrange formulation (also referred to as the variational formulation), is an equivalent formulation of the method presented in section 4.2.1, and is the more commonly used approach when applying RBF approximations in RBF-FD methods in PDE. The equivalence was first shown by Madych and Nelson [49, 82] who posed the variational problem in an infinite-dimension Hilbert space, while a more practical result by Wu and Shaback [83] showed that there exists a solution to the corresponding finite-dimensional minimisation problem, and hence an equivalent Lagrange-type representation of eq. (4.1) given by

$$f(x) = \sum_j \psi_j(x) f(x_j) \tag{4.3}$$

where $\psi_i(x_j) = \delta_{ij}$ are cardinal basis functions. Cardinal basis functions on a set of interpolation points x_j, $j = 1, ..., N$, have the property

$$\psi_j(x_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} .$$

It should be noted that these cardinal functions are distinct from the Kronecker delta itself, because although our approximation is computed at discrete points, the domain of interest is still continuous, and no restrictions are placed on the value of ψ_j away from the interpolation points. Figure 4.1 gives an example of the normalised sinc function, often seen in wavelet approximations, defined on a 1D grid. Such a function clearly satisfies eq. (4.3) at integer points, but still has expressive power on the entire domain.

Applying the linear operator to eq. (4.3), we obtain

$$f(x) = \sum_i \mathcal{L}\psi_i(x) f(x_i).$$

It so happens that there is a very neat closed-form expression for these cardinal function terms, given as a ratio of determinants of matrices of the RBFs [84], and

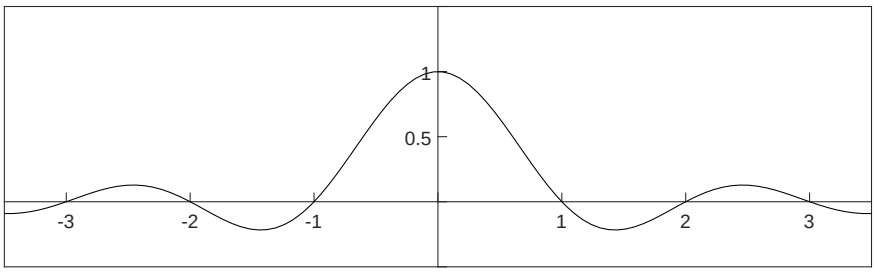


Fig. 4.1: Example cardinal basis function $\text{sinc}(r) = \sin(\pi r)/\pi r$, $r \neq 0$, on 1D unit grid.

in some situations it may be possible to calculate the action of the linear operator directly, however for practical purposes, the calculation of such determinants is difficult and generally unnecessary. As such, we prefer to simply compute the *values* of $\mathcal{L}\psi_i$ at a given approximation point x.

Letting $w_i = \mathcal{L}\psi_i(x)$, and using eq. (4.2), we obtain a linear system

$$Aw = \mathcal{L}B,$$

where $A_{ij} = \phi(r_{ij}) = \phi(\|x_i - x_j\|)$ and $B_i = \phi(r_i) = \phi(\|x - x_i\|)$. Once solved for w_i, the approximation is given at each x by

$$\mathcal{L}f(x) = \sum_i w_i f(x_i). \tag{4.4}$$

The primary advantage of this method is that the weights w_{ij} are related to the original function itself, evaluated at the points x_i, rather than the underlying RBFs, hence a differentiation matrix $D = A^{-1}\mathcal{L}B$ can be constructed such that

$$\mathcal{L}f = Df. \tag{4.5}$$

Both the standard and the variation formulations are readily localised, as the approximation at each point can be restricted in various ways to an appropriate number of neighbouring points, eliminating the need to invert a global matrix containing all points. In the RBF-FD method, this is generally done by way of a stencil, described in section 4.2.4.

4.2.3 Example

As a reasonably contrived example to demonstrate the machinery of section 4.2, consider the 1D function $f(x) = \sin(x)$. We wish to approximate the derivatives of f using RBF approximation. First let us choose a discrete set of points at which to anchor our approximations - 100 evenly spaced points across the

interval $[-2\pi, 2\pi]$, which we will denote x_i. Here we will use Wendland's C2 basis function, given by

$$\phi_i(x) = (1 - r_i)^4(4r_i + 1)$$

where $r_i(x)$ is the distance from x to the point x_i, normalised by a chosen support radius r_s using $r_i(x) = \|x - x_i\| r_s^{-1}$. The first two derivatives of this basis function are given by

$$\begin{aligned}\phi'(r_i) &= 20(x - x_i)(r_i - 1)^3 r_s^{-2} \\ \phi''(r_i) &= 20(r_i - 1)^2(4r_i - 1)r_s^{-2}.\end{aligned}$$

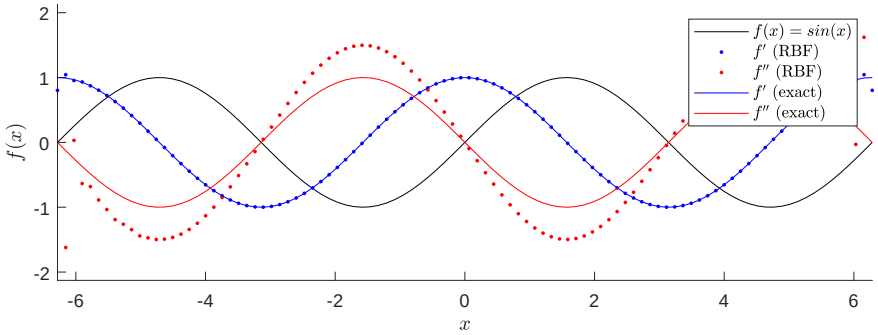
Evaluating eq. (4.1) at each interpolation point we get the linear system

$$f(x_j) = \sum_i \alpha_i \phi(r_i(x_j))$$

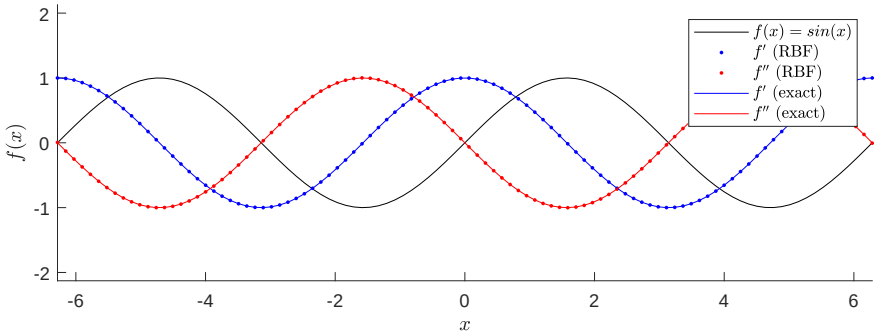
which we can solve for α_i. These α_i can then be used for all subsequent calculations. To approximate the derivatives of f at the point x, we then simply use eq. (4.2), i.e.

$$\begin{aligned}f'(x) &= \sum_i \alpha_i \phi'(r_i(x)) \\ f''(x) &= \sum_i \alpha_i \phi''(r_i(x)).\end{aligned}$$

Figure 4.2 shows the results of an RBF approximation compared to the exact derivatives. Here we can clearly see that for a large support radius (generally much greater than the domain) in fig. 4.2b, this approximation is quite accurate for both first and second order derivatives, even at the boundary, compared to the smaller radius of fig. 4.2a. This supports the notion that compact support generally detracts from smoothness of approximation [85], as extending the radius beyond the domain in essence removes compact support. In reality, care should be taken using large support radii (as well as non-compactly supported basis functions) in naive implementations of RBF methods, as they can result in a dense or ill conditioned linear system when solving for the α coefficients. Although compact support provides a natural localisation for the interpolation, that is, any given point is controlled by proximal points only, the reduction in smoothness can quickly become a limiting factor in approximating higher order derivatives. As such, the stencil method of localisation discussed in section 4.2.4 is more often employed in RBF-FD implementations, where accuracy and smoothness of derivatives is critical.



(a) $r_s = 1$



(b) $r_s = 1000$

Fig. 4.2: Approximation of derivatives using RBF method.

4.2.4 Stencil

To localise the approximation of RBF interpolations, RBF-FD methods often employ a stencil at each point. The most common approach of constructing stencils is a nearest neighbour approach, selecting the n_s nearest points to each point x in the domain as shown in fig. 4.3.

Once the stencils have been generated, we calculate the weights at each point by using eq. (4.4), which in general requires the inversion of the $n_s \times n_s$ RBF matrix for each stencil, hence for a domain of size N, the cost of this method is approximately $\mathcal{O}(Nn_s^3)$. For each point we then have a differentiation matrix D_x, which can be stacked together into a global differentiation matrix, allowing derivatives to be calculated on the domain with a single matrix multiplication. Since $n_s << N$, the resulting differentiation matrix is sparse, with approximately Nn_s non-zero elements. This helps greatly when required to solve linear systems in the approximation of PDE (e.g. Poisson's equation in section 4.3).

4.2.5 Choice of Basis Function

One of the main difficulties in any RBF method is the conditioning of the matrices involved, e.g. the differentiation matrix D in eq. (4.5), or the interpolation

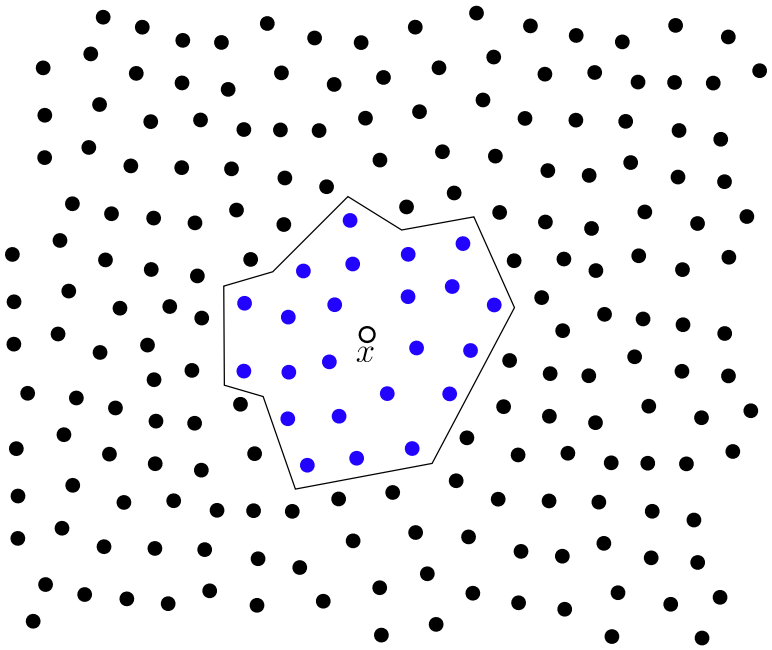


Fig. 4.3: Stencil of n_s nearest neighbours centred at the point x.

matrix H in eq. (2.7). In the case of general interpolation, the conditioning is important for accuracy, however when applying the approximations to derivatives for numerically solving PDEs, convergence of the numerical method must also be taken into account, which is significantly more sensitive to the conditioning of the matrices, as approximations are applied repeatedly. For infinitely smooth RBFs in the RBF-FD method, conditioning can be upheld for increasing resolution (shrinking radius) by increasing a shape parameter ϵ. The issue is that this causes convergence failure, as the more peaked the RBFs become, the less accurate the approximation is. To avoid this, it is possible to keep ϵ small and instead use a stabilisation algorithm [86]. However, as $\epsilon \to 0$, the function space spanned by the infinitely smooth RBFs is close to the space spanned by polynomials, hence adding polynomials in these cases provides little benefit [80]. In the case of PHS basis functions augmented with polynomials, it has been found that they provide a higher order algebraic convergence than the various infinitely smooth RBFs, and in fact numerical studies by Flyer et al. [80] have shown that the convergence is dominated by the higher order polynomial terms. In addition, PHS require no shape parameter optimisation, with the only significant parameter requiring selection is the order of the PHS itself. This order generally must be at least $d+1$, where d is the order of the derivative to be approximated. In the cases of order d or less, the PHS reduces firstly to a constant, then to zero, and the method reduces to polynomial interpolation.

For these reasons, we have opted to use PHS for our RBF-FD method. The

polyharmonic spline [87] basis functions are given by

$$\phi(r) = \begin{cases} r^k & \text{if } k \text{ odd} \\ r^k \log r & \text{if } k \text{ even} \end{cases}$$

4.3 Streamfunction-Vorticity Formulation for Unsteady, Incompressible, Viscous Flow

The formulation of the unsteady, incompressible Navier-Stokes equations in terms of a streamfunction and vorticity terms has considerable history, and was first used in a finite difference algorithm by Fromm [88] at Los Alamos laboratory. The formulation relies on introducing two scalar fields - a stream function ψ and the vorticity ω. In this way we are able to eliminate pressure and force terms (assuming their fields are conservative), resulting in a simple but powerful formulation.

4.3.1 Vorticity Transport Equation

To obtain the vorticity-transport equation, we begin with the incompressible Navier-Stokes equation

$$\frac{\partial V}{\partial t} + (V \cdot \nabla)V = \nu \nabla^2 V - \frac{1}{\rho}\nabla p + F. \tag{4.6}$$

Here V is velocity, ν is the kinematic viscosity, ρ is the density, p is the pressure, and F contains any body accelerations acting on the flow field, e.g. gravity. Next we take the curl of eq. (4.6) and apply the definition of vorticity, i.e. $\omega = \nabla \times V$. Since we assume variables and solutions have sufficient smoothness, we apply Schwarz's theorem allowing us to switch the order of the various linear operators to obtain

$$\frac{\partial \omega}{\partial t} + (V \cdot \nabla)\omega = \nabla \times \left(\nu \nabla^2 V - \frac{1}{\rho}\nabla p + F \right).$$

Assuming constant density and dynamic viscosity throughout the domain we get

$$\frac{\partial \omega}{\partial t} + (V \cdot \nabla)\omega = \nu \nabla \times \nabla^2 V - \frac{1}{\rho}\nabla \times \nabla p + \nabla \times F. \tag{4.7}$$

For conservative body forces, i.e. when F is the gradient of some sufficiently smooth scalar field ϕ, $\nabla \times F = \nabla \times \nabla\phi = 0$, and since pressure is already a

scalar field (smooth from previous assumption) we also have $\nabla \times \nabla p = 0$, hence eq. (4.7) becomes

$$\frac{\partial \omega}{\partial t} + (V \cdot \nabla)\omega = \nu \nabla \times \nabla^2 V.$$

Finally, again switching the order of linear operations and applying the definition of vorticity we arrive at vorticity-transport equation

$$\frac{\partial \omega}{\partial t} + V \cdot \nabla \omega = \nu \nabla^2 \omega. \tag{4.8}$$

This transport equation is for the general (3D) case. In two dimensions, we can simplify further. Expanding eq. (4.8) we obtain the equations

$$\begin{aligned}
\frac{\partial \omega_x}{\partial t} + u\frac{\partial \omega_x}{\partial x} + v\frac{\partial \omega_x}{\partial y} + w\frac{\partial \omega_x}{\partial z} &= \nu\left(\frac{\partial^2 \omega_x}{\partial x^2} + \frac{\partial^2 \omega_x}{\partial y^2} + \frac{\partial^2 \omega_x}{\partial z^2}\right) \\
\frac{\partial \omega_y}{\partial t} + u\frac{\partial \omega_y}{\partial x} + v\frac{\partial \omega_y}{\partial y} + w\frac{\partial \omega_y}{\partial z} &= \nu\left(\frac{\partial^2 \omega_y}{\partial x^2} + \frac{\partial^2 \omega_y}{\partial y^2} + \frac{\partial^2 \omega_y}{\partial z^2}\right) \\
\frac{\partial \omega_z}{\partial t} + u\frac{\partial \omega_z}{\partial x} + v\frac{\partial \omega_z}{\partial y} + w\frac{\partial \omega_z}{\partial z} &= \nu\left(\frac{\partial^2 \omega_z}{\partial x^2} + \frac{\partial^2 \omega_z}{\partial y^2} + \frac{\partial^2 \omega_z}{\partial z^2}\right).
\end{aligned}$$

Fortunately, we can eliminate two of these equations and simplify the third. From the definition of vorticity, $\omega = \nabla \times V$, we also have the equations

$$\begin{aligned}
\omega_x &= \frac{\partial w}{\partial y} - \frac{\partial v}{\partial z} \\
\omega_y &= \frac{\partial w}{\partial x} - \frac{\partial u}{\partial z} \\
\omega_z &= \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}.
\end{aligned} \tag{4.9}$$

Since we are now in two dimensions, $w = 0$, and any derivatives in the z direction vanish. Hence we have $\omega_x = \omega_y = 0$ and the expanded equations reduce to the single equation. We can further simplify by ensuring that our simulations have unit freestream velocity and characteristic length, in which case we now have

$$\frac{\partial \omega_z}{\partial t} + u\frac{\partial \omega_z}{\partial x} + v\frac{\partial \omega_z}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 \omega_z}{\partial x^2} + \frac{\partial^2 \omega_z}{\partial y^2}\right). \tag{4.10}$$

Combined with the stream function in section 4.3.2, this is the form of the vorticity transport that we will solve numerically.

4.3.2 Streamfunction

Since eq. (4.10) still has 3 unknowns u, v, and ω_z, we need more equations for a solution. Given our previous assumptions of smoothness on the velocity field components u and v, the velocity vector field itself is smooth (enough), and we can safely posit the existence of a scalar field ψ such that

$$u = \frac{\partial \psi}{\partial y} \tag{4.11}$$

$$v = -\frac{\partial \psi}{\partial x}. \tag{4.12}$$

Substituting this into eq. (4.10) we obtain

$$\frac{\partial \omega_z}{\partial t} + \frac{\partial \psi}{\partial y}\frac{\partial \omega_z}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega_z}{\partial y} = \nu \left(\frac{\partial^2 \omega_z}{\partial x^2} + \frac{\partial^2 \omega_z}{\partial y^2} \right),$$

and we have now reduced our unknowns and required equations to two. It should be noted that continuity of the flow using the stream function is automatically satisfied since

$$\nabla \cdot V = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{\partial^2 \psi}{\partial x \partial y} - \frac{\partial^2 \psi}{\partial y \partial x} = 0. \tag{4.13}$$

4.3.3 Poisson Equation

We still require an additional equation to be able to solve eq. (4.10). Recall eq. (4.9) defining the vorticity in terms of derivatives of the velocity. Now that we have defined the stream function ψ, we can express eq. (4.9) as

$$\omega_z = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = -\left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} \right).$$

Switching signs and dropping the z subscript for simplicity, we obtain the Poisson equation

$$\Delta \psi = -\omega.$$

We now have the following coupled equations, which are collectively known as the (non-dimensional) stream-function vorticity formulation for 2D incompressible flow:

$$\begin{aligned} \frac{\partial \omega_z}{\partial t} + \frac{\partial \psi}{\partial y}\frac{\partial \omega_z}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega_z}{\partial y} &= \frac{1}{Re}\left(\frac{\partial^2 \omega_z}{\partial x^2} + \frac{\partial^2 \omega_z}{\partial y^2} \right) \\ \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} &= -\omega. \end{aligned} \tag{4.14}$$

4.3.4 Boundary Conditions

An area of great interest when simulating the NS equations in stream-vorticity formulation is the determination of an appropriate boundary condition for vorticity. For some boundaries, the condition is obvious and/or simple to enforce, e.g. at a farfield freestream boundary, one would expect that $\omega = 0$ uniformly. However in the case of a solid wall, the boundary condition is either not apparent, or not easily applied.

In the case of the stream-vorticity formulation, the boundary conditions at an impermeable, no-slip wall are often stated as

$$\psi = 0, \quad \frac{\partial \psi}{\partial n} = 0.$$

The first of these conditions could be equivalently stated as requiring zero tangential derivative, and as such gives vanishing normal velocity, and hence impermeability. The second, vanishing normal derivative, gives zero tangential velocity, and hence provides a no-slip condition.

Unfortunately, as these are both conditions on the stream function ψ, i.e. a Cauchy-type boundary condition, the Poisson equation $\Delta\psi = -\omega$ often becomes ill-posed, and existence/uniqueness of a solution is no longer guaranteed. In addition, multiple conditions on a single boundary are difficult to apply in the RBF-FD case. Basic experiments with alternating Dirichlet/Neumann boundary points in an attempt to force the Cauchy conditions via mixed conditions, as well as least-squares approximation of the over-constrained problem, were largely unsuccessful. Hence it is desirable to use a method which provides a direct condition on ω.

Basic Formulation of Vorticity Boundary Condition

The most easily derivable boundary condition for ω, and likely the most natural, is to simply enforce the governing equations at the boundary using the Poisson equation, i.e. once the Poisson equation has been solved in the update loop, in can be used in reverse to specify ω at the boundary, i.e.

$$-\omega = \Delta\psi = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}.$$

This condition is natural and has been used with some success in various CFD configurations [89]. However, it may be more or less useful, or difficult to apply, depending on the method chosen to approximate the velocity derivatives, or indeed the overall method of solution of the governing equations. As such, some

further consideration must be given to the solution method when choosing a boundary condition for ω.

Vorticity Boundary Conditions for Meshless Methods

For meshed solvers, two common methods for obtaining boundary conditions for ω are that of Thom [90], and one attributed to Jensen[1] by Roache [92]. As we will see, Jensen's formula can be derived in a similar manner to Thom's, with a higher order starting approximation. Although these methods are not directly applicable to the current meshless case, we will introduce them, then discuss how they can be adapted to the meshless case.

In Thom's method, we first consider a wall of arbitrary orientation with normal n and moving at velocity v_t as shown in fig. 4.4, where ψ_{int} is the value of the streamfunction at some interior point at distance h, and ψ_{wall} is the value of the streamfunction at the wall. Note that both these quantities for ψ are known, ψ_{int} from the solve on the Poisson equation, and ψ_{wall} from the Dirichlet boundary conditions.

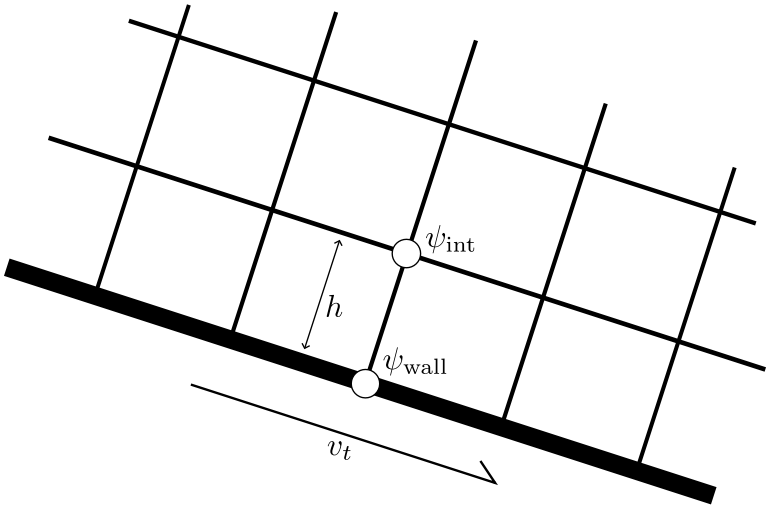


Fig. 4.4: Thom's method for arbitrary moving wall in meshed solver.

Taking a second order Taylor approximation for the streamfunction ψ at the wall in terms of ψ_{int}, we obtain

$$\psi_{\text{int}} = \psi_{\text{wall}} + h \left.\frac{\partial \psi}{\partial n}\right|_{\text{wall}} + \frac{h^2}{2} \left.\frac{\partial^2 \psi}{\partial n^2}\right|_{\text{wall}} + \mathcal{O}(h^3).$$

[1]Roache appears to have misspelt V.G. Jenson's name as 'Jensen' when referencing the original 1959 work [91]. Unfortunately for Jenson, this error has been replicated throughout almost all of the literature on the subject - a tradition I have elected not to break.

Observe from the interpretation of the streamfunction that $\frac{\partial \psi}{\partial n}\big|_{\text{wall}} = v_{\text{t}}$. Also, by the definition of vorticity, $\frac{\partial^2 \psi}{\partial n^2}\Big|_{\text{wall}} = -\omega_{\text{wall}}$. Hence we have

$$\psi_{\text{int}} = \psi_{\text{wall}} + h v_{\text{wall}} - \frac{h^2}{2}\omega_{\text{wall}} + \mathcal{O}(h^3).$$

Solving for ω_{wall} then gives a Dirichlet condition on the vorticity at a moving wall:

$$\omega_{\text{wall}} = \frac{2}{h^2}\left(\psi_{\text{wall}} - \psi_{\text{int}}\right) + \frac{2}{h} v_{\text{wall}} + \mathcal{O}(h). \tag{4.15}$$

Note that eq. (4.15) now has a first order truncation error. It is also interesting to note that although the error is first order, the convergence remains second order, as shown by Huang and Wetton [93]. The error can also be reduced to second order by using an additional interior point for ψ and a higher order Taylor approximation (Jensen's method) which results in:

$$\omega_{\text{wall}} = \frac{1}{2h^2}\left(7\psi_{\text{wall}} - 8\psi_{\text{int1}} + \psi_{\text{int2}} + 6h v_{\text{wall}}\right) + \mathcal{O}(h^2). \tag{4.16}$$

It should also be noted that we have never assumed zero normal velocity ($v_n = 0$) at the wall - the term simply never appears in the expression, since velocity normal to the wall does not produce any local vorticity. As such, this boundary condition is the *general* vorticity condition for non-slip walls, moving (normally and/or tangentially) or otherwise, and can be directly applied in the FSI case of chapter 5.

Turning to the meshless case, we have no adjacency information, hence no specific concept of the first or second interior points normal to the boundary. This is especially true if the point cloud is scattered, rather than in an organised grid. A number of ways have been proposed to solve this issue, including the use of locally orthogonal nodes near boundaries, or methods using ghost nodes external to the domain [94].

Here we propose a new method for the calculation of Dirichlet conditions, using only the normal direction of each boundary node. This method simplifies the calculations and eliminates the need for any complex near-boundary mesh-like structure, or the calculation of field values at external ghost nodes. We make use of a local approximation of ψ internal to the boundary, which we can then use in eq. (4.15) or eq. (4.16) directly, rather than having to discretise the more complex Poisson equation. Since most of the machinery (stencils, normals, basis functions, etc.) for interpolation is already setup from the construction of the

domain, the approximation of ψ at the appropriate interior points is reasonably straightforward. The basic geometry is shown in fig. 4.5.

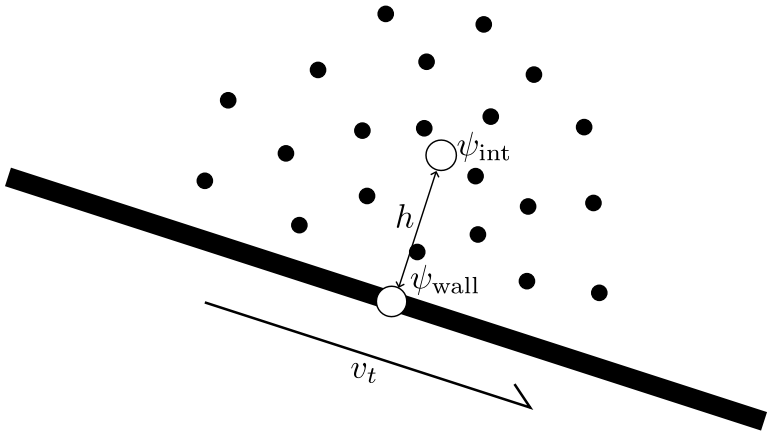


Fig. 4.5: Thom's method for arbitrary moving wall in meshless solver where ψ_{int} is approximated from ψ at the surrounding points.

As is usual for RBF methods, most of the calculation involved in the interpolation can be done prior to runtime. First, we must determine the location of the point at which we wish to approximate ψ. This is done using the normal of the boundary point, specified when creating the domain, combined with a distance h which is akin to the grid spacing parameter in the meshed case. This parameter can be specified manually (globally or locally), or calculated on the fly using a local density measure. In cases where the density of the boundary points vary significantly, a local density measure is preferred. An approximation for ψ_{int} can then be obtained using a stencil S about the interior point(s), either by nearest neighbour or radius. The stream function is then approximated in the standard way by

$$\psi_{\text{int}} = \sum_{i \in S} \alpha_i \phi_i(r)$$

where α_i can be determined ahead of time.

Of course RBF approximation is not the only possible means of determining ψ_{int}, any suitable meshless interpolation method can be used.

Below we give a summary of boundary conditions for geometrically stationary boundaries, both streamfunction and vorticity. Streamfunction boundary conditions for moving walls (normal motion) are discussed in chapter 5.

Wall (slip)

A slipping wall is perhaps the simplest boundary condition, as it generates no vorticity (the fluid does not get 'stuck' to the wall). The conditions on ψ and ω

are

$$\begin{aligned}\psi &= C \text{ , for some constant } C\\ \omega &= 0.\end{aligned}$$

Wall (non-slip)

A non-slip wall has the same condition on ψ as the slipping case (no fluid should be moving tangentially through the wall), and the vorticity produced is described by Taylor approximations normal to the wall, given in section 4.3.4 by either Thom's or Jensen's formulas. The conditions are

$$\begin{aligned}\psi &= C, \text{ for some constant } C\\ \omega &= \frac{2}{h^2}\left(\psi_{\text{wall}} - \psi_{\text{int}}\right) + \frac{2}{h}v_{\text{wall}} \text{ (Thom)}\\ \omega &= \frac{1}{2h^2}\left(7\psi_{\text{wall}} - 8\psi_{\text{int1}} + \psi_{\text{int2}} + 6hv_{\text{wall}}\right) \text{ (Jensen).}\end{aligned}$$

Inlet/Outlet

Assuming we have steady flow at a vertical inlet or outlet with a known (defined) normal velocity distribution $u = u(x_{\text{wall}}, y_{\text{wall}}), v = 0$, we have $\partial_x u = \partial_x v = 0$, hence by the continuity equation eq. (4.13), we also have $\partial_y v = 0$. Then by the definition of vorticity, the vorticity boundary condition is given by

$$\omega = -\frac{\partial u}{\partial y}.$$

Since we have defined the inlet/outlet velocity distribution, we can calculate the condition on ψ along the vertical as

$$\psi = \int u \, dy.$$

This condition can be adapted as required to horizontal inlets and outlets by switching u and v. For arbitrarily oriented inlets/outlets, the definition of vorticity can be used directly, since both u and v may be non-zero.

Corners

One issue of concern in using mesh-based finite difference methods with the stream-vorticity formulation, is that the vorticity produces a singularity at non-smooth boundary corners. A number of solutions have been proposed for this, including a small displacement of the computational domain from the physical surface to avoid the singularity [95], and an analytic solution applied over the computation solution [96]. In the current work, these issues are considered to be generally overcome by two features inherent in the RBF-FD method. Firstly, there is a smoothing effect of the RBFs, and hence the effect of the singularity can be somewhat mitigated by the numerics of the solution. Secondly, corners can easily be further artificially smoothed out by adjusting the normal at the corner point, often set to be an intermediate normal determined by normals of the two adjacent points when generating the domain. That said, as there is no adjacency or connectivity information between points, the notion of a 'corner' is somewhat non-existent in the meshless case, which may be reflected by the lack of discussion of the issue in the literature. See section 4.4.1 for comments on potential future work.

4.3.5 Hyperviscosity Term

One issue that arises in using the RBF-FD approach is that of instabilities caused by high frequency modes, often introduced by numerical artifacts. In the case of PDE without dissipative terms, this phenomenon is not limited to RBF-FD methods, but methods used in traditional FD solutions to deal with the issue are not generally applicable in the RBF-FD case. It has since been seen that the natural scattering of RBF-FD stencils can cause high frequency modes to grow even in PDEs with dissipative terms [80]. This is especially relevant to the current work, where at low Re, the dissipative viscosity can be relatively small.

To solve this issue, a number of methods were proposed by Fornberg and Lehto [97]. Firstly, on a 1D grid, a Laplacian can be approximated with regular FD methods. The maximum order of the Laplacian is dependent on the size of the stencil and the required accuracy - as order increases, accuracy necessarily decreases. Secondly, a global method in which the inverse of the RBF matrix A is used to filter high frequency oscillations. Since A^{-1} has the same eigenvectors with inverse eigenvalues, which have been shown experimentally to grow quickly [97], the effect of applying A^{-1} to the RHS of the equations is to damp low frequency eigenmodes slowly, but high frequency eigenmodes quickly. With appropriate scaling, the damping of the low frequency modes is negligible. This method also has advantage of being exceedingly simple, but does inherit some of the cost issues of global methods.

The most successful method, at least for fluid flows, has been shown to be that of a hyperviscosity operator applied at the stencil level, also proposed in [97]. This takes the form of a higher order Laplacian, $\gamma\Delta^k$, added to the RHS (i.e. with terms not explicit in time) of the equation, with γ a constant that is often exceedingly small, usually on the order N^{-k} where N is the number of nodes in the entire domain. For large domains, this constant can be well beyond machine precision. Methods of choosing the parameters k and γ are given in the appendix of [80]. That work also proposed improvements to the method that reduced the reliance on parameters that were often difficult to tune. Care must be taken to choose a sufficiently high k so as to only damp out frequencies beyond those of interest in the solution, however, depending on choice of RBF, k may be limited. For example, in the case of polyharmonic splines where $\phi(r) = r^m$, derivatives beyond $m-1$ are only piecewise smooth, and eventually zero, and we have the general restriction $m \geq 2k+1$.

Adding the hyperviscosity term to eq. (4.14) gives

$$\frac{\partial\omega}{\partial t} + \frac{\partial\psi}{\partial y}\frac{\partial\omega}{\partial x} - \frac{\partial\psi}{\partial x}\frac{\partial\omega}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2\omega}{\partial x^2} + \frac{\partial^2\omega}{\partial y^2}\right) + \gamma\left(\frac{\partial^{2k}\omega}{\partial x^{2k}} + \frac{\partial^{2k}\omega}{\partial y^{2k}}\right) \tag{4.17}$$

The calculation of the higher order derivatives required in eq. (4.17) will depend on the choice of basis function. Since our basis functions are radially symmetric, the Laplacian operator in dimension d is given by

$$\Delta = \frac{\partial^2}{\partial r^2} + \frac{d-1}{r}\frac{\partial}{\partial r}.$$

For PHS of order m in dimension 2, i.e. $\phi(r) = r^m$, the calculation is reasonably straightforward, with the Laplacian given by

$$\Delta r^m = \left(\frac{\partial^2}{\partial r^2} - r^{-1}\frac{\partial}{\partial r}\right) r^m = m^2 r^{m-2}.$$

This formula can then be repeatedly applied to obtain a direct expression for the hyperviscosity operator of order k as applied to a PHS basis function:

$$\Delta^k r^m = m^{2k} r^{m-2k}.$$

4.3.6 Solver Structure

We now must solve the coupled equations

$$\begin{aligned} \frac{D\omega}{Dt} &= \nu\nabla^2\omega + \gamma\Delta^k\omega \\ \nabla^2\phi &= -\omega \end{aligned}$$

At the highest level, we simply repeat a two-step process: solve the vorticity transport equation for ω, use the result to solve the Poisson equation for ψ, then use this result to again solve the vorticity transport, and so on. Of course there are a number of significant details that must be considered when implementing the process.

Figure 4.6 shows the general steps involved in the solution of the 2D streamfunction-vorticity formulation. Each step is detailed below.

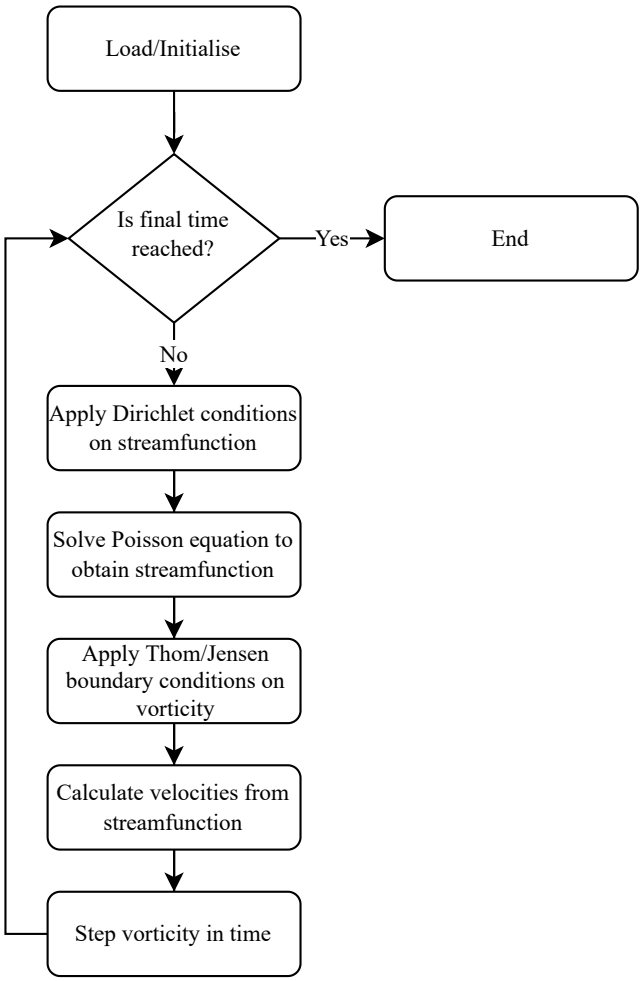


Fig. 4.6: Flow diagram for fluid solver with non-moving boundaries.

Step 0: Initialise Simulation

As ψ on the interior is determined explicitly before it is required by the vorticity transport equation (via velocity terms), there is no need to initialise ψ on the interior. For ω, we initialise the interior of the domain to zero. Dirichlet conditions on ψ and ω are set during the main loop as required, so do not require initialisation. Time is initialised to zero.

The main computational cost of initialisation is the calculation of the RBF-FD weights, although there are some significant savings to be made. These points are discussed further in section 5.2.5, as this cost is very much relevant in the FSI case, where it must not only be computed at initialisation, but at each time step.

Step 1: Apply Dirichlet Boundary Conditions

We begin the main loop by initialising ψ to be 0 inside the domain, and equal to the prescribed Dirichlet conditions at the boundaries. This is followed by initialising ω in a similar manner. For non-slip boundaries, i.e. boundaries where we intend to use eq. (4.15) to determine vorticity, the initial value will be overwritten immediately after solving the Poisson equation, and hence never used. For this reason, the initial ω at these boundaries can be arbitrary. Time is initialised to 0.

Step 2: Solve Poisson Equation

The stream function ψ on the interior is determined by solving

$$\nabla^2 \psi = -\omega$$

using the differentiation matrix A_{Poisson}, that is, solving the linear system

$$A_{\text{Poisson}} \psi = -\omega.$$

Step 3: Apply Thom's Equation for Dirichlet Condition on Vorticity

For each boundary point, ψ is approximated at an internal point in the normal direction as per section 4.3.4. This can be achieved using any interpolation method available.

Step 4: Calculate Velocities

The velocities are calculated by taking the directional derivatives of the stream-function ψ using the RBF-FD derivative matrices A_x and A_y, i.e.

$$\begin{aligned} v_x &= A_y \psi \\ v_y &= -A_x \psi \end{aligned}$$

Step 5: Step Vorticity Transport Equation in Time

The current solver uses a standard RK4 algorithm to step eq. (4.17) in time, preferred for its relative simplicity and stability. RK4 methods have been used

with success in RBF-FD solutions for Navier-Stokes equations, e.g. [80]. Given the equation

$$\frac{dy}{dt} = f(t, y),$$

where $f(t, y)$ is known for all t, and $y_n = y(t_n)$ is known at some time t_n, we can estimate $y_{n+1} = y(t_n + \Delta t)$ by

$$y_{n+1} = y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$\begin{aligned} k_1 &= f\left(t_n, y_n\right) \\ k_2 &= f\left(t_n + \Delta t/2, y_n + \Delta t k_1/2\right) \\ k_3 &= f\left(t_n + \Delta t/2, y_n + \Delta t k_2/2\right) \\ k_4 &= f\left(t_n + \Delta t, y_n + \Delta t k_3\right). \end{aligned}$$

4.3.7 Validation

Here we benchmark the solver against some standard test cases found in the literature.

Free Stream

As an initial sanity test, we use perhaps the most basic case possible, an empty horizontal freestream of velocity v_0 on a rectangular domain of unit height and length 2. This example demonstrates the setting of inlet/outlet boundary velocities (i.e. velocities normal to the boundary). In this case we have a very basic set of Dirichlet conditions for both the streamfunction ψ and the vorticity ω. As we are in a straight flowing freestream, $\omega = 0$ uniformly across the domain, including all boundaries. An impermeability condition on the top and bottom boundaries is given by

$$\left.\frac{\partial \psi}{\partial x}\right|_{\text{top}} = \left.\frac{\partial \psi}{\partial x}\right|_{\text{bottom}} = 0.$$

This is achieved by choosing ψ to be constant along these boundaries.

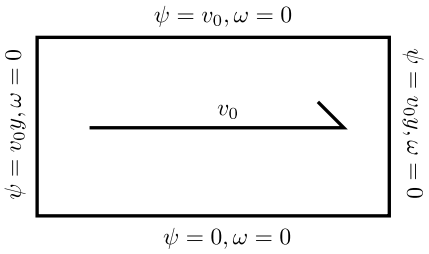
The inlet/outlet velocities are set by requiring that

$$\left.\frac{\partial \psi}{\partial y}\right|_{\text{inlet}} = \left.\frac{\partial \psi}{\partial y}\right|_{\text{outlet}} = v_0.$$

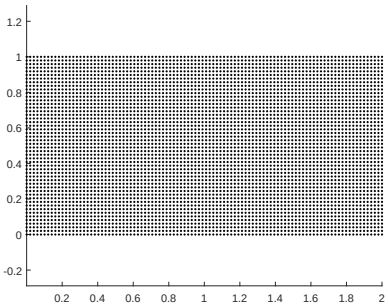
We integrate to obtain the Dirichlet condition

$$\psi|_{\text{inlet}} = \psi|_{\text{outlet}} = v_0 y + C$$

where we are of course free to set the integration constant as we please. Noting the additional restriction that ψ must be continuous around the boundary, the choice of constant for ψ on the top and bottom boundaries are set as v_0 and 0 respectively. These Dirichlet conditions are summarised in fig. 4.7a.



(a) Boundary conditions



(b) Computational domain (50×100)

Fig. 4.7: Freestream test case setup.

Given that the vorticity is uniformly zero on the entire domain, this cases reduces to solving the Poisson equation with Dirichlet conditions, and hence converges after a single timestep. The results are as expected, and the velocity at the inlet/outlet is replicated throughout the domain. The streamlines are shown in fig. 4.8, showing the expected inclined plane solution. The maximum error between the result of the RBF-FD Poisson solver and the analytic solution is found to be 5.18×10^{-4}.

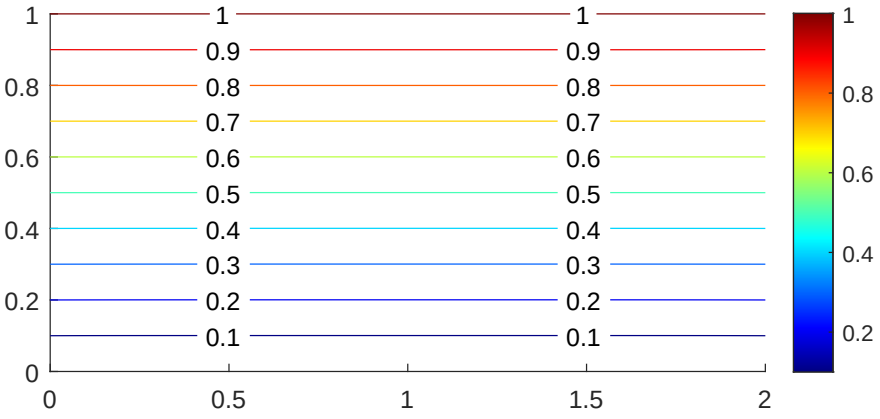


Fig. 4.8: Streamlines for freestream test case.

Pipe Flow

Flow in a pipe is modelled in a rectangular domain with unit height and length 4, with a computational grid of 25×100. The velocity is fixed at the left and right boundaries, with an impermeable wall condition on the top and bottom boundaries. The vorticity at the top and bottom walls is ω_{wall} given by eq. (4.15). The setup is shown in fig. 4.9.

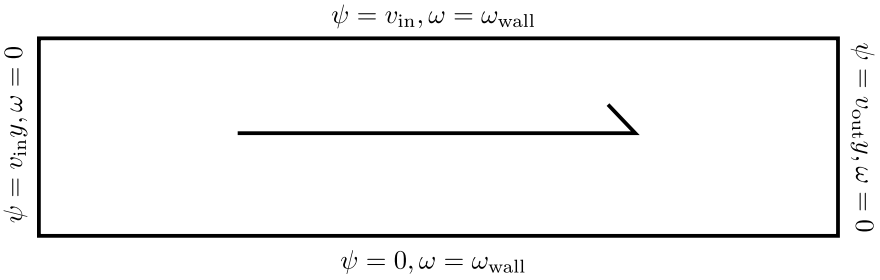


Fig. 4.9: Boundary conditions for 2D pipe flow.

Figure 4.10 shows the horizontal velocity is most developed at approximately $x = 3$. Figure 4.11 shows the velocity profile of the flow at regular intervals along the pipe. As the flow becomes fully developed, this profile is as expected for 2D turbulent pipe flow. Total flow through the midpoint at the most developed stage is approximately 0.9949, indicating that the mass conservation is captured.

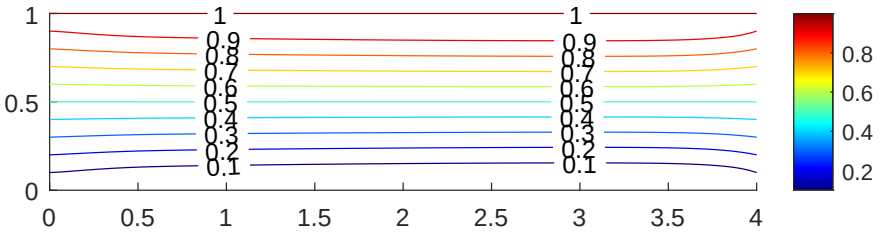


Fig. 4.10: Streamlines for 2D pipe flow.

Lid-Driven Cavity

Here we show the results of a standard square lid-driven cavity simulation. The domain is the region $[0, 1] \times [0, 1]$, and has been modelled with a 50×50 grid of points.

Since there is no flow in or out of the cavity, the entire boundary is a streamline, that is, a level set of the stream function, and can be set to an arbitrary constant, so for ease of calculation we choose $\psi = 0$ at all boundaries. The velocities at the left and right walls, as well as the floor, are also zero, corresponding to the non-slip condition. The velocity along the top is v_0. The velocity boundary

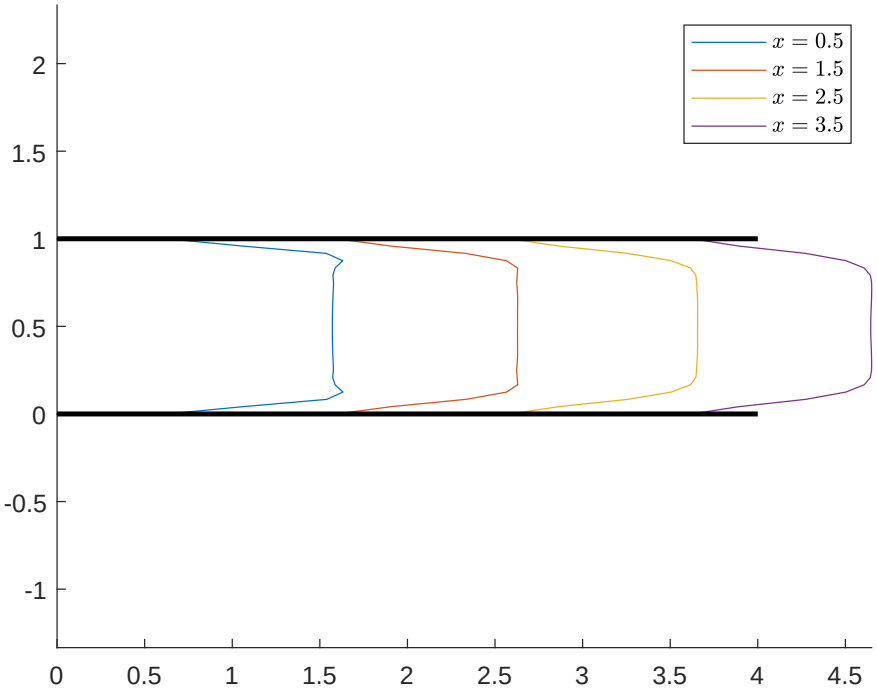


Fig. 4.11: Horizontal velocity profiles for 2D pipe flow.

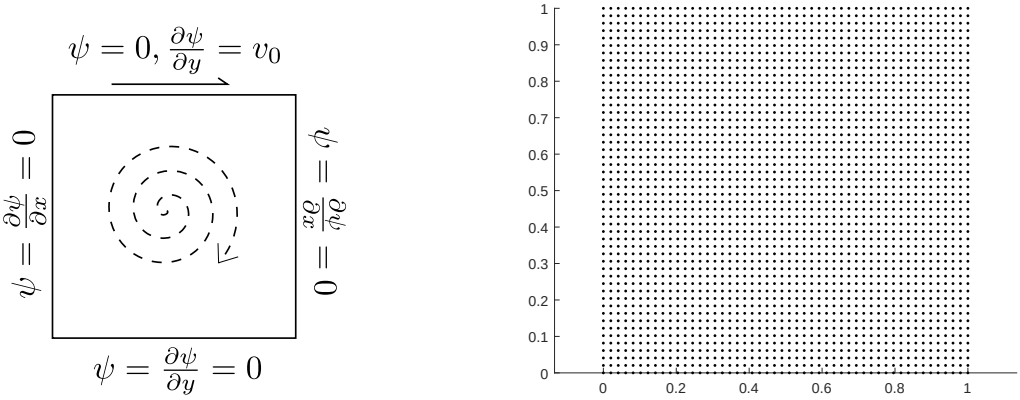
conditions are given by specifying $\frac{\partial \psi}{\partial x}$ or $\frac{\partial \psi}{\partial y}$ at each boundary, depending on the normal. The boundary conditions are summarised as follows:

$$\begin{aligned} \psi = 0, \frac{\partial \psi}{\partial x} = 0 \quad &\text{(left wall)} \\ \psi = 0, \frac{\partial \psi}{\partial x} = 0 \quad &\text{(right wall)} \\ \psi = 0, \frac{\partial \psi}{\partial y} = 0 \quad &\text{(bottom wall)} \\ \psi = 0, \frac{\partial \psi}{\partial y} = v_0 \quad &\text{(top wall)} \end{aligned} \tag{4.18}$$

Figure 4.12 shows both the domain with the boundary conditions of eq. (4.18), and the computational domain.

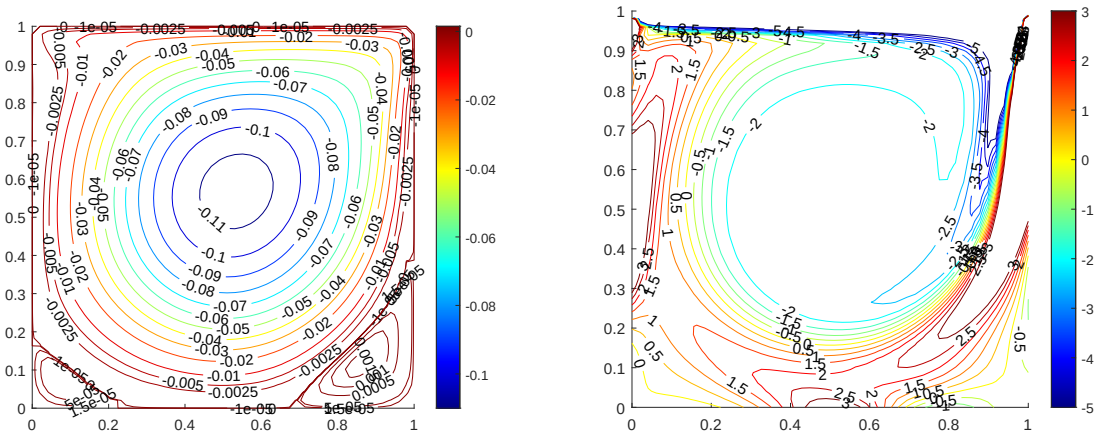
The simulation was run until $t = 30$s, with $\Delta t = 0.0005$. The resulting velocity and vorticity contours are shown in fig. 4.13. Both the primary and secondary vortices are captured well.

Figure 4.14 show the perpendicular velocities along vertical and horizontal cross-section through the geometric centres of the domain. These results are in close agreement with Ghia et al. [98] and Chinchapatnam et al.[94].



(a) Boundary conditions *(b)* Computational domain (50×50)

Fig. 4.12: Lid-driven cavity setup.



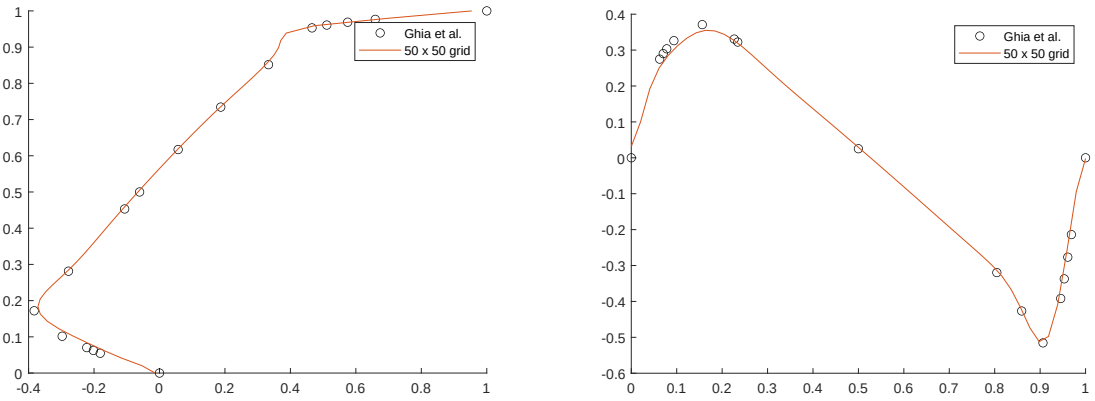
(a) Velocity *(b)* Vorticity

Fig. 4.13: Contours for lid-driven cavity with $Re = 1000$, 50×50 grid.

Flow Past a Square Cylinder

The final example we give is flow past a square cylinder. The setup is shown in fig. 4.15. We model the farfield as we did in the case of the freestream in section 4.3.7 with inlet and outlet having prescribed velocity, and impermeable top and bottom boundaries. We then add to this a non-slip unit square with blockage ratio 1/5. The grid density is set to a density of 20 points per unit length, resulting in a grid approximately 300×100. The flow was run at $Re = 40, 60, 80, 100$.

When the Reynolds number reaches a critical value ($Re \approxeq 60$), the flow behind the cylinder becomes unsteady, exhibiting the well known von Kármán vortex street structure, where alternating vortices are shed from the trailing edges of the cylinder. These vortices can be clearly seen in both the velocity magnitude field of fig. 4.16, and the vorticity fields of fig. 4.17 and fig. 4.18. To quantify the flow,



(a) x velocity (b) y velocity

Fig. 4.14: Comparison of velocities to previous results [98] along vertical and horizontal cross-sections through the geometric centres of the domain.

we use two calculated values, the Strouhal number St and the time-averaged drag coefficient $\overline{C}_D$.

The Strouhal number is a measure of the vortex shedding, which is calculated based on the shedding velocity by

$$St = \frac{fD}{u_0}.$$

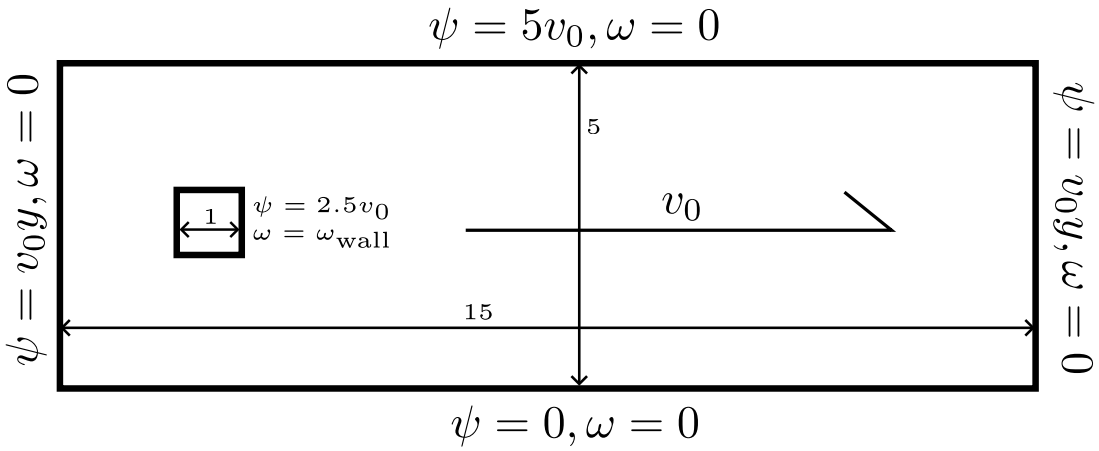
Since we have previously set the freestream velocity u_0 and the characteristic distance D to be 1 (so as to maintain the non-dimensional properties of eq. (4.14)), the Strouhal number reduces to the shedding frequency.

The time-averaged drag coefficient is a measure of the mean drag once the vortex shedding is fully developed. It is defined by

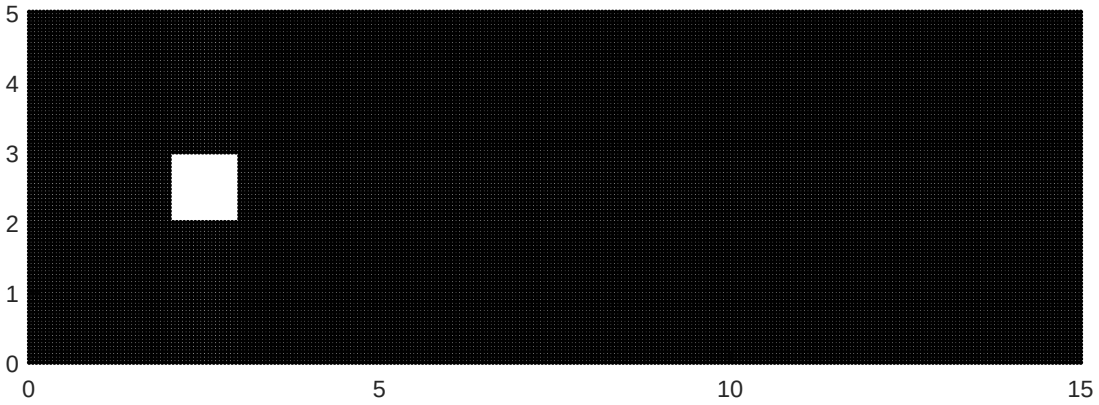
$$\overline{C}_D = \frac{1}{t_b - t_a} \int_T C_D dt$$

where $T = [t_a, t_b]$ is a sufficiently long time period during the fully developed vortex shedding.

Figure 4.19 shows a grid convergence study of Strouhal number St and average amplitude of C_L for $Re = 100$ at varying point cloud densities. Simulations were also run $\Delta t = 0.005, 0.001, 0.0005$ with similar results. Note that no variation was found in St as it can only be computed with accuracy proportional to Δt, i.e. 5×10^{-4}. Previous experiments and simulations of flow past a square cylinder have shown that St is expected to vary between 0.15 for sharp cornered squares, and 0.2 for chamfered squares (approaching a circle) [99, 100]. As the normals at the corners of the square have been set to point diagonally away from the square, this square can be considered slightly chamfered, and as such, we expect



(a) Boundary conditions.



(b) Computational domain 300×100

Fig. 4.15: Setup for square cylinder in a freestream flow.

St to be between the two cases. As seen in fig. 4.19, we have $St = 0.186$, which is in agreement with the range presented by Vikram [99], and in close agreement with similar simulations of Sahu et al. [101], although slightly different blockage ratios have been used.

Figure 4.20 shows the variation of the Strouhal number with Reynolds number. The results are compared against some similar simulations by Sahu et al. [101], although for slightly different blockage ratios $\beta = 1/4, 1/6$. The results show good qualitative agreement (increasing St with increasing Re), and show reasonable quantitative agreement with the $\beta = 1/6$ case. Differences are to be expected given the difference in blockage ratio, as well as a slightly different domain geometry.

4.4 Conclusions & Future Work

This chapter has described how RBF interpolation and approximation can be used to estimate derivatives via the RBF-FD method. The solution of the incompressible Navier Stokes equations in the streamfunction-vorticity formulation has been demonstrated and agrees well with existing literature, capturing the phenomenon of von Kármán vortex streets. A novel method of applying boundary conditions on vorticity was also presented, based on the traditional meshed methods of Thom and Jensen. It is hoped that this method can simplify existing implementations, and inform the development of other methods, as the application of boundary conditions in meshless methods is an ongoing challenge. Although the validations presented here have used primarily Cartesian point clouds, the RBF-FD method has been shown to apply equally well to less structured point clouds [94, 97, 80][2].

The primary goal of this chapter was to validate the fluid solver for use in the next chapter, where we couple it with a structural dynamics solver and a mesh motion algorithm to produce a partitioned FSI solver.

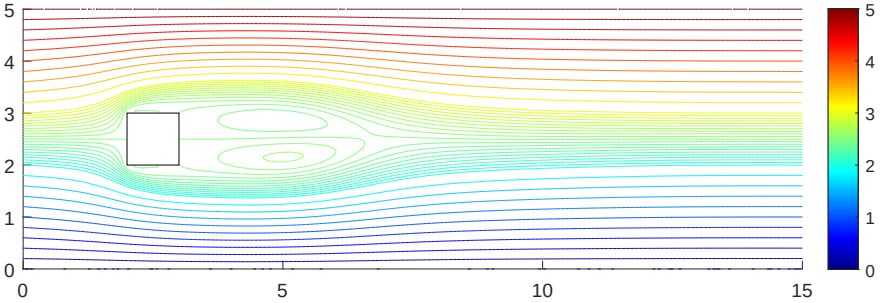
4.4.1 Domain Corners

The handling of sharp corners in the domain is currently somewhat unsatisfactory, both in this work and in the broader literature. The best method at the moment appears to be to soften corners via a spanning normal, i.e. a vector halfway between two adjacent wall normals. At sufficient resolutions, this approach appears to be reasonably robust, at least for the examples presented here, however, in coarse domains, the accuracy of this approximation may become problematic, and even for high fidelity applications, it may not be sufficient. The development of additional methods to handle sharp domain corners would be an important improvement to RBF-FD based methods.

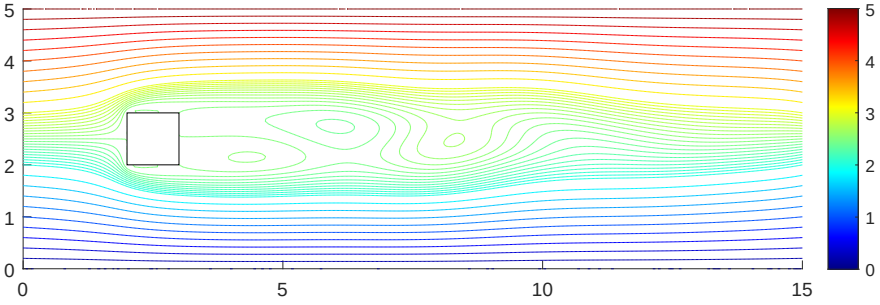
4.4.2 Automated Meshing

Mesh generation and adaptive meshing techniques are currently an active area of study for meshless PDE solvers, including RBF-FD based solvers [102, 103, 104]. Although meshless methods greatly reduce the difficulty of domain generation, it can still be a non-trivial task, even for simple geometry. Integrating these generative and adaptive techniques into the current RBF-FD solver would enable cases to be developed and solved with greater ease.

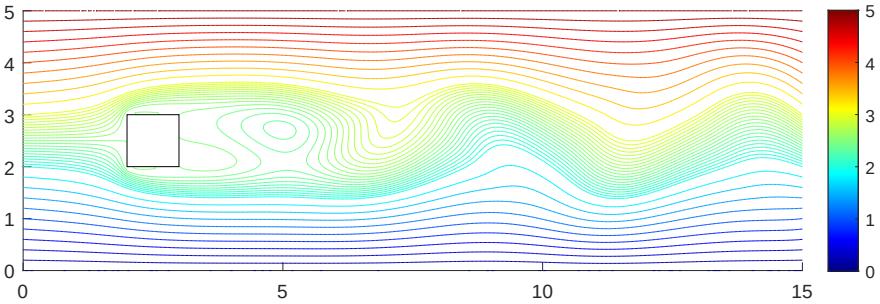
[2] In fact, in many cases, results have been shown to be 'more' accurate on non-Cartesian point clouds, e.g. by clustering nodes at corners, and around areas of complex flow.



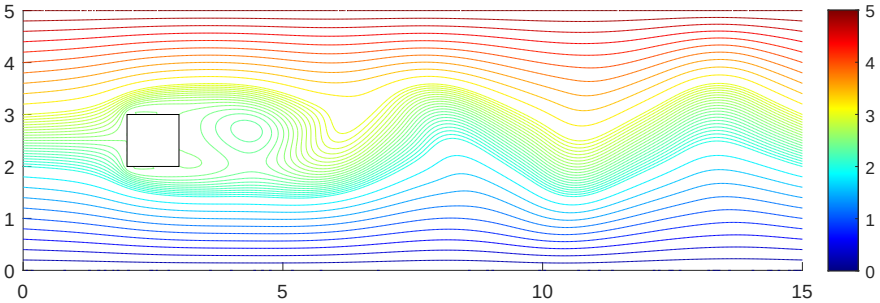
(a) $t = 10s$



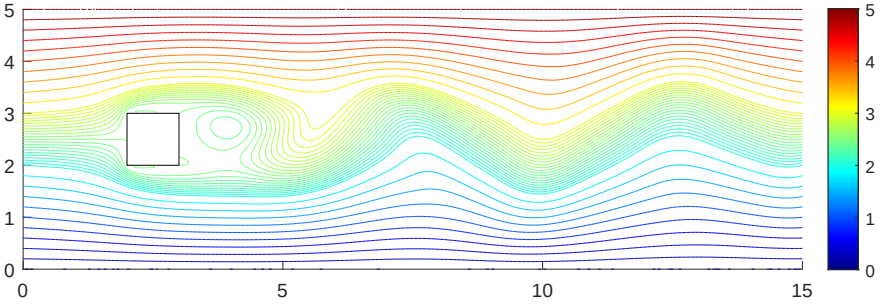
(b) $t = 20s$



(c) $t = 30s$

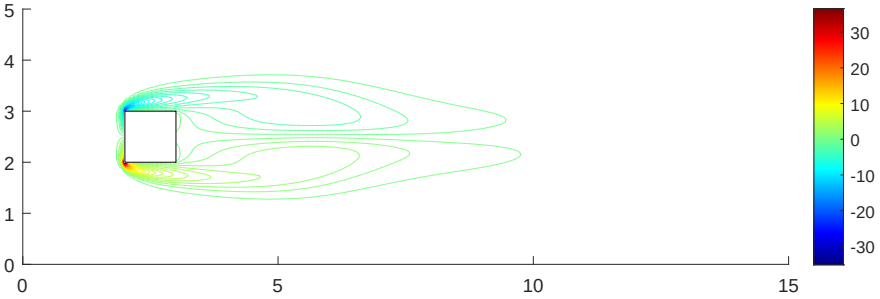


(d) $t = 40s$

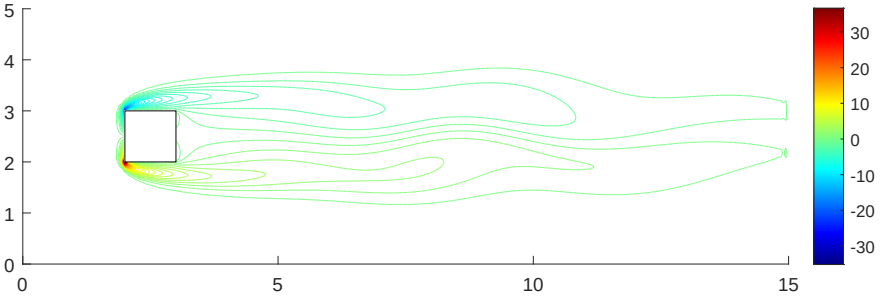


(e) $t = 50s$

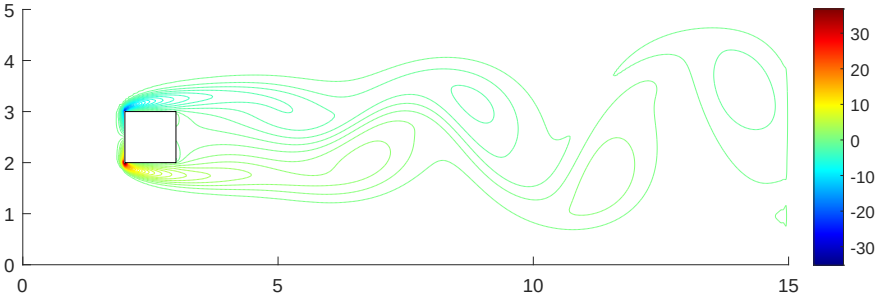
Fig. 4.16: Streamfunction contours of flow past a square cylinder at $Re = 100$, von Kármán vortex street is fully developed by $t = 50s$.



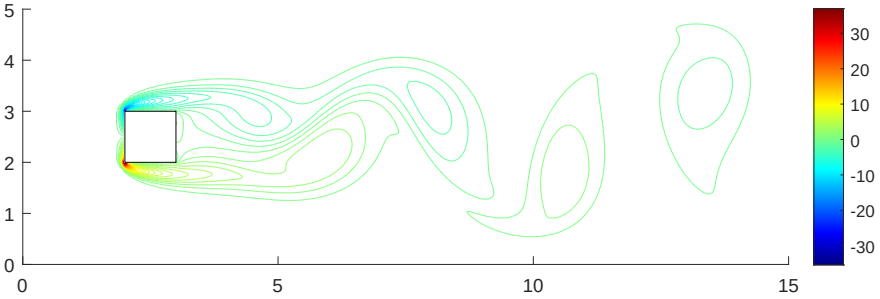
(a) $t = 10s$



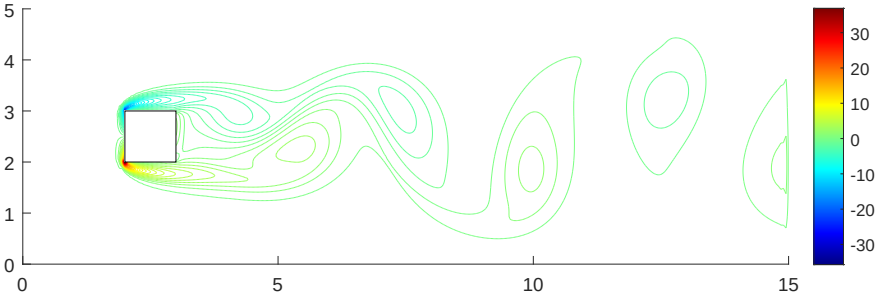
(b) $t = 20s$



(c) $t = 30s$

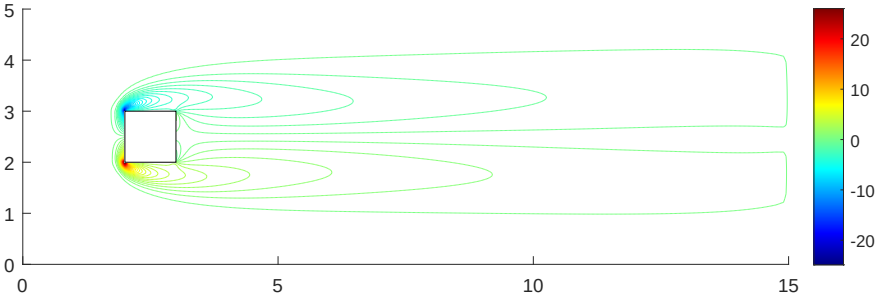


(d) $t = 40s$

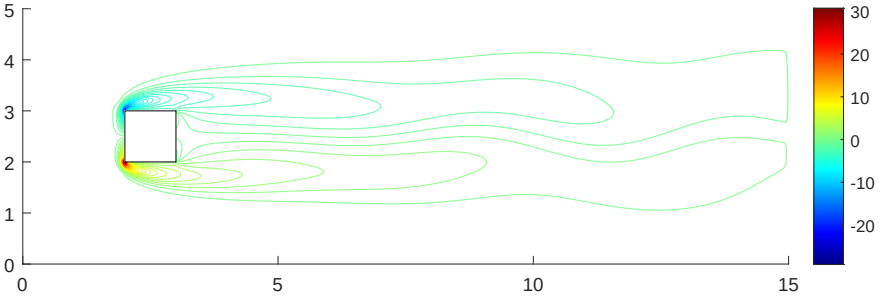


(e) $t = 50s$

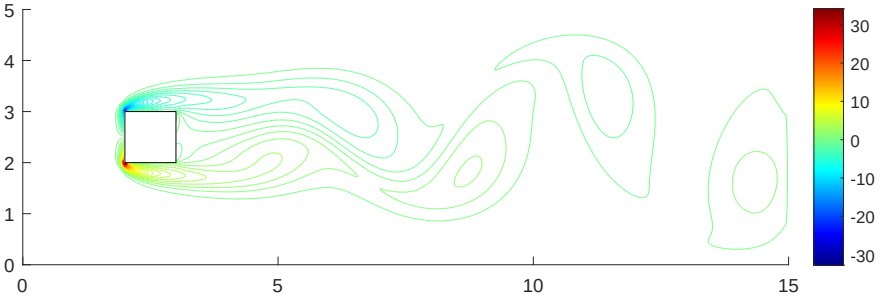
Fig. 4.17: Vorticity contours of flow past a square cylinder at $Re = 100$, von Kármán vortex street is fully developed by $t = 50s$.



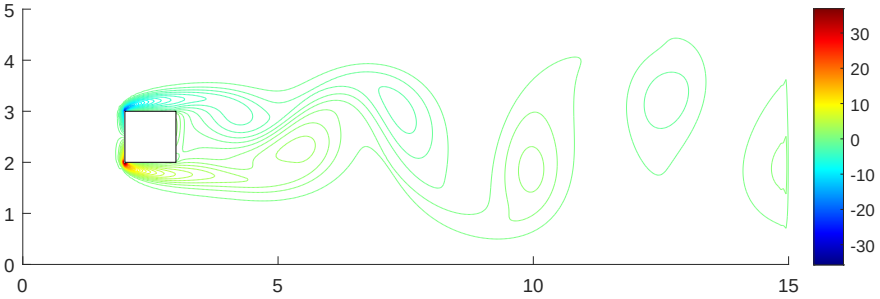
(a) $Re = 40$



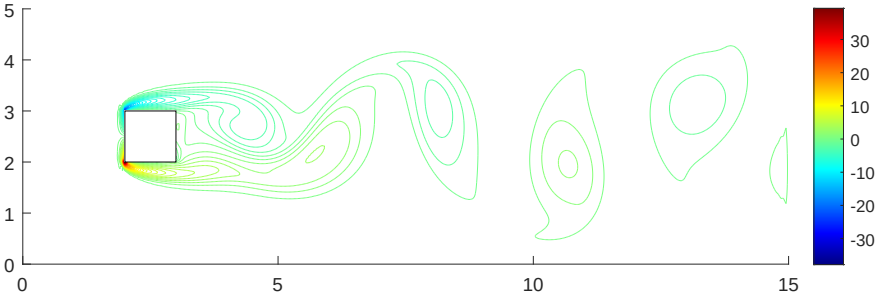
(b) $Re = 60$



(c) $Re = 80$



(d) $Re = 100$



(e) $Re = 120$

Fig. 4.18: Vorticity contours of flow past a square cylinder for varying Reynolds numbers at $t = 50s$ when von Kármán vortex street is fully developed.

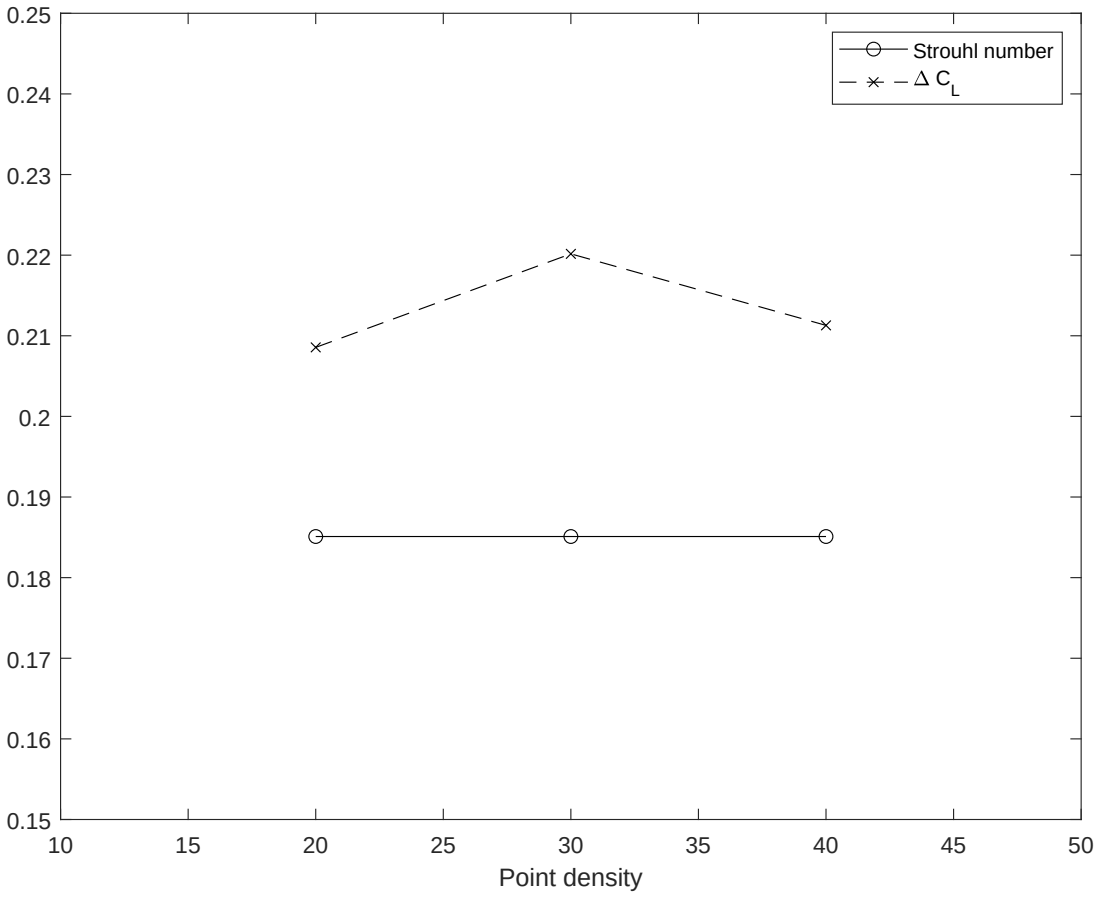


Fig. 4.19: Grid convergence study of Strouhal number and total average amplitude of C_L for $Re = 100$ with respect to point density per unit length.

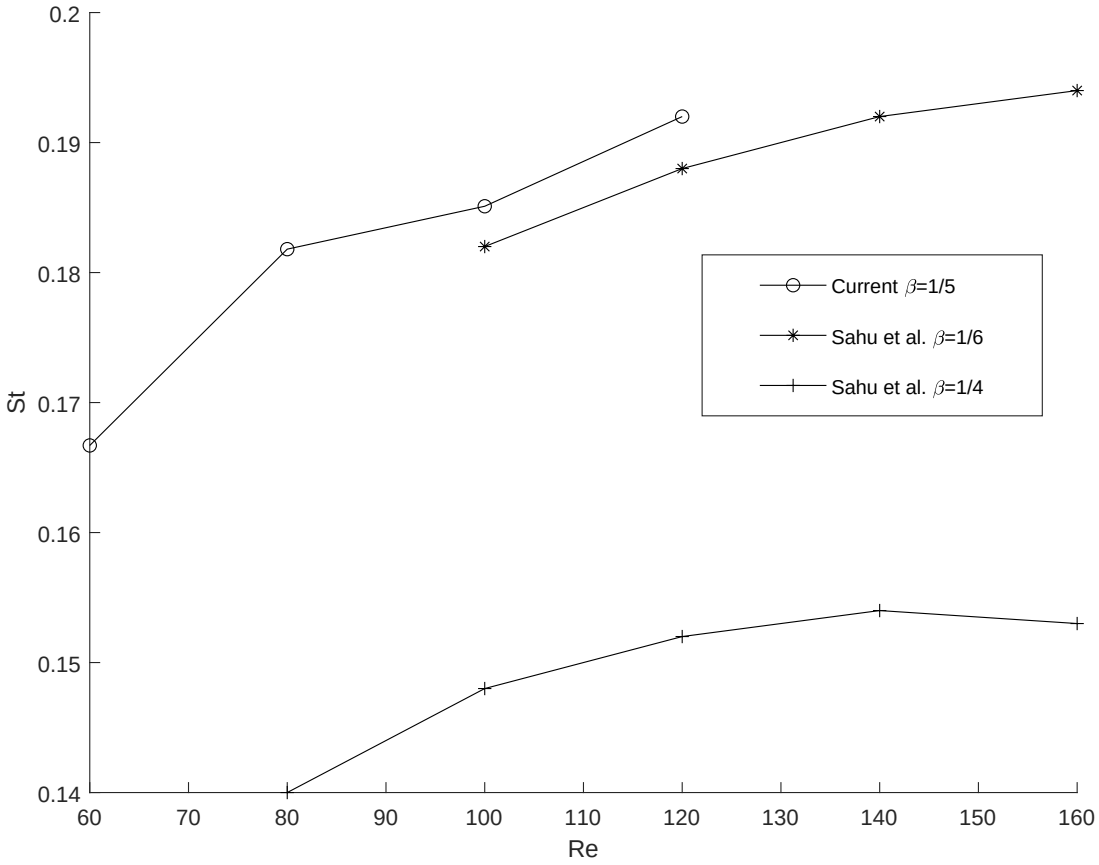


Fig. 4.20: Strouhal number variation with Reynolds number compared to similar simulations from Sahu et al. [101].

5. UNIFIED RBF FLUID FLOW AND FSI

5.1 Introduction

In this chapter we augment the RBF-FD fluid solver of the previous chapter with a mesh motion algorithm and integrate moving-wall boundary conditions to demonstrate a technique for the simulation of fluid-structure interactions. The result is a 2D partitioned, meshless FSI solver. The code for this solver and instructions for its use are available in appendix A. A description and examples of inputs are also provided[1].

We apply this new procedure to the benchmark case of section 4.3.7 in which a square cylinder is exposed to a horizontal flow.

5.1.1 Background

Overall, there appears to have been limited study into coupling the RBF-FD method with structural dynamics and moving boundaries. In particular, no study of the RBF-FD method for a locally deforming mesh is forthcoming in the literature, with locally deforming bodies still a relatively unexplored area more generally in meshless FSI methods [2].

Work by Chinchapatnam et al. [94] demonstrated that the RBF-FD method could be used both for both incompressible flow and structural dynamics independently, but stopped short of coupling the two, opting to use a simplified expression devised by Lam [105] to calculate directly the fluid forces on a near-wall cylinder in a flow.

The concept of a hybrid meshed/meshless scheme for body motion was originally developed by Ang et al. [106]. In such methods, a meshless method is used near moving bodies, while a meshed or Cartesian method is used for the farfield computations, with a region of overlap between them as seen in fig. 5.1. In this way, these methods leverage the power of meshless methods for complex boundary shapes, with the efficiency of meshed methods. RBF-FD was used as

[1]The code is provided in full in the appendix, but is also available online at `https://gitlab.com/ajmurra/rbf-fd-fsi`.

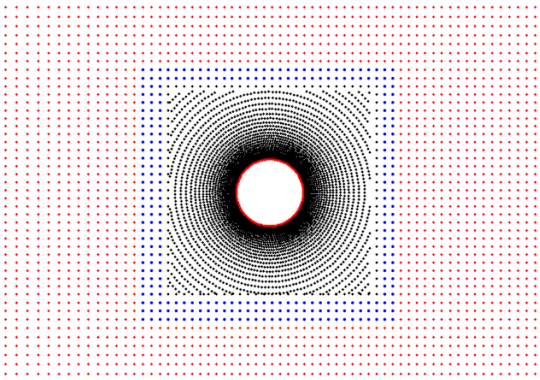


Fig. 5.1: Domain overlap between meshed and meshless solvers in hybrid method of [108]. A traditional meshed solver is used on the farfield (red) points, with RBF-FD employed on the nearfield (black) points around the moving body.

the meshless component by Javed [107]. The moving body along with the RBF-FD nodes are moved as a whole, with the overlapping region between the meshed and meshless methods absorbing the motion. In this way, no mesh deformation is necessary, but motions are limited to rigid body motions, and are required to remain within the near-field envelope. This method has since been successfully applied to vibrational problems for energy harvesters [108, 109].

More generally, although traditionally hindered by high costs, meshfree methods for FSI are gaining popularity. Han [110] coupled a Lattice-Boltzman fluid solver with a discrete element method solver for particle interaction, using an immersed boundary condition to handle the hydrodynamic interactions. The most popular approach to meshless FSI is currently partitioned approached using smoothed particle hydrodynamics (SPH) [2]. The SPH method of fluid flow is discussed in more detail in the introduction to chapter 4. The difficulties of boundary conditions in the SPH methods are still present in the FSI cases, and indeed are somewhat exacerbated by motion. A detailed overview of recent developments in SPH-based FSI are given by Zhang et al. [111].

Given the relative dearth of work on more general meshless FSI, the development of solvers and methods that can handle both rigid and soft body motions while leveraging the advantages of meshfree methods would appear to be highly desirable.

5.2 Formulation

Adapting the RBF-FD solver of chapter 4, we add a structural simulation, a mesh motion algorithm, and a coupling procedure to produce the partitioned solver. Each of these components are described in more detail below.

5.2.1 Structural Solver

In the current formulation, we loosely couple the RBF-FD fluid solver with a simple 2D mass-spring-damper system for simulation the structural dynamics. This allows rigid-body translation within the 2D plane. As the coupling is loose (i.e. partitioned solver), this structural solve can readily be substituted with structural dynamics as necessary for the problem at hand (e.g. see section 5.5.3 for discussion on non-rigid structural dynamics). As the pressure near each boundary point can be calculated from the velocity (via the streamfunction), we can in effect obtain normal force distributions along each boundary, as well as moments about a chosen centre, which can be then input as necessary to structural solvers as desired.

Calculation of Normal Forces at Walls

As the flow is separated from the cylinder, we assume frictional/shear forces are negligible compared to pressure forces, and it remains to compute the normal forces. To determine the normal force at each point along the wall, as in section 4.3.4 when applying formulas to calculate vorticity at the boundary, we make use of the normal information for each point on the moving wall, but this time approximate the velocity at a point a distance h from the wall instead of ψ. As we have already calculated the velocity field (required when iterating the vorticity-transport equation eq. (4.10)), this approximation can again be done relatively cheaply via RBF or other interpolation methods. The calculation of pressure is straightforward since due to the various normalisations applied to the governing equations and geometry, $p \propto v^2$. Once the pressure is known at each boundary point, overall forces and moments can be calculated by integrating (summing, in the discrete case) around the n_b boundary points:

$$F_{\text{total}} \propto \sum_{i=1}^{n} p_i$$

$$M_{\text{total}} \propto \sum_{i=1}^{n} p_i \Delta x$$

where Δx is the perpendicular distance to the rotational centre of the moving object.

Mass-Spring-Damper

As a simple method of producing rigid body dynamics, we can model a cantilever cylinder as a single point mass with dynamics described by uncoupled spring-mass-damper systems describing translation x and y, and rotation θ about a

chosen centre. This simplifies the fluid-to-structure coupling, as the forces and moments can be applied directly to the point mass.

For translation, the standard formulation of the mass-spring-damper begins with summation of forces at the mass node:

$$F_{\text{total}} = -kx - c\dot{x} + F_{\text{external}} = m\ddot{x}.$$

Here k, c, and m are the spring constant, damping factor, and mass respectively. F_{external} is the time varying force from the fluid solver obtained as described in section 4.3.7. Assuming unit mass and rearranging we obtain the usual form

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = F_{\text{external}} \tag{5.1}$$

where $\omega_n = \sqrt{k}$ is the (undamped) natural frequency (not to be confused with vorticity ω used throughout) and $\zeta = \frac{c}{2\omega_n}$ is the damping ratio. Assuming that the dynamics are all uncoupled, applying in both x and y directions gives:

$$\ddot{x} + 2\zeta_x\omega_{n,x}\dot{x} + \omega_{n,x}^2 x = F_x$$
$$\ddot{y} + 2\zeta_y\omega_{n,y}\dot{y} + \omega_{n,y}^2 y = F_y.$$

For rotational motion, the mathematical formulation is the same, with mass replaced by rotational inertia, and force replaced by moment, yielding

$$\ddot{\theta} + 2\zeta_\theta\omega_{n,\theta}\dot{\theta} + \omega_{n,\theta}^2 y = M_\theta.$$

This gives an uncoupled system of 3 equations (ultimately the system is somewhat loosely coupled via the fluid). Since the solver is loosely coupled, the dynamics can be readily modified or replaced as desired.

Discretising the equations using an explicit second order central difference scheme gives

$$\begin{aligned}
x^{t+} &= (1 + 2\Delta t\zeta_x\omega_{n,x})^{-1}\left[\Delta t^2 F_x^t + (2\Delta t\zeta_x\omega_{n,x} - 1)\, x^{t-} + \left(2 - \Delta t^2\omega_{n,x}^2\right) x^t\right] \\
y^{t+} &= (1 + 2\Delta t\zeta_y\omega_{n,y})^{-1}\left[\Delta t^2 F_y^t + (2\Delta t\zeta_y\omega_{n,y} - 1)\, y^{t-} + \left(2 - \Delta t^2\omega_{n,y}^2\right) y^t\right] \\
\theta^{t+} &= (1 + 2\Delta t\zeta_\theta\omega_{n,\theta})^{-1}\left[\Delta t^2 M_\theta^t + (2\Delta t\zeta_\theta\omega_{n,\theta} - 1)\, \theta^{t-} + \left(2 - \Delta t^2\omega_{n,\theta}^2\right) \theta^t\right].
\end{aligned}$$

This allows us to easily step the structural solution in time in line with the fluid solve. As a benchmark, the structural solver was compared against the analytical solution for the case $\omega_n = 0.2$ with $\zeta = 0.03$. The natural frequency was chosen to approximate to the shedding frequency seen in the static case of section 4.3.7. Time stepping $\Delta t = 0.005$ and total time was also chosen to match the coupled simulation. Figure 5.2 shows the results of the benchmark, with a maximum phase error of less than 0.1% ($< \Delta t$). As there is no approximation error of the

external forcing term F_{external}, it is sufficient to observe the homogeneous case of eq. (5.1), as long as all time derivative terms are exercised.

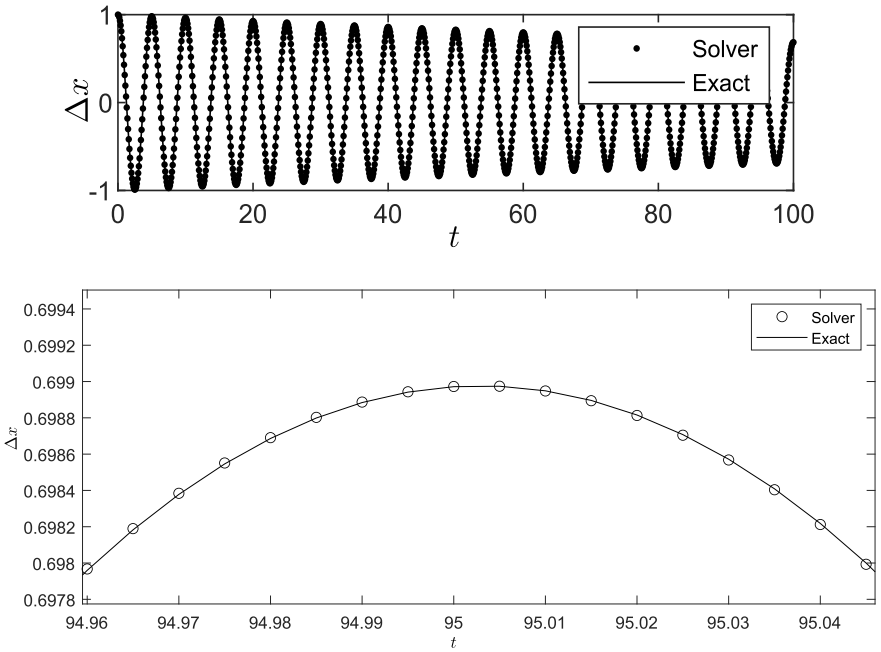


Fig. 5.2: Benchmark for structural solver (top) with detail (bottom) at $t = 95$. Phase error is less than 0.1% at $t = 100$.

5.2.2 *Moving Wall Boundary Condition*

Next we wish to determine appropriate boundary conditions at the moving surfaces. In section 4.3.4, we described the way in which we can apply boundary conditions on both the streamfunction ψ and the vorticity ω in the streamfunction-vorticity formulation of the incompressible Navier Stokes equations. To describe an object moving through the domain, we use both conditions.

A solid object moving through the domain is treated as a combination of wall points with non-zero velocities. From section 4.3.4 we recall that the way in which non-zero velocities at walls contribute to the fluid is twofold - the tangential component of velocities contribute via the vorticity boundary condition (Thom or Jensen formulas, eq. (4.15) and eq. (4.16) respectively), and normal components of velocities contribute via the Dirichlet boundary condition on the streamfunction ψ. The moving walls are comprised of a series of boundary points with their specified normal directions. For a rigid body motion (translation and rotation), these points and normals will stay in the same spatial configuration relative to each other, and any velocity of the object from the structural solver should be applied uniformly to all boundary points. At each point, we decompose this velocity vector into tangential and normal components (dictated by the

individual point normals). This allows us to apply the boundary conditions as described.

Applying the formulas of Thom or Jensen on vorticity using the tangential velocity is straightforward, and the boundary condition on ω is readily determined by direct calculation. Note that this boundary condition has already been applied in this manner on the lid of the (geometrically) static lid driven cavity case described in section 4.3.7, i.e. a wall moving parallel to itself.

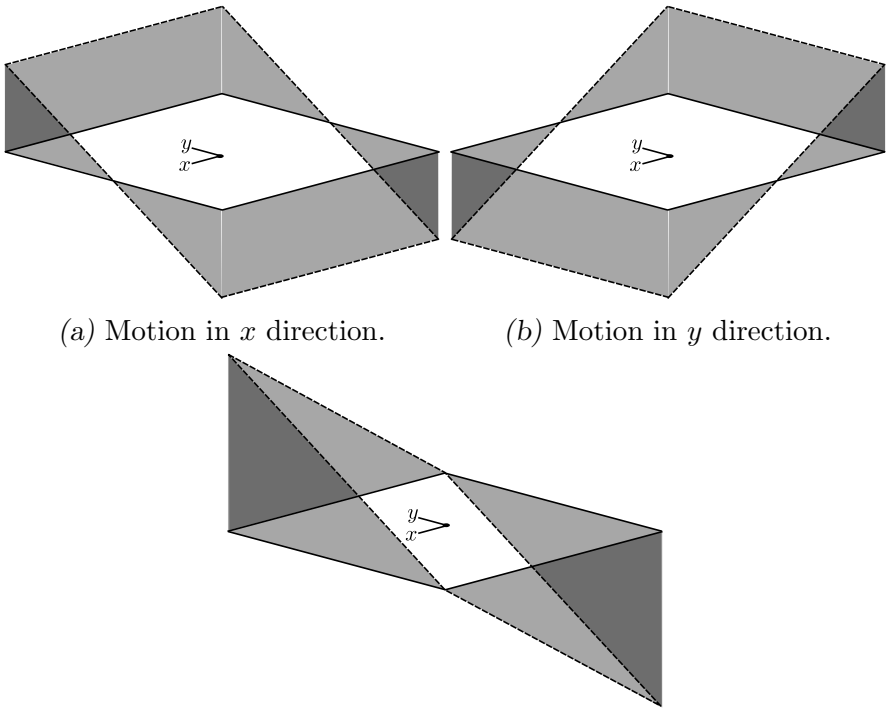
Applying the Dirichlet condition on the streamfunction ψ requires some additional care. From section 4.3.7, recall that applying a non-zero Dirichlet condition on ψ was required at the inlet and outlet of the domain. An inlet or outlet is, in essence, a permeable wall with prescribed flux velocity in the normal direction. In the case of a moving wall however, we wish the wall to remain impermeable, even with a flux velocity across it. In this case, the impermeability is recovered by the motion of the wall itself in the same direction as the flux, i.e. fluid flows across the boundary, but the boundary then moves an equal amount to cancel out this flux. This has the overall effect of an impermeable wall applying a velocity directly to the fluid in the normal direction. In practice, to apply such a condition requires a pair of skew transformations of the boundary in the ψ direction about the x and y axes. As these transformations are linear, it suffices to apply them individually and sum the results, i.e.

$$\psi_b = \psi_{b0} - v_y x + v_x y \tag{5.2}$$

where ψ_{b0} is an initial baseline boundary condition, which, given the impermeability of the moving boundary, is typically a constant. Here we assume that x and y coordinates are centred about a chosen centre of the object, ensuring that the offset in the ψ direction is minimised. Differentiating eq. (5.2) and restricting to the boundary, we can see that we recover the definitions of velocity via the stream function in eq. (4.12) and eq. (4.11). Figure 5.3 shows an example of applying this boundary condition to a moving square.

5.2.3 Motion of Grid Points

To move the points in the domain, we first move the points at the moving boundary manually given the displacements calculated from the structural solver, then interpolate the displacements to the domain using the method described in section 2.2.2. Figure 5.4 shows the process in more detail.



(a) Motion in x direction.

(b) Motion in y direction.

(c) Motion in x and y directions.

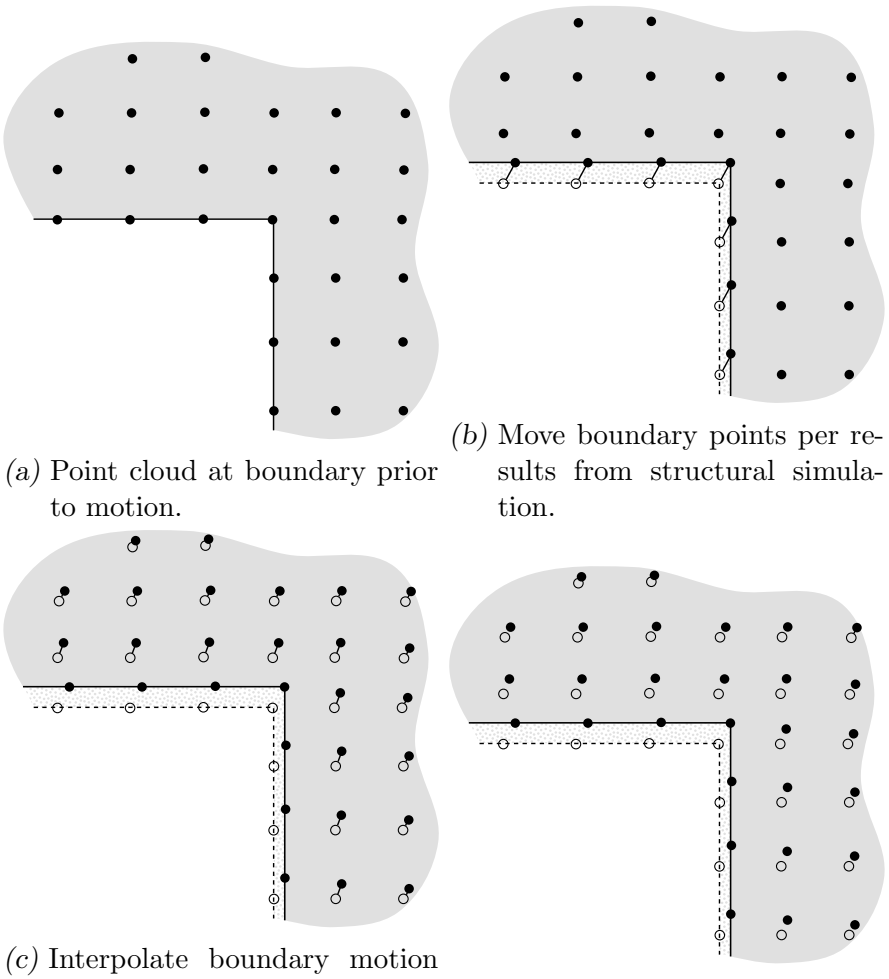
Fig. 5.3: Example of Dirichlet conditions of eq. (5.2) on ψ for a moving square.

5.2.4 *Interpolation of Vorticity*

At each timestep, once the nodes of the domain have been displaced, the stream-function and the vorticity must be interpolated to the new positions. This can be achieved using a number of interpolation schemes. At time t, after displacement of the nodes, i.e. once we are at the point of fig. 5.4d, we must use the latest value of ω to calculate vorticity at the new node positions to be used in the next time step (as the inhomogeneous term in the Poisson equation).

5.2.5 *Recalculation of RBF-FD Weights*

Once moving the points in the domain after the structural solve, the RBF-FD structure is invalidated for any point whose stencil includes a displaced node. As such, we must recalculate the RBF-FD weights at each timestep (see section 4.2). Unfortunately, this is an expensive operation, as it involves the solution of a number of linear systems, with a total cost $\mathcal{O}(Nn^3)$ in the naive case, where N is the total number of points in the domain, and n is the number of points in each stencil. More fortunately, there are a number of factors that aid us in performing these calculations reasonably efficiently:



(a) Point cloud at boundary prior to motion.

(b) Move boundary points per results from structural simulation.

(c) Interpolate boundary motion to domain internals using RBF interpolation matrix H.

(d) Final boundary and point positions (filled circles).

Fig. 5.4: Method of motion.

1. Each stencil can be handled independently, hence the N individual n^3 calculations are perfectly parallelisable.

2. n is not large, and certainly $n << N$, hence the cost (n^3) of solving the linear system for the weights of each individual stencil is small (and can potentially be further reduced by observation of the structural properties of the linear system, e.g. it will often be symmetric).

3. As N increases, it is not required that n also increases, hence for increasing domain size or point density, the cost only scales linearly with N.

4. Only points that are affected by the mesh motion are required to be recalculated, that is, only points that have themselves been moved, or had a

point in their stencil move, are required to be recalculated[2]. In particular, this will usually mean that far field points do not require recalculation.

In addition to these, the RBF-FD method requires no special treatment of the moved nodes, compared to the initialisation calculations made for the original domain, as the weights are a function only of their absolute position. As such, from an implementation perspective, no additional effort is required in this recalculation.

5.2.6 Final Solver Structure

Figure 5.5 shows the final structure of the flow solver with FSI enabled. In practice, the static and FSI solvers are the same code base, with a flag to enable the FSI functionality.

5.3 Simulations

The following section details results for the square cylinder case of section 4.3.7 with motion enabled, as well as a wake-induced vibration test case, similar to that by Bhatt and Alam [112]. In the single square cylinder case, we demonstrate x and y structural motion individually, then show a coupled case. The domain setup for the single square cylinder is shown in fig. 5.6, with the wake-induced vibration case shown in fig. 5.12.

5.3.1 Streamwise Motion

Here we constrain the structural motion to only the x direction, i.e. in the direction of the flow. The structural solver is configured with a natural frequency $\omega_n = 2.236$ and a damping ratio $\zeta = 0.03$. Figure 5.7 shows the response in the x (streamwise) direction of the square. As is expected, the square is forced steadily downstream by the oncoming flow, with the behaviour dominated by non-oscillatory motion towards a resting position approximately 1.5 units from the starting position. An oscillatory mode can be observed in fig. 5.7b, caused by the changing pressure distribution on the back of the square as vortices are

[2] In actual fact, a further distinction is that only points that move or have a point in their stencil move *relative* to the other points in the stencil, are required to be recalculated. However it is unusual that points in the cloud would move completely uniformly with their stencil, hence this is not a particularly helpful observation, as the cost of determining the condition would likely outweigh the savings.

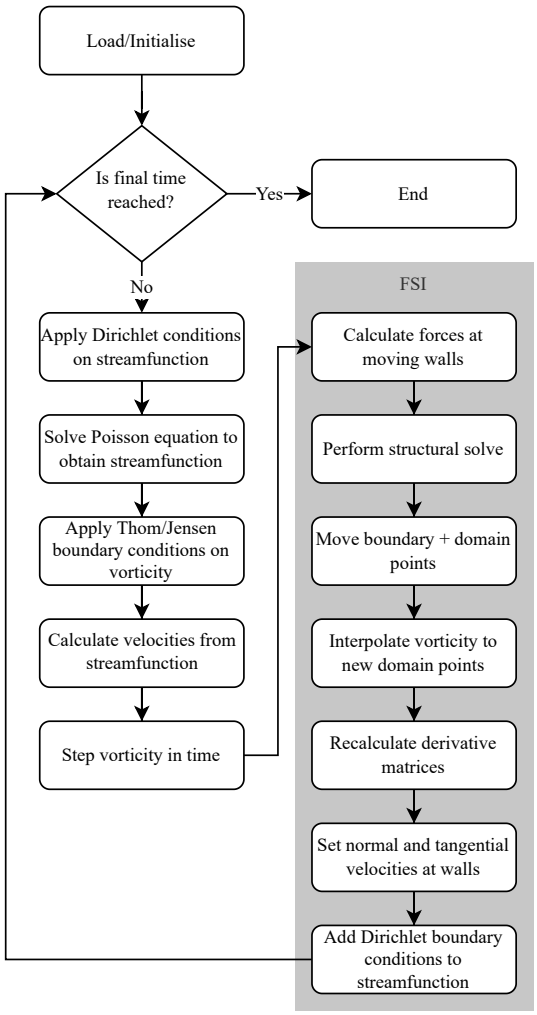


Fig. 5.5: Flow diagram for solver loop with FSI enabled.

shed. We see from the power spectrum in fig. 5.8a that these oscillations correspond exactly to vibrational modes of the Strouhal number 0.186 of the vortex shedding found in section 4.3.7, with most of the power centred around the second mode. This is likely due to two factors. Primarily, the interactions of the drag (pressure) forces with the forces from the shedding vortices. Since the drag force is greater when the square is travelling upstream (in the negative x direction), and is non-linear in the velocity of the square, this causes an asymmetric forcing to interact with existing shedding forces to excite the second mode more than the first. Secondly, there is additional (effective) velocity when the square is travelling upstream, which will affect the shedding frequency. This effect is likely less influential than the first. The lack of y motion prevents any other significant interplay between the moving square on the shedding frequency, and hence although the natural frequency of the structure is significantly different from St, the structure is excited close to (multiples of) the shedding frequency.

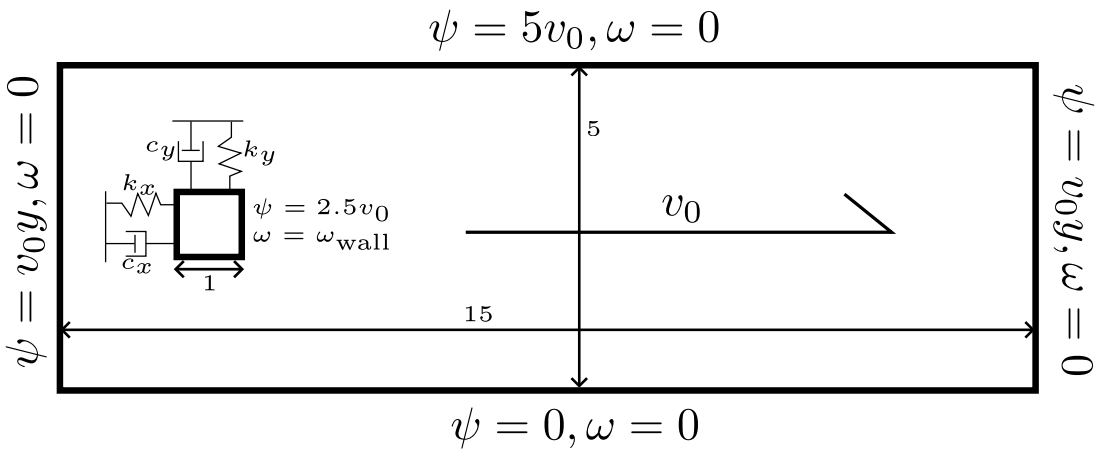
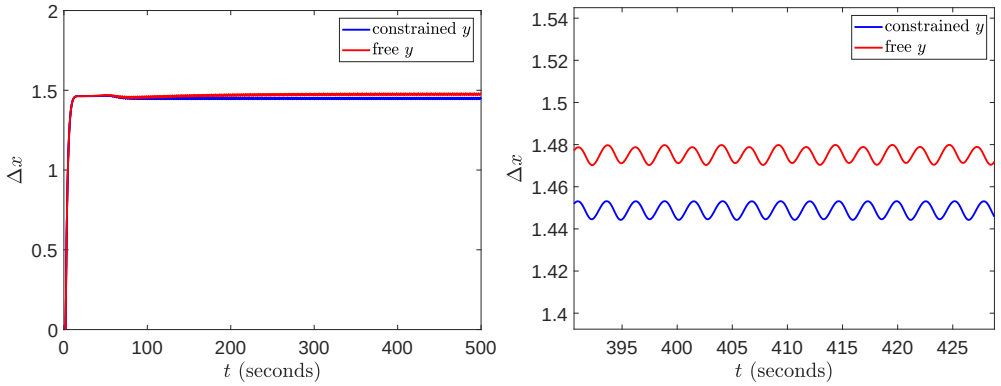


Fig. 5.6: Domain with boundary conditions and spring configuration for FSI case.



(a) x *(b) y*

Fig. 5.7: Response of moving square when constrained to allow only streamwise (x) motion, compared to unconstrained response (with detail).

5.3.2 *Perpendicular Motion*

Here we constrain the structural motion to only the y direction, i.e. perpendicular to the flow. The structural solver is configured with a natural frequency $\omega_n = 0.2$ close, but not equal, to the Strouhal number of the vortex shedding (0.186). The damping ratio is again set to $\zeta = 0.03$. Figure 5.9 shows the expected oscillatory behaviour, as well as a slow, non-oscillatory motion towards a steady position. This non-oscillatory behaviour is likely due to asymmetries in the flow caused by the sudden application of the uniform flow to the system at the beginning of the simulation, as some brief large forces are observed in the first few timesteps. This behaviour could potentially be avoided by running the simulation to a steady state before enabling the FSI, and fading in the structural forces and responses over multiple timesteps. The power spectrum for the oscillatory behaviour is given in fig. 5.8b. This response has a clear peak at 0.19347Hz, slightly higher than the Strouhal number in the static case, which is also reflected in the forces directly from the flow. This indicates that the vertical oscillations

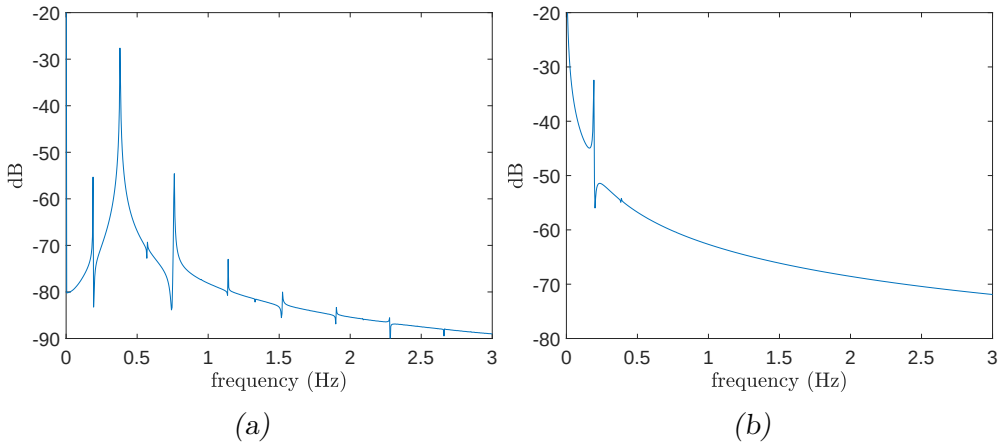


Fig. 5.8: Power spectrums for x and y displacements between $t = 250$ and $t = 500$ in the respective constrained cases. (a) shows power spectrum of x motion with y constrained, while (b) shows power spectrum of y with x constrained.

of the square interact with the vortex shedding regime to bring St towards the natural frequency of the structure.

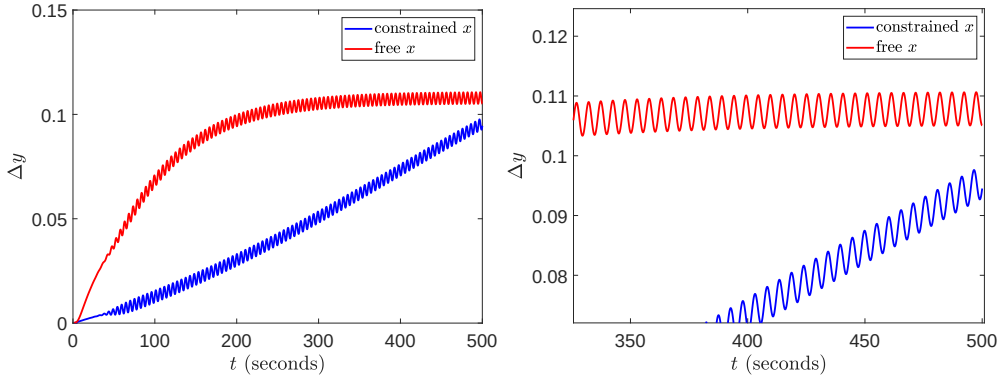
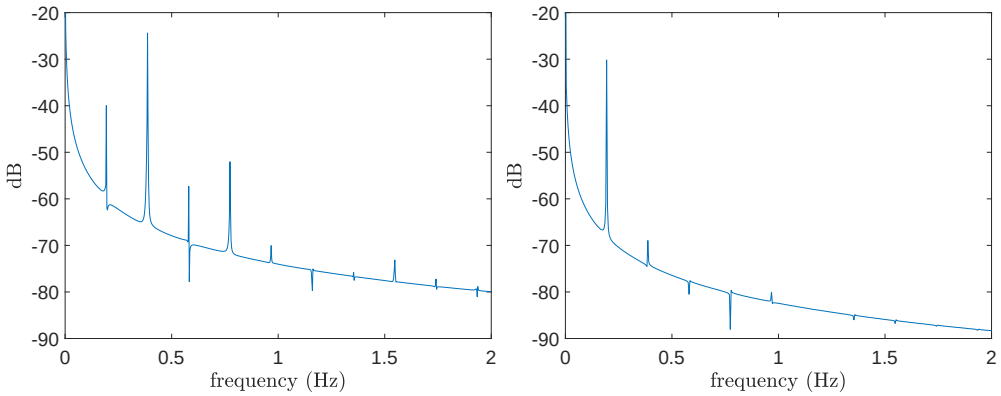


Fig. 5.9: Response of moving square when constrained to allow only perpendicular (y) motion, compared to unconstrained response (with detail).

5.3.3 *Streamwise & Perpendicular Motion*

We now allow both motion in the x and y directions, with structural properties in the respective directions as described in section 5.3.1 and section 5.3.2. Overall spatial responses in the x and y directions were similar to the constrained cases (again the y direction had a non-oscillatory motion), however fig. 5.10 shows a marked difference in the power spectrum. The x response is significantly more concentrated around the modes of the shedding frequency, while the y response now has additional modes that were no present in the constrained case. In both cases, the modes are again at the higher frequency of 0.193 observed in section 5.3.2, compared to the static shedding frequency of 0.186.



(a) x (b) y

Fig. 5.10: Power spectrums for x and y displacements coupled case.

Figure 5.11 shows the nodal positions of the domain after the square has reached its stationary position downstream. This shows clearly the RBF mesh motion using Wendland's compactly supported C2 function ($r = 5$), with the interpolation fixed at all boundaries, including the square. From the starting position ($x = 2.5$), the radius can clearly be seen, as the domain points accumulate at approximately half the radius downstream, corresponding to the steepest point on the basis function.

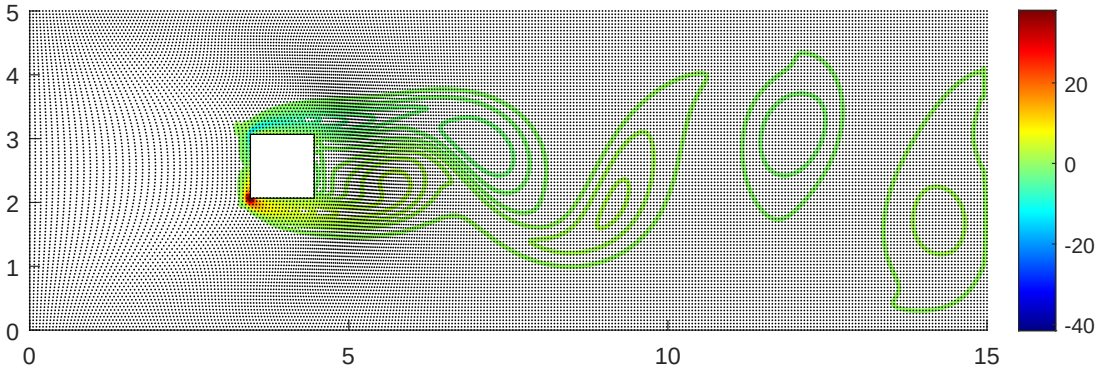


Fig. 5.11: Vorticity contours and mesh displacement for the FSI simulation at $t = 100$ with x and y motion enabled.

5.3.4 Wake-Induced Vibration

Here we describe a wake-induced vibration case, replicating the simulations of Bhatt and Alam [112]. The computational domain is shown in fig. 5.12. Here we have two unit cylinders, with the upstream cylinder fixed, and the downstream cylinder allowed to move in the vertical direction. The inlet, upper, and lower walls are a distance of 15 units from the center of the upstream cylinder, while

the outlet is a distance of 30 units from the same point. The downstream unit cylinder is L units from the center of the upstream cylinder.

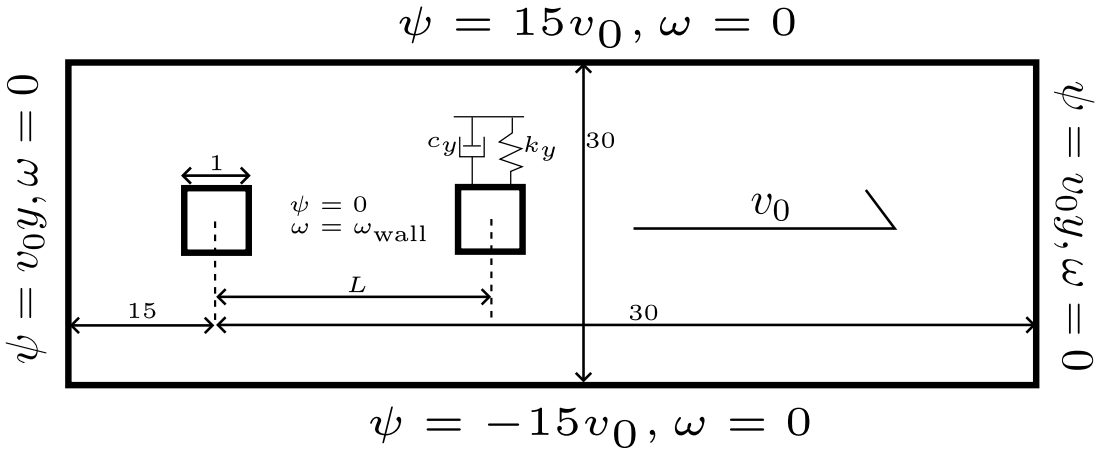
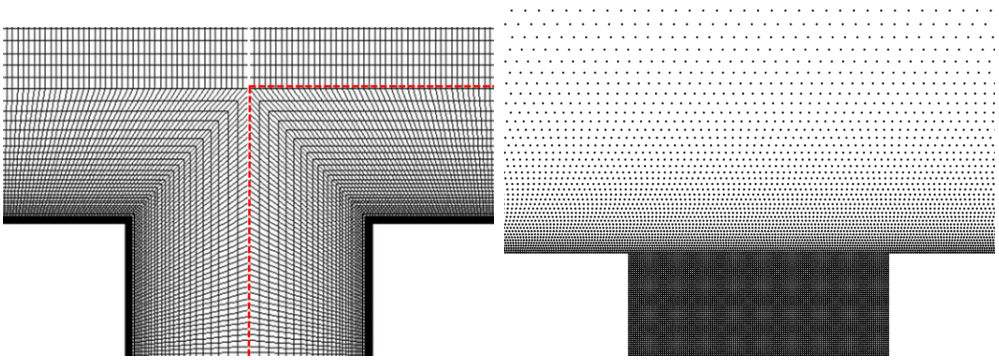


Fig. 5.12: Computational domain for wake-induced vibration case. The upstream cylinder is fixed, while the downstream cylinder is free to move in the y direction as per the structural solver, i.e. a mass-spring-damper model.

In [112], a structured cartesian mesh was used, as necessitated by the solver - Ansys Fluent 15, for unsteady, incompressible flow using the finite-volume method. In their study, a second-order upwind scheme is used for the convective components, with a central-difference scheme used for diffusive terms. The time is stepped using a first-order implicit formulation due to its unconditional stability and compatibility with the dynamic aspects of the solver. A pressure correction step is completed using a SIMPLE (semi-implicit method for pressure linked equations) to couple the velocity and pressure fields. This pressure correction step is similar to the solution of the Poisson equation in the streamfunction-vorticity formulation employed in this work, in that it couples the vorticity and the velocity fields.

One of the immediate advantages of the RBF-FD method is the ability to use a meshless domain to model the problem. Since we now have the freedom to place points in a more uniform manner, it is possible the drastically reduce the point count of the domain in the farfield, while maintaining the appropriate density in the nearfield at the boundaries of the cylinders. In the original work, a density of 125 points per unit length was used, with an expansion rate of 1.05 in the normal direction towards the edges of the domain. This resulted in a domain of 163,568 points for $L = 6$. The limitation of the cartesian mesh means that the high density at the cylinder walls must be carried through all the way to the farfield edge in at least 1 dimension, that is, the expansion rate can only be applied in a single direction, in this case the normal direction. This results not only in significantly more cells that necessary at the domain edges, but cells of a much poorer quality, as they are only expanded in one direction, hence their aspect ratio becomes large.

In the meshless case, we use the same density at the cylinder boundaries, and the same expansion rate, however we can expand in all directions at once, creating not only a much more uniform set of points, but one with fewer points overall, since without need for structured cells, the density at the cylinder surface can be resolved continuously towards the farfield boundary. In addition to this, as there is no adjacency information required in the point cloud, it can be generated with relative ease[3]. Since the farfield required significantly fewer points, additional points can be added between the cylinders to aid in the resolution of vortices. Here we have maintained the boundary density in the entire area between the cylinders as shown in fig. 5.13b. Even with this added density, for $L = 6$ this results in a domain of 149,568, a reduction of approximately 9%. In cases where the distance between the cylinders is reduced, the difference is even more drastic, with an almost 50% reduction when $L = 2$. The difference between the approaches is demonstrated in fig. 5.13.



(a) Structured mesh as used in [112]. *(b)* Current unstructured point cloud.

Fig. 5.13: A comparison of the structured mesh and unstructured point cloud in the area between the cylinders for $L = 2$. Since the density at the cylinder must be carried through to the domain edge in the structured case, significantly more points are required, and high mesh quality is difficult to achieve. In the unstructured case, the expansion can be done in all directions, maintaining a uniform distribution of points, while using fewer points overall. Savings in the far field can be used to increase density between the squares to better resolve complex flow phenomena.

To demonstrate the fidelity of the simulation, we can observe closely the vortex shedding process over a single period of vibration as described by Bhatt and Alam [112]. Here we have set the natural frequency of the wake cylinder to $f_n = 0.125$, which corresponds to the case of reduced velocity $U_r = 8$ in [112].

Figure 5.14 shows the process by which vortices are shed from the wake cylinder. Here we retain the notation from [112] for ease of reference, where they refer to vortices from the upstream cylinder as B, and vortices from the wake cylinder as C. At the start of the cycle, fig. 5.14a, the cylinder is in the bottom position, with

[3]For a handy utility function to help in the generation of such point clouds, see the method `FillTrap` in appendix A.

a high-vorticity layer along the top side of the cylinder. The incoming vortex B_1 from the upstream cylinder is beginning to impinge on the flow around the wake cylinder. As B_1 moves over the top edge of the wake cylinder, the lower pressure from increased velocity causes the vorticity shear layer to separate at the leading edge, and reattach at the trailing edge, resulting in a bubble along the top edge, as can clearly be seen in the detail inset of fig. 5.14b. This bubble causes a large region of low pressure above the cylinder, producing lift to move the cylinder upwards. As this bubble grows however, it collides with the impinging B_1 vortex, causing a weakening of the growth in the bubble, and a corresponding kink in the lift. As B_1 then passes over the trailing edge, it mixes with C_1 to create a binary vortex that is then shed downstream. The process then repeats on the lower edge, as can be seen in figs. 5.14d to 5.14f, with the vortices B_2 and C_2.

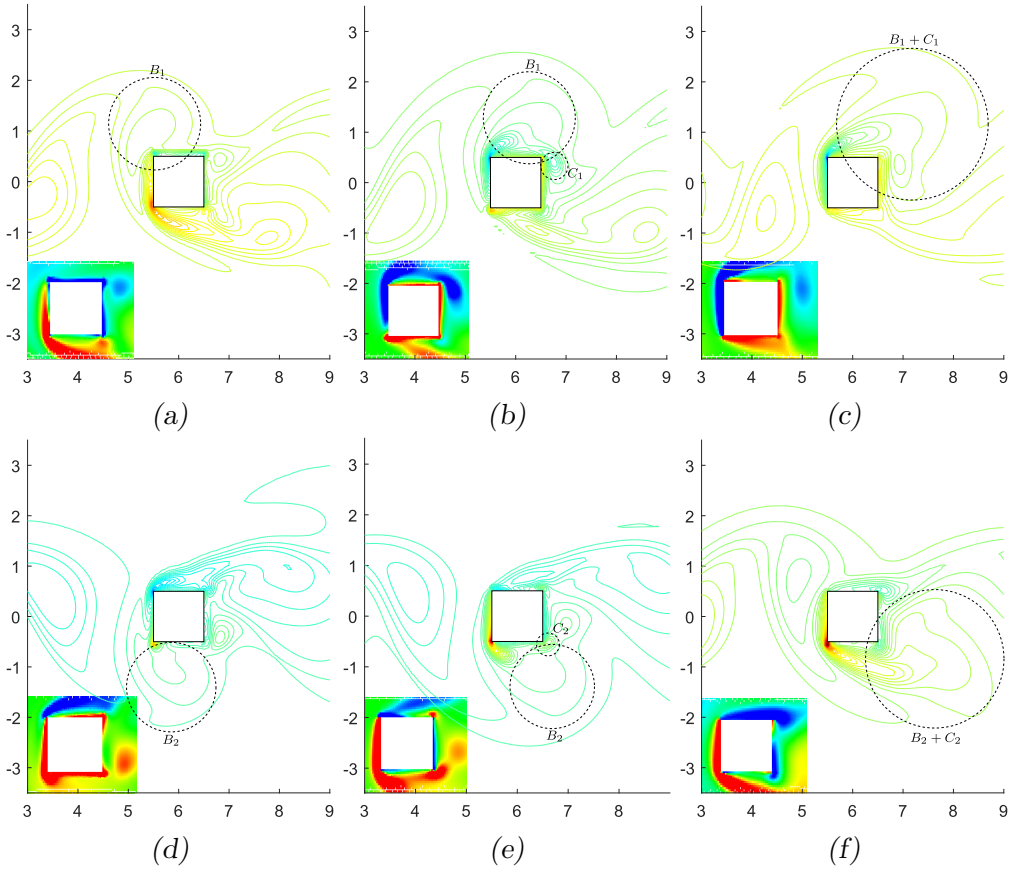


Fig. 5.14: Vortex shedding process from downstream cylinder. Vorticity contours are shown, with inset details of vorticity field close to the cylinder. The top and bottom rows of figures show the vortex shedding from the top and bottom surface of the cylinder respectively, with the subsequent mixing with vortices from the upstream cylinder.

The described process occurs on both the up and downstrokes of the cylinder, the kink in the lift causes the response of the cylinder to be periodic, but non-sinusoidal. Since the lift is almost exactly 180° out of phase with the response, as

it occurring at twice the frequency, the kink occurs during both the downstroke and upstrokes, interrupting the otherwise sinusoidal forcing. This causes the modified response as seen in fig. 5.15.

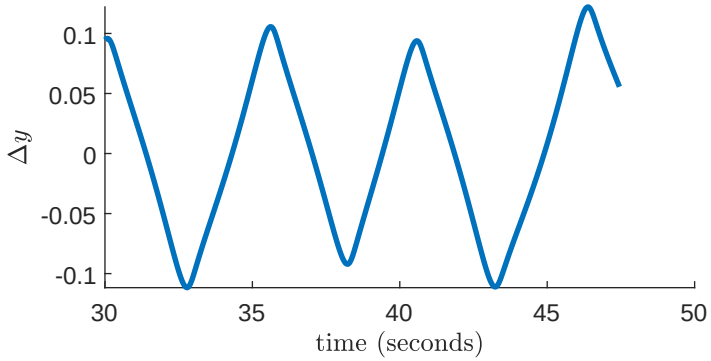


Fig. 5.15: Wake cylinder response demonstrating the effect of the interrupted lift caused by the impinging upstream vortices.

5.4 Scalability of Solver

All simulations in the current work were completed using a combination of the University of Sydney's *Artemis* HPC cluster, as well as a desktop computer. Here we perform some scalability tests on the desktop[4]. This study was run on an 8-core AMD Ryzen 7 5700X CPU @ 3.4GHz with 32 GB RAM. The study was conducted across the three mesh densities used for the validation of the RBF-FD solver in section 4.3.7.

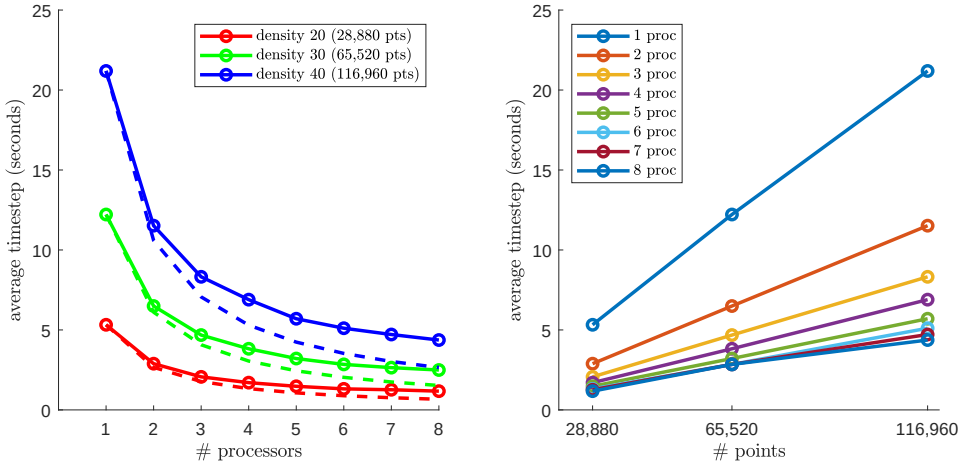
Figure 5.16 shows the results of the tests, varying both the number of processor cores and the domain point density. The increase in speed of a given workload with scaling hardware is given by Amdahl's law:

$$S = \frac{1}{\alpha + \frac{1-\alpha}{P}}, \tag{5.3}$$

where S is the total speedup compared to a single processor, P is the number of processors, and α is the proportion of the workload that is serial, i.e. not parallelisable. In the optimal case, all parts of the workload are perfectly parallelisable, i.e. $\alpha = 0$, and the speedup is equal to the number of processors.

In fig. 5.16, the dashed lines show this theoretical limit for scalability given by Amdahl's law. Solving eq. (5.3) for α and substituting the results from the

[4]Unfortunately, at time of writing, *Artemis* in in a degraded state, and has high variability of performance when under heavy load. This has meant that obtaining consistent timing measurements is difficult, hence we have opted to use the more stable desktop hardware, even though it only extends to 8 cores.



(a) *(b)*

Fig. 5.16: Scaling behaviour of solver for square flow case with varying processor count and domain point density on system with 8-core AMD Ryzen 7 5700X CPU @ 3.4GHz and 32 GB RAM. The dashed lines show the theoretical optimum scaling if all elements of the solver were perfectly parallel. Timesteps presented are taken as the average over 5 timesteps after the MATLAB worker pool has stabilised.

scalability tests of fig. 5.16a, we find across all core counts and domain densities that $\alpha \approx 0.096$, i.e. approximately 90% of the total workload for the FSI solver is parallelisable. This confirms that the expensive recalculation of the RBF-FD weights scales well due to the parallel nature of the calculations, as discussed in section 5.2.5. It is important to note that the optimal (dashed) lines in fig. 5.16 are approaching zero asymptotically in a $1/x$ manner, as predicted by Amdahl's law, rather than reaching a non-zero stagnation point. In the case of the real world solver's results however, these are asymptotically approaching the timestep size of the serial (non-parallelisable) elements of the solver, which is approximately 10% of the original single processor timestep (as $\alpha \approx 0.096$), i.e. approximately 2.5, 1.4, 0.6 seconds for mesh densities of 40, 30, and 20 points per unit length respectively. This accounts for the discrepancy between the optimal and the real world results in fig. 5.16a.

Further to the parallelisable nature of the computations, the linear scaling discussed in section 5.2.5 (i.e. that the cost scales linearly with the domain size N, since the n^3 cost of the stencil inversion is independent of domain size) is also confirmed in the results of fig. 5.16b, with the cost scaling linearly with mesh density for all numbers of processors.

5.5 Conclusions & Future Work

This chapter has presented a novel method of coupling an RBF-FD solver with structural dynamics. Results show that the solver is viable for small scale applications on relatively modest commercial hardware, and demonstrated that the scaling properties are favourable, allowing the method to be extended the method to larger scale applications in the future. The results of section 5.3.4 show that the current meshless FSI method can successfully reproduce detailed and complex flow phenomena as effectively as commercial FSI software (in this case, Ansys Fluent as used in [112]). Further to that, the advantages provided by the reduction in domain size and ease of mesh generation add to the utility of the current solver.

Here we comment on some possible directions for future improvements and applications of the method.

5.5.1 Application to Other Flow Regimes

In the current work, the coupling of the RBF-FD method with a simple but arbitrary structural solver has demonstrated that RBF-FD methods are able to be coupled with any structural solver of choice. As this coupling depends on no particular flow characteristics, and only on the RBF-FD solver itself, the method is applicable to any flow regime able to be modeled by RBF-FD methods. Work such as [113] has shown that RBF-FD methods are applicable to complex regimes with strong discontinuities and shocks, hence the application of the current method to transonic, supersonic, and hypersonic regimes is possible.

5.5.2 Further Optimisation

The current implementation in MATLAB 2022b, while concise, is not necessarily optimal. As an interpreted language, MATLAB overheads can sometimes be significant. Dynamic memory allocation, external library calls, cache misses, etc. are areas which could be improved upon if the solver were to be implemented in a lower level language. Of course MATLAB has many important strengths in relation to development time and the robustness of its routines, strengths that are particularly significant during the development of novel methods within the time constraints of the current work.

From section 5.4, the remaining 10% of the workload could likely be at least partially parallelised, e.g. at the time of writing, MATLAB does not appear to support (CPU-based) parallel matrix multiplication, which would seem to make up a significant proportion of the remaining serial workload. This would further

improve the scaling characteristics of the implementation. In addition to this, many aspects of the solver could be greatly enhanced through the use of GPU processing. Although this would likely only give a proportional performance increase, rather than a scaling improvement, it could still be significant, and would appear to be able to be done with relative ease within the current MATLAB implementation (hardware limitations restricted the use of GPU processing in the current implementation).

5.5.3 Application to Complex Structures

The rigid body translations presented here are merely demonstrative of the solver for more general FSI problems. Since neither the RBF-FD, mesh motion, nor fluid-structure coupling methods rely explicitly on these rigid-body dynamics, the methods are equally applicable to more general body motions, such as those resulting from flexible structures. Coupling the current work a higher fidelity structural solver, e.g. FEA or modal, would allow the study of more complex real world problems.

5.5.4 Unification of RBF Methods

In the current work, the mesh motion, the interpolation of the field to the new points, and the RBF-FD methods are all essentially separate. Since all are based on similar underlying RBF principles, it would seem that it may be possible to unify the calculations to some degree. This would both simplify the implementation, and provide additional performance benefits.

6. CONCLUSIONS

This thesis aimed to explore applications of radial basis functions to various aspects of the simulation of fluid-structure interaction problems. It has detailed two main applications of RBFs.

Firstly, applying RBFs to interpolation problems in mesh motion, and demonstrating a new framework in which to apply existing mesh motion solutions. This framework was found to reduce the costs of some large scale matrix operations required when applying motions to large numbers of domain volume points, on the order of 65% for a large scale simulation of physical experiments. RBF applications in numerical methods have been seen to provide great promise for many decades, with simple and robust formulations, but have often been hampered by high resource costs, with significant work dedicated to reducing computational overheads. The multistage method described in this thesis provides new avenues by which to explore the reduction of computational costs related to RBF interpolation.

Secondly, RBFs were applied in a meshless finite difference technique, known as RBF-FD, to approximate derivatives in order to numerically simulate PDEs. Here we developed a fluid solver, using a novel method to determine vorticity boundary conditions, and coupled it with a structural dynamics solver to produce a partitioned FSI solver, capable of arbitrary body and mesh motions. This appears to be the first RBF-FD based solver of this kind. Through the development and optimisation of this solver and its methods, new capabilities in simulating evermore complex FSI problems will be revealed.

6.1 Future Work

Here we provide a summary of potential directions for future research in the methods presented in this thesis. For more detail, see the conclusions section at the end of each individual chapter. The future work comments contained within the paper of appendix B may also be of interest.

6.1.1 Domain Corners in RBF-FD Method

The handling of sharp corners in the RBF-FD method may be of interest for both particularly coarse, and particularly fine-grained domains. Currently higher fidelity is achieved through increasing the resolution significantly at the corners, at significant cost. However, special handling methods may be able to improve the solution around corners without associated increases in computational costs.

6.1.2 Automated Meshing

One of the great benefits of meshless methods is that no connectivity information is required within the computational domain, hence points can be distributed in a much more uniform fashion, with smooth expansion factors. This avoids many of the issues of mesh generation, particularly in structured cases where detailed near-field meshes must be resolved into simpler far field grids. Automated meshing and mesh refinement techniques are a large field of study on their own, however the integration of existing techniques into the FSI solver developed in this thesis would be an excellent step in reducing case/domain development overheads. The method `FillTrap` included in appendix A is a utility function to generate a smoothly varying set of points within a symmetric trapezoid using a given expansion rate. Such a method could provide the basis for a simple meshless domain generation tool, and has been used with great success in the current work.

6.1.3 Optimisations to RBF-FD FSI Solver

The current implementation of the FSI solver (MATLAB 2022b) is likely suboptimal, as discussed in section 5.5.2. An implementation in a language more suited to high performance compute hardware would be desirable, e.g. FORTRAN, C++, etc. Further gains could be made by leveraging GPU hardware, particularly for some of the linear solves of the Poisson equations, which currently make up a significant proportion of the serial processing within the code. Care must be taken however, as data transfers between GPU and host architectures can incur significant overheads, eliminating any potential gains.

6.1.4 Application to Complex Structures

Since neither the mesh motion scheme nor handling of boundary conditions in the FSI solver presented in this thesis relies on motion of the body being rigid, replacing the current simple structural dynamics with a more comprehensive solver would allow for the simulation of more complex FSI problems.

More complex boundaries are also able to be handled with the RBF-FD method, though there are some additional practical considerations with domain and stencil generation. One of the primary concerns would be the stability of the stencil matrices for low-density point clouds around complex boundaries, where care would needed to ensure that both the stencils were picking points appropriately (e.g. not picking points across structural domains), as well as resolving the boundary appropriately within the stencil. Variable size stencils may be necessary for these cases, where greatly varying point cloud density would often be required to resolve complex boundary features.

6.1.5 Unification of RBF Methods

The FSI solver here makes use of a number of RBF-based methods. As such, exploring ways in which the underlying RBF structures could be exploited to unify and simplify various aspects of the solver would be another excellent way to progress towards the ability to simulate increasingly complex FSI problems.

6.2 Closing Remarks

This thesis provides the latest increment in a long line of methods, formulations, optimisations, and innovations for meshless numerical methods over the last century. It is hoped that the developments here are a small but worthy addition to the work of the countless individuals that have come before it.

APPENDIX

A. RBF-FD FSI SOLVER

A.1 User Guide

The source code for the RBF-FD FSI solver is provided in appendix A.3, or is also available at `https://gitlab.com/ajmurra/rbf-fd-fsi`.

In the directory containing the source code, create a directory *./cases/* alongside the .m files provided in the source code below. In the *./cases/* directory, create another directory with the case name to be run, e.g. *./cases/case123/*. Inside the case directory, place three files - *sets.txt*, *dom.txt*, and *bcs.txt*.

An example of *sets.txt* is shown in appendix A.2. It contains the various parameters for the simulation itself.

The *dom.txt* file contains information about the nodes in the domain, with each line having the format

`<index> <x position> <y position>`

where the index is an integer, and the x and y positions are floating point numbers.

The *bcs.txt* file contains information about the boundary and initial conditions of the case, with each line having the format

`<index> <type> <normx> <normy> <velx> <vely> <psi> <veln> <velt>`,

with each of the tags described below.

`<index>` - a reference to the point in *dom.txt*

`<type>` - the type of boundary, 1 = wall, 2 = inlet, 3 = free, 4 = moving

`<normx>,<normy>` - direction of the normal vector (normalised on load)

`<velx>,<vely>` - x and y velocities

`<psi>` - fixed streamfunction value

`<veln>,<velt>` - normal and tangential velocities

The velocity terms are legacy, and are now overwritten by the streamfunction value `<psi>`, which is used to determine fixed velocities at boundaries. Hence these can generally be set to 0.

Once all the above files have been created and placed in the correct directories, the simulation can be run using the command

```
MeshlessSolver('case123')
```

The output data will be written to *./output/case123/<timestamp>*.

A.2 Example Input

Below is the settings file for the square flow case of chapter 5. The domain and boundary conditions can be generated using the script provided.

../meshless–fsi/cases/fsi_square20Re100/sets.txt

```
#restartFile output/fsi_square20Re100/22-11-29_1709.23/restart_tstep15000.mat # name of restart file, comment out to start new sim
fsi         1       # 0 = no fsi, 1 = fsi
fsi_start   2       # time at which to start fsi
ord_phs     3       # order of polyharmonic spline to use
ord_pol     1       # order of polynomial terms
ord_hyp     0       # order of hyperviscosity term
sten        30      # stencil size
T           100     # the final simulation time
tout        0.05    # data output interval
dt          0.005   # time delta
h           0.05    # delta from wall for vorticity approximation at boundary
hv          0.05    # delta from wall for velocity approximation for pressure force calculation
r           5.0     # rbf radius for point motion
Re          100     # Reynolds number

# coordinates for centroid of moving points
cenx        2.5
ceny        2.5

# spring properties for structural solver
fsi_x       1
freq_x      2.236
damp_x      0.030
fsi_y       1
freq_y      0.200
damp_y      0.030
```

../meshless–fsi/gen_square.m

```
% generate a square in a flow

close all
clear all

% global settings
density = 20; % density - approx number of points along a unit length
fpref = 'SQUARE'; % filename prefix
vel = 1; % freestream velocity

% generate points
% left of square
xv1 = linspace(0,2,round(density*2));
yv1 = linspace(2,3,density);
[x1,y1] = meshgrid(xv1,yv1);
x1 = x1(:);
y1 = y1(:);
%right of square
xv2 = linspace(3,15,round(density*12));
yv2 = linspace(2,3,density);
[x2,y2] = meshgrid(xv2,yv2);
x2 = x2(:);
y2 = y2(:);
%above square
```

```
xv3 = linspace(2,3,density);
yv3 = linspace(3,5,round(density*2));
[x3,y3] = meshgrid(xv3,yv3);
x3 = x3(:);
y3 = y3(:);
%below square
xv4 = linspace(2,3,density);
yv4 = linspace(0,2,round(density*2));
[x4,y4] = meshgrid(xv4,yv4);
x4 = x4(:);
y4 = y4(:);
%top left
xv5 = linspace(0,2,round(density*2));
yv5 = linspace(3,5,round(density*2));
[x5,y5] = meshgrid(xv5,yv5);
x5 = x5(:);
y5 = y5(:);
%top right
xv6 = linspace(3,15,round(density*12));
yv6 = linspace(3,5,round(density*2));
[x6,y6] = meshgrid(xv6,yv6);
x6 = x6(:);
y6 = y6(:);
%bottom left
xv7 = linspace(0,2,round(density*2));
yv7 = linspace(0,2,round(density*2));
[x7,y7] = meshgrid(xv7,yv7);
x7 = x7(:);
y7 = y7(:);
%bottom right
xv8 = linspace(3,15,round(density*12));
yv8 = linspace(0,2,round(density*2));
[x8,y8] = meshgrid(xv8,yv8);
x8 = x8(:);
y8 = y8(:);

x = [x1;x2;x3;x4;x5;x6;x7;x8];
y = [y1;y2;y3;y4;y5;y6;y7;y8];

% remove duplicates
xy = unique([x,y],'rows');
x = xy(:,1);
y = xy(:,2);



pts = [];
for i = 1:length(x)
    pt = [i,x(i),y(i)];
    pts = [pts;pt];
end

fid = fopen(strcat(fpref,'_dom.txt'),'w');
fprintf(fid,'%i %f %f\n',pts');
fclose(fid);

% psi at square
ps = 2.5;

% do boundary conditions
bcs = [];
for i = 1:length(x)

    % square
    if (pts(i,2) == 2 && pts(i,3) == 3) % top left
        bcs = [bcs; i,1,-1,1,0,0,ps];
    elseif (pts(i,2) == 3 && pts(i,3) == 3) % top right
        bcs = [bcs; i,1,1,1,0,0,ps];
    elseif (pts(i,2) == 2 && pts(i,3) == 2) % bottom left
        bcs = [bcs; i,1,-1,-1,0,0,ps];
    elseif (pts(i,2) == 3 && pts(i,3) == 2) % bottom right
        bcs = [bcs; i,1,1,-1,0,0,ps];
    elseif (pts(i,3) == 3 && pts(i,2) > 2 && pts(i,2) < 3) % top
        bcs = [bcs; i,1,0,1,0,0,ps];
    elseif (pts(i,3) == 2 && pts(i,2) > 2 && pts(i,2) < 3) % bottom
        bcs = [bcs; i,1,0,-1,0,0,ps];
    elseif (pts(i,2) == 2 && pts(i,3) > 2 && pts(i,3) < 3) % left
        bcs = [bcs; i,1,-1,0,0,0,ps];
    elseif (pts(i,2) == 3 && pts(i,3) > 2 && pts(i,3) < 3) % right
        bcs = [bcs; i,1,1,0,0,0,ps];

    % domain
    elseif (pts(i,2) == 0) % inlet
        bcs = [bcs; i,3,1,0,0,0,pts(i,3)];
    elseif (pts(i,2) == 15) % outlet
        bcs = [bcs; i,3,-1,0,0,0,pts(i,3)];
    elseif (pts(i,3) == 5) % top
        bcs = [bcs; i,3,0,-1,0,0,5];
    elseif (pts(i,3) == 0) % bottom
        bcs = [bcs; i,3,0,1,0,0,0];
    end


```

```
end

fid = fopen(strcat(fpref,'_bcs.txt'),'w');
fprintf(fid,'%i %i %f %f %f %f %f\n',bcs');
fclose(fid);


figure
hold on
plot(x,y,'k.');
quiver(x(bcs(:,1)),y(bcs(:,1)),bcs(:,3),bcs(:,4));
axis equal

figure
plot3(pts(bcs(:,1),2),pts(bcs(:,1),3),bcs(:,7),'k.');
axis equal
```

A.3 Source Code

../rbf–fd–fsi/MeshlessSolver.m

```
function MeshlessSolver(caseName)


    fprintf("-----------------\nRBF-FD FSI SOLVER\n-----------------\n")
    % load settings
    %
    % we load up the text file using text scan
    % since this allows comments on lines with variable definitions,
    % but then we have to run through each variable and decide
    % if it's a number or not
    fid = fopen(sprintf("cases/%s/sets.txt",caseName),'r');
    set = textscan(fid,'%s %s','CommentStyle','#');
    fclose(fid);
    set = cell2struct(set{:,2},set{:,1},1);
    fn = fieldnames(set);
    for i=1:numel(fn)
        if all(ismember(set.(fn{i}),'0123456789+-.eEdD'))
            set.(fn{i}) = str2double(set.(fn{i}));
        end
    end

    fprintf("SETTINGS\n")
    disp(set);

    fprintf("--------------\nINITIALISATION\n--------------\n")

    % start the parallel pool
    tic
    gcp;
    fprintf("%fs\n",toc)

    % if this is a restart, just load the file
    restart = 0;
    if isfield(set,'restartFile')
        % it may be necessary to change the output directory if we've
        % changed the directory name since the simulation ran
        [outDirTmp,~,~] = fileparts(set.restartFile);
        restart = 1;
        % save a copy of the settings file to compare to what is in the
        % restart file, in case settings have changed and we need to
        % recalculate things
        tmpSet = set;

        % load in the restart file
        fprintf("Loading restart file %s... ",set.restartFile)
        tic
        load(set.restartFile)
        fprintf("%fs\n",toc)
        outDir = strcat(outDirTmp,'/');
        clear outDirTmp
    else
        % create the directory to output files to
        timestamp = datestr(datetime('now'),'yy-mm-dd_HHMM.SS');
        outDir = strcat(sprintf("output/%s/%s/",caseName,timestamp));
        mkdir(outDir);

        % load domain
        %
        % similar trick to the settings file, but no need to check
        % if things are numbers, since the data is more rigid, can convert
        % straight to an array
        fid = fopen(sprintf("cases/%s/dom.txt",caseName),'r');
        dom = cell2mat(textscan(fid,'%f %f %f','CommentStyle','#'));
```

```
64          fclose(fid);
65
66          % load boundary conditions
67          %
68          % similar to domain, but some extra columns
69
70          % define where to find the various things inside the bcs array
71          bciID = 1;
72          bciIND = 1;
73          bciTYPE = 2;
74          bciNORMx = 3;
75          bciNORMy = 4;
76          bciNORMxy = 3:4;
77          bciVELx = 5;
78          bciVELy = 6;
79          bciVELxy = 5:6;
80          bciPSI = 7;
81          bciVELn = 8;
82          bciVELt = 9;
83
84          % define some globals to enumerate the condition types
85          bcWALL = 1;
86          bcINLET   = 2;
87          bcFREE = 3;
88          bcMOVING = 4;
89
90          fid = fopen(sprintf("cases/%s/bcs.txt",caseName),'r');
91          bcs = cell2mat(textscan(fid, '%f %f %f %f %f %f %f','CommentStyle','#'));
92          fclose(fid);
93
94          % normalise normals
95          bcs(:,bciNORMxy) = bcs(:,bciNORMxy)./sqrt(sum(bcs(:,bciNORMxy).^2,2));
96
97          % calculate normal and tangential velocity components
98          tangents = [bcs(:,bciNORMy),-bcs(:,bciNORMx)];
99          bcs = [bcs, dot(bcs(:,bciVELxy),bcs(:,bciNORMxy),2),dot(bcs(:,bciVELxy),tangents,2)];
100
101         % now that we're done loading everything and matching boundary
102         % conditions to points, replace point ids with indices in the boundary
103         % conditions and delete ids from domain, since indexed access is faster
104         inds = [];
105         for i = 1:length(bcs)
106             ind = find(dom(:,1)==bcs(i,bciID));
107             inds = [inds;ind];
108         end
109         bcs(:,bciID) = inds;
110         dom = dom(:,2:3);
111     end
112
113     % display some warnings if necessary
114     if (mod(set.tout,set.dt) ~= 0)
115         warning("Data output delta is not divisible by time delta, output may be irregular.")
116     end
117
118     if (restart && tstep*tmpSet.dt == tmpSet.T)
119         warning("Restart time equals final time, no simulation will be run.")
120     end
121
122     if (set.ord_phs < 2*set.ord_hyp+1)
123         warning("Should have ord_phs >= 2*ord_hyp+1.")
124     end
125
126
127     % run the simulation
128
129     % if this isn't a restart, do all the initial setup, pulling out indices etc.
130     if ~restart
131
132         % get the total number of points in the domain
133         N = length(dom);
134
135         % get indices and number of boundary points
136         bpts = bcs(:,bciIND);
137
138         % indices of the individual bcs within the bcs array
139         bcs_w = find(bcs(:,bciTYPE)==bcWALL);
140         bcs_i = find(bcs(:,bciTYPE)==bcINLET);
141         bcs_f = find(bcs(:,bciTYPE)==bcFREE);
142         bcs_m = find(bcs(:,bciTYPE)==bcMOVING);
143
144         % indices of the individual bcs within the full domain
145         bpts_w = bcs(bcs_w,bciIND);
146         bpts_i = bcs(bcs_i,bciIND);
147         bpts_f = bcs(bcs_f,bciIND);
148         bpts_m = bcs(bcs_m,bciIND);
149
150         % find indices of all volume points
151         pts_vol = 1:size(dom,1);
152         pts_vol([bpts_w;bpts_i;bpts_f;bpts_m]) = [];
153
154         % store the dirichlet bcs for psi so we don't have to pull it out every
155         % loop
156         psi_w = bcs(bcs_w,bciPSI);
```

```
157         psi_i = bcs(bcs_i,bciPSI);
158         psi_f = bcs(bcs_f,bciPSI);
159         psi_m0 = bcs(bcs_m,bciPSI);
160         psi_m1 = zeros(size(psi_m0));
161
162         if (length(bpts_w)+length(bpts_i)+length(bpts_f)+length(bpts_m) ~= length(bpts))
163             warning("individual boundaries do to not sum to total");
164         end
165     end
166
167     % if new sim, or the stencil size has changed
168     if ~restart || (tmpSet.sten ~= set.sten)
169         % calculate stencils
170         if restart
171             set.sten = tmpSet.sten;
172         end
173         stencils = GetNearest(dom,set.sten);
174     end
175
176     % if new sim, or any of the RBF-FD settings have changed
177     if ~restart || (tmpSet.ord_phs ~= set.ord_phs) ...
178                 || (tmpSet.ord_pol ~= set.ord_pol) ...
179                 || (tmpSet.ord_hyp ~= set.ord_hyp)
180         % calculate derivative matrices / RBF-FD weights
181         fprintf("Calculating RBF-FD weights... ")
182         tic
183         CalcDerivMatrix;
184         fprintf("%fs\n",toc)
185     end
186
187     % if new sim with fsi, or restart where we have switched on fsi
188     if (~restart && set.fsi == 1) || (restart && set.fsi == 0 && tmpSet.fsi == 1)
189         % take a copy of domain at time 0 for keeping track of motion
190         dom0 = dom;
191
192         % setup motion matrix H
193
194         % stack the boundary points in order
195         bpts_ordered = [bpts_m;bpts_w;bpts_f;bpts_i];
196         nCtl = length(bpts);
197         nVol = size(dom,1) - length(bpts);
198         Css = zeros(nCtl);
199         Csv = zeros(nVol,nCtl);
200
201         % generate Css matrix
202         fprintf("Generating Css matrix for point motion... ")
203         tic
204         for i = 1:nCtl
205             for j = 1:nCtl
206                 indi = bpts_ordered(i);
207                 indj = bpts_ordered(j);
208                 Css(i,j) = RBF('C2',set.r,dom(indi,:),dom(indj,:));
209             end
210         end
211         fprintf("%fs\n",toc)
212
213         fprintf("Generating Csv matrix for point motion... ")
214         tic
215         parfor i = 1:nVol
216             for j = 1:nCtl
217                 indi = pts_vol(i);
218                 indj = bpts_ordered(j);
219                 Csv(i,j) = RBF('C2',set.r,dom(indi,:),dom(indj,:));
220             end
221         end
222         fprintf("%fs\n",toc)
223
224         fprintf("Inverting Css matrix... ")
225         tic
226         H = Csv*inv(Css);
227         fprintf("%fs\n",toc)
228
229         % initial variables for structural solve
230         xt = 0;
231         xtm1 = 0;
232         yt = 0;
233         ytm1 = 0;
234     end
235
236     % if we're restarting, overwrite the settings in the restart file with the ones in the
237     % configuration file, and delete the temp settings
238     % otherwise, set up all the initial conds etc.
239     if restart
240         clear tmpSet.restartFile; % don't want a restart file in a restart file
241         set = tmpSet;
242         clear tmpSet restart;
243     else
244         % STEP 0: initialise the simulation
245
246         % set psi to be zero everywhere to begin, boundary conditions will be
247         % applied once the sim is running
248         psi = zeros(N,1);
249
```

```
250         % set omega to be zero everywhere except at walls, where we use
251         % the boundary velocities to calculate vorticity, which then will
252         % carry the boundary velocity into the domain
253         omega = zeros(N,1);
254 
255         % set initial velocities zero everywhere, boundaries get set during loop
256         vel = zeros(N,2);
257 
258         % create an identity matrix for the velocity transport equations so we
259         % don't have to generate every loop
260         eye4vte = eye(length(bpts));
261 
262         % start at time = 0
263         tstep = 0;
264 
265         % write a file with the initial settings, so we always know
266         % what settings this file was run with, and we don't have to
267         % recalculate everything above
268         clear set.restartFile;
269         clear restart;
270         filename = strcat(outDir,'restart_tstep0.mat');
271         fprintf("Writing initial data to %s... ",filename)
272         tic
273         save(filename);
274         fprintf("%fs\n",toc)
275     end
276 
277     % if we're on a desktop, create a plot
278     if ispc
279         figure
280     end
281 
282     fprintf("----------------\nSIMULATION START\n----------------\n")
283     while tstep*set.dt < set.T
284 
285         % solve poisson equation for psi, setting psi @ boundary.
286         % note that the dirichlet boundary condition for psi is in omega for
287         % the poisson equation, i.e. this is NOT a vorticity boundary condition
288         % since the Apoiss matrix is modified to be identity at these points.
289         % the vorticity boundary condition is calculated in the NEXT step.
290         om = -omega;
291         psi_m = psi_m0 + psi_m1;
292         om(bpts_w) = psi_w;
293         om(bpts_i) = psi_i;
294         om(bpts_f) = psi_f;
295         om(bpts_m) = psi_m;
296         % solve the equation
297         psi = Apoiss\om;
298         % set psi on the boundary again for good measure
299         psi(bpts_w) = psi_w;
300         psi(bpts_i) = psi_i;
301         psi(bpts_f) = psi_f;
302         psi(bpts_m) = psi_m;
303 
304         % calculate omega at walls from psi, use Jensen's approximation
305         qpts1 = dom([bpts_w;bpts_m],:)+set.h*bcs([bcs_w;bcs_m],bciNORMxy);
306         qpts2 = dom([bpts_w;bpts_m],:)+2*set.h*bcs([bcs_w;bcs_m],bciNORMxy);
307         si = scatteredInterpolant(dom(:,1),dom(:,2),psi);
308         si.Method = 'natural';
309         psi_int1 = si(qpts1(:,1),qpts1(:,2));
310         psi_int2 = si(qpts2(:,1),qpts2(:,2));
311         % thom
312         omega([bpts_w;bpts_m]) = 2/set.h^2*([psi_w;psi_m] - psi_int1 + set.h*bcs([bcs_w;bcs_m],bciVELt));
313         % jensen
314         %omega([bpts_w;bpts_m]) = 1/2/set.h^2*(7*[psi_w;psi_m]-8*psi_int1 + psi_int2 + 6*set.h*bcs([bcs_w;
      bcs_m],bciVELt));
315         omega([bpts_i;bpts_f]) = 0;
316 
317         % calculate velocities
318         vel(:,1) = Ay*psi;
319         vel(:,2) = -Ax*psi;
320 
321         % set velocities at moving walls
322         vel(bpts_m,:) = bcs(bcs_m,bciVELxy);
323 
324         % create diagonal velocity matrices
325         dx = spdiags(vel(:,1),0,N,N);
326         dy = spdiags(vel(:,2),0,N,N);
327 
328         % build velocity transport equation matrix
329         vte = 1/set.Re*Alap - dx*Ax - dy*Ay - N^-set.ord_hyp*Ahyp;
330 
331         % set boundaries in matrix
332         vte(bpts,bpts) = eye4vte;
333 
334         % step in time using RK4
335         k1 = vte*omega;
336         k2 = vte*(omega+set.dt*k1/2);
337         k3 = vte*(omega+set.dt*k2/2);
338         k4 = vte*(omega+set.dt*k3);
339         omega = omega + set.dt/6*(k1 + 2*k2 + 2*k3 + k4);
340         tstep = tstep + 1;
341         t = tstep*set.dt;
```

```
342
343         % if fsi is enabled, move the boundary
344         if (set.fsi == 1 && t >= set.fsi_start)
345
346             % create interpolator before we move the points
347             si_omega = scatteredInterpolant(dom(:,1),dom(:,2),omega,'natural');
348
349             % calculate pressure around moving walls
350             qpts = dom(bpts_m,:)+set.hv*bcs(bcs_m,bciNORMxy);
351             si_velx = scatteredInterpolant(dom(:,1),dom(:,2),vel(:,1),'natural');
352             si_vely = scatteredInterpolant(dom(:,1),dom(:,2),vel(:,2),'natural');
353             velx = si_velx(qpts(:,1),qpts(:,2));
354             vely = si_vely(qpts(:,1),qpts(:,2));
355
356             % calculate forces by summing pressures in normal directions
357             % (negative since pressure force works against normal)
358             F = -sum((velx.^2+vely.^2).*bcs(bcs_m,bciNORMxy));
359
360             % step the structural solver
361             xtp1 = set.fsi_x*StructuralSolve(xt,xtm1,set.freq_x,set.damp_x,set.dt,F(1));
362             ytp1 = set.fsi_y*StructuralSolve(yt,ytm1,set.freq_y,set.damp_y,set.dt,F(2));
363
364             xtm1 = xt;
365             ytm1 = yt;
366             xt = xtp1;
367             yt = ytp1;
368
369             % calculate velocities of moving points
370             vx = (xt-xtm1)/set.dt;
371             vy = (yt-ytm1)/set.dt;
372
373             % move points
374             dom(bpts_m,:) = dom0(bpts_m,:) + [xt,yt];
375             displacements = [repmat([xt,yt],length(bpts_m),1);zeros(length(bpts)-length(bpts_m),2)];
376             dom(pts_vol,:) = dom0(pts_vol,:) + H*displacements;
377
378             % interpolate omega to new points
379             omega = si_omega(dom(:,1),dom(:,2));
380
381             % recalculate derivative matrices
382             CalcDerivMatrix;
383
384             % set tangential and normal velocity at the walls
385             bcs(bcs_m,bciVELxy) = repmat([vx,vy],length(bcs_m),1);
386             bcs(bcs_m,bciVELn) = dot(bcs(bcs_m,bciVELxy),bcs(bcs_m,bciNORMxy),2);
387             bcs(bcs_m,bciVELt) = dot(bcs(bcs_m,bciVELxy),tangents(bcs_m,:),2);
388
389             % shift moving points to origin and skew in z direction to
390             % calculate boundary condition for psi
391             pts = dom(bpts_m,:) - [set.cenx+xt,set.ceny+yt];
392             psi_m1 = vy*pts(:,1) + vx*pts(:,2);
393
394             fprintf("step = %i \tt = %f \tforce = [%d,%d] \tdisplacement = [%d,%d]\n",tstep,t,F(1),F(2),xt,
    yt);
395         else
396             fprintf("step = %i \tt = %f\n",tstep,t);
397         end
398
399         % if we're on desktop, plot the streamfunction and vorticity
400         if (ispc && mod(t,set.tout) < set.dt/2)
401             subplot(1,2,1)
402             plot3(dom(:,1),dom(:,2),omega,'k.')
403
404             subplot(1,2,2)
405             plot3(dom(:,1),dom(:,2),psi,'k.')
406
407             pause(0)
408         end
409
410         % output data
411         if (mod(t,set.tout) < set.dt/2)
412             filename = strcat(outDir,num2str(tstep),'.dat');
413             fprintf("Writing output to %s... ",filename)
414             fileID = fopen(filename,'w');
415             fprintf(fileID,'%f %f %f %f %f %f\n',[dom,vel(:,1),vel(:,2),omega,psi]');
416             fclose(fileID);
417             fprintf("%fs\n",toc)
418             if exist('F','var')
419                 fileID = fopen(strcat(outDir,"structural.dat"),"a+");
420                 fprintf(fileID,"%i %f %f %f %f %f\n",[tstep t F(1) F(2) xt, yt]);
421                 fclose(fileID);
422             end
423         end
424
425         % check for nan
426         if isnan(omega+psi)
427             fprintf("Solution contains NaN\n")
428             break;
429         end
430     end
431     fprintf("--------------\nSIMULATION END\n--------------\n")
432
433     % create a restart file at end of simulation
```

```
    clear set.restartFile;
    clear restart;
    filename = strcat(outDir,'restart_tstep', string(tstep),".mat");
    fprintf("Writing restart data to %s... ",filename)
    tic
    save(filename);
    fprintf("%fs\n",toc)
end
```

../rbf–fd–fsi/CalcDerivMatrix.m

```
% preallocate i,j,v arrays for use with sparse(i,j,v), we reshape these below
A_i = zeros(N,set.sten); % indices for matrices (all have the same sparsity pattern)
A_j = zeros(N,set.sten);
Ax_v = zeros(N,set.sten); % first deriv, x direction
Ay_v = zeros(N,set.sten); % first deriv, y direction
Alap_v = zeros(N,set.sten); % laplacian operator
Ahyp_v = zeros(N,set.sten); % second deriv, cross term

% extract the x and y coordinates of each stencil and store in matrix
% to ensure we can access quickly in the parallel step (i.e. sliced arrays)
sx = reshape(dom(stencils,1),[N set.sten]);
sy = reshape(dom(stencils,2),[N set.sten]);

% parallel loop through and calculate weights at each point/stencil and write to derivative matrices
parfor i = 1:N

    % extract x and y coords for current stencil
    stenx = sx(i,:)';
    steny = sy(i,:)';

    % find index of center point
    [tf, ind_cent] = ismember(dom(i,:),[stenx, steny],'rows');

    if tf

        j = stencils(i,:);
        w = RBFFDWeights(ind_cent,stenx,steny,set.ord_phs,set.ord_pol,set.ord_hyp);

        % write weights to appropriate arrays
        A_i(i,:) = i;
        A_j(i,:) = j;
        Ax_v(i,:) = w(:,1);
        Ay_v(i,:) = w(:,2);
        Alap_v(i,:) = w(:,3);
        if set.ord_hyp > 0
            Ahyp_v(i,:) = w(:,end);
        else
            Ahyp_v(i,:) = 0;
        end
    else
        warning("couldn't find stencil center in stencil");
    end
end

% reshape the matrices into columns
sz = [1 N*set.sten];
A_i = reshape(A_i,sz)';
A_j = reshape(A_j,sz)';
Ax_v = reshape(Ax_v,sz)';
Ay_v = reshape(Ay_v,sz)';
Alap_v = reshape(Alap_v,sz)';
Ahyp_v = reshape(Ahyp_v,sz)';

% create sparse matrices
Ax = sparse(A_i,A_j,Ax_v);
Ay = sparse(A_i,A_j,Ay_v);
Alap = sparse(A_i,A_j,Alap_v);
Ahyp = sparse(A_i,A_j,Ahyp_v);

% set up matrix for solving poisson equation with built in boundary cond
Apoiss = Alap;

% set the rows to zero, since coupling should only be 1-way, i.e. from
% boundary to domain
Apoiss(bpts_w,:) = 0;
Apoiss(bpts_i,:) = 0;
Apoiss(bpts_f,:) = 0;
Apoiss(bpts_m,:) = 0;

% set identities to setup dirichlet condition equations
Apoiss(bpts_w,bpts_w) = eye(length(bpts_w));
Apoiss(bpts_i,bpts_i) = eye(length(bpts_i));
Apoiss(bpts_f,bpts_f) = eye(length(bpts_f));
Apoiss(bpts_m,bpts_m) = eye(length(bpts_m));
```

../rbf–fd–fsi/StructuralSolve.m

```
% returns the displacement of mass-spring-damper system at  x^(t+1) when
```

```
% given the current displacment x^t
% and displacement at last timestep x^(t-1)
% uses second order central difference scheme

% xtp1 = x^(t+1), return value for new displacement
% xt = x^t, current displacement
% xtm1 = x^(t-1), displacement at last timestep
% dt = time delta
% omega_n = natural frequency
% zeta = damping ratio
% F = external force

function xtp1 = StructuralSolve(xt,xtm1,omega_n,zeta,dt,F)
    c0 = 1 + dt*zeta*omega_n;
    c1 = dt*dt;
    c2 = dt*zeta*omega_n - 1;
    c3 = 2 - dt*dt*omega_n*omega_n;
    xtp1 = (c1*F + c2*xtm1 + c3*xt)/c0;
end
```

../rbf–fd–fsi/RBFFDWeights.m

```
function w = RBFFDWeights (i,x,y,m,d,k)

% this function is adapted from
%
% "Enhancing finite differences with radial basis functions:
%  Experiments on the Navier-Stokes equations"
%
% by Natasha Flyer, Gregory A. Barnett, Louis J. Wicker, 2016

% Input parameters
% x,y   Column vectors with stencil node locations; approximation to
%       be accurate at x(1),y(1)
%   m   Power of r in RBF fi(r) = r^m, with m odd, >= 3.
%   d   Degree of supplementary polynomials (d = -1 no polynomials)
%   k   Degree of hyperviscosity term
%
% Output parameter
%   w   Matrix with three columns, containing weights for d/dx, d/dy,
%       and the Laplacian d2/dx2+d2/dy2, respectively.

% Shift nodes so stencil centered at origin
x = x-x(i);
y = y-y(i);
n = length(x);


% ------ RBF part ------------------------------------------------------
% calculate matrix of distances from each point to each other point
% note this is invariant under the translation to the origin above
dists = hypot(bsxfun(@minus,x,x'),bsxfun(@minus,y,y'));

% calculate matrix of r^m
A0 = dists.^m;

% RBF matrix

% radius is just distance to origin, which is the center of the stencil
% since we translated above, so use that rather than recalculate
r = dists(:,1);

L0 = m*(bsxfun(@times,r.^(m-2),...
    [...
     -x,... % dx
     -y,... % dy
     m*ones(n,1),... % laplacian
     ])); % RHSs

if k > 0
     hyperv = 1;
     for i=1:k
         hyperv = hyperv*(m-(2*i-1)).^2;
     end
     hyperv = hyperv.*r.^(m-2*k);
     L0 = [L0,hyperv];
end
% we can get a divide by zero above, but these should be zero
L0(isnan(L0))=0;

% ------ Polynomial part -----------------------------------------------
if d == -1
    % Special case; no polynomial terms,
    % i.e. pure RBF
    A = A0;
    L = L0;
else
    % Create matrix with polynomial terms and matching constraints
    X       = x(:,ones(1,d+1));
    X(:,1)  = 1;
    X       = cumprod( X,2);
```

```
    Y       = y(:,ones(1,d+1));
    Y(:,1)  = 1;
    Y       = cumprod( Y,2);
    np      = (d+1)*(d+2)/2; % Number of polynomial terms

    XY = zeros(n,np); col = 1; % Assemble polynomial matrix block

    for k = 0:d
        XY(:,col:col+k) = X(:,k+1:-1:1).*Y(:,1:k+1);
        col = col+k+1;
    end

    L1 = zeros(np,length(L0(1,:))); % Create matching RHSs

    if d >= 1
        L1(2,1) = 1;
        L1(3,2) = 1;
    end
    if d >= 2
        L1(4,3) = 2;
        L1(6,3) = 2;
    end

    A = [A0,XY;XY',zeros(col-1)]; % Assemble linear system to be solved
    L = [L0;L1]; % Assemble RHSs

end
% ------ Solve for weights -------------------------------------------------
W = A\L;
w = W(1:n,:); % Extract the RBF-FD weights
end
```

../rbf–fd–fsi/RBF.m

```
function out = RBF(rbf,r,X,Xc)
%Xnorm = sqrt((X(:,1) - Xc(:,1)).^2 + (X(:,2) - Xc(:,2)).^2 + (X(:,3) - Xc(:,3)).^2);
Xnorm = norm(X-Xc);
x = Xnorm/r;
switch rbf
    case 'C0'
        out = (1-x).^2;
        if Xnorm > r
            out = 0;
        end
    case 'C2'
        out = (1 - x).^4.*(4*x + 1);
        if Xnorm > r
            out = 0;
        end
    case 'C4'
        out = (1-x).^6.*(35*x.^2+18*x+3)/3;
        if Xnorm > r
            out = 0;
        end
    case 'C6'
        out = (1-x).^8.*(32*x.^3+25*x.^2+8*x+1);
        if Xnorm > r
            out = 0;
        end
    case 'Euclid'
        out = pi*((1/12*x.^3)-0.5^2*x+4/3*0.5^3)/(pi*(-0.5^2*0+4/3*0.5^3));
        if Xnorm > r
            out = 0;
        end
    case 'Multiquadric'
        out = sqrt(1+x.^2);
    case 'InverseMulti'
        out = 1./sqrt(1+x.^2);
    case 'TPS'
        out = x.^2*log(x);
    case 'Gaussian'
        out = exp(-x.^2);
    otherwise
        error('RBF not recognised.');
end
if isnan(out)
    out = 0;
end
end
```

../rbf–fd–fsi/GetNearest.m

```
function out = get_nearest(pts,k)
    kdts = KDTreeSearcher(pts);
    out = knnsearch(kdts,pts,'k',k);
end
```

../rbf–fd–fsi/FillTrap.m

```
% fills a trapezoid with smoothly spaced points
%
% s1 = length of bottom side
% s2 = length of top side
% h = height of trapezoid
% ptCnt = number of points along bottom size of trapezoid
% er = expansion rate

function [x,y] = FillTrap(s1,s2,h,ptCnt,er)

rise = h;
run = (s2-s1)/2;
slope = rise/run;

% calculate initial spacing
x = linspace(-s1/2,s1/2,ptCnt);
spc = (x(2)-x(1));

% calculate y spacing
y = 0;
ex = er;
while max(y) <= h
    y = [y,max(y)+spc*ex];
    ex = ex*er;
end



% if removing a row would get us closer to the height, do it
if y(end)-h > h-y(end-1)
    y = y(1:end-1);
end

% rescale y
y = y/max(y)*h;

% now go through generate the points
x = linspace(-s1/2,s1/2,ptCnt)';
out = [x,x*0+y(1)];
for i = 2:length(y)
    len = s1+2*y(i)/slope;
    spc = y(i)-y(i-1);
    num = round(len/spc)+1;
    x = linspace(-len/2,len/2,num)';
    out = [out;x,x*0+y(i)];
end

out = uniquetol(out,'ByRows',true);
x = out(:,1);
y = out(:,2);

end
```

B. AIAA SCITECH 2020 FORUM PAPER

The following paper [6] was delivered at SciTech2020. It is provided for interest, as an example of broader applications of RBFs.

Catastrophe Theoretic Modelling of Hysteresis in Transonic Shock Buffet

Adam Murray*, Nicholas Giannelis†, and Gareth A. Vio‡
School of Aerospace, Mechanical, and Mechatronic Engineering, University of Sydney, Australia

Given particular flow conditions within the transonic flow regime, undesirable dynamic lift and pitch profiles are produced due to the emergence of self sustaining oscillating shock waves known as shock buffet. Shock buffet onset has been observed to exhibit hysteresis with changes in angle of attack both experimentally and numerically. Here we model this hysteresis in the onset phase by way of catastrophe theory, a sub-branch of bifurcation theory in which bifurcations are studied using the geometry of a sufficiently smooth Lyapanov function. Numerical simulations are conducted for 2D flow over a thin aerofoil in the transonic regime and the nominal buffet response is discussed. The hysteresis behaviour is modelled using a cusp-catastrophic model of the interactions between Mach number, angle of attack, and lift coefficient, with buffet onset determined to be when the lift coefficient becomes oscillatory. The hysteresis boundary and cusp (bifurcation) points of the cusp-catastrophic model is then fit using a subset of the numerical data (including the cusp point) as a training set. The remaining numerical data is compared to the model predictions for the hysteresis boundary, both interior and exterior to the training set. The model is found to agree with numerical results within 3%. Future work will include investigating a dual cusp treatment for entry/exit into the buffet regime, expansion of the model to describe additional aspects of the buffet phenomenon, and the applicability of the model in the three dimensional case, which has clear importance for transonic applications.

I. Introduction

Catastrophe theory to model hysteretic aerodynamic phenomenon has recently been used by Li et al.[1], in which they predict intermediate values on the stall boundary for the 2D simulation of aerofoils. The stall boundary exhibits hysteresis with respect to angle of attack and their work highlights the success that can be achieved in the use of a simple mathematical model to predict aerodynamic phenomenon. Here we use the underlying mathematical theory to model transitions in and out of transonic shock buffet, a phenomenon which exhibits hysteresis with respect to Mach number and angle of attack.

A. Transonic Shock Buffet

In a transonic flowfield, variations in flight condition may induce large-scale unsteadiness in the form of oscillating shock waves and intermittently separated shear layers. Such autonomous shock oscillations, denoted as transonic shock buffet, are problematic for civil and military aircraft alike and often impose limitations on the flight envelope. Although the phenomenon evolves as a purely aerodynamic instability [2], the low frequencies of shock oscillation are typically of the same order as the fundamental structural modes, and in an aeroelastic system, may be a contributing factor to transonic limit cycle oscillations [3–8]; a detriment to both structural fatigue life and platform handling qualities [9].

The origin of the shock buffet instability remains contentious in the literature, with early works proposing onset due to shock-induced separation bubble bursting [10], an acoustic wave-propagation feedback mechanism [11] and an unstable shock wave/separation bubble interaction [12]. More recent works, however, support onset as a consequence of a global aerodynamic mode instability arising from a Hopf bifurcation [13–15]. Nonetheless, reliable and computationally cheap methods of buffet onset prediction remain elusive, with onset envelopes predicted either through empirical correlations with conservative safety margins or high-order CFD simulation.

Recent research in the field has also identified similarities between buffet cell propagation for three-dimensional swept wings and stall cells observed in subsonic flows [16, 17]. Even in a two-dimensional sense, the onset boundaries

*PhD Researcher
†PhD Researcher
‡Senior Lecturer

for the two phenomena are remarkably similar. An increase in angle of attack yields unsteady flow characteristics above a threshold (the onset condition), with hysteresis present in the response [18]. In this study, we draw on the presence of hysteresis and similarities to stall to develop an analytical model founded in Catastrophe theory for buffet onset prediction.

B. Catastrophe Theory

Catastrophe theory has its origins in Whitney's work on singularities of smooth mappings, and the work of Poincare and Andronov on bifurcations. The theory was first codified by Rene Thom in the 1960s, and soon became a popular method for exploring complex dynamical systems [19]. Similar to the theory of normal forms, the theory uses the geometry of potential functions around singularities to generalise nonlinear behaviour as part of a smooth, well-defined Lyapanov function. In this way, we can use familiar notions from geometry and calculus to shed light on the behaviour of badly behaved systems, e.g. discontinuous. The theory examines smooth Lyapanov functions and their degenerate critical points, i.e. those points and which not only the first, but higher order derivatives vanish. Since the potential function is smooth and well defined in the areas of interest, it is possible to generalise and classify behaviours of systems around such degenerate points through diffeomorphisms: smooth mappings of the specific Lyapanov function of the system to a simpler geometric form.

II. Cusp-Catastrophic Model

The canonical form for potential function of a cusp-catastrophic model with parameters x and y is given by

$$V(\varphi) = \varphi^4 + x\varphi^2 + y\varphi. \tag{1}$$

The critical points of V are given when $V' = 0$, i.e.

$$V'(\varphi) = 0 = 4\varphi^3 + 2x\varphi + y, \tag{2}$$

with *degenerate* critical points being when higher derivatives of the potential function are also zero. To calculate the degenerate critical points of the cusp catastrophe potential function, we take the second derivative to find

$$V''(\varphi) = 0 = 12\varphi^2 + 2x, \tag{3}$$

the parabolic nature of which can be seen in fig. 1 on the upper surface. These degenerate critical points are seen appear along the local extrema of the critical surface as expected. Projecting the degenerate points into local (parameter) coordinates, the catastrophe points produce the cusp geometry shown below the surface. To achieve this projection, we use V'' to eliminate the dependence on the variable φ in V' and find that the equation for hysteresis boundary with cusp point is given by

$$8x^3 + 27y^2 = 0. \tag{4}$$

Note that as parameter x passing over the cusp point corresponds to a change in the qualitative behaviour of the system, i.e. the transfer from non-history dependent behaviour to behaviour exhibiting hysteresis, while the two curves of the boundary correspond to onset and offset of shock-buffet in the hysteretic region (x < 0).

Although this standard form is only expected in general to produce qualitative behaviour of the system around the cusp point (i.e. local behaviour), we can perform a regression on the data in combination with a fixed smooth mapping to fit the simulation points to the standard cusp, and hence provide a qualitative physical model away from the cusp point as in the case of Li et al [1]. The detailed development and fitting of the model is discussed in section IV.

III. Buffet Modelling

A. Test case

The test case considered in this study is that of the thin NACA 64A204. This aerofoil was employed on the main wing of the F-16 fighter; a platform known to exhibit transonic limit cycle instabilities [20]. It must be stressed that to the authors' knowledge, no wind tunnel studies of shock buffet on this profile are available in open literature. Further, the findings of the computational studies related to this aerofoil are somewhat contradictory. Iovnovich & Raveh [21] have

found characteristic Type A shock motion at a flow condition of $M = 0.75$, $Re = 10 \times 10^6$ and $\alpha = 5.5°$. Bhamidipati et. al. [22] found that while they were able to produce shock buffet at this condition in an early study [23], a damped response was observed at higher levels of spatial resolution and spanwise extent. Ultimately, the difficulty in assessing the buffet response of this aerofoil is the lack of experimental data. Herein, methods that have effectively reproduced the shock buffet phenomenon in preceding studies by the authors [24, 25] are employed on the NACA 64A204 at equivalent conditions to Iovnovich & Raveh [21].

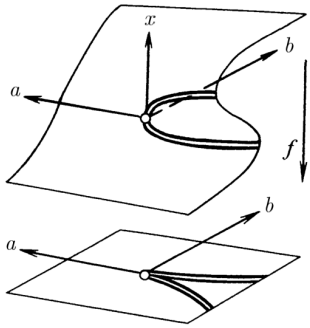


Fig. 1 Projection of cusp catastrophe [19]

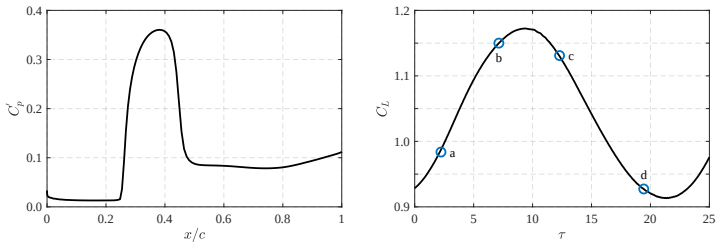
B. Numerical method

Simulations throughout this study have been performed with the cell-centred finite volume code ANSYS Fluent R18.2 [26]. Transonic flow over the NACA 64A204 profile is considered, with the two-dimensional pressure-based implicit solver used to formulate the coupled set of continuity and momentum equations, with the energy equation solved in a segregated manner. The inviscid fluxes are resolved with an upwind Roe flux difference splitting scheme, and second-order upwind differencing is used for extrapolation of the convective quantities. The diffusive fluxes are treated with a second-order accurate central-difference scheme. Gradients for the convective and diffusive terms are computed at cell faces through a Green-Gauss reconstruction scheme, in addition to a differentiable gradient limiter to mitigate spurious shock oscillations. Viscous closure of the Navier-Stokes equations is achieved with the Stress-Omega Reynolds Stress Model (SORSM), derived from the omega equations and the Launder-Reece-Rodi (LRR) model [27]. All turbulent transport equations are solved segregated from the coupled set of continuity, momentum and energy equations, with second-order accurate upwind discretisation of the turbulent quantities. Regarding the computational domain, a two-dimensional, CH-type grid is used with an upstream through to wall-normal boundary of 40 chords and downstream domain length of 60 chords. All computations presented in this paper are performed on a grid of approximately 67000 grid points with a nondimensional time-step (with respect to the speed of sound) of $\tau = 0.008$, as determined through the spatial and temporal convergence study provided in Giannelis et. al. [28].

C. Nominal buffet response

For the nominal buffet condition, fig. 2 shows the root-mean-square pressure coefficient and lift coefficient time-history during a buffet cycle. At this condition, the shock traverses 20% of the aerofoil chord, with the aerodynamic coefficients fluctuating in an approximately sinusoidal manner. Such a response is typical of Type A Tijdeman [29] shock motion, observed in the vicinity of buffet onset on two-dimensional profiles [25].

In fig. 3, Mach number contours are provided at time instances during the buffet cycle corresponding to the data points in fig. 2(b). Figure 3(a) shows the shock travelling downstream and gaining strength. With the shock at its most aft position in fig. 3(b), the high pressure imparted aft of the shock foot by the separation bubble causes the shock shift



(a) Root-mean-square pressure coefficient (b) Lift coefficient time-history

Fig. 2 RMS pressure coefficient and lift time-history at nominal buffet conditions $M = 0.75$, $\alpha = 5.5°$ & $Re = 10 \times 10^6$. Data points correspond to Mach number contour snapshot presented in fig. 3.

upstream. As identified by Iovnovich & Raveh [21], the shock-induced separation produced by the pressure rise through the shock behaves as a geometric wedge, increasing the shock strength with an increase in separation. This is evident in fig. 3(c), where a pronounced slanting of the shock is observed during its upstream excursion. As the shock reaches its most upstream position in fig. 3(d), the separated region encompasses approximately 70% of aerofoil chord. A weakening of the shock results in reattachment of the flow immediately aft of the sonic region. This reattachment point progresses aft towards the trailing edge as the shock begins moving downstream, repeating the cycle.

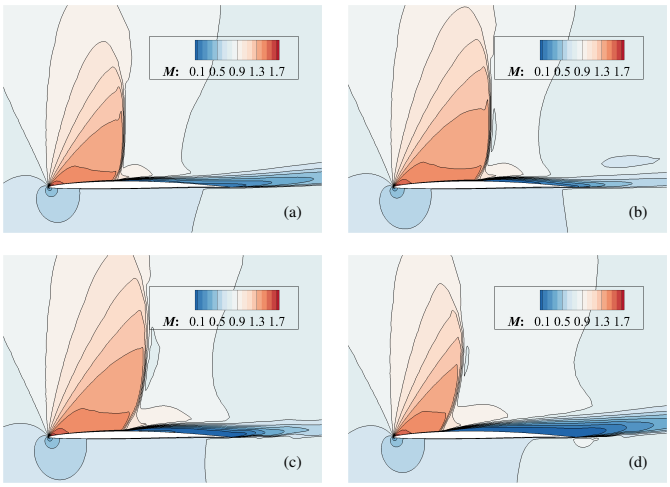
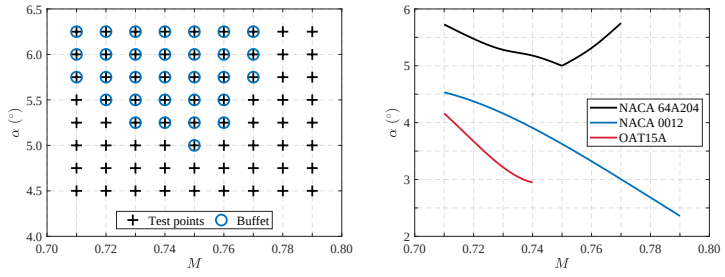


Fig. 3 NACA 64A204 Mach contours over a single buffet cycle ($M = 0.75$, $\alpha = 5.5°$, $Re = 10 \times 10^6$).

D. Buffet boundary

Building from the nominal buffet condition, a test matrix within Mach number (M) and angle of attack (α) space has been investigated to determine the extent of freestream conditions for which shock buffet is evident. fig. 4 provides this test space, in addition to the conditions where large-scale shock motion has been observed. In fig. 4(b), the computed onset boundary for the NACA 64A204 is compared with the onset boundaries of two thicker profiles for a corresponding Mach number range; the OAT15A supercritical aerofoil [25] and the symmetric NACA 0012 [24]. While the present work is not intended to provide an extensive comparison of onset conditions between different profiles, an interesting distinction is noted between the thick and thin aerofoils. For the OAT15A and NACA 0012, the onset incidence is shown to increase with a reduction in Mach number. Although this is also observed at lower Mach numbers for the NACA 64A204, a local minimum onset angle of attack is found at $M = 0.75$, with onset incidence increasing at higher Mach numbers. The unique onset behaviour of the thin profile may be representative of a distinct buffet response, warranting further investigation in a future study.

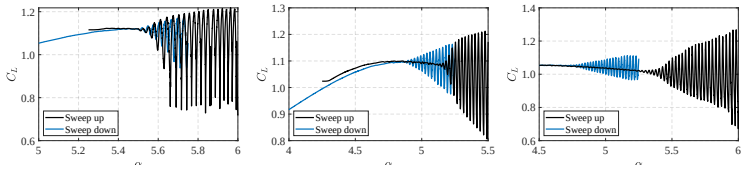


(a) NACA 64A204 buffet envelope.

(b) Comparison of onset boundary of various profiles.

Fig. 4 NACA 64A204 buffet envelope and comparison of onset boundary with the OAT15A [25] and NACA 0012 [24].

Having identified the test conditions for which shock buffet occurs, angle of attack sweeps are then performed for Mach numbers between $0.71 < M < 0.78$. Both increasing and decreasing sweeps are conducted, with the sweeps beginning at a minimum of $\Delta\alpha = \pm 0.5°$ from the corresponding onset condition in fig. 4(a). The angle of attack sweeps are performed in a quasi-steady manner, with an angular velocity of $\pm 0.002°$ per travelled chord length. Similar to Nitzsche [18], these sweeps identify a hysteretic region of incidence, with large-scale unsteadiness continuing to lower angles of attack on the downward sweep. The extent of this hysteretic zone varies with Mach number, generally increasing from $M = 0.71$ with an increase in Mach number as shown in fig. 5.



(a) $M = 0.71$

(b) $M = 0.74$

(c) $M = 0.76$

Fig. 5 Onset and offset of oscillatory shock behaviour as a function of angle of attack with increasing Mach number.

Figure 5(a) indicates the at $M = 0.71$ the hysteretic region has effectively dissipated, with the shock buffet onset condition coincident between the upward and downward sweeps. A representation of the hysteretic region across each of the Mach numbers considered is also shown in fig. 6. The hysteretic zone is clearly seen to diminish with a decrease in Mach number, forming a cusp at the $M = 0.71$ condition. In subsequent sections, this data will be employed to explore the efficacy of a cusp-catastrophe model on capturing this shock buffet hysteresis.

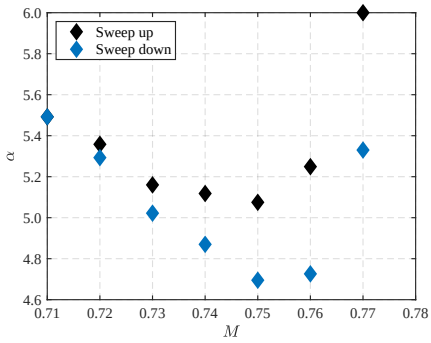


Fig. 6 NACA 64A204 shock buffet hysteretic region with incidence sweeps.

IV. Modeling of Buffet Hysteresis

Table 1 summarises the angles of attack at which shock buffet begins during the increasing sweep (onset) and ceases during the decreasing sweep (offset) as found by the numerical simulation of the previous section.

Mach	Sweep Up	Sweep Down
0.71	5.492	5.492
0.72	5.358	5.293
0.73	5.160	5.022
0.74	5.118	4.870
0.75	5.075	4.695
0.76	5.249	4.726
0.77	6.000	5.33

Table 1 Points of shock buffet on/offset during angle of attack sweeps with varying Mach number.

Figure 6 shows clearly the natural cusp behaviour of the system as it approaches Mach 0.71. Currently the behaviour at higher speeds (M > 0.77) is outside of consideration, but will be the topic of future work, both modeling the behaviour separately and unifying the overall system with the single cusp model as presented here. The envelope is highly non-linear, both up and downward sweeps, although both the upper and lower boundaries mimic each other's geometry.

The task of fitting the numerical data to the projected cusp model geometry of fig. 1 can be achieved in 3 individual coordinate transforms as shown in fig. 7. As the topology of the data and the cusp model of equation 1 is the same, the mappings are constructed to be continuous and invertible, and hence it is not necessary to calculate them all in the same direction. This reduces the mathematical complexity of calculating the mappings. In the following we use the coordinate notation of fig. 7, i.e. (x, y), (η, ξ), (γ, ζ) and (X, Y) for the cusp, symmetric linear, asymmetric linear, and numerical results respectively.

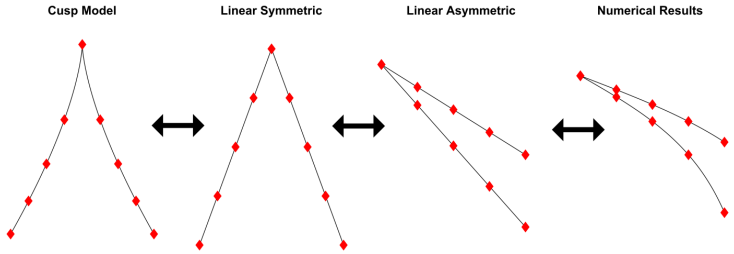


Fig. 7 Steps to transform between canonical cusp model and numerical data.

1. Transform 1: Cusp Model to Symmetric Linear Model

By making use of eq. (4), we note that the expressions for the cusp and linear symmetric models respectively are given by

$$y = -3x^{\frac{2}{3}}, \quad \xi = -|\eta|, \tag{5}$$

the mapping of points between the cusp model and the linear model follows trivially. It is possible to state the expression for the symmetric linear model in more generality by introducing further parameters, e.g. a slope, however as the model serves as an interim space between the cusp model and the simulation data, the simplest expression is preferred.

2. Transform 2: Symmetric Linear Model to Asymmetric Linear Model

The transform from the symmetric linear model to asymmetric linear model is affine, and hence can easily be split into a linear transform and a translation. Once the translation is calculated by matching the cusp points (here we assume both are translated to the origin for simplicity), the linear transform is able to be calculated from the action of the transform on two points. The equation to be solved is then simply

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} \eta_1 & \eta_2 \\ \xi_1 & \xi_2 \end{bmatrix} = \begin{bmatrix} \gamma_1 & \gamma_2 \\ \zeta_1 & \zeta_2 \end{bmatrix} \tag{6}$$

with a, b, c, d unknown. The solution given by

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} \gamma_1 & \gamma_2 \\ \zeta_1 & \zeta_2 \end{bmatrix} \begin{bmatrix} \eta_1 & \eta_2 \\ \xi_1 & \xi_2 \end{bmatrix}^{-1} = \frac{1}{\eta_1\xi_2 - \eta_2\xi_1} \begin{bmatrix} \gamma_1 & \gamma_2 \\ \zeta_1 & \zeta_2 \end{bmatrix} \begin{bmatrix} \xi_2 & -\eta_2 \\ -\xi_1 & \eta_1 \end{bmatrix}. \tag{7}$$

As in the case of the previous transform, the asymmetric linear model is an interim space, and hence the two target points (γ_i, ζ_i) can be chosen somewhat arbitrarily. However it is usually helpful at this stage to choose target points to match the extremities of numerical data, reducing the need for scaling in the final transform.

3. Transform 3: Asymmetric Linear Model to Data

While the previous two transformations have been determined completely, mapping the asymmetric linear model to the numerical results necessarily requires a data fitting method. Li et al. have used a powerful data fitting tool in RBF based neural nets. This allows capture of non-linear aspects to the data, and provides a more generally applicable method for more complex hysteresis envelope geometry. An RBF neural net implementation (`newrb` or `newrbe`) is available in MATLAB via the Deep Learning Toolbox.

To map between the data and the straight line of the asymmetric linear model, we use the neural net to find fit on the difference between the numerical results and the linear model as a function of Mach number, i.e. $\Delta_i = \zeta_i - Y_i$. Using a

difference rather than a direct fit also has the advantage of providing an invertible transform, when assuming that the cusp point is fixed (at the origin or otherwise). As discussed in the previous transform, the selection of the lines (i.e. slopes) of the asymmetric model are somewhat arbitrary, however it is best practice to match at least the cusp.

A. Model Results

Figure 8 shows the results of the RBF neural net fitting. Now that three transforms above have been constructed, we have a continuous, invertible map between the canonical cusp model and the numerical data.

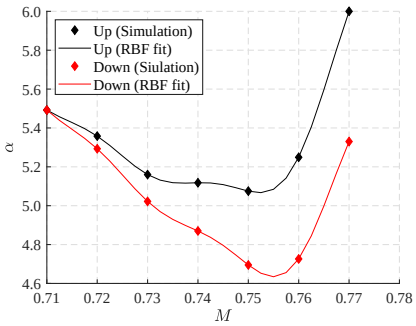


Fig. 8 Results of RBF neural net fitting.

Passing a test Mach numbers through the cusp model and comparing to additional numerical simulations (fig. 9), we see that the model predicts the hysteresis boundary to within 3%.

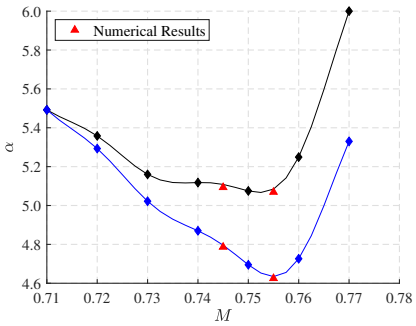


Fig. 9 Testing the model at interim Mach numbers.

V. Conclusions and Future Work

This work has shown that the hysteresis of transonic shock buffet over an aerofoil exhibits cusp catastrophic behaviour. Using simple mathematical modeling techniques, it has been shown that accurate predictions of the onset and offset of shock buffet can be determined interior to numerical data, eliminating at least some need for computationally expensive CFD simulations or experimentation.

A. Applicability to 3D Wings

It should be noted that the method is generic and applies to any cusp catastrophic behaviour observed in physical phenomena. However the exploration of hysteresis envelopes for transonic shock buffet has been so far limited. As such it is unknown as to whether cusps exist in all cases. Further investigation of the hysteresis envelopes of various aerofoils/wing configurations is necessary to determine the overall efficacy of the method to the wider problem of transonic shock buffet.

B. Prediction of Cusp Point

It was initially hoped that the method could provide some means to predict the cusp point of a shock buffet envelope, however due to the highly non-linear nature of the hysteresis envelope, it is expected that this will ultimately prove impossible in any general sense. However, with more extensive study on a range of aerofoils and/or wing types, it may be possible to give accounts of cusp points in certain sub cases.

C. Full Buffet Characteristics

As seen in fig. 1, the cusp boundary of degenerate points is a single curve within a 3D manifold sitting in $\mathbb{R}^3$. A mapping method similar in theme to the one presented here may be able to be developed to capture certain aspects of the buffet regime using the full surface as a model, however additional mathematical machinery may need to be employed due to the additional dimension.

D. Dual Cusp

To describe the boundary of the full transonic shock buffet hysteresis envelope, it may be possible to unify two cusp models into a single model. This would allow a cohesive description of both the entry ($M < 1$) and exit ($M > 1$) of the buffet phenomenon in a single model. Towards this purpose, the following Lyapanov function is proposed:

$$V(\varphi) = \varphi^4 + z\varphi^2 + y\varphi. \tag{8}$$

Here z is a new parameter that depends smoothly on the original parameter x from eq. (1) by

$$z(x) = e^{x^2} - 2. \tag{9}$$

Following the derivation of section II, the degenerate points and hence model for the hysteresis boundary is given implicitly by

$$8\left(e^{x^2} - 2\right)^3 + 27y^2 = 0. \tag{10}$$

Figure 10 shows the above model, which is locally similar to the single cusp model at each extremity, but globally distinct, as it now is able to model the entire envelope. It is hoped that this model will provide a more complete picture of the hysteresis boundary for the transonic shock phenomenon.

References

[1] Li, Z., Peng, Z., Pan, T., Li, Q., Zhang, J., and H. Dowell, E., "Catastrophe-Theory-Based Modeling of Airfoil-Stall Boundary at Low Reynolds Numbers," *AIAA Journal*, Vol. 56, 2017, pp. 1–10. doi:10.2514/1.J056048.

[2] Lee, B. H. K., "Self-sustained shock oscillations on airfoils at transonic speeds," *Progress in Aerospace Sciences*, Vol. 37, No. 2, 2001, pp. 147–196.

[3] Giannelis, N. F., Murray, A. J., and Vio, G. A., "Application of the Hilbert-Huang Transform in the identification of frequency synchronisation in transonic aeroelastic systems," *in: AIAA Scitech 2019 Forum*, AIAA Paper 2019-1341, San Diego, CA, 2019.

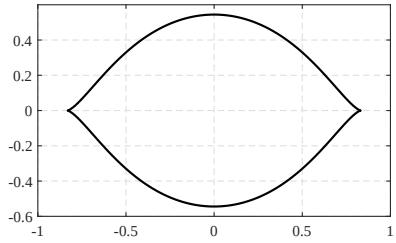


Fig. 10 Hysteresis boundary in double cusp model.

[4] Raveh, D. E., and Dowell, E. H., "Aeroelastic Responses of Elastically Suspended Airfoil Systems in Transonic Buffeting Flows," *AIAA Journal*, Vol. 52, No. 5, 2014, pp. 926–934.

[5] Giannelis, N. F., and Vio, G. A., "Investigation of frequency lock-in phenomena on a supercritical aerofoil in the presence of transonic shock oscillations," *Proceedings of the 17th International Forum on Aeroelasticity and Structural Dynamics, Como, Italy*, 2017.

[6] Gao, C., Zhang, W., Li, X., Liu, Y., Quan, J., Ye, Z., and Jiang, Y., "Mechanism of frequency lock-in in transonic buffeting flow," *Journal of Fluid Mechanics*, Vol. 818, 2017, pp. 528–561.

[7] Giannelis, N. F., Vio, G. A., and Dimitriadis, G., "Dynamic Interactions of a Supercritical Aerofoil in the Presence of Transonic Shock Buffet," *Proceedings of the 27th International Conference on Noise and Vibration Engineering, Leuven, Belgium*, 2016.

[8] Giannelis, N. F., Geoghegan, J. A., and Vio, G. A., "Gust Response of a Supercritical Aerofoil in the Vicinity of Transonic Shock Buffet," *Proceedings of the 20th Australasian Fluid Mechanics Conference, Perth, Western Australia*, 2016.

[9] Giannelis, N. F., Vio, G. A., and Levinski, O., "A review of recent developments in the understanding of transonic shock buffet," *Progress in Aerospace Sciences*, Vol. 92, 2017, pp. 39–84. doi:10.1016/j.paerosci.2017.05.004.

[10] Pearcey, H. H., and Holder, D. W., "Simple methods for the prediction of wing buffeting resulting from bubble type separation," NPL AERO-REP-1024, National Physics Laboratory, 1962.

[11] Lee, B. H. K., "Oscillatory shock motion caused by transonic shock boundary-layer interaction," *AIAA Journal*, Vol. 28, No. 5, 1990, pp. 942–944.

[12] Raghunathan, S., Mitchell, R. D., and Gillan, M. A., "Transonic shock oscillations on NACA0012 aerofoil," *Shock Waves*, Vol. 8, No. 4, 1998, pp. 191–202.

[13] Crouch, J. D., Garbaruk, A., Magidov, D., and Travin, A., "Origin of transonic buffet on aerofoils," *Journal of Fluid Mechanics*, Vol. 628, 2009, pp. 357–369.

[14] Sartor, F., Mettot, C., and Sipp, D., "Stability, Receptivity, and Sensitivity Analyses of Buffeting Transonic Flow over a Profile," *AIAA Journal*, Vol. 53, No. 7, 2014, pp. 1980–1993.

[15] Timme, S., and Thormann, R., "Towards three-dimensional global stability analysis of transonic shock buffet," *Proceedings of the AIAA Atmospheric Flight Mechanics Conference, Washington, D.C.*, 2016, pp. 2016–3848.

[16] Iovnovich, M., and Raveh, D. E., "Numerical study of shock buffet on three-dimensional wings," *AIAA Journal*, Vol. 53, No. 2, 2015, pp. 449–463.

[17] Plante, F., Dandois, J., Beneddine, S., Sipp, D., and Laurendeau, É., "Numerical simulations and global stability analyses of transonic buffet and subsonic stall," *AAAF AERO2019, PARIS, France.*, 2019.

[18] Nitzsche, J., "A numerical study on aerodynamic resonance in transonic separated flow," *Proceedings of the International Forum on Aeroelasticity and Structural Dynamics, Seattle, WA*, 2009.

[19] Arnold, V., *Catastrophe Theory*, Springer-Verlag, 1992.

[20] Denegri, C. M., "Limit cycle oscillation flight test results of a fighter with external stores," *Journal of Aircraft*, Vol. 37, No. 5, 2000, pp. 761–769.

[21] Iovnovich, M., and Raveh, D. E., "Reynolds-averaged Navier-Stokes study of the shock-buffet instability mechanism," *AIAA Journal*, Vol. 50, No. 4, 2012, pp. 880–890.

[22] Bhamidipati, K. K., Reasor, D. A., and Pasiliao, C. L., "Unstructured Grid Simulations of Transonic Shockwave-Boundary Layer Interaction-Induced Oscillations," *Proceedings of the 22nd AIAA Computational Fluid Dynamics Conference, Dallas, TX*, 2015.

[23] Bhamidipati, K. K., Reasor, D. A., Lechniak, J. A., Pasiliao, C. L., and Kielb, R., "Unsteady Reynolds-averaged Navier–Stokes simulation of shock-buffet instability on the NACA 64A204 using unstructured meshes," *Proceedings of USNCCM12-975: Symposium on Advances in Nonlinear Unsteady Aerodynamic Flows, Raleigh, NC*, 2013.

[24] Giannelis, N. F., and Vio, G. A., "On the effect of control surface deflections on the aeroelastic response of an aerofoil at transonic buffet conditions," *Proceedings of the 28th International Conference on Noise and Vibration Engineering, Leuven, Belgium*, September 2018.

[25] Giannelis, N. F., Levinski, O., and Vio, G. A., "Influence of Mach number and angle of attack on the two-dimensional transonic buffet phenomenon," *Aerospace Science and Technology*, Vol. 78, 2018, pp. 89–101.

[26] ANSYS, *Fluent 18.2 Theory Guide*, ANSYS Inc, 2017.

[27] Launder, B. E., Reece Jr, G., and Rodi, W., "Progress in the development of a Reynolds-stress turbulence closure," *Journal of Fluid Mechanics*, Vol. 68, No. 03, 1975, pp. 537–566.

[28] Giannelis, N. F., Murray, A. J., and Vio, G. A., "Influence of control surface deflections on a thin aerofoil at transonic buffet conditions," *Proceedings of the AIAA Scitech 2019 Forum*, AIAA Paper 2019-1339, San Diego, CA, January 2019.

[29] Tijdeman, H., "Investigations of the transonic flow around oscillating airfoils," PhD Thesis, TU Delft, Delft University of Technology, 1977.

BIBLIOGRAPHY

[1] S. Le Borne and W. Leinen, "Guidelines for RBF-FD Discretization: Numerical Experiments on the Interplay of a Multitude of Parameter Choices," *J. Sci. Comput.*, vol. 95, feb 2023.

[2] F. Mazhar, A. Javed, J. T. Xing, A. Shahzad, M. Mansoor, A. Maqsood, S. I. A. Shah, and K. Asim, "On the meshfree particle methods for fluid-structure interaction problems," *Engineering Analysis with Boundary Elements*, vol. 124, pp. 14–40, 2021.

[3] T. C. S. Rendall and C. B. Allen, "Unified fluid-structure interpolation and mesh motion using radial basis functions," *International Journal for Numerical Methods in Engineering*, vol. 74, pp. 1519–1559, jun 2008.

[4] L. Kedward, C. B. Allen, and T. C. Rendall, "Efficient and exact mesh deformation using multiscale RBF interpolation," *Journal of Computational Physics*, vol. 345, pp. 732–751, 2017.

[5] N. Flyer, G. B. Wright, and B. Fornberg, *Radial Basis Function-Generated Finite Differences: A Mesh-Free Method for Computational Geosciences*, pp. 2635–2669. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

[6] A. Murray, N. Giannelis, and G. Vio, "Catastrophe Theoretic Modelling of Hysteresis in Transonic Shock Buffet," in *AIAA Scitech Forum*, 2020.

[7] A. Murray, B. Thornber, M. Flaig, and G. Vio, "Highly Parallel, Multi-stage Mesh Motion using Radial Basis Functions for Fluid-Structure Interaction," in *AIAA Scitech Forum*, 2019.

[8] R. W. Douglass, G. F. Carey, D. R. White, G. A. Hansen, Y. Kallinderis, and N. P. Weatherill, "Current views on grid generation: Summaries of a panel discussion," *Numerical Heat Transfer, Part B: Fundamentals*, vol. 41, no. 3-4, pp. 211–237, 2002.

[9] S. MM and K. RP, "Mesh Deformation Approaches – A Survey," *Journal of Physical Mathematics*, vol. 7, no. 2, 2016.

[10] J. T. Batina, "Using Unstructured Dynamic Meshes," *AIAA Journal*, vol. 28, no. 8, pp. 1381–1388, 1990.

[11] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne, "Torsional springs for two-dimensional dynamic unstructured fluid meshes," *Computer Methods in Applied Mechanics and Engineering*, vol. 163, no. 1-4, pp. 231–245, 1998.

[12] F. J. Blom, "Considerations on the spring analogy," *International Journal for Numerical Methods in Fluids*, vol. 32, no. 6, pp. 647–668, 2000.

[13] D. Zeng and C. R. Ethier, "A semi-torsional spring analogy model for updating unstructured meshes in 3D moving domains," *Finite Elements in Analysis and Design*, vol. 41, no. 11-12, pp. 1118–1139, 2005.

[14] C. L. Bottasso, D. Detomi, and R. Serra, "The ball-vertex method: A new simple spring analogy method for unstructured dynamic meshes," *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 39-41, pp. 4244–4264, 2005.

[15] G. A. Markou, Z. S. Mouroutis, D. C. Charmpis, and M. Papadrakakis, "The ortho-semi-torsional (OST) spring analogy method for 3D mesh moving boundary problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 196, no. 4-6, pp. 747–765, 2007.

[16] Y. Yang, S. Özgen, and H. Kim, "Improvement in the spring analogy mesh deformation method through the cell-center concept," *Aerospace Science and Technology*, vol. 115, p. 106832, 2021.

[17] M. R. Lashkariani and A. R. Firoozjaee, "An improved node moving technique for adaptive analysis using collocated discrete least squares meshless method," *Engineering Analysis with Boundary Elements*, vol. 130, no. May, pp. 322–331, 2021.

[18] A. A. Johnson and T. E. Tezduyar, "Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces," *Computer Methods in Applied Mechanics and Engineering*, vol. 119, no. 1-2, pp. 73–94, 1994.

[19] Z. Yang and D. J. Mavriplis, "Unstructured dynamic meshes with higher-order time integration schemes for the unsteady Navier-Stokes equations," *43rd AIAA Aerospace Sciences Meeting and Exhibit - Meeting Papers*, no. January, pp. 15051–15063, 2005.

[20] E. J. Nielsen and W. K. Anderson, "Recent improvements in aerodynamic design optimization on unstructured meshes," *39th Aerospace Sciences Meeting and Exhibit*, vol. 40, no. 6, 2001.

[21] Z. Yang and D. J. Mavriplis, "Mesh deformation strategy optimized by the adjoint method on unstructured meshes," *AIAA Journal*, vol. 45, no. 12, pp. 2885–2896, 2007.

[22] K. Takizawa, T. E. Tezduyar, and R. Avsar, "A low-distortion mesh moving method based on fiber-reinforced hyperelasticity and optimized zero-stress state," *Computational Mechanics*, vol. 65, no. 6, pp. 1567–1591, 2020.

[23] L. R. Herrmann, "Laplacian-Isoparametric Grid Generation Scheme," *Journal of Engineering Mechanics-asce*, vol. 102, pp. 749–907, 1976.

[24] W. J. Gordon and C. A. Hall, "Construction of curvilinear co-ordinate systems and applications to mesh generation," *International Journal for Numerical Methods in Engineering*, vol. 7, no. 4, pp. 461–477, 1973.

[25] L. Ding, Z. Lu, and T. Guo, "An efficient dynamicmesh generationmethod for complex multi-block structured grid," *Advances in Applied Mathematics and Mechanics*, vol. 6, no. 1, pp. 120–134, 2014.

[26] A. Garon and M. Delfour, *Chapter 6: Explicit Moving Mesh Computations*, pp. 107–131. Society for Industrial and Applied Mathematics, 2007.

[27] P. M. Bartier and C. P. Keller, "Multivariate interpolation to incorporate thematic surface data using inverse distance weighting (IDW)," *Computers and Geosciences*, vol. 22, no. 7, pp. 795–799, 1996.

[28] C. Ware, W. Knight, and D. Wells, "Memory intensive statistical algorithms for multibeam bathymetric data," *Computers and Geosciences*, vol. 17, no. 7, pp. 985–993, 1991.

[29] Y. Zhao and A. Forhad, "A general method for simulation of fluid flows with moving and compliant boundaries on unstructured grids," *Computer Methods in Applied Mechanics and Engineering*, vol. 192, no. 39-40, pp. 4439–4466, 2003.

[30] E. Luke, E. Collins, and E. Blades, "A fast mesh deformation method using explicit interpolation," *Journal of Computational Physics*, vol. 231, no. 2, pp. 586–601, 2012.

[31] X. Liu, N. Qin, and H. Xia, "Fast dynamic grid deformation based on Delaunay graph mapping," *Journal of Computational Physics*, vol. 211, no. 2, pp. 405–423, 2006.

[32] A. de Boer, A. H. van Zuijlen, and H. Bijl, "Review of coupling methods for non-matching meshes," *Computer Methods in Applied Mechanics and Engineering*, vol. 196, no. 8, pp. 1515–1525, 2007.

[33] T. Rendall and C. Allen, "Improved Radial Basis Function Fluid-Structure Coupling via Efficient Localised Implementation," in *38th Fluid Dynamics Conference and Exhibit*, (Reston, Virigina), American Institute of Aeronautics and Astronautics, jun 2008.

[34] T. C. S. Rendall and C. B. Allen, "Efficient mesh motion using radial basis functions with data reduction algorithms," *Journal of Computational Physics*, vol. 228, no. 17, pp. 6231–6249, 2009.

[35] A. Samareh, "Application Deformation of Quaternions for Mesh Deformation," *NASA TM*, no. April, pp. TM–2002–211646, 2002.

[36] D. Maruyama, D. Bailly, and G. Carrier, "High-quality mesh deformation using quaternions for orthogonality preservation," *AIAA Journal*, vol. 52, no. 12, pp. 2712–2729, 2014.

[37] D. R. Mcdaniel and S. A. Morton, "Efficient mesh deformation for computational stability and control analyses on unstructured viscous meshes," *47th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, no. January, pp. 1–21, 2009.

[38] R. Melville, "Nonlinear Simulation of F-16," *39th AIAA Aerospace Sciences Meeting & Exhibit*, 2001.

[39] C. B. Allen, "Parallel universal approach to mesh motion and application to rotors in forward fligh," *International*, no. 69, pp. 2126–2149, 2007.

[40] A. de Boer, M. S. van der Schoot, and H. Bijl, "Mesh deformation based on radial basis function interpolation," *Computers and Structures*, vol. 85, no. 11-14, pp. 784–795, 2007.

[41] M. Hounjet and J. Meijer, "Evaluation of elastomechanical and aerodynamic data transfer methods for non-planar configurations in computational aeroelastic analysis," tech. rep., NLR, 1994.

[42] S. Spekreijse, B. B. Prananta, and J. C. Kok, "A Simple, Robust and Fast Algorithm to Compute Deformations of Multi-Block Structured Grids," in *National Aerospace Laboratory NLR*, 2002.

[43] A. Beckert and H. Wendland, "Multivariate interpolation for fluid-structure-interaction problems using radial basis functions," *Aerospace Science and Technology*, vol. 5, no. 2, pp. 125–134, 2001.

[44] F. Zhang, *The Schur complement and its applications*, vol. 4. Springer, 2005.

[45] N. Flyer, B. Fornberg, V. Bayona, and G. A. Barnett, "On the role of polynomials in RBF-FD approximations: I. Interpolation and accuracy," *Journal of Computational Physics*, vol. 321, pp. 21–38, 2016.

[46] V. Bayona, N. Flyer, B. Fornberg, and G. A. Barnett, "On the role of polynomials in RBF-FD approximations: II. Numerical solution of elliptic PDEs," *Journal of Computational Physics*, vol. 332, no. December, pp. 257–273, 2017.

[47] V. Bayona, N. Flyer, and B. Fornberg, "On the role of polynomials in RBF-FD approximations: III. Behavior near domain boundaries," *Journal of Computational Physics*, vol. 380, pp. 378–399, 2019.

[48] R. L. Harder and R. N. Desmarais, "Interpolation using surface splines.," *Journal of Aircraft*, vol. 9, no. 2, pp. 189–191, 1972.

[49] W. R. Madych and S. A. Nelson, "Multivariate Interpolation and Conditionally Positive Definite Functions," *Approximation Theory and its Applications*, vol. 4, pp. 77–89, 1988.

[50] H. Wendland, *Scattered Data Approximation*. Cambridge: Cambridge University Press, 2004.

[51] H. Wendland, "Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree," *Advances in Computational Mathematics*, vol. 4, no. 1, pp. 389–396, 1995.

[52] A. S. F. Wong, H. M. Tsai, S. P. Drive, J. Cai, Y. Zhu, and F. Liu, "Unsteady Flow Calculations with a Multi-Block Moving Mesh Algorithm," *38th Aerospace Sciences Meeting 81 Exhibit*, no. c, 2000.

[53] D. P. Jones and A. L. Gaitonde, "Moving mesh generation for unsteady flows about deforming complex configurations using multiblock methods," *Japan Society of CFD/CFD Journal*, pp. 430–439, 2001.

[54] R. Craig and M. Bampton, "Coupling of Substructures for Dynamic Analyses To cite this version : HAL Id : hal-01537654 Coupling of Substructures for Dynamic Analyses," *AIAA Journal*, vol. 6, no. 7, pp. 1313–1319, 1968.

[55] C. B. Allen, "Towards automatic structured multiblock mesh generation using improved transfinite interpolation," *International Methods for Numerical Metrhods in Engineering*, no. 74, pp. 697–733, 2008.

[56] J. L. Wagner, K. M. Casper, S. J. Beresh, P. S. Hunter, R. W. Spillers, J. F. Henfling, and R. L. Mayes, "Fluid-structure interactions in compressible cavity flows," *Physics of Fluids*, vol. 27, no. 6, 2015.

[57] B. Thornber and D. Drikakis, "Large-Eddy Simulation of Shock-Wave-Induced Turbulent Mixing," *Journal of Fluids Engineering*, vol. 129, no. 12, pp. 1504–1513, 2007.

[58] Z. A. Rana, B. Thornber, and D. Drikakis, "Dynamics of Sonic Hydrogen Jet Injection and Mixing Inside Scramjet Combustor," *Engineering Applications of Computational Fluid Mechanics*, vol. 7, no. 1, pp. 13–39, 2013.

[59] D. Linton, B. Thornber, and R. Widjaja, *A Study of LES Methods for Simulation of Ship Airwakes*, pp. 1–14. AIAA, 2016.

[60] D. L. Youngs and B. Thornber, "Buoyancy–Drag modelling of bubble and spike distances for single-shock Richtmyer–Meshkov mixing," *Physica D: Nonlinear Phenomena*, vol. 410, p. 132517, 2020.

[61] H. S. Park, D. Linton, and B. Thornber, "Rotorcraft fuselage and ship airwakes simulations using an immersed boundary method," *International Journal of Heat and Fluid Flow*, vol. 93, p. 108916, 2022.

[62] S. Turek and J. Hron, "Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow," *Lecture Notes in Computational Science and Engineering*, vol. 53, pp. 371–385, 2006.

[63] S. Turek, J. Hron, M. Razzaq, H. Wobker, and M. Schäfer, "Numerical Benchmarking of Fluid-Structure Interaction: A Comparison of Different Discretization and Solution Approaches," *Fluid Structure Interaction II*, vol. 73, 2010.

[64] J.-S. Chen, M. Hillman, and S.-W. Chi, "Meshfree Methods: Progress Made after 20 Years," *Journal of Engineering Mechanics*, vol. 143, no. 4, 2017.

[65] L. B. Lucy, "A numerical approach to the testing of the fission hypothesis.," *Astron. J.*, vol. 82, pp. 1013–1024, dec 1977.

[66] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars," *Monthly Notices of the Royal Astronomical Society*, vol. 181, no. 3, pp. 375–389, 1977.

[67] L. D. Libersky and A. G. Petschek, "Smooth particle hydrodynamics with strength of materials," in *Advances in the Free-Lagrange Method Including Contributions on Adaptive Gridding and the Smooth Particle Hydrodynamics Method* (H. E. Trease, M. F. Fritts, and W. P. Crowley, eds.), (Berlin, Heidelberg), pp. 248–257, Springer Berlin Heidelberg, 1991.

[68] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner, "SPH Fluids in Computer Graphics," in *Eurographics 2014 - State of the Art Reports* (S. Lefebvre and M. Spagnuolo, eds.), The Eurographics Association, 2014.

[69] M. S. Shadloo, G. Oger, and D. Le Touzé, "Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, Current state, And challenges," *Computers and Fluids*, vol. 136, pp. 11–34, 2016.

[70] P. S. Jensen, "Finite difference techniques for variable grids," *Computers & Structures*, vol. 2, no. 1, pp. 17–29, 1972.

[71] B. Nayroles, G. Touzot, and P. Villon, "Generalizing the finite element method: Diffuse approximation and diffuse elements," *Computational Mechanics*, vol. 10, pp. 307–318, sep 1992.

[72] T. Belytschko, Y. Y. Lu, and L. Gu, "Element-Free Galerkin Methods," *International Journal for Numerical Methods in Engineering*, vol. 37, pp. 229–256, 1994.

[73] L. Wing Kam, J. Sukky, and Z. Yi Fei, "Reproducing kernel particle methods," *International Journal for Numerical Methods in Fluids*, vol. 20, no. 8-9, pp. 1081–1106, 1995.

[74] C. Franke and R. Schaback, "Solving partial differential equations by collocation using radial basis functions," *Applied Mathematics and Computation*, vol. 93, no. 1, pp. 73–82, 1998.

[75] R. L. Hardy, "Multiquadric equations of topography and other irregular surfaces," *Journal of Geophysical Research*, vol. 76, no. 8, pp. 1905–1915, 1971.

[76] E. J. Kansa, "Multiquadrics-A scattered data approximation scheme with applications to computational fluid-dynamics-I surface approximations and partial derivative estimates," *Computers and Mathematics with Applications*, vol. 19, no. 8-9, pp. 127–145, 1990.

[77] A. S. M. Wong, Y. C. Hon, T. S. Li, S. L. Chung, and E. J. Kansa, "Multizone decomposition for simulation of time-dependent problems using the multiquadric scheme," *Computers & Mathematics with Applications*, vol. 37, no. 8, pp. 23–43, 1999.

[78] E. J. Kansa and Y. C. Hon, "Circumventing the ill-conditioning problem with multiquadric radial basis functions: Applications to elliptic partial differential equations," *Computers & Mathematics with Applications*, vol. 39, no. 7, pp. 123–137, 2000.

[79] Y. C. Hon and R. Schaback, "On unsymmetric collocation by radial basis functions," *Applied Mathematics and Computation*, vol. 119, no. 2, pp. 177–186, 2001.

[80] N. Flyer, G. A. Barnett, and L. J. Wicker, "Enhancing finite differences with radial basis functions: Experiments on the Navier-Stokes equations," *Journal of Computational Physics*, vol. 316, pp. 39–62, 2016.

[81] A. Javed, K. Djidjeli, J. T. Xing, and S. Cox, "A Hybrid Mesh Free Local RBF- Cartesian FD Scheme for Incompressible Flow around Solid Bodies," *International Journal of Computational Engineering*, vol. 7, no. 10, pp. 60–70, 2013.

[82] W. R. Madych and S. A. Nelson, "Multivariate Interpolation and Conditionally Positive Definite Functions II," *Mathematics of Computation*, vol. 54, no. 189, pp. 211–230, 1990.

[83] Z.-m. Wu and R. Schaback, "Local error estimates for radial basis function interpolation of scattered data," *IMA Journal of Numerical Analysis*, vol. 0, pp. 13–27, 1993.

[84] B. Fornberg, G. Wright, and E. Larsson, "Some Observations Regarding Interpolants in the Limit of Flat Radial Basis Functions," *Computers and Mathematics with Applications*, vol. 47, no. 1, pp. 37–55, 2004.

[85] M. E. Biancolini, A. Chiappa, U. Cella, E. Costa, C. Groth, and S. Porziani, "Radial Basis Functions Mesh Morphing," in *Computational Science – ICCS 2020* (V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, eds.), (Cham), pp. 294–308, Springer International Publishing, 2020.

[86] B. Fornberg, E. Lehto, and C. Powell, "Stable calculation of Gaussian-based RBF-FD stencils," *Computers & Mathematics with Applications*, vol. 65, no. 4, pp. 627–637, 2013.

[87] J. Duchon, "Splines minimizing rotation-invariant semi-norms in Sobolev spaces," in *Constructive Theory of Functions of Several Variables: Proceedings of a Conference Held at Oberwolfach April 25 – May 1, 1976* (W. Schempp and K. Zeller, eds.), pp. 85–100, Berlin, Heidelberg: Springer Berlin Heidelberg, 1977.

[88] J. Fromm, "The time dependent flow of an incompressible viscous fluid," *Methods in Computational Physics*, vol. 3, pp. 345–382, 1964.

[89] L. Carnevale, G. Anjos, and N. Mangiavacchi, "Stream Function-Vorticity Formulation Applied in the Conjugated Heat Problem Using the FEM With Unstructured Mesh," in *Brazilian Congress of Thermal Sciences and Engineering*, 2018.

[90] A. S. Thom, "The flow past circular cylinders at low speeds," *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 141, pp. 651–669, 1933.

[91] V. G. Jenson, "Viscous flow round a sphere at low Reynolds numbers (<40)," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 249, pp. 346–366, 1959.

[92] P. J. Roache, *Computational Fluid Dynamics*. Lecture Series, Hermosa Publishers, 1976.

[93] H. Huang and B. R. Wetton, "Discrete compatibility in finite difference methods for viscous incompressible fluid flow," *Journal of Computational Physics*, vol. 126, no. 2, pp. 468–478, 1996.

[94] P. P. Chinchapatnam, K. Djidjeli, P. B. Nair, and M. Tan, "A compact RBF-FD based meshless method for the incompressible Navier-Stokes equations," *Proceedings of the Institution of Mechanical Engineers Part M: Journal of Engineering for the Maritime Environment*, vol. 223, no. 3, pp. 275–290, 2009.

[95] C. Fletcher and K. Srinivas, "Stream Function Vorticity Revisited," *Computer Methods in Applied Mechanics and Engineering*, 1983.

[96] H. J. Lugt and E. W. Schwiderski, "II. Analysis of regular and singular motions," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 285, no. 1402, pp. 400–412, 1965.

[97] B. Fornberg and E. Lehto, "Stabilization of RBF-generated finite difference methods for convective PDEs," *Journal of Computational Physics*, vol. 230, no. 6, pp. 2270–2285, 2011.

[98] U. Ghia, K. N. Ghia, and C. T. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," *Journal of Computational Physics*, vol. 48, no. 3, pp. 387–411, 1982.

[99] H. V. R. Vikram C. K, Y. T. Krishne Gowda, "Numerical investigation on flow past square cylinders with different corner shapes," *International Journal of Scientific & Engineering Research*, vol. 5, no. 10, pp. 479–486, 2014.

[100] Y. G. Zhitian Zhang, Xianxiong Zhang, "Motion-induced vortex shedding and lock-in phenomena of a rectangular section," *Nonlinear Dynamics*, vol. 102, pp. 2267–2280, 2020.

[101] A. K. Sahu, R. P. Chhabra, and V. Eswaran, "Two-dimensional laminar flow of a power-law fluid across a confined square cylinder," *Journal of Non-Newtonian Fluid Mechanics*, vol. 165, no. 13-14, pp. 752–763, 2010.

[102] B. Tóth and A. Düster, "h-Adaptive radial basis function finite difference method for linear elasticity problems," *Computational Mechanics*, vol. 71, pp. 1–20, 2022.

[103] J. Li, S. Zhai, Z. Weng, and X. Feng, "H-adaptive RBF-FD method for the high-dimensional convection-diffusion equation," *International Communications in Heat and Mass Transfer*, vol. 89, pp. 139–146, 2017.

[104] D. T. Oanh and N. M. Tuong, "An approach to adaptive refinement for the RBF-FD method for 2D elliptic equations," *Applied Numerical Mathematics*, vol. 178, pp. 123–154, 2022.

[105] K. Y. Lam, Q. X. Wang, and Z. Zong, "A Nonlinear Fluid-Structure Interaction Analysis of a Near-bed Submarine Pipeline in a Current," *Journal of Fluids and Structures*, vol. 16, no. 8, pp. 1177–1191, 2002.

[106] S. J. Ang, K. S. Yeo, C. S. Chew, and C. Shu, "A singular-value decomposition (SVD)-based generalized finite difference (GFD) method for close-interaction moving boundary flow problems," *International Journal for Numerical Methods in Engineering*, vol. 76, no. 12, pp. 1892–1929, 2008.

[107] A. Javed, *Investigation on meshfree particle methods for fluid structure interaction problems*. PhD thesis, University of Southampton, sep 2015.

[108] A. Javed, K. Djijdeli, and J. T. Xing, "A coupled meshfree-mesh-based solution scheme on hybrid grid for flow-induced vibrations," *Acta Mechanica*, vol. 227, 2016.

[109] M. Jamil, A. Javed, S. Shah, M. Mansoor, A. Hameed, and K. Djidjeli, "Performance Analysis of Flapping Foil Flow Energy Harvester Mounted on Piezoelectric Transducer using Meshfree Particle Method," *Journal of Applied Fluid Mechanics*, vol. 13, pp. 1759–1872, 2020.

[110] K. Han, Y. T. Feng, and D. R. J. Owen, "Numerical Simulations of Irregular Particle Transport in Turbulent Flows Using Coupled LBM-DEM," *Computer Modeling in Engineering & Sciences*, vol. 18, no. 2, pp. 87–100, 2007.

[111] A.-m. Zhang, P.-n. Sun, F.-r. Ming, and A. Colagrossi, "Smoothed particle hydrodynamics and its applications in fluid-structure interactions," *Journal of Hydrodynamics, Ser. B*, vol. 29, no. 2, pp. 187–216, 2017.

[112] R. Bhatt and M. M. Alam, "Vibrations of a square cylinder submerged in a wake," *Journal of Fluid Mechanics*, vol. 853, 2018.

[113] I. Tominec and M. Nazarov, "Residual Viscosity Stabilized RBF-FD Methods for Solving Nonlinear Conservation Laws," *International Journal of Heat and Fluid Flow*, vol. 94, 2022.