Reduced Order Modeling Techniques for Mesh Movement Strategies as Applied to Fluid Structure Interactions

Alfred E.J. Bogaers

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Reduced order modeling techniques for mesh movement strategies as applied to fluid structure interactions

by

Alfred E.J. Bogaers

A dissertation submitted in partial fulfillment
of the requirements for the degree

Master of Engineering

in the

Faculty of Engineering, the Built Environment and Information Technology

University of Pretoria
Pretoria

April 2010

Abstract

Title: Reduced order modeling techniques for mesh movement strategies as applied to fluid structure interactions

Author: Alfred Edward Jules Bogaers

Supervisors: Dr. S. Kok
Dr. A.G. Malan

Department: Department of Mechanical and Aeronautical Engineering

Degree: Master of Engineering

Keywords: Unstructured mesh movement, radial basis function interpolation, mesh optimization, elastic deformation, reduced order modeling, proper orthogonal decomposition

In this thesis, the method of Proper Orthogonal Decomposition (POD) is implemented to construct approximate, reduced order models (ROM) of mesh movement methods. Three mesh movement algorithms are implemented and comparatively evaluated, namely radial basis function interpolation, mesh optimization and elastic deformation. POD models of the mesh movement algorithms are constructed using a series of system observations, or snapshots of a given mesh for a set of boundary deformations. The scalar expansion coefficients for the POD basis modes are computed in three different ways, through coefficient optimization, Galerkin projection of the governing set of equations and coefficient interpolation. It is found that using only coefficient interpolation yields mesh movement models that accurately approximates the full order mesh movement, with CPU cost savings in excess of 99%.

We further introduce a novel training procedure whereby the POD models are generated in a fully automated fashion. The technology is applicable to any mesh movement

method and enables potential reductions of up to four orders of magnitude in mesh movement related costs. The proposed model can be implemented without having to pre-train the POD model, to any fluid-structure interaction code with an existing mesh movement scheme.

Acknowledgments

I would like to thank my supervisors, Schalk Kok and Arnuad Malan, for their continued advice and guidance throughout the project. Special mention need also be made to *Project Fluxion*, for providing financial assistance.

Contents

Nomenclature

$\boldsymbol{a}$	POD expansion coefficients, eigenvectors of modified correlation tensor
C	Fourth order elasticity tensor
d	Boundary displacement
D	Arbitrary linear differential operator
f	Arbitrary function, or externally applied force
f_{ss}	Shape-size quality metric, ranges between 0 and 1 for degenerate and perfect elements respectively
$\boldsymbol{f}$	Objective function to be minimized, or displacement vector
J^e	Jacobian of element
k	Number of retained modes
$\boldsymbol{K}$	Stiffness matrix
M	Number of snapshots
$M_{b,b}$	Matrix containing evaluations of radial basis functions
$\boldsymbol{n}$	Outward pointing unit normal
n_b	Number of boundary nodes
N	System DOF, number of nodal co-ordinates within mesh, or Finite element displacement shape functions
N_2	Quadratic differential operator
N_3	Cubic differential operator
p	Linear polynomial
P_b	Matrix containing boundary co-ordinates in the form $[1\, x_{b_i}\, y_{b_i}\, z_{b_i}]$
$\boldsymbol{u}$	Set of system observations or snapshots, or displacement function, or displacement vector
$\boldsymbol{U}$	Observation matrix where each row vector is a system snapshot
r	Radial basis function support radius
R	Residual
$\mathcal{R}$	Modified correlation matrix, obtained via method of snapshots
$\mathbf{R}$	Autocorrelation tensor
s	RBF interpolation function
t	Time

W	Weighting function
$\boldsymbol{x}$	Nodal co-ordinates of a mesh
$\boldsymbol{x}_{bi}$	Boundary nodal co-ordinates
$\boldsymbol{X}$	Observation matrix, where each row vector is a snapshot containing mesh nodal co-ordinates

Greek symbols

α	Area of triangle
$\boldsymbol{\alpha}$	POD expansion coefficients
$\boldsymbol{\beta}$	Coefficients of linear polynomial p
$\boldsymbol{\varepsilon}$	Strain tensor, or convergence criteria
λ	Eigenvalues
ξ	Compact radial basis functions scaling factor, or local co-ordinates in the finite element formulation
$\boldsymbol{\sigma}$	Stress tensor
ϕ	Radial basis function
$\boldsymbol{\varphi}_j$	j-th POD basis mode
Ω	Spatial domain
$\partial\Omega$	Bounding surface to domain Ω
χ	Stiffening constant
Ξ	Element local domain

Mathematical operators

$\langle . \rangle$	Averaging operator
$(.,.)$	Inner product of two functions over a predefined interval
$\|.\|$	Modulus
$\|\|.\|\|$	L^2 -norm, or Euclidean distance
∂	Partial derivative
δ	Dirac delta functions

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The numerical simulation of flow across a boundary arises in many engineering related problems, e.g. flutter simulations of wings, blood flow through veins and arteries and parachute dynamics. These, and other fluid-structure interaction problems involve flow induced moving boundaries, and in order to accurately complete these unsteady flow simulations it becomes necessary for the computational grid to conform to the new displaced domain.

In ensuring a boundary conforming mesh, the obvious choice would be to simply re-generate the mesh at each time step. To regenerate the mesh of a complex geometry requires the use of an automatic mesh generator which is an expensive task, especially for three-dimensional problems. The cost can to a certain extent be mitigated by restoring grid quality through localized grid coarsening and refinement [5]. By altering or regenerating a mesh the grid connectivity information changes, and it requires that the solution be projected from the old mesh to the new one. Not only is this a computationally expensive process, but may lead to conservation issues arising within the fluid domain [41]. As such, a method or algorithm is desired to displace the given mesh to conform to a dynamically moving boundary in a computationally efficient manner.

Several algorithms have been developed that can effectively adapt unstructured meshes to the new displaced boundary without changing the grid connectivity. This is referred to as mesh movement, examples of which include the spring analogy [12], solving a set of Laplacian or Bi-harmonic equations [14], radial basis function (RBF) interpolation [9, 35] or through mesh optimization [4, 11]. Despite the successes of these algorithms in reducing the frequency and necessity for re-meshing, they may still account for a significant percentage of CPU time for a FSI simulation involving large complex boundary deformations [34].

The aim of this thesis is to address the computational cost deficiency of typical mesh movement strategies via the use of effective reduced order models. This is to be done

such that the quality of the resulting meshes remains comparable to the original mesh movement methods. The reduced order modeling technique that we make use of is the method of Proper Orthogonal Decomposition (POD), also commonly referred to as Principle Component Analysis (PCA), Singular Value Decomposition (SVD) or Karhunnen-Loéve (KL) decomposition.

1.2 Scope of Work

The research activities conducted for the thesis are:

- Researching the fundamentals and application of POD, and its application to ROM, as well as the various mesh movement methodologies.
- Implementation and rigorous comparison of three mesh movement strategies for FSI modeling purposes: Radial basis function interpolation, mesh optimization and solving a set of linear elasticity equations.
- Investigating for the first time the application of POD towards the generation of ROMs of mesh movement algorithms.
- A novel training procedure is developed, where POD is used to automatically facilitate reductions in mesh movement related computational cost of first time FSI simulations. The mesh movement model is generated without having to pre-train the POD model.

1.3 Outline of Thesis

The thesis is divided into seven chapters, including an introduction and conclusion. Below is a brief outline of each of the chapters:

- *Chapter one: Introduction.* Provides background to the work presented in the thesis and provides an overall outline.
- *Chapter two: Proper Orthogonal Decomposition.* This chapter provides an overall introduction into the method of POD. The derivation and fundamental principles are outlined along with the "method of snapshots" for the calculation of POD basis modes. The chapter concludes with an example of a simple mesh movement problem.
- *Chapter three: System Approximations and Reduced Order Modelling Using POD.* Chapter three introduces various methods of weighted residuals (MWR). The MWRs are introduced specifically with the aim of projecting a set of governing

partial differential equations onto POD basis modes. In so doing it allows for the solution of the original problem in a transformed subspace with a significantly reduced problem size.

- *Chapter four: Unstructured Mesh Movement.* Description, detailed investigation and comparative evaluation of three mesh movement methods. Each method is compared to one another in terms of both computational efficiency and quality of final meshes produced.

- *Chapter five: POD based ROM for Mesh Movement.* Documents the application of the method of POD to build ROMs of the mesh movement methods. Different methods for equating the POD models are investigated, as well as the quality of the approximations and computational efficiencies.

- *Chapter six: Mesh Movement ROM Applied to a Benchmark FSI problem and Adaptive Model Training.* In this chapter, a ROM is applied to effect mesh movement of a widely used FSI benchmark problem. We further introduce a novel training procedure whereby the method of POD is used to effect reductions in mesh movement computational costs for first time FSI simulations, in a fully automated and robust manner. The method is a fully automatic wrapper, that can be applied to any existing mesh movement method resulting in competitive mesh qualities, for large unstructured grids and orders of magnitude reductions in computational times.

- *Chapter seven: Conclusion.* The work presented in the thesis is summarized briefly.

Chapter 2

Proper Orthogonal Decomposition

2.1 Introduction

Proper Orthogonal Decomposition (POD) is a mathematical procedure aimed at finding low-dimensional approximate descriptions of high-dimensional systems. POD is in essence an empirical spectral method, similar to Fourier decomposition, where field variables are approximated using expansions of a set of projected basis functions or modes. POD obtains these basis functions from a set of observations, where these observations can be obtained either experimentally or through numerical simulations of a real system. What makes POD remarkable is that the selected modes are not only appropriate but make up the optimal linear basis for describing any given system.

POD, also commonly referred to in literature as Karhunnen-Loéve decompositions, Principal Component Analysis (PCA) or Singular Value Decomposition (SVD), was first used in the 1940s, by Karhunnen and Loéve for continuous systems [21, 26]. Since its initial use, POD has been applied in a wide range of disciplines including image processing, data compression, control in chemical engineering and oceanography. The first use of POD in the field of Fluid Mechanics was by Lumley [27] as a post processing step for determining coherent structures within turbulent flow. Since then POD has been applied successfully to many engineering problems including the characterization of dominant turbulent flow properties [15, 31], aircraft flutter prediction [25] and reduced order models for multidisciplinary optimization [7, 23].

The rest of the chapter is dedicated to an overview of the underlying theory regarding PODs, followed by a short example demonstrating the application thereof. For an easy to understand introduction into POD refer to [6] and for a detailed and comprehensive derivation and discussion on PODs refer to Holmes et al. [15].

2.2 Theoretical Background

The basic idea of POD is relatively simple. Suppose we have a set of discrete observations, $\{\boldsymbol{u}^k\}$ for $k = 1, 2, ..., M$, as a function of x. These observations, or snapshots, may be obtained either experimentally or through numerical simulation, describing for instance the flow velocity through a domain. The observations may be for various instances of time, varying flow domain geometries or for varying flow properties. In the case of mesh movement, these observations are the nodal coordinates for particular domain boundaries. It is possible to approximate these observations by decomposing the system into a linear combination of basis functions $\{\boldsymbol{\varphi}_j\}$ such that

$$\boldsymbol{u}^k \approx \sum_{j=1}^{M} \alpha_j^k \boldsymbol{\varphi}_j(x) \tag{2.1}$$

where $\boldsymbol{\alpha}^k$ is an appropriate set of scalar expansion coefficients, relating to the k$^{\text{th}}$ snapshot, $\boldsymbol{u}^k$. It can be shown that the observations $\boldsymbol{u}^k$ can be exactly reproduced if M tends to infinity [16].

There are various approximation techniques by which to compute the basis functions $\{\boldsymbol{\varphi}_j\}$, such as Fourier series decomposition, or Legendre and Chebyshev polynomials. The distinction between POD and the various methods, is that POD provides the *optimal linear* basis functions to describe a set of observations.

The notation we have adopted is that $\{.\}$ denotes a collection of a set of vectors. $\boldsymbol{u}^k$ represents the k$^{\text{th}}$ snapshot vector of size N, where N is the total degrees of freedom (DOF) of a system. Furthermore, $\boldsymbol{u}$ denotes a continuous function describing the system observations or in the discrete form a $N \times M$ matrix, where each column vector is a snapshot.

2.2.1 Autocorrelation Matrix of a Finite Dimensional POD

In equation (2.1), it was stated that a set of observations, $\{\boldsymbol{u}^k\}$ can be decomposed linearly in terms of $\{\boldsymbol{\varphi}_j\}$. These POD modes are chosen in such a way, that regardless of the number of retained modes M, the approximation in equation (2.1) is as good as possible in a least square sense. This *optimality* of the POD modes is found by maximizing the average projection of the ensemble set of observations $\{\boldsymbol{u}^k\}$ onto the normalized basis modes $\boldsymbol{\varphi}$

$$\max_{\varphi \in H} \frac{\left\langle |(\boldsymbol{u}, \boldsymbol{\varphi})|^2 \right\rangle}{||\boldsymbol{\varphi}||^2}, \tag{2.2}$$

where we assume that all observations $\{\boldsymbol{u}^k\}$ form part of an infinite-dimensional Hillbert space H, with an associated inner product. The notation used for an inner product of

two functions over a predefined interval is $(\cdot,\cdot)$, and $\langle\cdot\rangle$ denotes an averaging operator, which can be an ensemble average over many experimental realizations or simply be a time-average of several samples of a single experiment. $|\cdot|$ denotes the modulus and $||\cdot||$ the L^2-norm generated by the inner product, defined as

$$||f|| = (f, f)^{1/2}. \tag{2.3}$$

For the mesh movement problem, our system observations $\{\boldsymbol{u}^k\}$, are not infinite dimensional functions, but rather a set of discrete vectors containing nodal coordinate information. We therefore limit the rest of our discussion of the POD method to the finite dimensional case.

Solving the maximization problem (2.2), with the constraint that $||\boldsymbol{\varphi}||^2 = 1$, it can be shown [15] that the optimal POD modes are the eigenvectors of the autocorrelation function, $\mathbf{R}$, which for the finite dimensional case is an $N \times N$ tensor given by

$$\mathbf{R} = \langle \boldsymbol{u} \otimes \boldsymbol{u} \rangle, \tag{2.4}$$

where N is the total degrees of freedom (DOF) of the system and $\otimes$ is the tensor product.

The POD basis functions to the maximization problem stated in (2.2) is the eigenvectors $\boldsymbol{\varphi}$ of $\mathbf{R}$ i.e.

$$\mathbf{R}\boldsymbol{\varphi} = \lambda\boldsymbol{\varphi}. \tag{2.5}$$

The exact definition of the eigenvalues, λ, in (2.5) depends specifically on the problem being solved. For example, when applied to incompressible fluid flow λ is a measure of twice the kinetic energy pertaining to the flow. If the POD modes are derived in terms of the H^1 Sobolov norm rather than L^2 norm, then λ would be a measure of the dissipation $\left(||\nabla\boldsymbol{u}||^2\right)$ or vorticity $(\nabla \times \boldsymbol{u})$ [16].

For the mesh movement problem, λ is the square of the magnitude of displacements captured by each of the POD modes. In other words, the POD basis mode associated with the highest ranked eigenvalue contains the dominant system behavior. By ordering the eigenvalues and associated eigenvectors from largest to smallest, one can significantly reduce the order of the POD model by retaining only the K^{th} most 'energetic' modes. The observations $\{\boldsymbol{u}^k\}$ can then be successfully decomposed as a lower-dimensional model i.e.

$$\boldsymbol{u}^k \approx \sum_{j=1}^{K} \alpha_j^k \boldsymbol{\varphi}_j. \tag{2.6}$$

2.2.2 Method of Snapshots

In the previous section, it was shown that with a set of observations an autocorrelation matrix could be set up from which the POD modes can be computed. The computation involves solving for the eigenvectors for this autocorrelation matrix, $\mathbf{R}$. The observations, $\{\boldsymbol{u}^k\}$, for mesh movement, are the coordinates of the discrete grid points of the computational mesh. To save all the snapshot data used in the construction of $\mathbf{R}$ requires a $N \times M$ matrix, where N is the total degrees of freedom (DOF) of the system and M is the number of snapshots deemed reasonable to accurately describe the system of interest; each snapshot is for a different domain boundary movement. To obtain the POD modes therefore requires the solution of a $N \times N$ eigenvalue/vector problem. In most engineering problems, the size of N is large, often making the associated cost in computing these eigenvectors prohibitively expensive.

As an example, the computational meshes for CFD simulations of realistic engineering problems can easily have millions of nodal grid points. Solving for the eigenvectors of a million by million matrix is unrealistic. The cost can, to a certain extent, be mitigated by using iterative solvers. Fortunately, Sirovich [37] came up with the "method of snapshots", an elegant procedure which reduces the eigenvalue problem from $N \times N$ to $M \times M$, where $M \ll N$. There is unfortunately no rigorous procedure by which to determine the minimum number of snapshots to be used, though it can safely be stated that M is several orders of magnitude smaller than N, in the majority of cases.

Previously we saw that the observations $\{\boldsymbol{u}^k\}$ can be decomposed as a linear combination of the basis modes $\boldsymbol{\varphi}$. Conversely, $\boldsymbol{\varphi}$ is an eigenvector, that from the span of the basis mode may be expressed as a linear combination of the observed snapshots [15]

$$\boldsymbol{\varphi} = \sum_{k=1}^{M} \boldsymbol{a}_k \boldsymbol{u}^k, \tag{2.7}$$

where $\boldsymbol{a}_k$ are coefficients still to be determined. Assuming that the averaging operator of a discrete function can be defined as

$$\langle f \rangle = \frac{1}{M} \sum_{k=1}^{M} f^k \tag{2.8}$$

and substituting the modal decomposition (2.7) into (2.5) one obtains

$$\frac{1}{M} \sum_{i=1}^{M} \boldsymbol{u}^i \left(\boldsymbol{u}^i, \sum_{i=1}^{M} \boldsymbol{a}_k \boldsymbol{u}^k \right) = \lambda \sum_{k=1}^{M} \boldsymbol{a}_k \boldsymbol{u}^k. \tag{2.9}$$

Rearranging the left hand side of equation (2.9), by using the property $(x + y, z) = (x, z) + (y, z)$, gives

$$\frac{1}{M}\sum_{i=1}^{M}\boldsymbol{u}^i\left(\boldsymbol{u}^i,\sum_{i=1}^{M}\boldsymbol{a}_k\boldsymbol{u}^k\right)=\frac{1}{M}\sum_{i=1}^{M}\boldsymbol{u}^i\left[\sum_{k=1}^{M}\boldsymbol{a}_k\left(\boldsymbol{u}^i,\boldsymbol{u}^k\right)\right] \tag{2.10}$$

$$=\sum_{i=1}^{M}\left[\sum_{k=1}^{M}\frac{1}{M}\left(\boldsymbol{u}^i,\boldsymbol{u}^k\right)\boldsymbol{a}_k\right]\boldsymbol{u}^i. \tag{2.11}$$

Therefore a sufficient condition for the solution of (2.5) is

$$\sum_{k=1}^{M}\frac{1}{M}\left(\boldsymbol{u}^i,\boldsymbol{u}^k\right)\boldsymbol{a}_k=\lambda\boldsymbol{a}_i\quad\text{for}\,i=1,2,...,M. \tag{2.12}$$

Using the method of snapshots, we now have a new modified autocorrelation matrix of the form

$$\boldsymbol{\mathcal{R}}=\frac{1}{M}\left(\boldsymbol{u}^i,\boldsymbol{u}^j\right), \tag{2.13}$$

where $\boldsymbol{u}^i$ represents the i^{th} snapshot of the solution. Since we are dealing with discrete points, the modified $M\times M$ autocorrelation matrix is equal to

$$\boldsymbol{\mathcal{R}}=\frac{1}{M}\boldsymbol{U}\boldsymbol{U}^T, \tag{2.14}$$

where $\boldsymbol{U}$ is the $M\times N$ observation matrix where each row vector of the matrix represents a snapshot. The eigenvectors $\boldsymbol{a}$ of $\boldsymbol{\mathcal{R}}$ are now computed as an intermediate step to determining the basis modes, i.e.

$$\boldsymbol{\mathcal{R}}\boldsymbol{a}=\lambda\boldsymbol{a}. \tag{2.15}$$

The POD modes can then be calculated as

$$\boldsymbol{\varphi}^k=\sum_{i=1}^{M}a_i^k\boldsymbol{u}^i\ \ \text{for}\,k=1,2,...,M \tag{2.16}$$

where a_i^k is the i^{th} element of eigenvector $\boldsymbol{a}$ corresponding to λ_k. It should be noted, that the eigenvectors $\boldsymbol{a}$ computed above, are the same expansion coefficients needed in equation (2.1), if the observations or snapshots $\left\{\boldsymbol{u}^k\right\}$ are to be reproduced. In Chapter 3 it will be shown however that these coefficients need to be re-computed if a reduced order model of a given system is desired at different conditions to the original snapshot matrix.

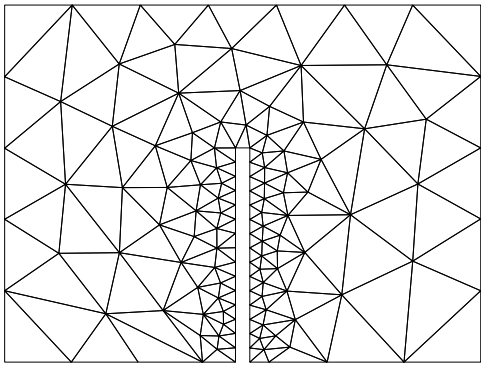


Figure 2.1: The mesh used for the example of a flexible beam. The mesh is generated using NETGEN [36].

2.3 Example of POD application

Let us consider a simple mesh movement problem of a slender 2D beam subjected to a bending moment, presented in Figure 2.1. The mesh contains a total of 81 grid points. We would like to generate a lower dimensional model of the mesh movement as the beam displaces. We generate 5 snapshots of the mesh nodal co-ordinates for the beam in various degrees of pure bending, illustrated in Figure 2.2. The mesh movement is performed using the method of radial basis function interpolation, discussed in detail in Section 4.2.

Here follows a summary of the required procedure to generate a lower-dimensional model using PODs:

1. The observations are gathered and assembled into a $M \times N$ snapshot matrix $\boldsymbol{U}$, where M is the number of snapshots (5 for this example) and N the number of interior degrees of freedom, the x and y coordinates (162 for this example).

2. The covariance matrix is computed as

$$\boldsymbol{\mathcal{R}} = \frac{1}{M}\boldsymbol{U}\boldsymbol{U}^T.$$

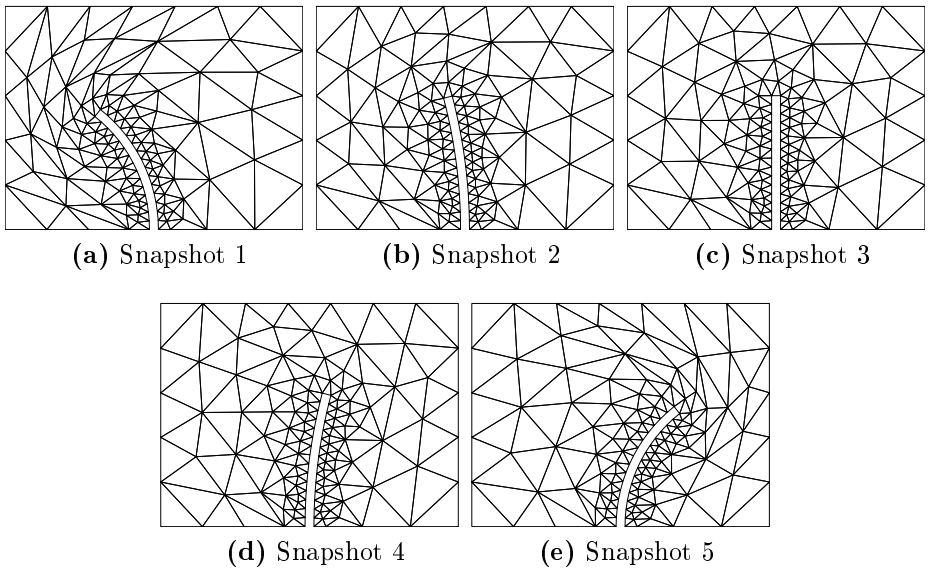
3. The eigenvalues/eigenvectors are computed from

$$\boldsymbol{\mathcal{R}}\boldsymbol{a} = \lambda\boldsymbol{a}.$$

4. The eigenvectors $\boldsymbol{a}$ are sorted in order of descending magnitude of the associated eigenvalues λ.

5. The K^{th} most dominant POD modes are computed

$$\boldsymbol{\varphi}^k = \sum_{i=1}^{M} a_i^k \boldsymbol{u}^i \quad k = 1, 2, ..., M$$



(a) Snapshot 1 **(b)** Snapshot 2 **(c)** Snapshot 3

(d) Snapshot 4 **(e)** Snapshot 5

Figure 2.2: Snapshots of a slender beam in pure bending.

where a_i^k is the i^{th} element of vector $\boldsymbol{a}^k$ corresponding to λ_k, and $\boldsymbol{u}^i$ is the i^{th} snapshot vector of $\boldsymbol{U}$.

6. The POD modes are normalized to unit length.

7. Finally, the mesh nodal coordinates $\boldsymbol{u}$ may be approximated through a linear combination of the K$^{\text{th}}$ most dominant POD modes

$$\boldsymbol{u} = \sum_{i=1}^{K} \alpha_i^k \boldsymbol{\varphi}_i^k.$$

It should be noted that if the expansion coefficients α_i^k are chosen as $\boldsymbol{a}^k$, then $\boldsymbol{u}$ would be an approximation of the k^{th} snapshot.

2.3.1 Results of surface approximation

Applying the POD procedure outlined above, the POD modes for the mesh movement were computed and ordered in order of descending eigenvalue magnitudes. A plot of the ordered eigenvalues is presented in Figure 2.3, where the magnitudes are 7.9×10^4, 2.5×10^3, 2.0, 1.7×10^{-2} and 3.5×10^{-12}. Since the eigenvalues provide an indication as to how much of the system information is captured by the associated POD modes, these values suggest that the first two modes contain over 99% of the system information. We approximate the error E of retaining only the first K most dominant POD modes by

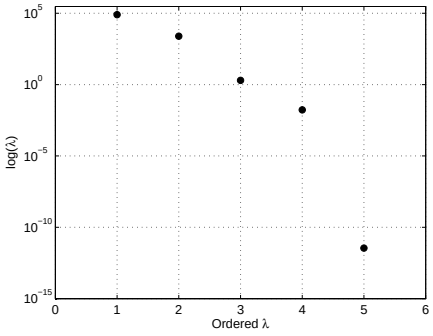


Figure 2.3: Ordered eigenvalues for slender beam mesh movement.

$$E = 1 - \frac{\sum_{j=1}^{K} \lambda_j}{\sum_{i=1}^{M} \lambda_i}. \tag{2.17}$$

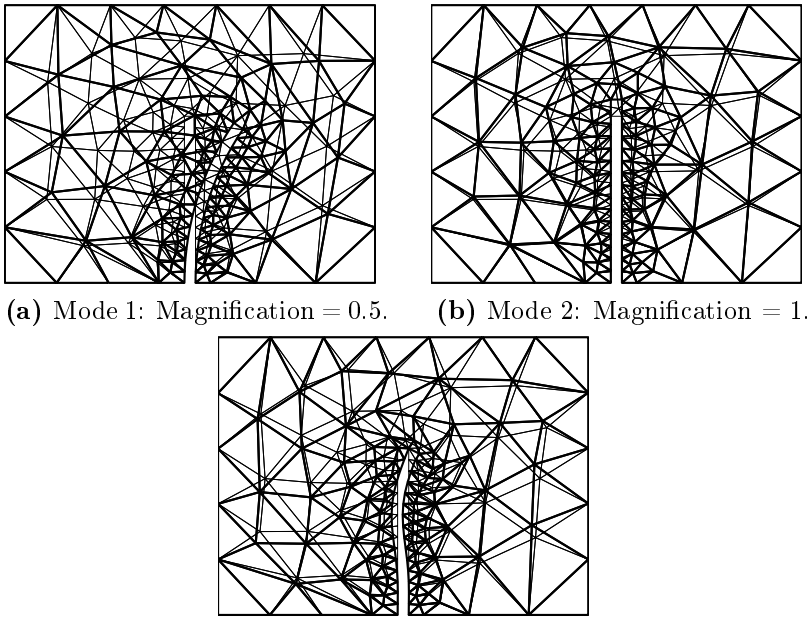
To illustrate the contributions made by each of the modes, representations of the first three modes, for varying degrees of magnifications, are plotted in Figure 2.4. From the representative plots of the modes, it becomes clear that the first mode, qualitatively a pure bending mode, does in fact contain the majority of the system information. The expansion coefficients $\boldsymbol{\alpha}$ controls the magnitude and direction of the associated movement. The second mode stretches the elements at the beam tip, which prevents element inversion for large displacements. In comparison to the first two modes, the remaining three modes contribute an almost insignificant amount. The plot of mode 3 is for a magnification factor of 30, and appears to move the mesh for axial and higher order bending displacements of the beam.

To demonstrate the ability of the POD method, we will attempt to approximate snapshot 5 (Figure 2.2(e)). To do so, we set our expansion coefficients to the first K components of the eigenvector associated with the 5^{th} snapshot, to produce a K^{th} mode approximation. The approximate mesh solutions are shown in Figure 2.5, for only one and two retained POD modes. We find that using only two modes, the approximate mesh solution is almost indistinguishable from the exact solution.

To mathematically quantify the difference between the POD approximations and the exact solution of snapshot 5, we define an error function normalized to element size as

$$||e||_2 = \frac{1}{N}\left(\frac{\sum_{i=1}^{N}\sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_{i\,\text{exact}})^2}}{l}\right), \tag{2.18}$$

where l is the average length of all edges connected to node i. A plot of the error as a function of the number of retained modes, K, is shown in Figure 2.6. From an engineering point of view, the mesh approximation retaining only the first two modes is



(a) Mode 1: Magnification = 0.5. **(b)** Mode 2: Magnification = 1.

(c) Mode 3: Magnification = 30.

Figure 2.4: Representations of the first three POD modes at various magnification factors. Thick line = POD basis mode, thin line = initial mesh.

sufficient, with $||e||_2 = 2.6 \times 10^{-4}$. On the other hand, if we retain all five modes,$||e||_2 = 5.1 \times 10^{-9}$.

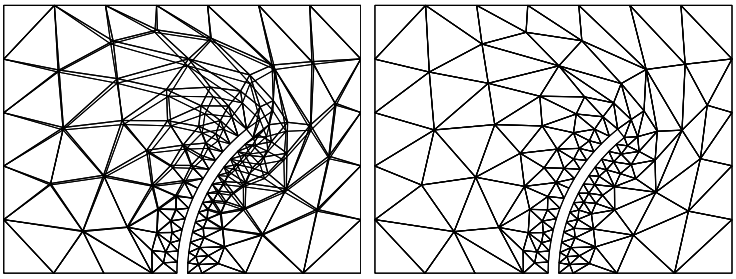
This example demonstrates the power of POD as applied to mesh movement; the original system can now be accurately reproduced with only two pertinent pieces of information. Furthermore, the approximate model is linear in nature, thus simple to compute. It is this powerful ability of POD to *cheaply* and *accurately* approximate a system that makes it a viable tool in the creation of reduced order models.

In this particular example, POD was used to reproduce a known mesh for a boundary movement already computed (snapshots). It would be far more useful if one is able to generate a reduced model of the mesh movement problem that is applicable for arbitrary boundary deformations. To do so would require the computation of an appropriate set of expansion coefficients, $\boldsymbol{\alpha}$. To this end, Chapter 3 provides a broad overview of the method of weighted residuals which have been successfully applied to obtain system approximations using the method of POD.

2.4 Conclusion

In this chapter, the method of proper orthogonal decomposition was briefly introduced. The basic concepts were outlined and an overview of its derivation was provided. Furthermore, the method of snapshots was discussed, and demonstrated as a means to

efficiently compute the POD modes from a set of system observations. The chapter closed with a simple example demonstrating the ability of POD to generate an accurate approximate model of mesh movement.



(a) First Mode Approximation **(b)** Two Modes Approximation

Figure 2.5: POD approximation of snapshot 5. Thick line = approximate solution, thin line = exact solution.

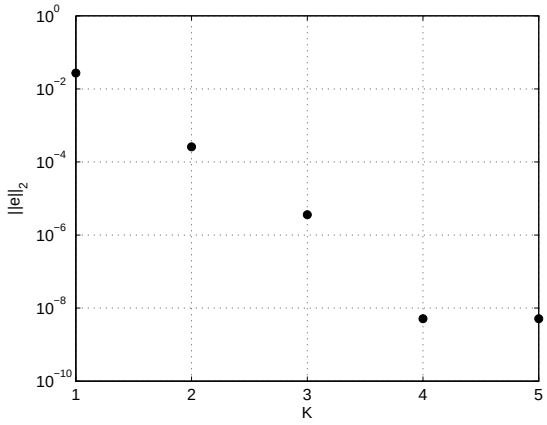


Figure 2.6: A plot of the error function, normalized to element sizes, comparing the difference in nodal coordinates between the approximate and exact solution for snapshot 5 as a function of the number of retained modes.

Chapter 3

Reduced Order Modelling Using POD

The purpose of this chapter is to introduce the concept of POD in the framework of reduced order modeling (ROM) and solution approximations.

3.1 Introduction

In the preceding chapter it was shown that given a set of snapshots, POD modes $\boldsymbol{\varphi}_i$ can be found such that a lower-order approximation to these snapshots can be generated as a linear combination of the POD modes:

$$\boldsymbol{u}(\boldsymbol{x},t) = \sum_{j=1}^{M} \alpha_j(t)\boldsymbol{\varphi}_j(\boldsymbol{x}). \tag{3.1}$$

If the coefficients α_j are chosen as the eigenvalues of the autocorrelation matrix in (2.4), then using equation (3.1) we are able to reproduce $\{\boldsymbol{u}^k\}$, our ensemble of training snapshots. It is however, in the context of ROM, fairly pointless to simply be able to reproduce information that we already have. In Chapter 2, an example was presented where the mesh of one of the training snapshots was approximated using POD. In order for the ROM to be of any real value, we would like to use these computed POD basis modes to solve the mesh movement problem for boundary motions other than those used in the generation of the snapshot information. To do so, some means is required to solve for an appropriate set of expansion coefficients $\boldsymbol{\alpha}$.

This chapter will focus on the discussion of PODs within this framework of generating approximate reduced order models of a given system, specifically for parameter changes. The discussion will focus on systems that can be described by a set of equations (for example, mesh motion based on the solution of a set of partial differential equations (PDE), such as the equations of linear elasticity).

The main aim of reduced order modeling, as the name suggests, is to as accurately as possible find a solution to the system of equations, but at a reduced computational cost. POD decomposition provides an elegant tool to this end. The model is trained with various snapshots that are deemed representative of the system. The POD process then extracts the dominant system information in terms of POD modes. Essentially, the system is transformed and rotated into a subspace, where the solution coefficients are no longer x and y coordinates at time t, but rather the expansion coefficients α_j. To do this, the set of governing equations are projected onto the basis modes through some or other method of weighted residuals (MWR). Using equation (3.1), one can then map back from the POD subspace to the Euclidean space.

By projecting the governing equations onto the POD basis modes, we still inherently solve the original problem, but now only in an approximation subspace with a significant reduction in the magnitude of problem size. The total degrees of freedom (DOF) of the POD based model is equal to the number of retained modes, M, in equation (3.1), where M is typically several orders of magnitude smaller than the DOF of the original system. Furthermore, because the POD basis modes are orthonormal to one another, the projection of a set of coupled PDEs onto these basis modes, will result in a decoupled set of ordinary differential equations.

The remainder of the chapter will briefly discuss the most popular forms of the MWR.

3.2 Methods of Weighted Residuals (MWR)

The MWR is a commonly used numerical approximation technique for the solution of partial differential equations. The most commonly used MWR is known as the Galerkin method; other methods include the collocation method, method of least squares and the sub-domain method.

Suppose we have a linear differential operator D, which acts on a function u to produce a function f

$$D(u(x)) = f(x). \tag{3.2}$$

We are able to linearly decompose and approximate u according to equation (3.1) via our POD basis modes (or any other linearly independent basis functions for that matter). To distinguish between our approximate and real solution, let us define our approximate solution as $\tilde{u} \simeq u$.

Using the approximation of (3.1) and substituting it in the differential operator of equation (3.2), we will not exactly reproduce $f(x)$. As such we can define an error, or residual as

$$R(x) = D(\tilde{u}(x)) - f(x). \tag{3.3}$$

The idea of the MWR is now to try and force the residual to zero in some average sense over a predefined interval or general domain. It does so by means of a weighting function:

$$(W_j, R(x)) = 0 \quad j = 1, 2, ..., M \tag{3.4}$$

where W_j is the weighting functions which are still to be chosen, and M is the number of basis modes used for the decomposition of $\tilde{u}$ in equation(3.1). The distinction between the various MWR is the choice of weighting functions [8].

3.2.1 Collocation Method

In the collocation method, the weighting functions are chosen from the family of Dirac delta functions. The weighting function is set to

$$W_i = \delta(x - x_i) = \begin{cases} 1 & x = x_i \\ 0 & x \neq x_i \end{cases} \tag{3.5}$$

where i is the chosen collocation point. Hence the residual of equation (3.4) is forced to zero at a pre-selected number of points. As the number of basis modes used is increased, the residual is forced to zero at more points. The collocation method is by far the cheapest computational MWR, but the obtained results are largely dependent on the choice of collocation points. If the number of points are increased to include all the grid points within the computational domain, the method closely resembles the method of least squares.

3.2.2 Method of Least Squares

The method of least squares attempts to minimize the continuous sum, of the square of the residuals, at each of the computational grid points to zero. In other words, a minimum is to be found of

$$\mathcal{R} = \int_X R^2(x), \tag{3.6}$$

or for a discrete computational mesh

$$\mathcal{R} = \sum_{i,j} R^2(x) \tag{3.7}$$

where i and j are the nodal coordinates. In order to achieve a minimum, the derivative of (3.7) in terms of the expansion coefficients need to be set to zero. Thus the weighting functions become

$$W_i = \frac{\partial \mathcal{R}}{\partial \alpha_i}. \tag{3.8}$$

Unfortunately, the problem can become rather expensive for large computational domains. In an attempt to decrease the associated cost, the method of least squares and collocation method can be joined. Instead of choosing i and j in (3.7) as the nodal coordinates of each of the computational grid points, they may be chosen as collocation points, where the number of points may be varied.

3.2.3 Galerkin Projection

Galerkin projection is perhaps the most well known MWR, where the weigthing function is chosen to be the same as the basis functions, in our case $\boldsymbol{\varphi}_i$.

Following the discussion presented in [1], let us consider the following generic time-dependent PDE for $u(x,t)$,

$$\frac{\partial u}{\partial t} = Lu + N_2(u,u) + N_3(u,u,u) \tag{3.9}$$

where L, N_2, N_3 are respectively linear, quadratic and cubic operators. Using Galerkin projection to project equation (3.9) onto each of the POD basis modes φ_j, we obtain

$$\left(\frac{\partial u}{\partial t}, \varphi_j\right) = (Lu, \varphi_j) + \left(N_2\left(u,u\right), \varphi_j\right) + \left(N_3\left(u,u,u\right), \varphi_j\right). \tag{3.10}$$

Now, substituting the linear POD decomposition of (3.1) into (3.10), and applying the rules of inner products and the fact that the POD basis modes are orthogonal, gives us:

$$\frac{d\alpha_k}{dt} = \sum_l \alpha_l \left(\varphi_k, L(\varphi_l)\right) + \sum_{l,m} \alpha_l \alpha_m \left(\varphi_k, N_2\left(\varphi_l, \varphi_m\right)\right) + \sum_{l,m,n} \alpha_l \alpha_m \alpha_n \left(\varphi_k, N_3\left(\varphi_l, \varphi_m, \varphi_n\right)\right). \tag{3.11}$$

This is now the reduced order model of the time depended PDE (3.9), a time dependent set of ODEs, where the order is equal to the number of retained POD modes, $k = 1, 2, .., M$. The POD/Galerkin model in equation (3.11) be fairly expensive to solve, can despite the great reduction in complexity. Fortunately, since the inner products are functions of the POD basis modes, which are already known, they may be pre-computed, thus to a certain extent lessening the computational cost.

3.3 Conclusion

In this chapter, a broad overview of the various popular methods of weighted residuals were provided, in the context of POD applications. The MWR can be used to project the system equations onto the POD basis modes, and in so doing, allow for the solution of the system in terms of the model expansion coefficients.

Chapter 4

Unstructured Mesh Movement

The aim of this chapter is to describe and compare various mesh movement strategies for unstructured grids.

4.1 Introduction

FSI simulations involve boundary deformations. Whether these deformations are prescribed or a structural deflection due to the flow, it becomes necessary for the computational grid to conform to the new displaced domain when using a Lagrangian family of descriptions. It is often inadvisable to regenerate or locally adapt a mesh at each instance of mesh motion. Not only can this process quickly become very expensive, but mesh topology is altered requiring that information be mapped from the old mesh to the new one. Projection of information from one mesh to another is another expensive process, and can easily lead to conservation issues arising within the fluid domain [32, 41]. To this end, several mesh movement algorithms have been developed with the primary aim to move a mesh such that the total frequency and necessity for re-meshing or localized adaptation is reduced.

Moving a structured mesh is a well defined problem and several efficient algorithms exist to this end, for example the Transfinite Interpolation method [43]. The structured nature of the grid allows the motion of internal grid points to be interpolated, along grid lines, based on the boundary displacements. These methods however do not apply to unstructured meshes, which are typically preferred when a mesh is to be automatically generated across complex geometries.

Under mesh movement, there are three main schools of thoughts or approaches to dealing with the movement of unstructured meshes. The three approaches can be classified as follows: through the use of radial basis function (RBF) interpolation, through mesh quality optimization or through the solution of a set of partial differential equations

(PDE). Each of the methods differ in terms of their robustness, quality of meshes provided and their associated computational cost.

Mesh movement using Radial basis function (RBF) interpolation has become very popular in recent years because of its ability to move meshes in a computationally inexpensive manner [9, 35]. RBF interpolation is a well established tool for interpolating scattered data and has for some time been used in fluid-structure interaction computations to transfer information across discrete fluid-structure interfaces. In mesh movement, RBFs are used to interpolate the motion of internal grid points based on the known displacement of the domain boundary. The method offers the advantage that no grid connectivity information is required and only a small system of equations needs to be solved, involving only the nodes on the flow domain boundary. The method does however suffer from path dependency, and while the quality of meshes produced are comparable to most available methods, RBF interpolation does not offer direct control over element qualities.

The second approach is based on optimization of each element within the computational grid through some objective function [4, 11]. The objective function can be defined to measure one or other mesh property. Most often it is defined as a scalar function that in some fashion represents the mesh quality in terms of element shapes and sizes. In so doing, the mesh vertices are allowed to reposition themselves to lead to an optimum mesh quality in a global sense. Of all the available mesh movement methods, optimization arguably produces the highest quality meshes, though at a high computational cost. The associated cost is often prohibitively expensive (of order equal to a full CFD solution) to be used in an actual FSI simulation.

The last major approach is to solve a set of partial differential equations (PDE) that describes the position and motion of internal grid points. Examples include representing the grid points by a series of springs [3, 12]; the grid is defined by a network of fictitious tension/compression springs, where the stiffness of the springs are chosen to be inversely proportional to the length of supporting edges. Other methods include solving the mesh as a solid body elastic deformation problem by means of the finite element method [28, 29, 39]. Here the motion of nodes are governed by the equations of linear elasticity, where the stiffness of elements can be altered throughout the domain to allow for optimal mesh qualities. A fair amount of success has also been attained by using Laplacian and bi-harmonic operators [14]. The use of bi-harmonic operators offer the advantage that two conditions can be specified at a moving boundary (for instance the position and normal mesh spacing). All of the above mentioned methods requires solving a set of PDEs for all the grid points, making the methods computationally expensive. Furthermore, for large boundary deformations, these methods can lead to tangled meshes.

In the remainder of this chapter we aim to investigate and discuss three mesh movement algorithms, one from each major class of algorithms, namely RBF interpolation, mesh optimization and the elastic deformation method. We rigorously compare the three methods in terms of computational/memory cost, robustness, mesh quality and

repeatability. The discussions examine the underlying theory pertaining to the mesh movement algorithms to which we apply the reduced order modeling concepts later in Chapters 5 and 6.

4.2 Radial Basis Function Interpolation

4.2.1 Formulation

The following discussion on radial basis function interpolation applied to mesh movement follows the discussion presented by de Boer *et al.* [9].

RBF interpolation can be used to determine the motion of internal nodes given the displacement of the boundary. The displacements of nodes are characterized by the interpolation function $\boldsymbol{s}$, which is defined by the summation of a set of radial basis functions:

$$\boldsymbol{s}_j = \boldsymbol{s}(\boldsymbol{x}_j) = \sum_{i=1}^{n_b} \boldsymbol{\alpha}_i \boldsymbol{\phi}\left(||\boldsymbol{x}_j - \boldsymbol{x}_{bi}||\right) + \boldsymbol{p}(\boldsymbol{x}_j), \quad j = 1, 2, ..., N, \tag{4.1}$$

where N is the total number of nodal coordinates, $\boldsymbol{x}_j = [x_j,\, y_j,\, z_j]$ is the location at which the function $\boldsymbol{s}_j = \left[s_{j_x},\, s_{j_y},\, s_{j_z}\right]$ is evaluated. Furthermore, $\boldsymbol{x}_{bi} = [x_{bi}, y_{bi}, z_{bi}]$ are the known boundary nodes spatial positions, n_b is the number of boundary nodes, $\boldsymbol{p}(\boldsymbol{x}_j) = [p_x,\, p_y,\, p_z]$ is a linear polynomial, $\boldsymbol{\phi} = [\phi_x,\, \phi_y,\, \phi_z]$ is the given radial basis functions and $||\cdot||$ denotes the Euclidean distance.

The coefficients of the polynomial $\boldsymbol{p}\left(\boldsymbol{x}_j\right)$ and the coefficients $\boldsymbol{\alpha}_i = \left[\alpha_{i_x},\, \alpha_{i_y},\, \alpha_{i_z}\right]$ in equation (4.1) can be found by realizing that the interpolation function at the boundaries should be equal to the boundary displacement

$$\boldsymbol{s}\left(\boldsymbol{x}_{bi}\right) = \boldsymbol{d}_{bi}, \tag{4.2}$$

where $\boldsymbol{d}_{bi} = \left[d_{bi_x},\, d_{bi_y},\, d_{bi_z}\right]$ are the discrete values of the known boundary displacements. When the polynomial is included, the system is completed with the additional side condition/constraint that

$$\sum_{i=1}^{n_b} \boldsymbol{\alpha}_i \boldsymbol{q}(\boldsymbol{x}_{bi}) = 0, \tag{4.3}$$

for all polynomials $\boldsymbol{q}(\boldsymbol{x}_j)$ of degree equal to or less than $\boldsymbol{p}(\boldsymbol{x}_j)$. According to de Boer et al. [9], the minimal degree of $\boldsymbol{p}(\boldsymbol{x}_j)$ depends on the choice of basis functions, where a unique interpolant exists if the basis functions are positive definite. If the basis functions are conditionally positive definite for orders 2 or less, then a linear polynomial can be

used. It should be noted, that through the use of a linear polynomial, that rigid body translation and shear will be exactly recovered. A set of basis functions that adhere to these criteria are shown in Table 4.1.

Using the conditions specified in (4.2) and (4.3), the following matrix problem can be setup to solve for the coefficients $\boldsymbol{\alpha}_i$ and those of the linear polynomial $\boldsymbol{p}(\boldsymbol{x}_j)$

$$\begin{bmatrix} \boldsymbol{d}_b \\ \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M}_{bb} & \boldsymbol{P}_b \\ \boldsymbol{P}_b^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix}, \tag{4.4}$$

where $\boldsymbol{\alpha}$ is a matrix containing the coefficient sets $\boldsymbol{\alpha}_i$, and $\boldsymbol{\beta}$ are the coefficients of the linear polynomial $\boldsymbol{p}$. $\boldsymbol{M}_{bb}$ is a $n_b \times n_b$ matrix containing the evaluations of the basis functions

$$\boldsymbol{M}_{bb} = \begin{bmatrix} \phi_{s_1 s_1} & \phi_{s_1 s_2} & \cdots & \phi_{s_1 s_{n_b}} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{s_{n_b} s_1} & \phi_{s_{n_b} s2} & \cdots & \phi_{s_{n_b} s_{n_b}} \end{bmatrix} \tag{4.5}$$

with $\phi_{s_1 s_2} = \phi\left(||x_{s_1} - x_{s_2}||\right)$. Furthermore, $\boldsymbol{P}_b$ in (4.24), is an $n_b \times 4$ matrix, where each row i is given by $[1\, x_{b_i}\, y_{b_i}\, z_{b_i}]$, and as before, $\boldsymbol{d}_b$ refers to the known displacements along the boundaries.

Once the coefficients $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ have been solved for, the interpolation function in equation (4.1) can be used to, point for point, compute the displacements of the internal mesh grid points

$$\boldsymbol{d}_{\text{internal}} = \boldsymbol{s}(\boldsymbol{x}_{\text{internal}}). \tag{4.6}$$

Since the displacement of internal points are interpolated separately, no mesh connectivity information is required, where the total size of the matrix system to be solved is $(n_b + 4) \times (n_b + 4)$. This is primarily why mesh movement based on RBFs is computationally so effective, since the number of nodes on the boundaries in any typical FSI or CFD simulation is orders of magnitude smaller than the number of internal nodes.

Several radial basis functions are cited in literature, and for a more comprehensive list view [9]. The functions can essentially be divided into two main categories: namely functions with global support and functions with compact support. Functions with compact support have the following property:

$$\phi(x) = \begin{cases} f(x) & 0 \leq x \leq 1 \\ 0 & x > 1 \end{cases} \tag{4.7}$$

The first four basis functions in Table 4.1 are examples of basis functions with compact support. The functions are generally scaled with a support radius r, such that $\phi_r = \phi(\xi)$

where $\xi = x/r$. The support radius implies that only mesh nodes within the sphere of influence defined by r, surrounding the centers of known movement $\boldsymbol{x}_{b_i}$, are influenced. Thus increasing the size of the support radius r, leads to an improvement in the quality of the final mesh produced by the interpolation function, but it increases the density of the matrix to be solved. Inversely, a smaller radius will result in a sparser matrix leading to faster computation times.

Global support functions, examples of which include the last four functions in Table 4.1, are in general more expensive to solve than compact functions. This is due to the fact that global support functions are not zero outside some specified radius, but cover the whole space. The parameter a for the MQB and IMQB functions control the shape of the functions. Large values of a gives a flat sheet like function whereas smaller values give a cone like shape. Typical values for a fall in the range 10^{-5} to 10^{-3} [9].

Name	Definition
CP C^0	$(1-\xi)^2$
CP C^2	$(1-\xi)^4(4\xi+1)$
CTPS C^0	$(1-\xi)^5$
CTPS C^1	$1+\frac{80}{3}\xi^2-40\xi^3+15\xi^4-\frac{8}{3}\xi^5+20\xi^2\log(\xi)$
Thin plate spline (TPS)	$x^2\log(x)$
Multiquadratic biharmonic (MQB)	$\sqrt{a^2+x^2}$
Inverse multiquadratic biharmonics (IMQB)	$\sqrt{\frac{1}{a^2+x^2}}$
Gaussian (G)	e^{-x^2}

Table 4.1: Radial basis functions

4.2.2 Mesh Movement using RBF

To demonstrate the use of RBF as a mesh movement strategy, a simple rotation and translation test is presented. The test problem consist of a square domain with an inner rectangle that undergoes extreme translation and rotation. The square domain is of size 200×200 units with an inner rectangle of size 30×10. The initial mesh for the test problem is shown in Figure 4.1, generated using NETGEN [36]. The mesh has a total of 492 inner grid points and 74 boundary nodes, with 26 nodes on the boundary of the inner rectangle.

In order to allow for comparison, all mesh movement test results will be cited in terms of the element quality metric discussed in section 4.3.1.1. Briefly, the metric f_{ss} is a shape-size metric. f_{ss} ranges between 0 and 1, where 1 denotes the perfect element and 0 a degenerate element. The initial mesh has a minimum and mean element quality of 0.7917 and 0.9743 respectively.

In de Boer et al. [9] several different radial basis functions were implemented and compared for a variety of test cases. From their results, it was shown that the TPS

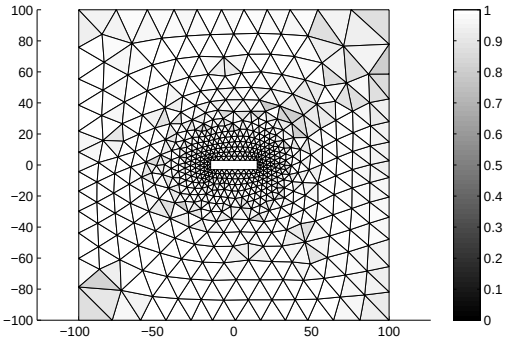


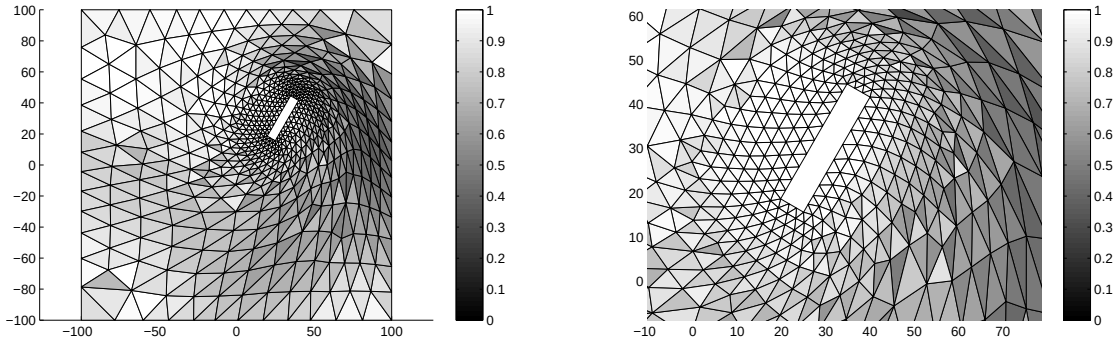
Figure 4.1: Initial Mesh. Element qualities: minimum = 0.7917, mean = 0.9743.

basis function in general performed the best in terms of overall mesh quality, though it tended to be more expensive than the other basis functions. Further, they showed that the CP C^2 function offers the best trade off between computational efficiency and final mesh quality.

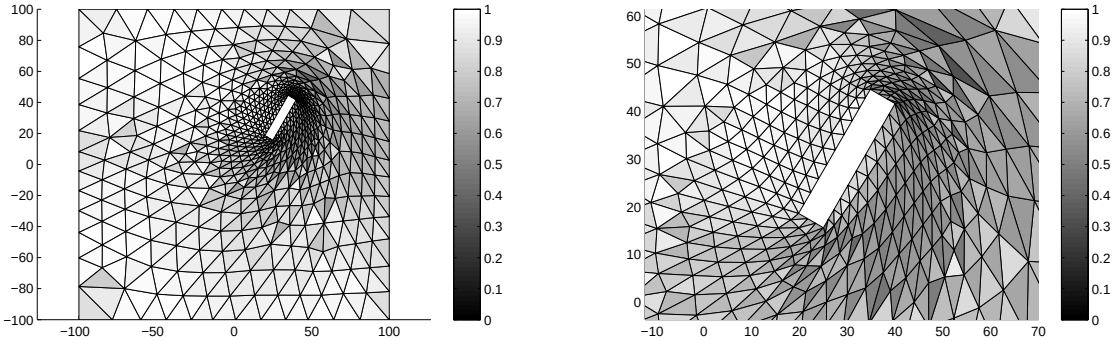
To test the mesh movement strategy independently, four basis functions are implemented, namely the CP C^2, CTPS C^0, TPS and MQB functions. The inner rectangle is translated in the x and y directions by 30 units, and a rotation of 60° anti-clockwise. The support radius length that is chosen for the compact basis functions is 2.5 times the characteristic length of the test problem domain, as suggested by de Boer *et al.* [9]. Examples of the mesh movement performed using the CP C^2 and MQB functions are illustrated in Figure 4.2.

By choosing a support radius of 2.5 times the characteristic length of the domain we essentially change the local support functions into global functions. We do so to ensure that the results we obtain are independent of the choice of support radius. Varying the length of the support radius has two distinct effects. Firstly, there are significant computational advantages to be gained by reducing the support radius. The greatest component of computational cost for the RBF method is to compute the basis functions. These have to be computed for each node that falls within the sphere of influence defined by the support radius. For all nodes outside this sphere, the value may be pre-set to zero and thus does not have to be computed. Furthermore, the solution matrices themselves become significantly more sparse, and hence cheaper to compute. By reducing the radius, one does however sacrifice on the actual quality of the mesh movement process. These effects are to a certain extent investigated in Section 4.2.4.

The mesh quality using RBF interpolation improves if more than one intermediate step is used from the initial mesh to the fully deformed state. All four basis functions are implemented with intermediate steps ranging from 1 to 15. A comparison of the mesh qualities for the various number of intermediate steps is presented in Figure 4.3;



(a) CP C^2



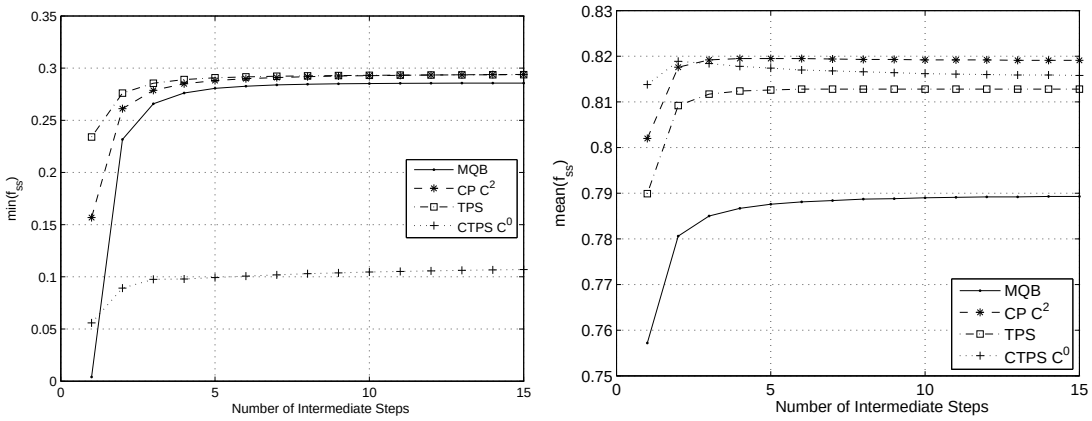
(b) MQB

Figure 4.2: Mesh after rotation and translation for 15 intermediate steps

a comparison is made between the lowest quality element within the mesh and the average over all the elements, for each of the basis functions. From the comparative plots, it can be seen that the CP C^2 function performs the best of the four in terms of average mesh quality, and performs second best in terms of minimum mesh quality.

The real attraction of the CP C^2 function is the comparative CPU cost. Figure 4.4 depicts the CPU scaling for the four RBFs as a function of the mesh size. All four RBFs have a near one to one scaling, where the CP C^2 function requires the least computational time, followed by CTPS C^0, MQB and lastly the TPS function. The difference in computational time is attributed to the complexity of the functions. The CP C^2 and CTPS C^0 functions requires the evaluation of a fifth order polynomial, where the other functions either require the computation of a logarithmic function or square root.

A complete discussion on the behavior and accuracy of most of the known basis func-



(a) Quality of the worst element in the mesh **(b)** The mean quality of all the elements in the mesh

Figure 4.3: Comparison of the minimum and mean qualities for the four basis functions

tions can be found in [9]. In the article, several test cases are investigated, including the application of the basis functions to the movement of a three-dimensional mesh. In general, mesh movement through the use of RBF interpolation is shown to be a fast and efficient way of moving unstructured meshes.

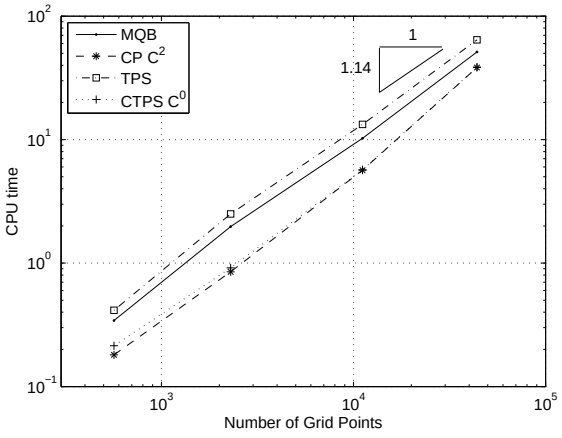


Figure 4.4: CPU time required for a single RBF increment as a function of mesh size.

4.2.3 Path Dependency and Limitation of RBF Interpolation for Mesh Movement

The implementation of RBF to the problem of mesh movement in general yields final meshes of comparable qualities to other mesh movement strategies [35]. Along with the fact that they often take but a fraction of the time to solve make RBF a very attractive

option. Unfortunately the method is path dependent. An example of this was first seen in Figure 4.3 where the quality of the final mesh for the rotation and translation test case was shown to be dependent on the number of displacement increments used. Path dependency is a serious concern for transient FSI simulations where repeatability is of critical importance.

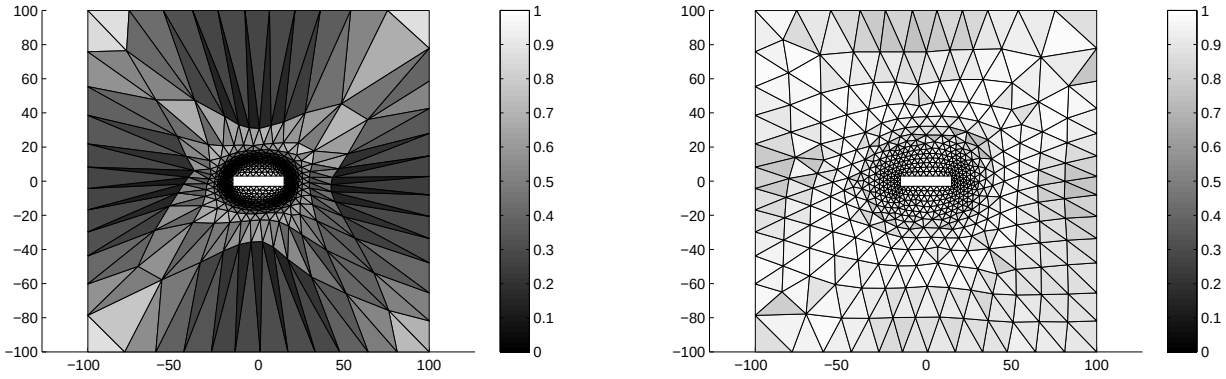
The problem of path dependency is especially highlighted when rotations are present. In the RBF formulation a linear polynomial is used and as a result rigid body translations, of the entire mesh domain, can be recovered exactly. However a mesh tends to distort quickly when rotations are present. To illustrate this problem, the inner rectangle of the mesh presented in Figure 4.1 is rotated between $-60°$ and $+60°$ five times before returning to the original position.

The test was performed for a total of 18 and 80 intermediate steps with the final meshes shown in Figure 4.5. Rotating the block with only 18 incremental steps results in an unacceptable mesh, with an average mesh quality of 0.4824 and the lowest element being of a quality 0.1109. The mesh is considerably improved by using 80 incremental steps, where the mesh in Figure 4.5(b) has a mean of 0.9318 and a minimum of 0.6841, which is more acceptable, though still less than the original mesh, with mean and minimum qualities of 0.9743 and 0.7917 respectively.

This example highlights the extent to which a mesh can distort when moved using a path dependent method. In the case of using RBF interpolation this problem is especially prominent when rotations are present. To somewhat rectify the problem several iterations need to be used, which tends to make the mesh movement problem expensive. In practice, there are several FSI problems where the boundary undergoes an oscillating type motion, for example in the flutter simulation of a wing. The boundary will typically oscillate or move backwards and forwards several times before a limit state solution is obtained, if at all. Since RBF functions offer no control, and rotational movements are inherently present in these type of motions, the mesh quality will likely deteriorate to unacceptable levels. This in turn can possibly force an increase in the frequency of re-meshing.

4.2.4 Influence of Support Radius for Compact Functions

The last concept that remains to be investigated in the use of RBF is the choice of support radius r, for compact support functions. For the investigation we revisit the extreme rotation and translation of the inner rectangle ($\triangle x = 30$, $\triangle y = 30$ and $\phi = 60°$CCW) for a selection of mesh sizes. The RBF implemented for the investigation is the CP C^2 function with 15 displacement increments. In Figure 4.6 we plot the minimum element qualities, computational time for a single displacement increment and the memory requirements to store the matrix in equation (4.24), all as a function of the chosen support radius.



(a) 18 intermediate steps. Min: 0.1109, Mean: 0.4824. **(b)** 80 intermediate steps. Min: 0.6841, Mean: 0.9318.

Figure 4.5: Final meshes for rotating the inner rectangle between $-60°$ to $+60°$six times and back to the original position, using the CP C^2 basis function.

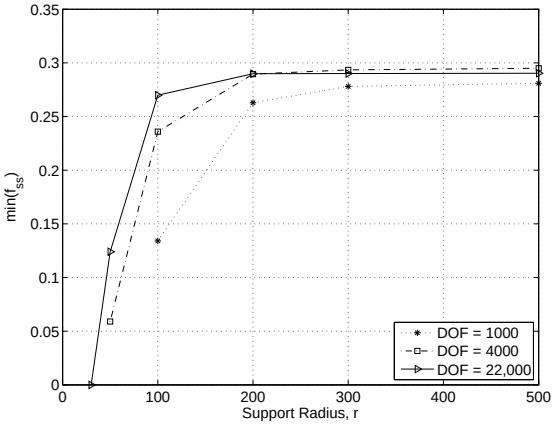
As alluded to in the preceding section, changing the support radius has two competing effects. By decreasing the support radius, the sparsity of the solution matrix increases resulting in decreased memory usage and a speed up in solution time. A further speed up is attained by only having to solve the radial basis function for internal nodes located within the sphere of influence. On the contrary however, decreasing the radius results in a decrease in the quality of the final meshes produced. The extent or impact on element quality due to a change of support radius depends strongly on the size of the mesh being moved.

To better aid the discussion, let us analyze the results in Figure 4.6(a). Choosing the support radius r to less than 100 for the 1000 DOF mesh yields unacceptable meshes. In this instance we refer to an entangled mesh as unacceptable. Furthermore, setting $r = 100$ for the same mesh yields an acceptable mesh, but suffers from a significantly reduced minimum element quality of close to 60% as compared to a support radius which covers the entire mesh domain. By choosing the support radius to be larger than the domain size one essentially renders the compact support function a global function.

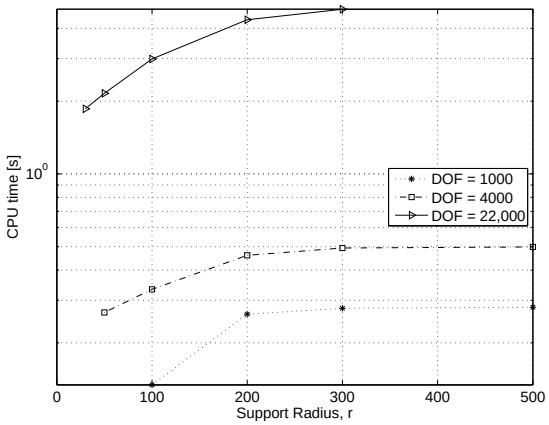
In contrast to the smaller mesh, choosing $r = 100$ for the 22,000 DOF mesh results in only an 8% reduction in minimum element quality. The choice of support radius then truly becomes a decision of weighing criteria. In the case of the 22,000 DOF mesh, an 8% sacrifice of element quality may be justifiable by gaining close to 40% and 70% reduction in computation time and memory usage respectively.

The choice of support radius is hence an important additional heuristic to consider when implementing a compact support function. In this study however, we regard mesh quality as a priority, therefore for all future applications of RBF in this study

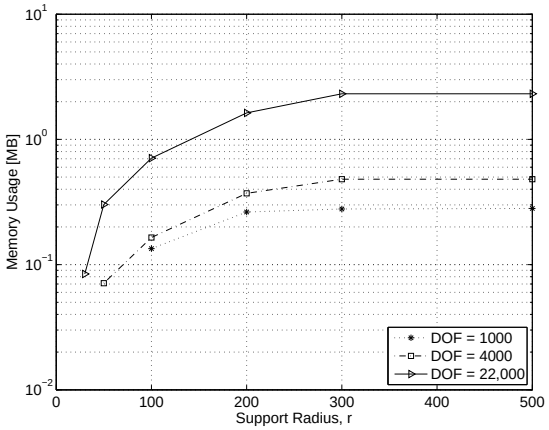
we will make use of an infinite support radius. We merely highlight these results to point out that for larger meshes, there are potential computational and memory savings available at minor reductions in the resulting mesh quality.



(a) Minimum element qualities for $0 \leq f_{ss} \leq 1$.



(b) CPU time for a single displacement increment.



(c) Memory requirements for storage of the matrix in equation (4.24).

Figure 4.6: Plots demonstrating the effect of the support radius to mesh quality, CPU time and memory usage. Results are for the CP C^2 function for inner rectangle motion of $\triangle x = 30$, $\triangle y = 30$, $\phi = 60°$CCW, over 15 displacement increments.

4.3 Optimization Methods for Mesh Quality Improvement

The problem of mesh movement through mesh quality optimization is not so much a mesh movement technique but rather a mesh smoothing operation. Each time the flow domain boundary is moved, the mesh is regularized or smoothed by allowing each of the mesh vertices to move, through the use of some optimization algorithm. The mesh is optimized according to an objective function which in some fashion describes the overall quality of the mesh. Formally, the mathematical optimization problem is stated as follows:

$$\underset{\text{w.r.t}\,\boldsymbol{x}}{\text{minimize}}\, F(\boldsymbol{x}),\ \ \boldsymbol{x} = [x_1, x_2, ..., x_n]^T \tag{4.8}$$

where $F(\boldsymbol{x})$ is the objective function to be optimized and $\boldsymbol{x}$ in our particular case is the x and y co-ordinates of the interior mesh vertices that are allowed to move.

In order to solve the problem posed in equation (4.8), two choices have to be made. Firstly, we require an objective function that adequately describes the mesh quality. For this particular study, four mesh quality metrics are investigated, where they describe the quality of each element within the mesh; the mesh quality metrics are scalar descriptors of an element's shape, size or a combination of shape and size. The objective function is then the summation of the scalar values of the mesh element qualities.

The second requirement is an appropriate optimization algorithm. Several options are available to this end, including gradient based algorithms and numerous evolutionary, population based methods. It is however the opinion of most researchers that if the objective function is well defined, continuous and differentiable everywhere, that gradient based methods are the quickest and most efficient of the available methods [38].

Within the class of unconstrained gradient based optimization, several feasible algorithms exist, and to name but a few include conjugate-gradient methods, Newton and quasi-Newton methods such as DFP or BFGS. Each of these methods offer their own properties in terms of the type of problems they can handle, computational cost and convergence rates. For the purpose of this study, we will implement and compare Newton's method (with line search) and a conjugate gradient line search method.

Conjugate gradient methods are memory efficient, first order line search algorithms that attempt to minimize a given objective function. The conjugate gradient method has the desirable property of quadratic termination, but only makes use of first order gradient information. As a result they are quick to solve per solution iteration, however suffer from poor convergence properties for non-quadratic functions far away from the solution.

Newton's method on the other is a second order method that attempts to solve the optimality criterion that all gradient components equal 0. Newton's method requires

the evaluation of a Hessian matrix and the solution of an $N \times N$ linear system (where N is the total DOF of the system), which may become computationally intensive for large problem sets. The method does however offer the desirable property of being quadratically convergent if it converges. Unfortunately convergence is not guaranteed and the solution may sometimes diverge even when close to the solution. To alleviate the convergence issue associated with Newton's method we modify Newton's method by including a line search. In so doing, the Newton step is then used as a search direction, with the step length determined using a general line search descent algorithm. The implementation of the conjugate gradient method and Newton's method with line search are detailed in Appendices A and B respectively.

As a final note, in the context of mesh movement, the optimization approach offers some drawbacks. Firstly, for large grids (common in CFD simulations), the procedure is computationally intensive (of approximately the same order as the CFD simulations). Secondly, the maximum boundary displacement cannot be larger than the smallest element size along the boundary. Large displacements lead to the inversion of the boundary elements, which will result in the failure of any optimization algorithm to successfully find an optimal and feasible mesh. To circumvent the problem, the total boundary displacement has to be broken into smaller increments, where each increment will entail a full order optimization problem. Because mesh movement through optimization offers the possibility of high quality meshes, at high computational costs, it constitutes an ideal candidate to which to apply the reduced order modeling techniques described in the preceding chapters.

4.3.1 Mesh Quality Metrics

In literature and practice, there are several quality metrics available to choose from, and there is no consensus within the field of practitioners as to which of the quality metrics offer the best description of element qualities. Inherently, the definition of a good quality mesh is linked to the solution error; in other words, a mesh is considered good if the error based on a simulation is below some acceptable level. The simulation however cannot be performed unless a mesh is already available. To this end, geometric mesh quality metrics are used to provide an indication as to whether elements are of appropriate sizes and shapes.

Primarily, experience is used to judge what is considered as an appropriate size and shape of an element. For instance, it is known that geometrically poor elements (distorted, overly skewed, or with large aspect ratios), result in ill-conditioned matrices, which slow or entirely prevent convergence when using an iterative solver. Prior to knowledge of the solution, it is best to assume isotropic physics, thus the optimal shape for triangular elements would be an equilateral triangle.

In this study, mesh movement is applied only to triangular unstructured meshes, accordingly only quality metrics pertaining to triangular elements will be investigated.

There are a multitude of metrics that were not investigated that may work as well if perhaps not better than the ones implemented; quality metrics are also readily available for all element types.

4.3.1.1 Quality Metric 1

The first quality metric investigated is a shape-size metric proposed by Knupp [22], and is based on the Jacobian matrix and ideas from linear algebra. The metric is a dimensionless quantity defined by the weighted product

$$f_{\text{ss}} = \sqrt{f_{\text{size}}} f_{\text{shape}}, \tag{4.9}$$

where the range of f_{ss} is

$$0 \leq f_{\text{ss}} \leq 1, \tag{4.10}$$

where an element with $f_{\text{ss}} = 1$ refers to the perfect element and $f_{\text{ss}} = 0$ a degenerate element. f_{size} and f_{shape} denote metrics associated with the size and shape respectively. f_{size} is square rooted, since changes in element size have a smaller influence to mesh quality in comparison to distorted elements.

Consider a triangular element, with the coordinates of the vertices defined by (x_k, y_k), $k = 1, 2, 3$, where k is the vertices of the triangle. A Jacobian matrix for the element, around node k is

$$\boldsymbol{A}_k = \begin{bmatrix} x_{k+1} - x_k & x_{k+2} - x_k \\ y_{k+1} - y_k & y_{k+2} - y_k \end{bmatrix}. \tag{4.11}$$

The Jacobian matrix is not independent on which node it is computed, though the shape and size metrics are, therefore the subscript k can be dropped. The determinant of $\boldsymbol{A}$ is twice the area of the triangle, independent of which node is used for the Jacobian. Thus let us set

$$\alpha = \det(\boldsymbol{A}). \tag{4.12}$$

Furthermore, a "metric" tensor matrix $\boldsymbol{\lambda}$ can be computed by

$$\boldsymbol{\lambda} = \boldsymbol{A}^T \boldsymbol{A} \tag{4.13}$$

which is a 2×2 symmetric matrix. The tensor matrix has three distinct components λ_{ij}, $i, j = 1, 2$. Geometrically λ_{11} and λ_{22} are measures of the squared lengths of two sides within the triangle and λ_{12} a measure of the included angle.

The shape metric can now be defined as

$$f_{\text{shape}} = \frac{\sqrt{3}\alpha}{\lambda_{11} + \lambda_{22} - \lambda_{12}}. \tag{4.14}$$

f_{shape} ranges between 0 and 1, where a value of 1 denotes an equilateral triangle. The size metric is defined as

$$f_{\text{size}} = \min(\tau, 1/\tau) \tag{4.15}$$

where $\tau = \alpha/w$, is a ratio between the actual area α, of the triangle and some reference area w. In our specific case, by assuming that the initial mesh is optimal in its own right, the reference area is the initial area of the undeformed mesh. For a gradient based optimization algorithm, we require that the functions be differential everywhere, which is not the case for the size metric in equation (4.15). An alternative size metric, suggested by Knupp [22] is

$$f_{\text{size}} = \frac{2\tau}{1 + \tau^2} \tag{4.16}$$

which is indeed differentiable everywhere.

Using this metric, the objective function to be minimized is then

$$F = -\sum_{\text{elements}} f_{\text{ss}}. \tag{4.17}$$

4.3.1.2 Quality Metric 2

The second metric investigated is a robust method based on the quotient of the radii of the circumscribed and inscribed circles of a given triangle [4]. The ratio given by

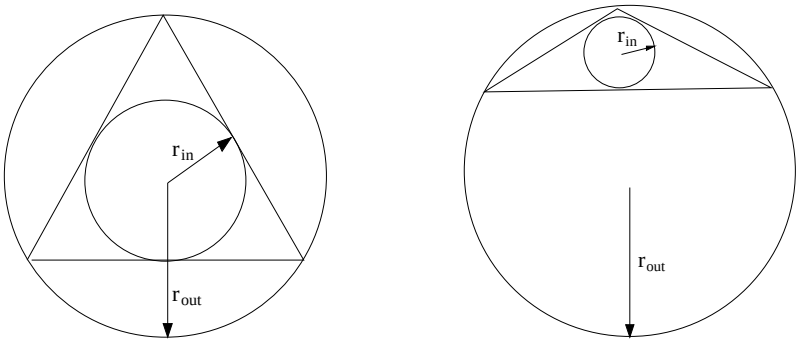
$$\frac{r_{\text{out}}}{r_{\text{in}}} \tag{4.18}$$

can directly give a measure of the shape quality of an element. Consider the triangles shown in Figure 4.7. For an equilateral triangle, the ratio is 3, and as the element degenerates, this ratio tends to infinity.

The objective function of the metric proposed in [4] is

$$F = \sum_{\text{elements}} \left(\frac{r_{\text{out}}}{r_{\text{in}}}\right)^{\nu} \left(\frac{r_{\text{out}}}{r_0}\right)^{\mu}, \tag{4.19}$$

where ν, μ and r_0 are positive constants. The choice of constants have little impact on the solution of the optimal mesh, and for this studies the chosen values are $\nu = 3$, $\mu = 1$ and $r_0 = 1$, as proposed in [4]. The quotient $\left(\frac{r_{\text{out}}}{r_{\text{in}}}\right)^{\nu}$ ensures that the element shapes are favorable and $\left(\frac{r_{\text{out}}}{r_0}\right)^{\mu}$ controls the size of the elements, ensuring that the elements do not become too large.



(a) Equilateral Triangle **(b)** Badly distorted triangle

Figure 4.7: Comparison of the inner and outer circles for an equilateral and degenerate triangle

4.3.1.3 Quality Metric 3

The third metric investigated is similar to equation (4.19), but instead of optimizing for both the shape and size, the objective function only focuses of the element shapes. The objective function is defined as

$$F = \sum_{\text{elements}} \left(\frac{r_{\text{out}}}{r_{\text{in}}}\right)^{\nu}. \tag{4.20}$$

Once again, the choice of ν has little impact on the solution.

4.3.1.4 Quality Metric 4

The final quality metric is a combination of the size metric presented in section 4.3.1.1, and quality metric 3. The shape of the elements is controlled by the ratio of inner and outer radii, while the element size is controlled by equation (4.16). The objective function to be minimized is then

$$F = \sum_{\text{elements}} \left(\frac{r_{\text{out}}}{r_{\text{in}}}\right)\left(\frac{1}{f_{\text{size}}}\right). \tag{4.21}$$

4.3.2 Test Case Comparing Quality Metrics

To test the applicability of each of the quality metrics, the same mesh movement problem used in section 4.2.2, with the initial mesh shown in Figure 4.1 will be used. The inner rectangle once again will be translated in the x and y directions by 30 units and rotated a total of 60° counter-clockwise. For the mesh movement tests, it will be assumed that an appropriate optimization algorithm is made use of, as discussed in Section 4.3.3. The final meshes are compared to the benchmark set by the RBF interpolation method, using the CP C^2 function.

Before continuing the discussion and comparison of the four metrics, mention must be made that we have modified metric 1 for the purposes of the mesh movement. In section 4.3.1.1, we defined f_{ss} such that it ranges between 0 and 1. The definition as is provides for intuitive comparisons between meshes, as the extremes are well defined, with 0 being a degenerate element and 1 the optimal. Metric 1 however is ill-suited towards the application in an optimization algorithm, as the extremes are not harsh enough. Letting a few elements become degenerate will not have a significant impact on the total cost function. The optimization algorithm will thus allow for a few element qualities to approach 0 in order to attain an overall improvement in the mesh quality. To demonstrate this, Figure 4.8 shows the final mesh as optimized for using the original form of metric 1. The mean quality for the mesh, as measured by metric 1, is 0.8404 while the minimum element quality is 0.0003, which for all practical purposes is degenerate, and will lead to an ill-conditioned solution matrix.

We propose the following modified form of metric 1

$$f_{\text{ss modified}} = \frac{1}{\sqrt{f_{\text{size}} f_{\text{shape}}}}. \tag{4.22}$$

In so doing, f_{ss} now ranges between ∞ and 1 for a degenerate and perfect element respectively.

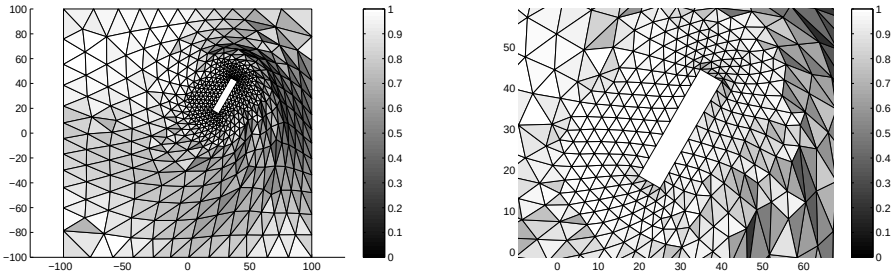


Figure 4.8: Mesh after rotation and translation using the original form of metric 1.

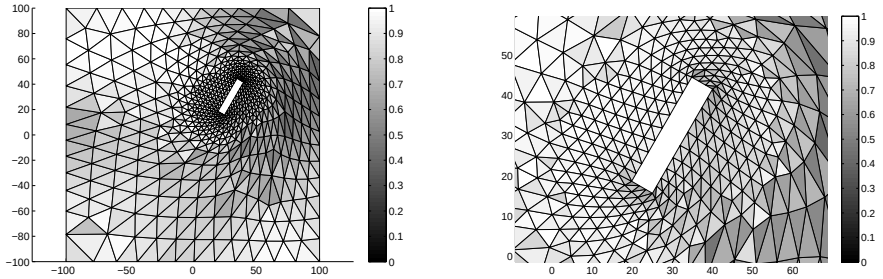
The first important consideration when comparing the four quality metrics is the computational cost associated with each of the metrics. The total number of iterations required to reach convergence for each of the metrics are shown in Table 4.2. All the metrics require a similar number of iterations, both for the conjugate Gradient and Newton's method. There is also little difference in the cost per iteration between each of the metrics, therefore there is no overriding distinction on the basis of computational cost. The four metrics must accordingly be compared purely on merit of the quality of their respective final meshes.

Metric	Conjugate Gradient	Newton
1	141	6
2	304	7
3	176	6
4	201	7

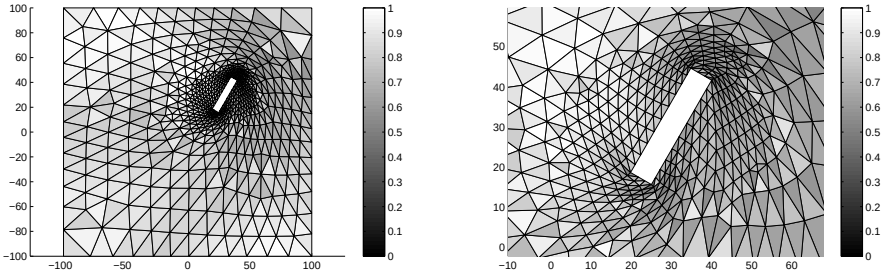
Table 4.2: Number of iterations required to reach convergence for inner rectangle motion of $\triangle x, y = 30$, $\phi = 60°$CCW.

In order to allow for a fair comparison of the quality metrics, the initial mesh is pre-optimized with the respective metrics prior to the mesh movement; in so doing, we remove any bias towards the initial mesh. Using the same initial mesh for all four metrics would unfairly disadvantage metrics 2 and 3, as they contain no information pertaining to the original mesh.

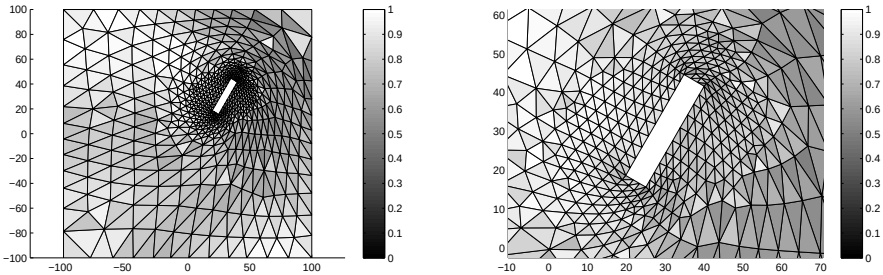
The final meshes for each of the metrics are presented in Figure 4.9. The gray-scale mapping is a measure of the element qualities based on the original form of metric 1, where $0 \leq f_{ss} \leq 1$ for degenerate and perfect elements respectively. By studying the figures, it becomes apparent that the mesh movement produced through optimization is heavily dependent on the choice of quality metrics.



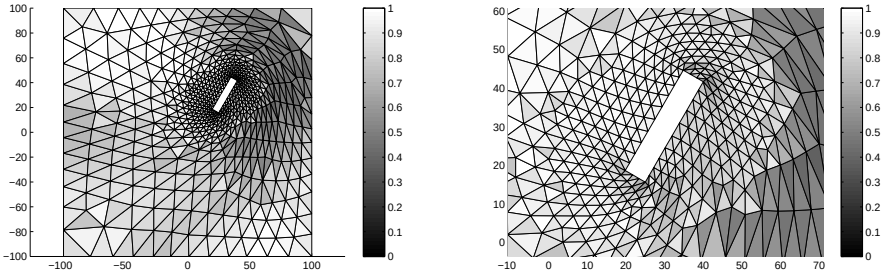
(a) Metric 1 (modified).



(b) Metric 2.



(c) Metric 3.



(d) Metric 4.

Figure 4.9: Mesh movement after rotation and translation for all four metrics. Inner rectangle movement of $\triangle x, y = 30$, $\phi = 60°$ CCW. The gray-scale is a measure of element quality using the original form of metric 1, $0 \leq f_{ss} \leq 1$.

As an example, the meshes produced using metric 2 results in elements along the inner rectangle being severely distorted (Figure 4.9(b)). Because of the quotient $\left(\frac{r_{\text{out}}}{r_0}\right)$ in metric 2, the optimization algorithm attempts to minimize the volume of all mesh elements. Since the large majority of elements are located close to the inner rectangle boundary, the optimization algorithm will favor these elements to be small. Having small elements close to the flow boundary is desirable for fluid simulations, as this is the region where most of the complex flow behavior occurs. Having said that, while it is beneficial to have the smaller elements on the boundary, it is far more detrimental for unsteady fluid simulations if these elements are distorted.

Metric 1 on the other hand appears to produce a final mesh with high element qualities along the inner boundary (Figure 4.9(a)), but generates distorted elements further away from the boundary. In an attempt to produce meshes that compromise between the good inner element qualities of metric 1, with the overall improved performance of metric 2, metric 2 was modified by removing the size quotient resulting in the current form of metric 3. The size quotient in metric 2 is to ensure that elements do not become overly large. Since the starting mesh consists of a set of closely packed elements there is little room for extreme size changes; controlling the element sizes is therefore unnecessary. Using metric 3 we attain an overall improvement in the quality of elements along the boundary (Figure 4.9(c)).

Metric 4 was created by taking this notion one step further. Since most elements are located along the inner domain boundary, any optimization algorithm will favor these elements. Thus by inserting the size measurement of $\frac{1}{f_{\text{size}}}$ we force the optimization algorithm to keep the element volumes along the boundary as close to the original mesh as possible. The result of this is a further improvement in the quality of elements along the inner boundary, while not sacrificing the overall mesh quality (Figure 4.9(d)).

To numerically quantify and compare the performance of each of the metrics we report on the worst and mean element qualities in Tables 4.3 and 4.4. To ensure that we compare like quantities, we report the element qualities using all 4 metrics for each of the mesh movements. The top performing metric for each measure is highlighted for ease of viewing.

In terms of worst element qualities (Table 4.3), metric 2 consistently provides the best results. All four metrics are however formulated to maximize mean element qualities, and hence we are not that interested in which metric performs best in terms of worst element qualities, save that the final meshes should contain no degenerate elements. If worst element qualities are the desired property of a mesh, then the cost functions should be chosen or formulated accordingly. Metric 2 produces high worst quality elements because the metric heavily penalizes distorted elements. The optimization routine therefore spends considerable effort ensuring that there are as few distorted elements as possible. The exact opposite is true for the original formulation of metric 1, where near degenerate elements were not penalized enough.

Since none of the final meshes produce near degenerate elements, the top performing

metric will be chosen solely on the basis of mean element qualities. In Table 4.4, one may observe that the top performing metrics, for each metric measure, sits along the diagonal of the table. We ignore these diagonal terms, because optimizing and measuring the mesh quality using the same metric will always produce the best results. Based on the results with the diagonal terms omitted, we note that mesh movement based on metric 4 produces the overall highest quality meshes. Metric 4 will therefore be the metric of choice for all future optimization based mesh movements.

Mesh moved according to	Starting mesh optimized according to and measured by								
	Metric 1 (mod.)		Metric 2		Metric 3		Metric 4		
	Worst	Rank	Worst	Rank	Worst	Rank	Worst	Rank	Rank
Starting Mesh	1.5995	-	3.7560×10^3	-	5.3607	-	5.0526	-	-
RBF CP C^2	13.2927	-	6.1894×10^5	-	24.8947	-	30.5798	-	-
Metric 1 (mod.)	**6.3359**	1	1.4995×10^5	4	15.7634	4	15.8788	4	4
Metric 2	6.3548	2	$\mathbf{8.2735 \times 10^3}$	1	**9.8429**	1	**10.6980**	1	1
Metric 3	9.5118	4	3.7729×10^4	2	11.6018	2	11.1999	2	2
Metric 4	7.3339	3	6.6899×10^4	3	12.6984	3	11.3971	3	3

Table 4.3: Comparison of worst element quality of all four quality metrics. Results are for inner rectangle movement $\triangle x = 30$, $\triangle y = 30$, $\phi = 60°$CCW. Meshes are pre-optimized with each of the respective quality metrics to allow for fair comparisons.

Mesh moved according to	Starting mesh optimized according to and measured by								
	Metric 1 (mod.)		Metric 2		Metric 3		Metric 4		
	Mean	Rank	Mean	Rank	Mean	Rank	Mean	Rank	Rank
Starting Mesh	1.0576	-	623.9486	-	4.0934	-	3.7597	-	-
RBF CP C^2	1.8947	-	9.2598×10^3	-	5.6679	-	5.4564	-	-
Metric 1 (mod.)	**1.5197**	1	2.9973×10^3	4	5.1308	3	4.8346	3	3
Metric 2	1.7975	4	$\mathbf{1.3569 \times 10^3}$	1	5.2559	4	5.0141	4	4
Metric 3	1.7068	3	2.1173×10^3	3	**4.9804**	1	4.7825	2	2
Metric 4	1.6592	2	1.9653×10^3	2	5.0451	2	**4.2787**	1	1

Table 4.4: Comparison of mean element qualities of all four quality metrics. Results are for inner rectangle movement $\triangle x = 30$, $\triangle y = 30$, $\phi = 60°$CCW. Meshes are pre-optimized with each of the respective quality metrics to allow for fair comparisons.

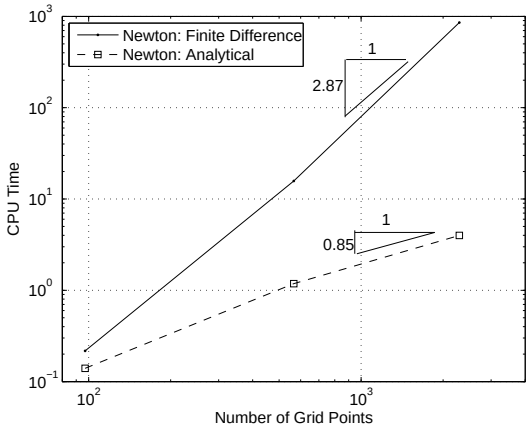
4.3.3 Optimization Algorithms

In this section, we aim to compare the computational cost and efficiency of the conjugate gradient method and Newton's method (with line search). For the sake of brevity the implementation of the algorithms will not be discussed here, but to ensure reproducibility of the work, detailed discussions of the algorithms are attached in the appendices.

Since the cost associated with each of the metrics are similar to one another, we will focus the rest of our discussion on the comparison of the optimization algorithms using metric 4 only. The quality metric is continuous and differentiable everywhere, allowing for the first derivatives and Hessian required by the optimization algorithms to be computed analytically. To take advantage of the sparsity of the Hessian matrix, only the non-zero entries are saved. A sparse matrix solver with LU factorization is made use of along with node re-numbering to produce LU factors with near minimal bandwidth. The re-numbering scheme used is the reverse Cuthill-McKee ordering scheme available in MATLAB [13]. An analysis and comparison of solver technologies is covered in greater detail in Section 4.3.4.

The gradients and Hessian are computed analytically to decrease the computational cost. Using finite differencing (FD), while being more expensive does offer some advantages. Using FD allows for a generic solution, and can inter-changeably be used on any objective function or quality metric without requiring any additional coding. FD is however considerably more expensive, and a comparison of the CPU scaling is shown in Figure 4.10.

In the cases where FD is used to compute the first and second derivatives, Newton's method is a poor choice for the optimization problem, as it scales poorly for an increase in problem size. When using FD, the computation/construction of the Hessian accounts for close to 99% of the solution time, thus quasi-Newton methods such as DFP or BFGS would be better suited. Both the DFP and BFGS algorithms never compute the Hessian but rather make approximations thereof, mitigating a large portion of the computation time. They do however sacrifice on the convergence rate, changing Newton's second order convergence properties to super linear.



(a) CPU time.

Figure 4.10: Computational cost scaling comparison between Newton solver with derivatives and Hessian computed analytically and through finite differencing.

To compare the conjugate gradient and Newton optimization algorithms, we make use of the same rotation and translation problem as in the preceding section, for increasing

mesh sizes. The convergence criteria used is the normalized error

$$||e||_2 = \frac{1}{N}\left(\frac{\sum_{i=1}^{N}\sqrt{(x_{i\text{current}} - x_{i\text{old}})^2}}{l}\right) < 1 \times 10^{-5} \tag{4.23}$$

where l is the average length of all edges connected to node i. We also use a single RBF displacement increment to initially displace the boundary and mesh prior to starting the mesh optimization.

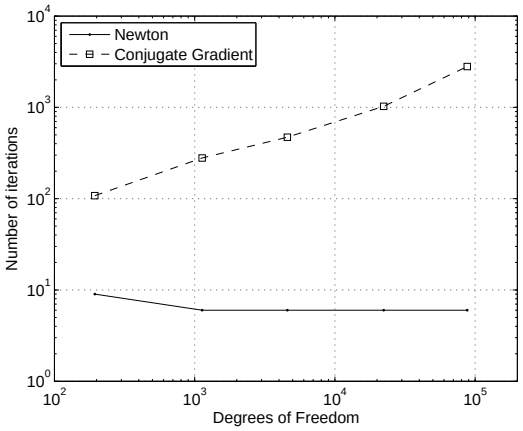
We make use of a single RBF increment because the maximum displacement that the boundary may move, for the optimization algorithm, is the minimum element size along the boundary. Boundary displacements larger than this results in element inversion, and the subsequent failure of the optimization algorithm. Large boundary displacements therefore have to be divided into several smaller increments, where each increment is fairly expensive. As an example, for the finest mesh implemented in this study (DOF$\approx$160000), over 500 increments are required for the inner rectangle movement of $\triangle x = 30$, $\triangle y = 30$, $\phi = 60°$CCW. By using a single initial RBF increment, we are able to cheaply move the mesh to the desired boundary location, while ensuring that none of the elements are inverted. We implement the same method later on in Chapter 5, when we train our reduced order model.

The computational time comparisons for the two algorithms are shown in Figure 4.11. Figure 4.11(a) depicts the total number of iterations required to reach convergence, Figure 4.11(b) the time per iteration and Figure 4.11(c) the total time required for a displacement increment, all as a function of problem size.

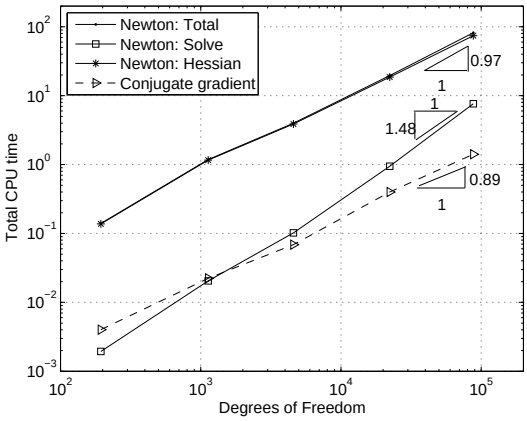
The conjugate gradient method is known to scale well with problem size, and is expected to outperform second order methods for larger problems. For the mesh movement problem, we find that Newton's method performs better for an increase in problem size. For smaller problems the Conjugate gradient method performs better, with a cross over point for a mesh of around 1000 DOF. The cross over point itself will vary from computational platform and implementation, but a mesh of 1000 DOF is still insignificant in comparison to the size of meshes used in actual simulations. For any practical applications, Newton's method would therefore be the preferred optimization method.

Per iteration, the Conjugate Gradient method is cheaper, but requires a significantly larger number of iterations to reach convergence, and hence the poor CPU cost scaling. The reason for this is based on the manner in which the two methods obtain their respective search directions. The Conjugate Gradient method only uses first order, local gradient information. As a result, elements far away from one another have no inter-relating information. For example, if the elements along the inner rectangle boundary are distorted, the elements further away at the edge of the domain have no gradient information regarding this distortion. It takes several iterations for this information

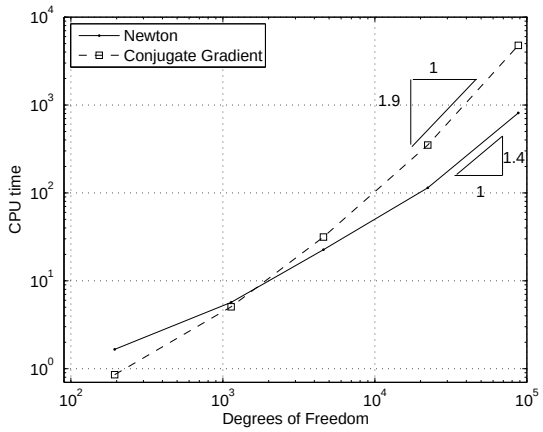
to propagate through the mesh. On the other hand, Newton obtains search directions based on a Hessian, which incorporates information for the whole mesh domain; while being more expensive per iteration, Newton requires substantially fewer iterations to reach convergence.



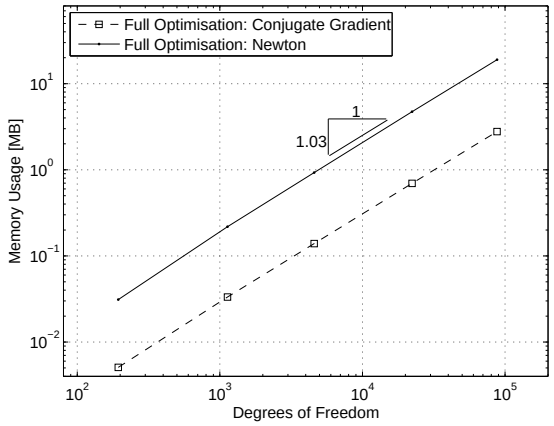
(a) Comparison of number of iterations required to reach convergence.



(b) Comparison of CPU time per iteration.



(c) Comparison of total CPU time per displacement increment.



(d) Comparison of memory usage.

Figure 4.11: CPU time and memory usage comparisons between Conjugate Gradient and Newtons method as a function of problem size.

Despite the clear advantage Newton has in terms of computational time, it does come at a larger memory cost. Figure 4.11(d) is a comparison of the memory scaling as a function of problem size for the two optimization algorithms. The fact that the Hessian is so sparse, and only the non-zero entries are saved, results in Newton's method scaling similarly to that of the Conjugate gradient method, at a near one to one ratio. To provide an indication to the sparsity of the matrix, consider the mesh in Figure 4.1. The Hessian matrix is of size $[1132 \times 1132]$ while containing only 13200 non-zero entries,

accounting for slightly more than 1% of the total number of entries in the full matrix. Despite scaling the same, Newton's method is significantly more memory expensive, and in applications where memory is the defining constraint, the Conjugate gradient method would be the preferred method.

4.3.4 Brief Analysis of Solver Technologies Available in MATLAB

In the preceding section the Newton mesh optimization problem was solved using a direct sparse solver. In this section we aim to briefly explore the validity of our solver choice. In large problem sizes typically expected in CFD simulations of realistic problems it is commonly accepted that direct solvers are infeasible, and one would rather opt for a matrix free iterative solver. Note however that the study of solver technologies and effective pre-conditioners are entire research fields of their own and beyond the scope of this study.

Nevertheless, it is important to use a suitable solver that scales well in terms of both CPU usage and in memory as the aim of this thesis is CPU savings. In this section we therefore try to demonstrate that the direct sparse solver is the most efficient solver available in MATLAB for the class and size of problems with which we are dealing. Yet despite using a direct matrix based solver, all the reported savings reported in Chapters 5 and 6 hold true even if an iterative solver is implemented instead.

In general, the computational cost of using direct solvers on a full matrix, such as Gauss elimination, is of $\mathcal{O}(N^3)$, with memory requirements of $\mathcal{O}(N^2)$. Direct matrix solvers therefore become too expensive to be implemented for large problems. Due to the sparsity of the matrix being solved, we used MATLAB's sparse matrix solver. Numerical tests showed the sparse solver to achieve a CPU scaling factor of approximately $\mathcal{O}(N^{1.5})$ (for our particular application, illustrated in Figure 4.12). The memory requirements of the solver to our mesh problems is approximately of $\mathcal{O}(100N)$.

In contrast, matrix free, iterative methods such as Krylov subspace solvers, with a good pre-conditioner can have computational costs of $\mathcal{O}(N \log N)$ with memory requirements of $\mathcal{O}(N)$ [10]. In MATLAB, there are several iterative methods to choose from, and the one that for our particular application yielded the best results is the symmetric LQ method [2]. The performance of iterative methods depend largely on the choice of pre-conditioners used. The use of pre-conditioners is often problem specific, and may for example be used to improve the stability of the solution when solving the set of Navier-Stokes equations for flow problems. In our application, we use a pre-conditioner to improve the computational time performance of the iterative solver by primarily reducing the number of required iterations to reach convergence.

To briefly illustrate the effect of pre-conditioners, we plot the CPU scaling for the symmetric LQ method in Figure 4.12 for no pre-conditioner and a very simple pre-conditioner. Consider a matrix problem of the form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{4.24}$$

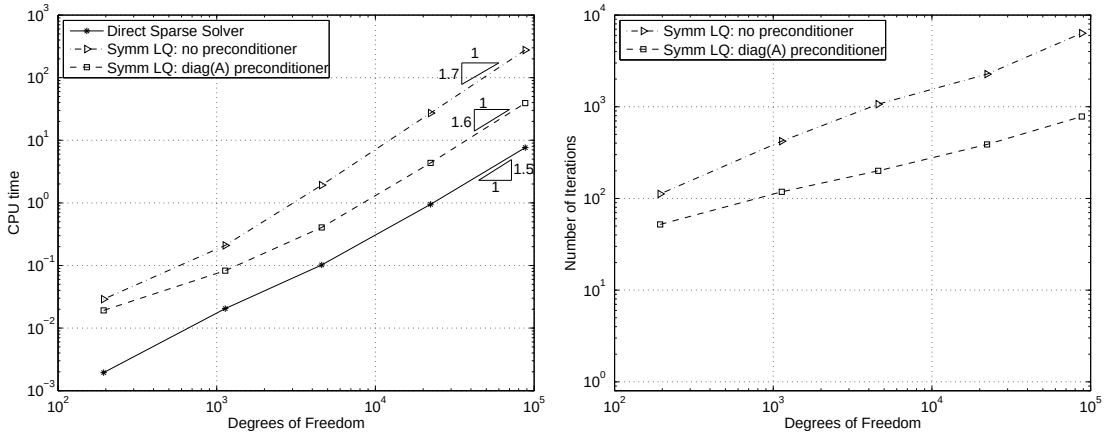
where $\boldsymbol{A}$ and $\boldsymbol{b}$ is a known matrix and vector respectively, solving for an unknown vector $\boldsymbol{x}$. The symmetric LQ method solves the problem in (4.24), given a positive definite pre-conditioner $\boldsymbol{M}$, by solving first for $\boldsymbol{y}$

$$\left(\sqrt{\boldsymbol{M}}\right)' \boldsymbol{A} \left(\sqrt{\boldsymbol{M}}\right)' \boldsymbol{y} = \left(\sqrt{\boldsymbol{M}}\right)' \boldsymbol{b} \tag{4.25}$$

then solves for $\boldsymbol{x}$ by

$$\boldsymbol{x} = \left(\sqrt{\boldsymbol{M}}\right)' \boldsymbol{y} \tag{4.26}$$

where $(.)'$ denotes the inverse of a matrix. The square root of a matrix is defined such that $\boldsymbol{M} = \sqrt{\boldsymbol{M}}\sqrt{\boldsymbol{M}}$, which exists if $\boldsymbol{M}$ is positive semi-definite [20]. The symmetric LQ function never fully computes the inverse, but approximates it iteratively. Based on equations (4.25) and (4.26) it may be noted if a pre-conditioner of $\boldsymbol{M} = \boldsymbol{A}$ is supplied, the symmetric LQ method would solve the matrix problem in a single iteration, provided that the computation of the inverse is exact. A good pre-conditioner choice for the symmetric LQ method is thus all the diagonal terms of matrix $\boldsymbol{A}$, provided $\boldsymbol{A}$ is diagonally dominant. Both these sets of results are shown in Figure 4.12, for both CPU cost and the number of iterations required to reach convergence. The convergence criteria used is that the normalized residual be less than 1×10^{-6}.



(a) CPU scaling. **(b)** Number of iterations.

Figure 4.12: a) CPU scaling and b) number of iterations required for the symmetric LQ iterative method for solving Newton's method, using different pre-conditioners.

In our application of the symmetric LQ method it appears as though the choice of pre-conditioner has little impact on the solution scaling of the problem, but does impact

the overall solution time. We do not quite attain CPU scalings of $\mathcal{O}(N \log N)$, but the results for the iterative solver is nonetheless competitive. Furthermore, for the size of problems we are dealing with, the direct sparse solver scales similarly but at a fraction of the cost. In terms of the solver technologies available in MATLAB, the direct sparse solver is hence the most efficient in terms of CPU time, provided the system is very sparse.

4.4 Mesh Movement Using Equations of Linear Elasticity

The final mesh movement technique which is evaluated is the elastic deformation method, where the motion of internal nodes is governed by a set of linear elasticity equations. The method is an example of mesh movement techniques based on the solution of a set of partial differential equations. Within the class of PDE based techniques, several methods have been proposed in an attempt to handle the compromise between quality of meshes produced, the extent of boundary deformations that can be handled and the associated cost.

The method of elastic deformation has not gained popularity as a mesh movement technique because for large deformations the method often leads to mesh entanglement or hanging nodes. To rectify the problem, Stein *et al.* [39] proposed the use of a Jacobian based stiffening factor. By introducing the stiffening factor, the amount of deformation of elements can be controlled.

It is possible to selectively treat the mesh deformation without the inclusion of this stiffening factor, by adjusting the Lamé constants of the elasticity equations. The inclusion of the stiffening factor however allows for a far greater degree of control. The stiffening factor causes smaller elements to be rendered "stiffer" in comparison to the larger elements and thus undergo a smaller degree of distortion due to the boundary deformation. Since most of the small elements in a mesh are usually located near the boundary, the method allows for mesh movements whereby the amount of element distortion near the flow boundary may be minimized to the desired level.

4.4.1 Model Formulation

4.4.1.1 Equations of Linear Elasticity and Finite Element Formulation

Let Ω be a spatial domain bounded by $\partial\Omega$. The domain boundary is partitioned into $\partial\Omega_u$ and $\partial\Omega_t$ corresponding to the Dirichlet and Neumann-type boundary conditions, where prescribed displacements $\bar{\boldsymbol{u}}$ and tractions $\bar{\boldsymbol{t}}$ are applied. The equation governing the internal node motion may be written as [17]

$$\nabla \cdot \boldsymbol{\sigma} + \boldsymbol{f} = \mathbf{0} \text{ in } \Omega, \tag{4.27}$$

$$\boldsymbol{\sigma} = \boldsymbol{C}\boldsymbol{\varepsilon} \text{ in } \Omega, \tag{4.28}$$

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\nabla \boldsymbol{u} + \nabla^T \boldsymbol{u}\right) \text{ in } \Omega, \tag{4.29}$$

$$\boldsymbol{u} = \bar{\boldsymbol{u}} \text{ on } \partial\Omega_u, \tag{4.30}$$

$$\boldsymbol{\sigma}\boldsymbol{n} = \bar{\boldsymbol{t}} \text{ on } \partial\Omega_t \tag{4.31}$$

where $\boldsymbol{\sigma}$ denotes the stress tensor, $\boldsymbol{f}$ the external force, $\boldsymbol{C}$ the fourth order elasticity tensor, $\boldsymbol{\varepsilon}$ the strain tensor, $\boldsymbol{u}$ the displacements and $\boldsymbol{n}$ an outward pointing unit normal.

Since it is in general not possible to solve the strong form of the boundary value problem (4.27)-(4.31), we use Galerkin's method to formulate the weak form

$$\int_\Omega \boldsymbol{\sigma} \cdot \nabla \mathbf{w} d\Omega = \int_{\partial\Omega_t} \bar{\mathbf{t}} \cdot \mathbf{w} d\beta + \int_\Omega \mathbf{w} \cdot \mathbf{f} d\Omega, \tag{4.32}$$

where $\mathbf{w}$ is an arbitrary weighting function such that $\mathbf{w} = \mathbf{0}$ along $\partial\Omega_u$. In the Finite Element method, we choose the weighting function to be similar to the displacement function,

$$\mathbf{u} = [N]^T \{U_e\} \text{ and } \mathbf{w} = [N]^T \{W_e\}, \tag{4.33}$$

where $[N]$ is the displacement shape functions, $\{U_e\}$ is the nodal displacement vector and $\{W_e\}$ the nodal weigthing function vector. Thus the global integrals of equation (4.32) takes the form

$$\int_\Omega [...] \, d\Omega = \sum_{\text{elements}} \int_\Xi [...]^e J^e d\Xi, \tag{4.34}$$

where Ξ represents the element local domain in terms of local co-ordinates $\boldsymbol{\xi}$, $[...]$ represents what is being integrated and J^e is the elemental Jacobian defined by

$$J^e = \det \left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}}\right)^e. \tag{4.35}$$

For the mesh movement, we make use of the constant strain triangle (CST) element [19], and thus no numerical integration is necessary. Furthermore, we make a plane stress assumption since we are dealing with two dimensional meshes only.

4.4.1.2 Jacobian Stiffening

The Jacobian stiffening factor alters the way in which the Jacobian is handled. We modify the finite element integration (4.34) as follows

$$\int_{\Omega} [...]\, d\Omega = \sum_{\text{elements}} \int_{\Xi} [...]^e \, J^e \left(\frac{J^0}{J^e}\right)^{\chi} d\Xi, \tag{4.36}$$

where χ is a non-negative stiffening constant, and J^0 is an arbitrary scaling constant inserted to make the alteration dimensionally consistent [39].

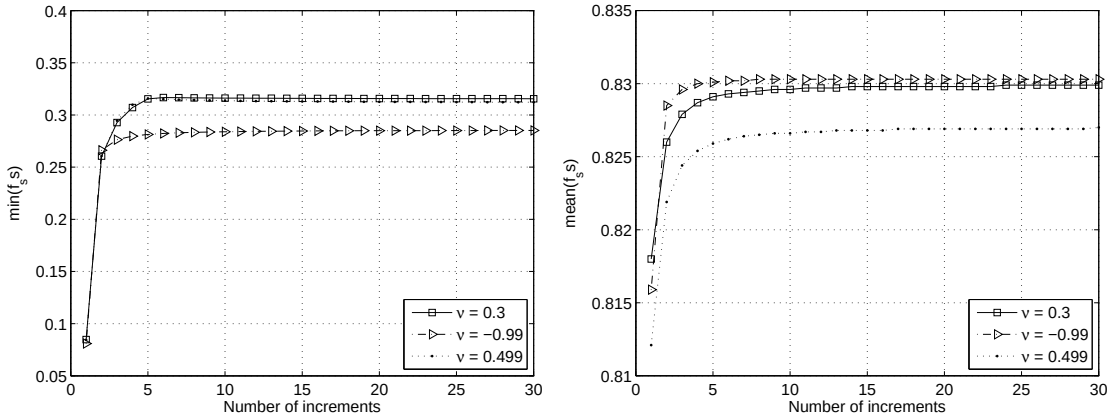
The Jacobian of an element is linked to the elemental volume, e.g. for the CST element $J^e = 2 \times \text{Area}$. Therefore, by inserting the stiffening factor, we cause smaller elements to stiffen up. The amount of elemental stiffening can be controlled by altering the stiffening constant χ. For $\chi = 0$ we retrieve the original FEM formulation, and with $\chi = 1$ we remove the Jacobian altogether. Furthermore, for $\chi = 2$ the smallest elements within the mesh will be forced to remain almost unchanged.

4.4.2 Rotation and Translation Test Case

To test the application of the linear elastic deformation method, the same mesh movement problem used in section 4.2.2, with the initial mesh shown in Figure 4.1 will be used. The inner rectangle once again will be translated in the x and y directions by 30 units and rotated a total of 60° counter-clockwise. We use CST elements to model the triangular mesh, and assume plane stress. For comparison purposes we make use of element quality metric 1 defined in section 4.3.1.1, where f_{ss} ranges between 0 for degenerate elements and 1 for the good elements. We move the boundary over 15 displacement increments.

To illustrate the dependency of the elastic deformation method on number of increments and Poisson's ratio, we plot the minimum and mean element qualities in Figure 4.13 for $\nu = -0.99$, 0.3 and 0.499 with a stiffening constant of $\chi = 1$.

To demonstrate the effect of the stiffening factor on mesh movement using the equations of linear elasticity, we set the stiffening constant $\chi = 0$, 1 and 2. The final meshes for the three stiffening constants are shown in Figure 4.14 (for Poisson's ratio of $\nu = 0.3$). By setting $\chi = 0$ we retrieve the standard Finite Element formulation, and it is evident that the smaller elements near the boundary perform poorly. They undergo large distortions and stretching, and there is severe tangling of elements near the tip of the inner rectangle. By setting $\chi = 1$, we essentially remove the Jacobian from the FEM formulation, resulting in the stiffening of the smaller elements near the boundary, and an overall improvement of the mesh. For a stiffening constant of $\chi = 2$, the smallest elements in the mesh remain essentially unchanged. Thus near the inner rectangle, we

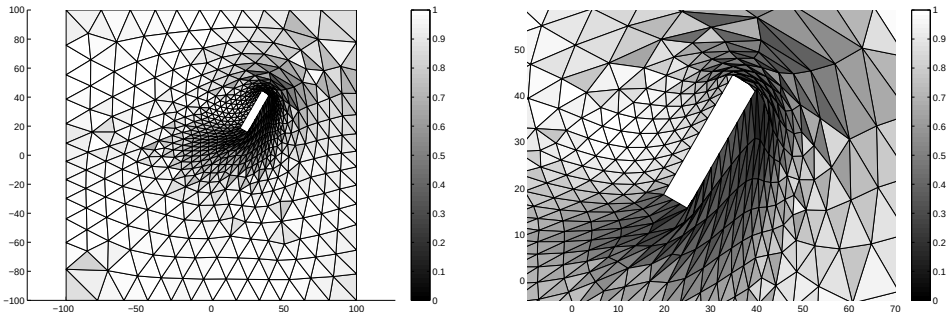


(a) Minimum element qualities. **(b)** Mean element qualities.

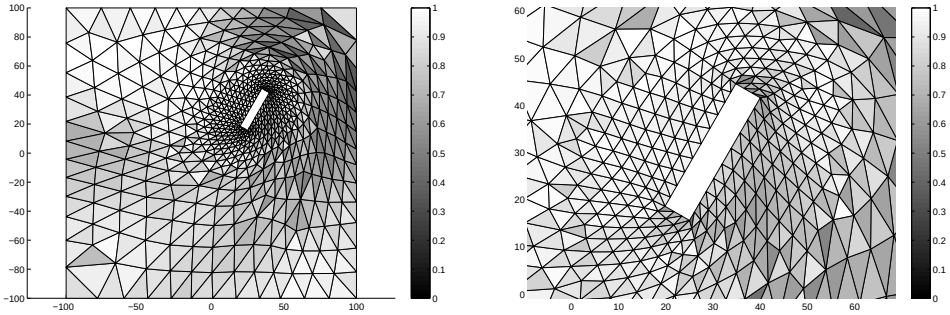
Figure 4.13: Comparison of the mean and minimum element qualities for the elastic deformation method as a function of the number of displacement increments and Poisson's ratio ν. Quality measured using $0 \leq f_{ss} \leq 1$.

obtain high quality elements, but the larger elements further away from the boundary are severely distorted and stretched.

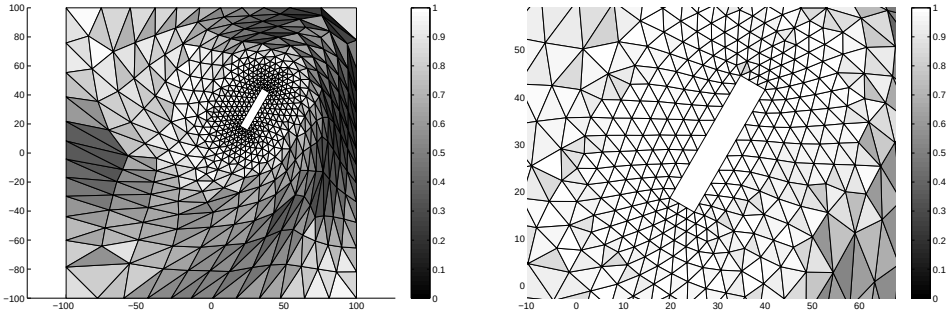
The mesh movement using equations of linear elasticity in conjunction with the stiffening factor allows the user fine control over the desired element qualities near the boundaries. Figure 4.15 depicts the minimum and mean element qualities of the mesh for the range of $0 < \chi < 2$. For $1 \leq \chi \leq 1.1$ we obtain a good compromise between minimum and mean element qualities for the rotation and translation test case.



(a) $\chi = 0$



(b) $\chi = 1$



(c) $\chi = 2$

Figure 4.14: Final meshes using equations of linear elasticity, after rotation and translation for different values of χ

Furthermore, the elastic deformation method performs well in comparison to other mesh movement techniques. In Table 4.5, we compare the minimum and mean element qualities, for the same mesh movement problem, using RBF interpolation, mesh optimization and now the linear elastic deformation. As expected, mesh optimization provides the highest element qualities, but the elastic deformation method performs better than RBF interpolation.

The elastic deformation method with Jacobian stiffening is a particularly effective mesh movement technique. It provides a high degree of control over the element qualities,

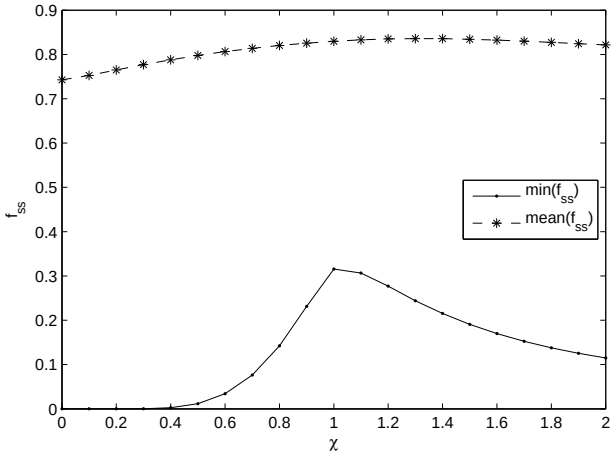


Figure 4.15: Element quality measure for rotation and translation test case for varying χ (see Figure 4.14).

No.	Mesh Movement	Minimum	Mean
1	RBF CP C^2	0.2724	0.8125
2	Opt: $\left(\frac{r_{\text{out}}}{r_{\text{in}}}\right)\left(\frac{1}{f_{\text{size}}}\right)$	0.3747	0.8306
3	Elasticity: $\chi = 1$	0.3157	0.8298

Table 4.5: Comparison of element qualities for rotation and translation test for all three mesh movement techniques. Mesh quality measure by metric 1 (Section 4.3.1.1).

and is computationally efficient, as will be shown in the following section.

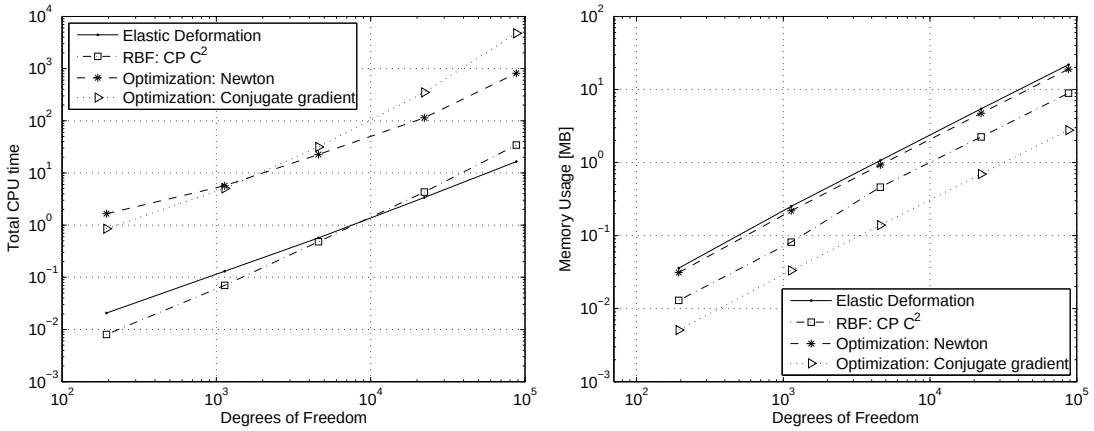
4.4.3 Computational Cost

For the solution of the linear elastic equations, we make use of the Finite Element method. To improve the memory usage of the solution we save only the non-zero entries of the stiffness matrix, and use a sparse matrix solver to find the displacements. To improve the computational performance we re-number the mesh nodal connectivity (using the reverse Cuthill-McKee ordering scheme [13]) to allow for a diagonally banded stiffness matrix, and use LU factorization to improve memory performance. The CPU and memory usage for the elastic deformation method is shown in Figure 4.16, and is compared to the RBF interpolation and optimization methods.

In terms of computational cost for large problems, the elastic deformation method is shown to be an efficient mesh movement method. For large problems the elastic deformation method outperforms RBF interpolation using a fixed support radius. It should however be noted, based on the discussion in Section 4.2.4 that for larger meshes the computational times using RBF can be improved by decreasing the support radius.

By decreasing the support radius, RBF interpolation does become the cheapest of the three methods with only a minor sacrifice in terms of final mesh qualities.

The elastic deformation method is however, despite the sparse implementation, the most memory intensive method of the three, though compares similarly to Newton's method. As with the Newton optimization routine, the system can be solved using an iterative solver, which would reduce the memory requirements, while increasing the overall solution times.



(a) CPU time scaling. **(b)** Memory usage scaling.

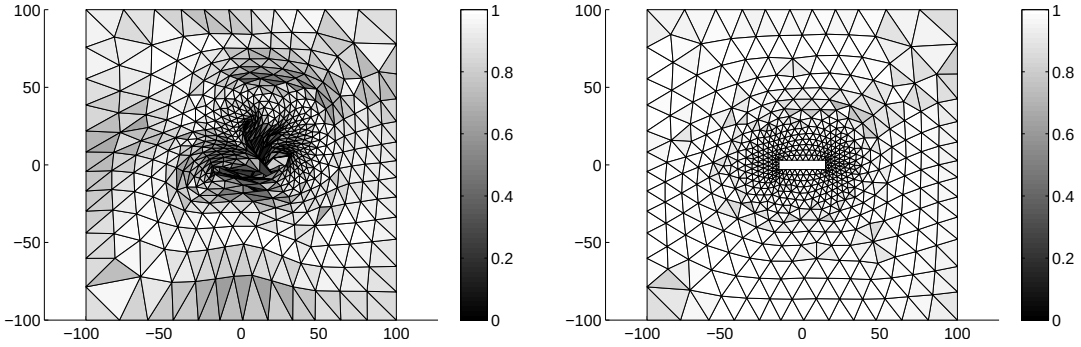
Figure 4.16: CPU and memory scaling comparisons of all three mesh movement methods.

4.4.4 Path Dependency of Linear Elastic Formulation

The elastic deformation method, like RBF interpolation, is a path dependent mesh movement scheme. To demonstrate this, we revisit the example used in Section 4.2.3, where the inner rectangle is rotated between $\pm 60°$ five times before returning to the original position. The test is performed using 18 and 80 intermediate steps, with the final meshes shown in Figure 4.17.

Using only 18 intermediate steps, the resulting mesh is completely tangled and consequently unusable. For 80 intermediate steps the minimum and mean element qualities is 0.6065 and 0.9637 respectively. This is a marginal decrease compared to the starting mesh which had minimum and mean element qualities of 0.7917 and 0.9743 respectively.

The elastic deformation method is path dependent due to the small strain and displacement assumption made for classic linear elastic analysis. For meshes undergoing large deformations, these assumptions are no longer valid, and in order to attain meshes of reasonable qualities the total displacements need to be subdivided into smaller increments. This path dependency can be removed by modifying the constitutive equations to a general non-linear elasticity formulation [44].



(a) 18 intermediate steps. Tangled mesh.

(b) 80 intermediate steps. Min: 0.6065, Mean: 0.9637.

Figure 4.17: Final meshes produced for the inner block rotated between $+60°$ and $-60°$ five times and back to the original position using the linear elastic deformation method.

Using a non-linear elastic formulation will have two effects. Firstly, for any given deformation there would exist a unique mesh solution. Much like the optimization methods, the non-linear elastic method will be capable of regularizing a mesh once it has been deformed. Unfortunately it will also suffer from the same drawbacks as the optimization method. The method itself will be more expensive than the standard linear elastic formulation as the solution involves solving a set of non-linear equations and no elements may be inverted prior to solving the constitutive equations. For this reason, the maximum boundary displacement per increment will also be limited to the size of the smallest element along the boundary edge, or an initial, inversion free guess of the mesh would need to be provided.

4.5 Conclusion

Three mesh movement methods were investigated, namely radial basis function interpolation, mesh optimization and elastic deformation. Of the three methods, mesh movement through optimization, offered the highest quality final meshes but proved to be computationally the most expensive. Elastic deformation, with the added Jacobian stiffening factor, was the fastest of the three with the second highest quality meshes. It was however demonstrated that the CPU scaling using RBF can be improved by decreasing the support radius size for compact support functions with only a small sacrifice in element quality.

For the optimization method, it was shown that using a Newton solver offered better CPU scaling in comparison to conjugate gradient methods. A sparse solver, with analytically determined gradients and Hessian were implemented and near one to one scaling

factors were obtained in terms of both CPU and memory scaling. Mesh optimization offers the advantage that for any given boundary displacement there exists a unique mesh. The method as implemented is not applicable to hybrid meshes and also suffers from the limitation that any given displacement increment cannot be larger than the smallest element lengths along the moving boundary. As such, large displacements has to be broken into many smaller displacement increments, each of which is an expensive full optimization problem or an initial inversion free guess needs to be provided.

Both the RBF interpolation and elastic deformation methods were demonstrated to be path dependent. While a single displacement increment can be used regardless of the motion magnitude, using larger number of displacement increments result in significantly improved mesh qualities. Due to the path dependency both these methods may experience mesh degeneration, especially for transient FSI simulations. The RBF interpolation method however requires no grid connectivity information and is therefore suited to hybrid meshes without any changes to the formulation or implementation.

Chapter 5

POD Based ROM for Mesh Movement

The aim of this chapter is to generate efficient POD based reduced order models of mesh movement algorithms.

5.1 Introduction

In Chapter 4 mesh movement techniques were introduced, namely radial basis function interpolation, mesh optimization and elastic deformation. All three techniques proved to be relatively costly operations. The aim of this chapter is to test the possibility of generating reduced order models of these mesh movement techniques based on the method of proper orthogonal decomposition, introduced in Chapters 2 and 3.

POD enables the definition of a lower-dimensional model to describe a higher dimensional system. In essence, it creates a subspace to the original system, which requires far fewer degrees of freedom to describe a much larger system. In order to apply the POD techniques to reduced order modeling requires that the system's basis modes be computed. To do so the model needs to be trained by generating a set of snapshots that is representative of the original system.

In terms of mesh movement, these snapshots are the mesh nodal coordinates associated with the meshes of a pre-defined set of domain boundary movements. Since a POD based model can only reproduce information within close proximity to the information in these snapshots, the boundary movements must be chosen as those most likely to be encountered in the actual simulation. This inherently means that certain *a priori* knowledge is required as to how the actual system will behave. Fortunately, in most engineering problems, a substantial amount of information relating to a problem, such as the predominant physics is usually known or can be estimated prior to the simulation. If no such knowledge exists, there are several techniques available, such as Latin hypercube sampling, that can be used to aid in the choice of snapshots.

The generation of a POD based model for mesh movement can be viewed as a pre-processing step to the actual simulation. Once the initial cost of generating the snapshots and determination of the orthogonal modes have been completed, the mesh movement problem becomes trivially cheap. The second advantage is that once a POD model has been generated, it can be applied repeatedly, whether it is for the same problem or a problem with the same domain but different simulation parameters. There exists, as an example, many scenarios where a simulation is to be run on the same flow domain, with different model parameters such as inlet velocities, density, viscosity or even structurally related properties of the FSI problem. Using the conventional mesh movement techniques would imply that a large fraction of CPU time is spent on solving the mesh movement problem that would be similar in all these simulations. On the other hand, once a POD model has been trained, it can be used in all these cases, at a mere fraction of the cost.

In order to demonstrate the POD technique applied to mesh movement, the simple rotation and translation of an inner rectangle used in sections 4.2.2 and 4.3.2 will be used again. For the purposes of this study we will attempt to generate reduced order models of the optimization and elastic deformation mesh movement methods.

5.2 Snapshot Generation for Simple Rotation and Translation Test Case

The first requirement for generating a POD model is to acquire a set of snapshots. When generating the snapshots, it is crucial that we capture the kind of mesh movements that is representative of the expected boundary movements. Since the motion of the internal rectangle is merely a benchmark problem, and not related to any actual physical problem, there is no possible prior knowledge as to what the rectangle motion is going to be. Our choice of snapshots is merely based on what we will attempt to reproduce. For example, if we wish to use the POD model to rotate the rectangle, then our training snapshots will be of the inner rectangle at various degrees of rotation.

For this particular test, we however wish to produce or generate a POD model that can translate and rotate the inner rectangle anywhere within a ± 40 translational and $\pm 90°$ rotational region in the square domain. To do so, we require snapshots of appropriate meshes for various instances of the inner rectangle translated through several x, y coordinates and angles of rotation. Rather than simply choosing a set of random motions and rotations, we will make use of the statistical method of Latin Hypercube sampling (LHS) [30, 40].

Latin hypercube sampling is often used in uncertainty analysis, and was developed primarily to generate a distribution of plausible collections of parameter values from a multidimensional distribution [30]. To aid in the definition of LHS, let us consider a Latin Square. A Latin square is a statistical sampling where the parameter domain is

divided into a square grid, where sample points are distributed within this grid. The square is considered a Latin square if and only if there exists only one sample point within each row and column. Figure 5.1 is a representation of a Latin square with 4 sample points. The Latin Hypercube is the generalization of this concept, where the dimension of the cube is extended to an arbitrary amount N, where N is the number of parameters. In the Latin cube, each axis-aligned hyperplane contains only one sample point. Each of the parameter domains is then divided into M equally spaced intervals, where M is the number of sample points used.

A detailed discussion of LHS, and computation methods for the sample points are available in [40], or can readily be found in several other literature sources.

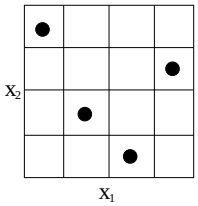


Figure 5.1: Latin Hypercube with number of variables $N = 2$, and number of sample points $M = 4$.

To illustrate the benefit of LHS, let us consider a cube with two parameters, x_1 and x_2, and 1000 sample points, where the values of x_1 and x_2 range continuously between 0 and 1. The sample points along with histogram plots of x_1 and x_2 as computed by LHS are shown in Figure 5.2. Using LHS, the sample points are chosen randomly, with the exception that the distribution of each parameter is chosen uniformly along the length of the parameter intervals, which is illustrated by the histogram plots.

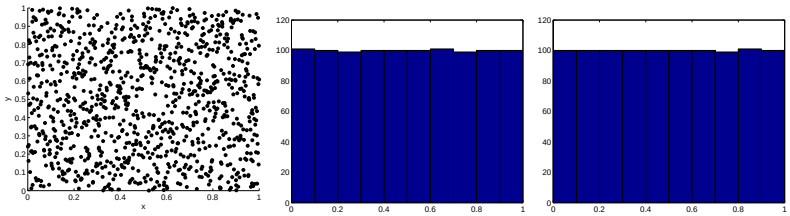


Figure 5.2: Example of a 2D Latin hypercube sampling with 1000 data points ranging between 0 and 1. (a) x and y coordinates of data points. (b) Histogram plot of x_1 dimension. (c) Histogram plot of x_2 dimension.

We use LHS to generate a set of 60 data points for three variables, x and y translation and rotation ϕ; we then generate a snapshot for each of these data points. The number of snapshots necessary is not know prior to an actual analysis, and the choice of 60 snapshots is selected rather arbitrarily. Because of the relative uniform distribution

offered by the Latin hypercube sampling, we should in theory now have information covering most combinations of translations and rotations in the sampled region. It should be noted that the sampling was taken for a maximum x and y translation ranging between $+40$ and -40 and rotations between $+90°$ and $-90°$. These values represent the limit of the mesh movement algorithms for this particular mesh; any movements greater than these values result in meshes that contain one or more elements with too poor a quality for the effective use in simulations.

5.3 POD Model of Optimization Mesh Movement

The snapshots are generated using the optimization based method of Chapter 4, where RBF interpolation is used to initially move the mesh from the origin to the Latin hypercube specified points.

Once we have our snapshots, we can now compute our POD basis modes, as described in Chapter 2. The mesh coordinates may be approximated by the summation of expansion coefficients and the POD modes

$$\boldsymbol{x} \approx \sum_{i}^{M} \alpha(\boldsymbol{x}_b)_i \boldsymbol{\varphi}_i, \tag{5.1}$$

where $\boldsymbol{x}$ is a vector containing the x and y nodal coordinates, $\boldsymbol{\varphi}$ is the POD modes and $\boldsymbol{\alpha}(\boldsymbol{x}_b)$ is the expansion coefficients as a function of the boundary nodal coordinates, $\boldsymbol{x}_b$.

With the POD modes known, all that is needed for the ROM to fully describe the mesh nodal coordinates is the expansion coefficients, $\boldsymbol{\alpha}$. These coefficients have to be computed by some means when the domain boundary is moved. In the original mesh optimization method, we optimized the mesh quality with respect to an objective function $F(\boldsymbol{x})$ defined in terms of the inner nodal coordinates:

$$\underset{\text{w.r.t. } \boldsymbol{x}}{\text{minimize}}\, F(\boldsymbol{x}),\ \boldsymbol{x} = [x_1, x_2, ..., x_n]^T, \tag{5.2}$$

where the total degrees of freedom of the problem was twice the number of nodal coordinates. Using the POD modes, we now have a subspace of the original problem, where we can now solve (5.2), in terms of only the expansion coefficients, and consequently substantially fewer DOF. The original optimization problem is solved in terms of $\boldsymbol{\alpha}$ by substituting (5.1) into (5.2), such that we minimize

$$\underset{\text{w.r.t. } \boldsymbol{\alpha}}{\text{minimize}}\, F(\boldsymbol{x}(\boldsymbol{\alpha})),\ \boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_M]^T. \tag{5.3}$$

The total DOF of the problem to be solved is now equal to the number of modes M that we decide to retain. If we study the eigenvalues associated with the POD

modes (Figure 5.3), we note that the first 10 modes are dominant, and contain over 99.99% of the system information. The system can therefore be accurately reproduced by retaining only the first 10 most dominant modes. This means that our systems DOF will be reduced from $2N$ to around 10, which signifies a remarkable decrease in computational cost.

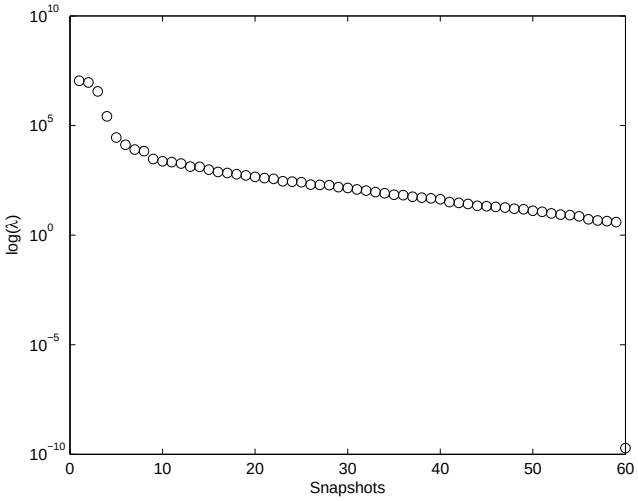


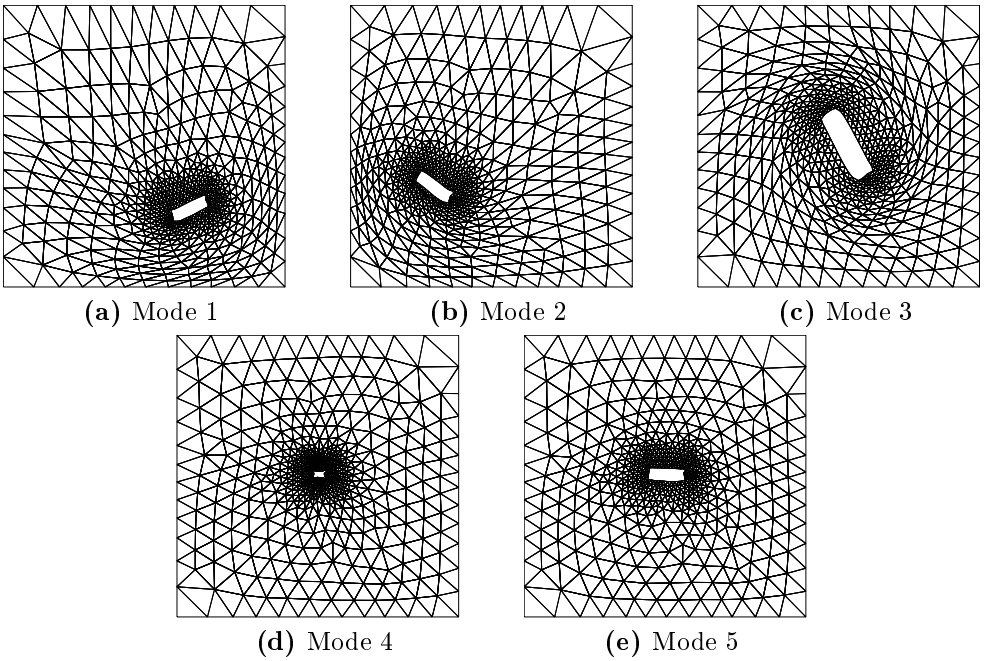
Figure 5.3: Ordered eigenvalues for associated POD modes for the 60 snapshots.

To briefly illustrate the contributions made by each of the POD basis modes, representations of the first 5 modes are shown in Figure 5.4. The first two modes appear to contain the predominant system information, and handles all the x and y translations as well as some rotation information for the inner block movement. The third POD mode contains information with regards to the block rotation and expands elements away from the inner block, and mode 4 controls the degree of element contractions around the inner rectangle. The type of information contained within modes 5 and upwards cannot be expressed in terms of simplistic motions, however the magnitude of their contributions is minimal in comparison to the first 4 modes.

It should be noted that the basis modes representations in Figure 5.4 does not include the inner rectangle boundary. The boundary itself is prescribed and as such does not form part of the information contained within the POD modes. The external boundary is also prescribed, however since they are known and for this problem remain unchanged are included in the basis modes plots to assist in providing a concrete and easy to understand definition of the boundary.

The expansion coefficients $\boldsymbol{\alpha}$ scale the magnitude and directions of each POD mode to the contribution of the overall solution. There is however a slight problem in trying to solve the mesh movement by optimizing (using equation (5.3)) for the coefficients $\boldsymbol{\alpha}$. The cost function in terms of the coefficients is populated with multiple local minima. To illustrate, a random search direction is chosen and the function $F(\boldsymbol{x}(\boldsymbol{\alpha}))$ is evaluated

along the path. The plot of the function profile is shown in Figure 5.5. Because of these local minima, there is no gradient based optimization routine available that will guarantee convergence of the solution to the global minimum, unless the starting points are chosen to be close to the exact solution, or more appropriately, within the valley of the global minimum.



(a) Mode 1 **(b)** Mode 2 **(c)** Mode 3

(d) Mode 4 **(e)** Mode 5

Figure 5.4: Representations of the first 5 POD modes. Magnification factor of 0.3.

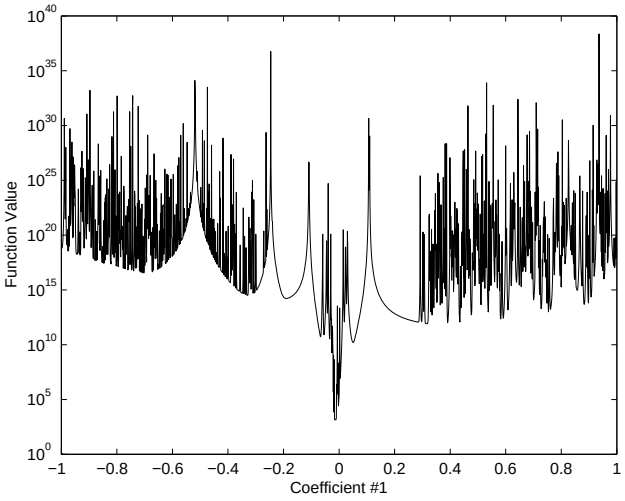


Figure 5.5: Function profile of the cost function, $F(\boldsymbol{x}(\boldsymbol{\alpha}))$.

The presence of these local minima is attributed to element inversion. For any instance

in which there are elements within the mesh that are inverted a local minimum will exist, for example the mesh depicted in Figure 5.6. The mesh itself consist of high quality elements, save for the inverted elements located along the inner rectangle boundary. A similar problem exists within the full order mesh optimization, but never arises as a computational issue. Although the optimization algorithm individually moves each of the nodal coordinates, the cost function gradient provides search directions that never move nodes to locations that will result in element inversion. When defining the objective cost function in terms of the coefficients $\boldsymbol{\alpha}$, these local minima are an acute problem, because even a minor change in one of the dominant expansion coefficients, has an effect on all the nodal coordinates.

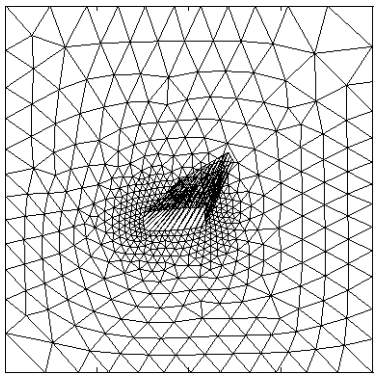


Figure 5.6: Example of a local minimum.

We propose using an interpolation method to obtain an approximation to the expansion coefficients $\boldsymbol{\alpha}$ as an initial starting guess for the optimization algorithm. Interpolation would then be used to equate a function relating the expansion coefficients to the moving boundary coordinates based on information gathered from the generated set of snapshots.

The most commonly used method for generating surrogate surface models is the method of Least Squares (LS) using some polynomial basis [33]. Briefly, LS attempts to find an approximate description $f(\boldsymbol{x})$ that approximates, in the least-square sense, a function f_i at locations $\boldsymbol{x}_i$ where $i = 1, ..., N$. The approximate function is defined as

$$f(\boldsymbol{x}) = \boldsymbol{b}(\boldsymbol{x})^T \boldsymbol{c}, \tag{5.4}$$

where $\boldsymbol{b}(\boldsymbol{x}) = [b_1(\boldsymbol{x}),\, b_2(\boldsymbol{x}),\, ...,\, b_k(\boldsymbol{x})]^T$ is the polynomial basis vector and $\boldsymbol{c} = [c_1,\, ...\,,\, c_k]^T$ is a vector of unknown coefficients that need to be solved. The coefficients $\boldsymbol{c}$ can be solved by solving the following matrix problem:

$$\sum_i \boldsymbol{b}(\boldsymbol{x}_i)\boldsymbol{b}(\boldsymbol{x}_i)^T \boldsymbol{c} = \sum_i \boldsymbol{b}(\boldsymbol{x}_i) f_i. \tag{5.5}$$

Different approximations can be obtained by using various order polynomials, for example a linear polynomial with three dimensions would be $\boldsymbol{b}(\boldsymbol{x}) = [1, x, y, z]^T$ or a second order polynomial with two dimensions is $\boldsymbol{b}(\boldsymbol{x}) = [1, x, y, xy, x^2, y^2]^T$.

In our particular application, the LS method with linear polynomials provides unsatisfactory interpolations of the expansion coefficients. To demonstrate the ability of interpolation methods to approximate the coefficients we move the inner rectangle along a predefined path. The path is depicted in Figure 5.7, where the inner rectangle moves from $\triangle x = 30$, $\triangle y = 30$ and $\phi = 60°$CCW (position A), through the origin (B) to $\triangle x = -30$, $\triangle y = -30$ and $\phi = 60°$CCW (C). The approximations of the first 6 coefficients are shown in Figure 5.8, along with the exact values of the respective expansion coefficients $\boldsymbol{\alpha}$.

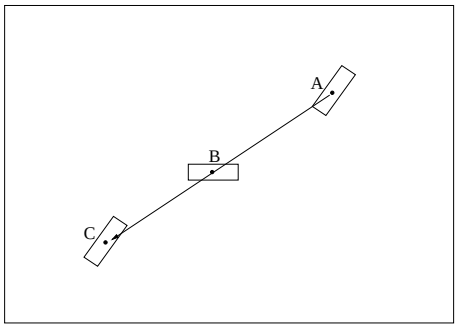
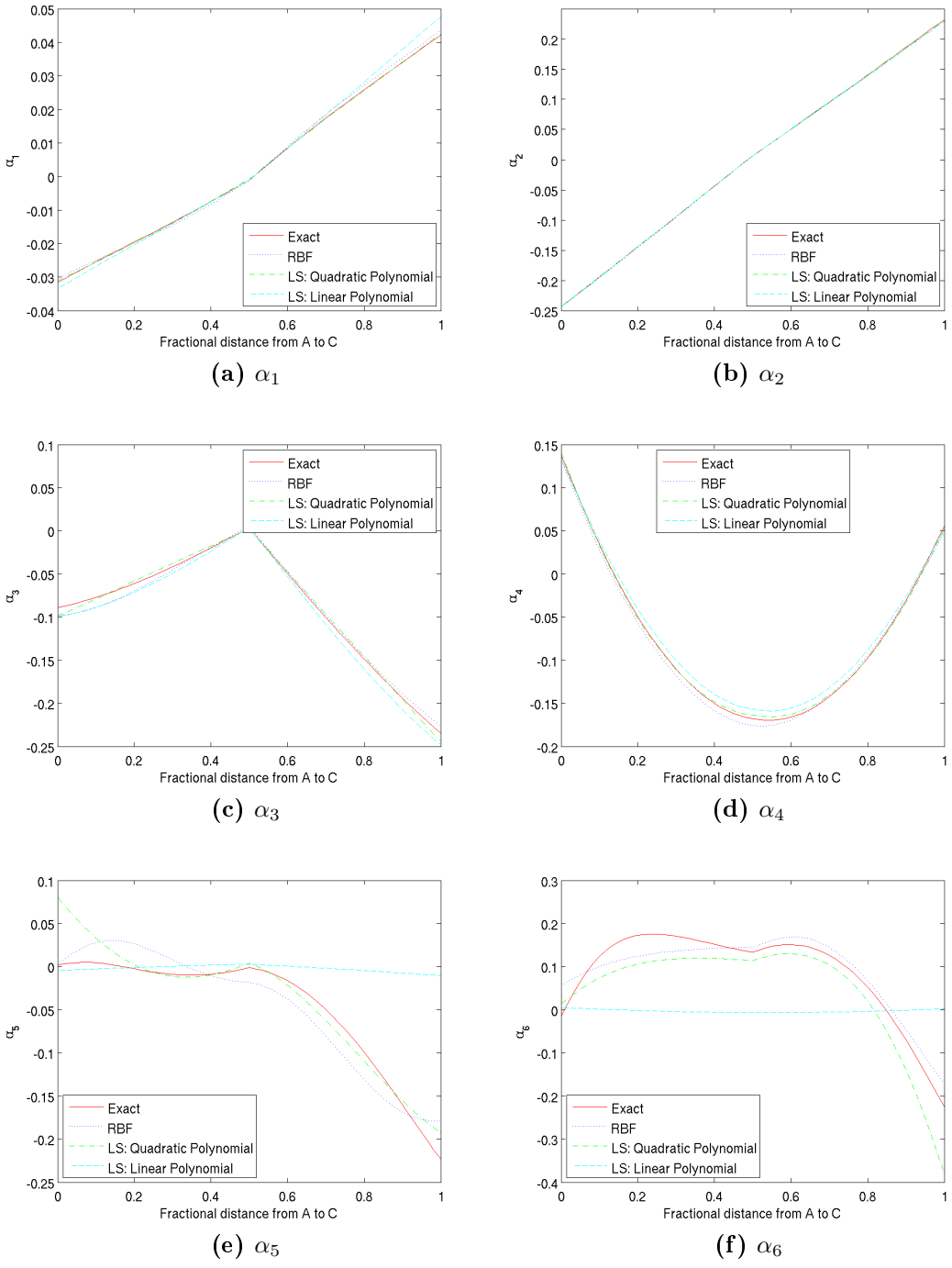


Figure 5.7: Pre-defined path for the motion of the inner rectangle between $\triangle x = 30$, $\triangle y = 30$, $\phi = 60°$CCW to $\triangle x = -30$, $\triangle y = -30$, $\phi = 60°$CCW . The expansion coefficients $\boldsymbol{\alpha}$ for the POD model along the path is shown in Figure 5.8.

From the coefficient plots in Figure 5.8, it may be noted that the linear LS function adequately approximates the first 4 expansion coefficients. However, from α_5 onwards, the relationship between boundary positions and expansion coefficients become too non-linear for a linear basis polynomial. This approximation can be significantly improved by using a higher basis polynomial; the approximations based on a quadratic polynomial is also shown in Figure 5.8. It is certainly conceivable, that using a cubic, or even higher polynomial basis would provide even better approximations.

Using higher order polynomials does however have serious cost implications. For example, for a 2D mesh, the size of the matrix problem (equation (5.5)) to be solved using a linear polynomial is of size $[2n_b + 1] \times [2n_b + 1]$, where n_b is the number of boundary nodes used to generate the approximation. In contrast, using a quadratic polynomial, the matrix problem size increases to $[n_b^2 + 3n_b + 1] \times [n_b^2 + 3n_b + 1]$, where each matrix problem will have to be solved K times, where K is the number of expansion coefficients to be solved for. The cost implications can become rather severe, especially considering that the matrices to be solved consist entirely of non-zero entries. As an example, for a problem with 100 boundary nodes, a linear approximation requires the solution of a 201×201 size matrix, compared to a 10301×10301 sized matrix using a quadratic polynomial basis.



(a) α_1 **(b)** α_2

(c) α_3 **(d)** α_4

(e) α_5 **(f)** α_6

Figure 5.8: Plots of the first 6 expansion coefficients $\boldsymbol{\alpha}$ along the predefined path illustrated in Figure 5.7.

The costs are not that severe if one considers that the matrix problem only has to be solved once off, and can be done using an iterative solver. Furthermore, it is not neces-

sary that all the boundary nodes be used in the interpolation function. For example, in our problem with the rectangle, only the corner nodes need be used, as they completely describe the position and orientation of the block. On the other hand, if the boundary surface is more complex, for example an aircraft wing, only every n^{th} node needs to be accounted for, where n can be changed to include all the nodes or only a small number of key nodes.

Having said that, it would be preferable if an interpolation method is used that can generate relationships between highly non-linear data, while only using a linear polynomial basis. Several such methods are available, to name but a few, include the Weighted Least Squares (WLS), Moving Least Squares (MLS) [24, 33] and RBF interpolation. In contrast to the classical LS, the WLS and MLS do not approximate the data in a global sense, but rather locally with a weighting function. The WLS approximations are computed around discrete points, whereas the MLS is computed continuously over the domain.

While both the WLS and MLS can effectively be used to describe highly non-linear relationships between data, another effective interpolation method is through radial basis functions. Since RBF interpolation has already been used in the preceding part of the work, a fully functioning RBF code is already available, and is hence the interpolation method chosen. The approximations produced using RBF interpolation, with CP C^2 radial basis functions, is also shown in Figure 5.8. RBF provides as good an approximation as 2nd order LS, while the size of the matrix to be solved is only $[2n_b + 1] \times [2n_b + 1]$.

Using RBF interpolation, we now have an effective means by which to approximate the POD coefficients. These approximations can now be used as the initial starting point for the optimization routine. The RBF approximations are in fact so good, that further optimization of the expansion coefficients is in most cases unnecessary, unless a very accurate solution is desired.

5.3.1 Results of Simple Translation and Rotation Problem

To summarize, at this point we generated 60 snapshots of the mesh at various positions and angles of rotation based on hypercube generated sampling points. The POD basis modes were then computed based on these snapshots. We then suggested two ways for which to solve for the POD coefficients: i) to approximate the coefficients using RBF interpolation, and ii) to use these approximations as an initial starting point for an optimization routine to solve equation (5.3).

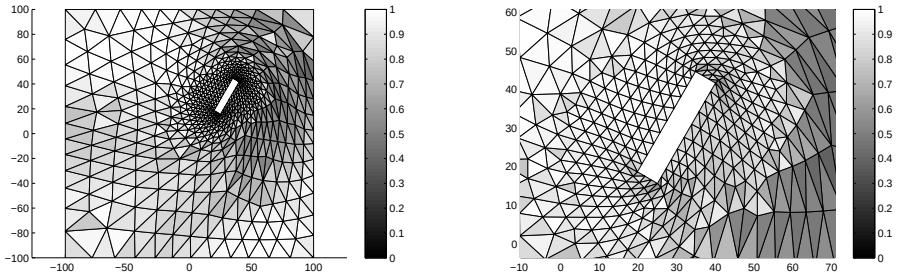
It remains to quantify how well the POD model is able to reproduce the meshes for various boundary motions. The optimization routine that will be made use of for the test is the Newton-Raphson method. The DOF for the ROM is small enough that it will be computationally efficient, and we gain from second order convergence.

Secondly, since the starting point for the optimization will be a good one, we are mostly guaranteed of convergence.

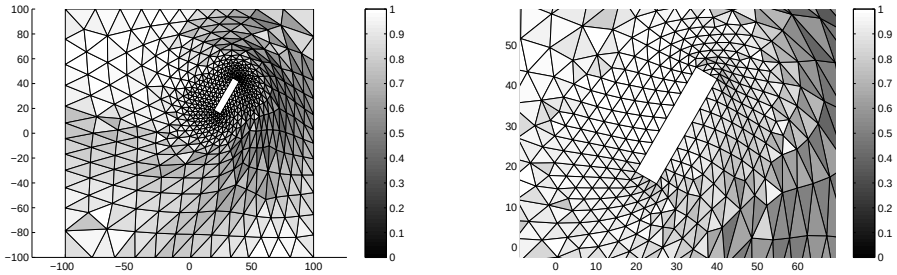
To demonstrate the ability of the POD based ROM, the inner rectangle is translated by 30 units in the x and y directions and rotated by 60° counter clock-wise. The meshes produced by the ROMs for the mesh movement are shown in Figure 5.9. From the figure, we note that the POD based ROM is capable of reproducing the full order mesh movement. It may also be noted that the approximations based on the RBF interpolation is in fact very good.

Comparing the element qualities of the ROM with the full order optimization, we find that using only interpolation of the coefficients the mean percentage difference in element qualities is 0.3477%, and using Newton to further optimize the coefficients we find a difference of 0.0274%. Figure 5.10 presents histograms comparing the percentage difference in terms of each element size and shape for the mesh generated by the ROM and the full order mesh optimization. Using only RBF interpolation we find that 97.83% of the elements' size differ by less that 5% and 86.11% of the elements' shapes differ by less than 5%. There is only a minor improvement in the use of Newton optimization, with 98.68% and 88.28% of the elements' size and shape respectively differing by less than 5%. The histograms show that some of the elements experience either a positive or negative size and shape change. The positive changes indicate that the element shape and sizes improved in comparison to the full order optimization. Notice that the histograms are centered about zero, indicating that on average the mesh quality does not deteriorate significantly.

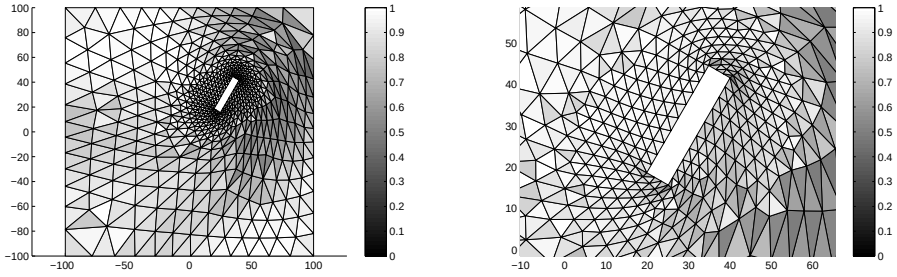
The initial results appear to be promising. If we further consider that the inner rectangle motion of $\triangle x = 30$, $\triangle y = 30$, $\phi = 60°$CCW is close to the edge of the training region specified by the Latin hypercube sampling points, the results are more attractive. POD basis modes are far better at interpolating between information contained within the snapshots, as opposed to extrapolating. Figure 5.11 presents histograms comparing the difference in element sizes and shapes for an inner rectangle motion of $\triangle x = 10$, $\triangle y = 10$, $\phi = 10°$CCW, which lies further in the interior of the training region. We find that all the element sizes and shapes, for both the use of coefficient interpolation and Newton optimization, differ by less than 5%.



(a) Full order mesh optimization.

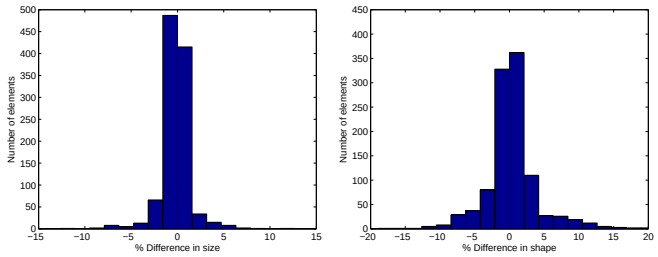


(b) POD ROM with coefficient interpolation only.

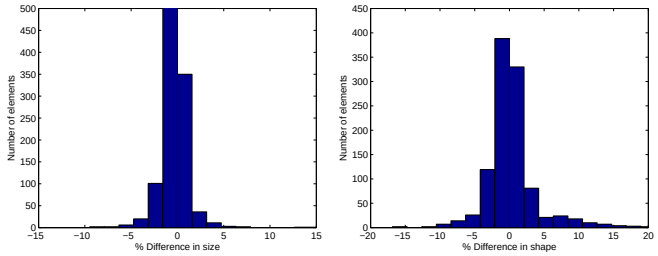


(c) POD ROM using Newton's method to solve the coefficients $\boldsymbol{\alpha}$.

Figure 5.9: Comparison of final meshes after rotation and translation of full order optimization and ROM using 10 coefficients.

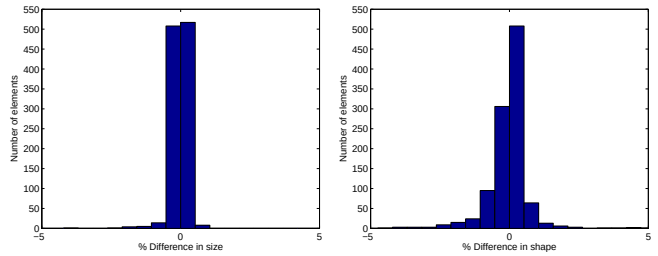


(a) POD ROM with coefficient interpolation only.

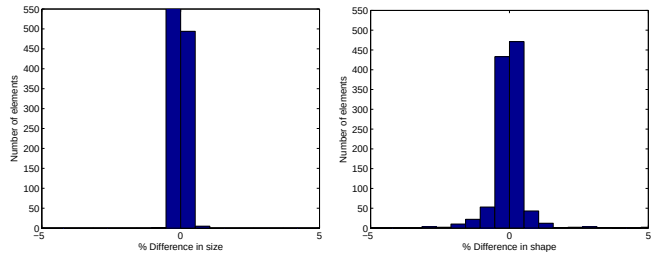


(b) POD ROM with Newton's method to solve for coefficients.

Figure 5.10: Histogram plot of the percentage change in shape and size for each element in the mesh of the ROM compared to the full order mesh optimization. For $\triangle x = 30$, $\triangle y = 30$, $\phi = 60°$CCW.



(a) POD ROM with coefficient interpolation only.



(b) POD ROM with Newton's method to solve for coefficients.

Figure 5.11: Histogram plot of the percentage change in shape and size for each element in the mesh of the ROM compared to the full order mesh optimization.For $\triangle x = 10$, $\triangle y = 10$, $\phi = 10°$CCW.

Of more interest to us than the actual differences in terms of shape and size, is whether the meshes produced by the reduced order models are good enough to be used in a simulation. To this end, in Table 5.1 we summarize the element qualities for the ROMs as well as for the full mesh optimization, for inner rectangle movements ranging from well within the training region to the extremes of the snapshot training region. The element qualities in Table 5.1 are the minimized cost functions based on quality metric 4 (Section 4.3.1.4), which was the cost function that was minimized.

				Quality Metric 4: $\sum_{\text{elements}} \left(\frac{r_{\text{out}}}{r_{\text{in}}}\right)\left(\frac{1}{f_{\text{size}}}\right)$		
No.	$\triangle x$	$\triangle y$	$\phi°$ CCW	Full Optimization	ROM Coefficient Interpolation	ROM Optimization
1	0	0	0	3.9538×10^3	3.9561×10^3	3.9548×10^3
2	10	10	10	4.0227×10^3	4.0281×10^3	4.0249×10^3
3	20	20	20	4.2231×10^3	4.2339×10^3	4.2261×10^3
4	30	30	60	5.0024×10^3	5.0750×10^3	5.0527×10^3
5	40	40	90	6.3248×10^3	6.3782×10^3	6.3624×10^3

Table 5.1: Mesh quality for various inner rectangle movement based on mesh quality Metric 4.

Unfortunately the cost function metric is difficult to interpret, as the values themselves, ranging from 1 to ∞, have little meaning save that they should be as low as possible. For this reason, in Table 5.2 we report the minimum and mean element qualities, but now using $0 \leq f_{\text{ss}} \leq 1$. This quality metric is well bounded and hence allows for intuitive comparisons to be made, where values of 0 and 1 implies degenerate and perfect elements respectively.

Based on the results in Table 5.2 it may be noted that for all the positions both the ROM based on coefficient interpolation and optimization provide adequate meshes and compare favorably with the full optimization method. The only exception is for the extreme boundary movement ($\triangle x = 40$, $\triangle y = 40$, $\phi = 90°$CCW), for which the minimum element quality produced by ROM coefficient interpolation is somewhat less than that of the ROM optimization and full optimization.

				Full Optimization		ROM Coefficient Interpolation		ROM Optimization	
No.	$\triangle x$	$\triangle y$	$\phi°$ CCW	min (f_{ss})	mean (f_{ss})	min (f_{ss})	mean (f_{ss})	min (f_{ss})	mean (f_{ss})
1	0	0	0	0.7844	0.9780	0.7799	0.9777	0.7806	0.9779
2	10	10	10	0.7329	0.9648	0.7067	0.9641	0.7230	0.9645
3	20	20	20	0.5753	0.9312	0.5265	0.9305	0.5824	0.9308
4	30	30	60	0.3812	0.8297	0.3770	0.8326	0.3563	0.8299
5	40	40	90	0.2789	0.7898	0.1393	0.7869	0.2485	0.7855

Table 5.2: Element qualities for various movements of the inner rectangle. $0 < f_{\text{ss}} < 1$.

The fact that the ROM based on coefficient interpolation produces results that so closely matches those of ROM based on Newton optimization is probably not that surprising.

The objective cost function in terms of the expansion coefficients is populated with a large number of local minima (Figure 5.5). We use RBF interpolation to gain an approximation of the POD expansion coefficients and use these approximations as the starting point for Newton's method. Newton's method will only be able to find the minimum of the valley in which the RBF interpolation places the starting guess, and there is no guarantee that this valley in fact contains the global minimum. The results obtained by the POD ROMs are however accurate enough to suggest that our computations of the expansion coefficients are at the very least close to the global minimum.

RBF interpolation however provides sufficiently good results that it is not necessary to further optimize the coefficients, except perhaps in extreme cases. An additional benefit in using RBF to interpolate the POD expansion coefficients is that we can move the boundary to any position in a single increment regardless of the size of displacement. In the full order optimization, if the boundary displacement is larger than the smallest element along the boundary, we have to divide the displacement into smaller increments. Thus, if we use a fine mesh, and large displacements are experienced we need to perform the expensive full optimization many more times, or use RBF interpolation to initially move the mesh, which is still fairly expensive. The ROM has no such limitations.

Figure 5.12 shows the mean difference in element qualities between the ROMs and the full order mesh optimization as a function of the number of coefficients used in the approximation. From the figure, it may be noted that the ROM can accurately reproduce the full mesh movement by retaining as few as 4 POD basis modes. This is perhaps not entirely suprising for this problem since there are really only a small number of degrees of freedom in the motion, i.e. 2 translations and 1 rotation. The implications thereof is that the total problem size to be solved is then essentially reduced from $2N$ down to just 4.

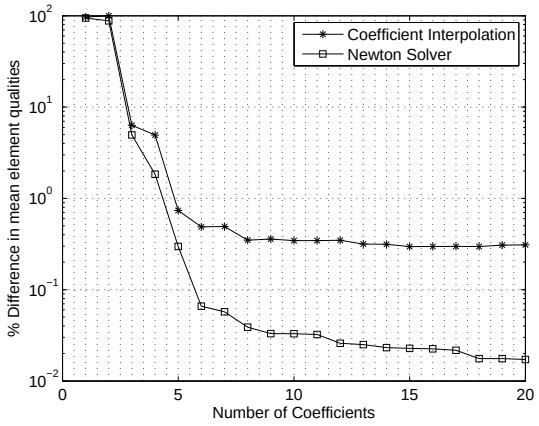


Figure 5.12: Plot of percentage difference in mean element qualities for the ROM compared to the full order optimization, as function of the number of coefficients. Inner rectangle movement of $\triangle x = 30$, $\triangle y = 30$ and $\phi = 60°$CCW.

The final question that remains to be answered is how many snapshots is an appropriate

choice. Unfortunately there is no way of knowing prior to generating a POD model how many snapshots are actually necessary to sufficiently train a ROM. To demonstrate the dependence of the ROM on the number of snapshots, we plot the difference in mean element qualities between the ROM based on coefficient optimization and the full order mesh movement method in Figure 5.13 as a function of the number of snapshots. The mean difference reported on is the average value of the mean element differences for the inner rectangle at four different positions within the domain. Furthermore, to ensure that we maintain a consistent comparison, all LHS points are saved for each successive addition of snapshots, and the number of LHS data points (number of snapshots) is doubled to ensure that our LHS data distribution retains uniformity.

From the plot, it may be noted that the initial choice of 60 snapshots is far more than necessary. In fact, as few as 5 snapshots are sufficient to adequately describe the system. In retrospect, this is perhaps not that surprising, considering that our LHS specified snapshots are uniformly distributed throughout the domain, and the POD method extracts from this information the predominant system variances. Considering further that the first 4 modes contain all the really important system information, no more than 5 snapshots and 4 retained POD modes are necessary.

These properties are however entirely dependent on the type of problem that one attempts to approximate, and the choice of training snapshots. Unless significant information is known about a system prior to a simulation there will always be a risk of either generating too many snapshots, or more detrimentally, too few. Naturally there also exists the possibility of generating snapshots that do not provide sufficient descriptions of the system on hand, or spending valuable computational time generating snapshots of scenarios that never feature within the real simulation.

In an attempt to alleviate this concern, a novel adaptive training procedure is introduced in Chapter 6. The training procedure allows for the POD method to be used in conjunction with a full order mesh movement scheme and generate snapshots only when necessary. The training procedure therefore will require no prior knowledge of a system, or the expected behavior thereof.

5.3.2 Computational cost of the ROM

Figure 5.14 is a comparison of the CPU time required for the ROM based on coefficient interpolation and optimization, and is compared to the cost of the full order optimization. Both ROMs are based on 10 expansion coefficients, and the gradients and Hessian required by Newton's method for the coefficient optimization are computed using finite differencing.

The figure highlights why it is such an attractive option to use RBF interpolation only to compute the expansion coefficients. The computational time for the interpolation remains almost unchanged for an increase in problem size; the cost is only a function of the number of coefficients used, which is kept constant at 10, and the number of

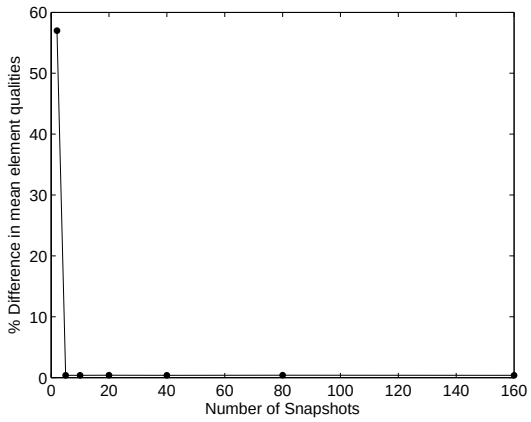


Figure 5.13: Plot of percentage difference in mean element quality for ROM using coefficient optimization as a function of the number of snapshots used in the computation of the POD modes.

boundary nodes, which is much smaller than the total DOF of the mesh. For the smallest mesh, the ROM results in a 96.84% saving in CPU time, and further increases to 99.99% savings or almost 5 orders of magnitude speed up. The ROM based on using Newton's method to optimize the coefficients performs less favorably, but none the less produces CPU time reductions ranging from 87.35% to 96.51% from the smallest to largest mesh.

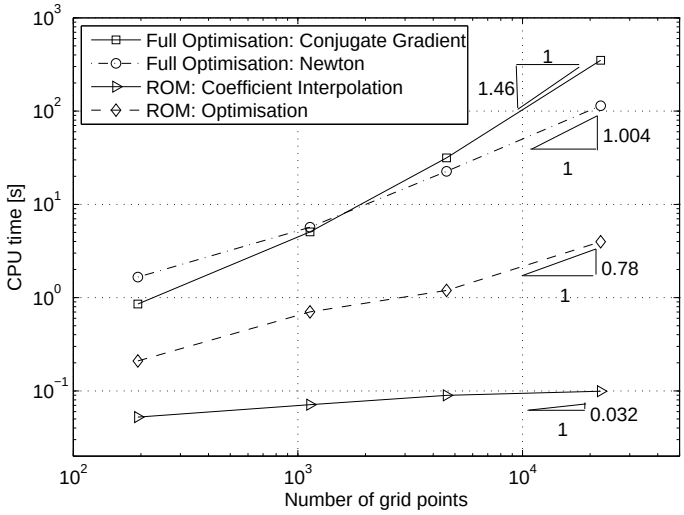


Figure 5.14: CPU scaling of ROMs in comparison to full order optimization.

As a final note, it is possible to compute the gradients and Hessian matrix for the ROM based on those computed for the full optimization. The gradients and Hessian for the full optimization can be found analytically, and thus are fairly cheap to compute. The derivatives of the objective function, in terms of the coefficients can be computed by

$$\frac{dF}{d\boldsymbol{\alpha}} = \left(\frac{d\boldsymbol{x}}{d\boldsymbol{\alpha}}\right)^T \frac{dF}{d\boldsymbol{x}}. \tag{5.6}$$

And since $\boldsymbol{x} = \sum_i^M \alpha_i \boldsymbol{\varphi}_i$, $\frac{d\boldsymbol{x}}{d\boldsymbol{\alpha}}$ is the basis modes. Therefore the differential becomes

$$\frac{dF}{d\boldsymbol{\alpha}} = [\boldsymbol{\varphi}]^T \frac{dF}{d\boldsymbol{x}}, \tag{5.7}$$

where $\frac{dF}{d\boldsymbol{x}}$ is the analytical derivatives of the full order system, and $[\boldsymbol{\varphi}]$ is a $[N \times M]$ matrix containing all the basis modes. Similarly the Hessian can be computed as

$$\frac{d^2F}{d\boldsymbol{\alpha}^2} = [\boldsymbol{\varphi}]^T \frac{d^2F}{d\boldsymbol{x}^2} [\boldsymbol{\varphi}], \tag{5.8}$$

where $\frac{d^2F}{d\boldsymbol{x}^2}$ is the analytical Hessian for the full order system.

The CPU time for Newton's method using the above analytical gradients, as well as those using finite differencing are shown in Figure 5.15. The plots are for CPU times as a function of the number of coefficients used, for three different mesh sizes consisting of 100, 550 and 2000 nodal coordinates.

For problems where a small number of coefficients and POD modes are retained it is beneficial to use finite differencing. For more than approximately 14 coefficients for smaller meshes to 19 coefficients for larger meshes it becomes cheaper to compute the gradients analytically. For our mesh movement problem we need no more than 10 coefficients, but it is conceivable that a mesh movement problem with complex boundary movements would require a greater number of coefficients.

It is the opinion of the author that using only RBF interpolation to solve the ROM is sufficient. There is a clear computational advantage, and only a slight improvement in the ROM accuracy by further optimizing on the RBF interpolation approximations.

5.4 POD Model of Elastic Deformation Mesh Movement

The final mesh movement algorithm to which we will apply the concepts of POD based reduced order modeling is the elastic deformation method.

In the previous section we have shown that POD can be used to effectively generate reduced order models of mesh movement. Using RBF interpolation to compute the expansion coefficients, any mesh movement technique can be approximated cheaply and accurately, including the elastic deformation method. The elastic deformation method is however based on a set of partial differential equations, which allows us to apply

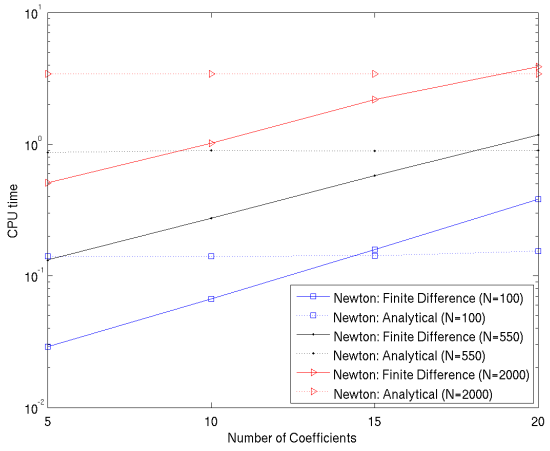


Figure 5.15: CPU time per iteration, as a function of number of coefficients used for Newton's method using analytical and finite difference gradients.

one of the weighted residual methods discussed in Chapter 3 to compute the respective expansion coefficients. In so doing, the set of equations can be projected onto the POD basis modes, allowing for the original set of equations to be solved but for a reduced number of DOFs. The primary aim of this section is thus to investigate whether there are any advantages to solving for a projected set of equations rather than simply using RBF to interpolate the coefficients.

For the purposes of the investigation, we will make use of Galerkin's projection, discussed in Section 3.2.3. For the original finite element formulation, the solution of the equations of linear elasticity results in

$$\mathbf{K}\boldsymbol{u} = \boldsymbol{f}, \tag{5.9}$$

where $\mathbf{K}$ is the stiffness matrix of size $N \times N$, $\boldsymbol{f}$ is the force vector and $\boldsymbol{u}$ is the displacement vector to be solved. Using Galerkin's method to project the POD basis modes onto the equations of linear elasticity, we end up with a similar matrix problem,

$$\mathbf{K}_\alpha \boldsymbol{u}_\alpha = \boldsymbol{f}_\alpha, \tag{5.10}$$

where $\mathbf{K}_\alpha$ is now a $M \times M$ stiffness matrix in terms of the expansion coefficients $\boldsymbol{\alpha}$, $\boldsymbol{f}_\alpha$ is the force vector now of size M and $\boldsymbol{u}_\alpha$ are the displacement in terms of coefficients. The solution of the ROM is now a matrix problem of size M rather than N, where $M \ll N$.

By using Galerkin projection, and since we are using CST elements that require no integration, we find that

$$\mathbf{K}_\alpha = [\boldsymbol{\varphi}]^T \boldsymbol{K} [\boldsymbol{\varphi}] \tag{5.11}$$

where $[\boldsymbol{\varphi}]$ is the $N \times M$ set of basis modes. Similarly

$$\boldsymbol{f}_\alpha = [\boldsymbol{\varphi}] \boldsymbol{f} \tag{5.12}$$

and the actual system nodal coordinates displacements are found by

$$\boldsymbol{u} = [\boldsymbol{\varphi}] \boldsymbol{u}_\alpha. \tag{5.13}$$

To solve the reduced system in (5.10) is therefore significantly cheaper, but setting up the solution matrix problem is slightly more expensive than setting up the initial system. It is hence expected that the ROM based on Galerkin's projection will only offer slight computational benefits for small mesh movement problems. As the problem size increases, a greater percentage of the CPU time will be spent solving the matrix problem (5.9), in which case the Galerkin based ROM will provide increasing computational savings.

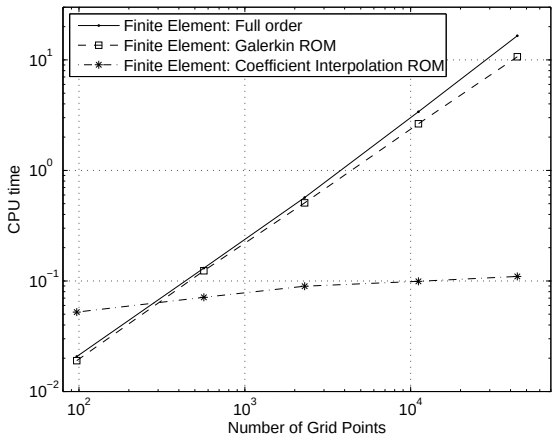


Figure 5.16: Comparison of CPU times for the full order elastic deformation and ROM based on interpolation and Galerkin projection.

To demonstrate, we generate 60 snapshots for stiffening constant of $\chi = 1$, with the magnitudes of rotation and translation selected using Latin hypercube sampling, in a similar fashion to the optimization based ROM. Figure 5.16 is a comparison of the computational cost of the full order elastic deformation method, the ROMs based on Galerkin projection and ROMs using coefficient interpolation.

As expected, the computational savings obtained using the Galerkin based ROM increases as a function of the problem size, with CPU time savings ranging from 8.06%

to 35.40% from the smallest to largest mesh respectively. Despite the promising CPU times for the Galerkin based ROM, using coefficient interpolation only we still obtain significantly larger computational savings, with up to 99.33% savings for the largest mesh. Another advantage of using RBF interpolation to compute the coefficients is that only a single displacement increment is required. The times shown in Figure 5.16 is for only a single displacement increment, but the ROM based on Galerkin projection requires as many displacement increments as the full order system solution did; for our rotation and translation test problem we used 15 increments. Thus for large displacements, the computational benefits using coefficient interpolation increases further.

Furthermore, the computational times of the ROM are compared to the linear elastic formulation using direct matrix solvers. In Section 4.3.4 we demonstrated that an iterative solver scales similarly to the sparse direct matrix solver, only at a factor more expensive. Therefore, these computational percentage savings would be significantly increased if an iterative solver were rather made use of. Especially considering that the ROM is small enough that direct solvers can comfortably be used.

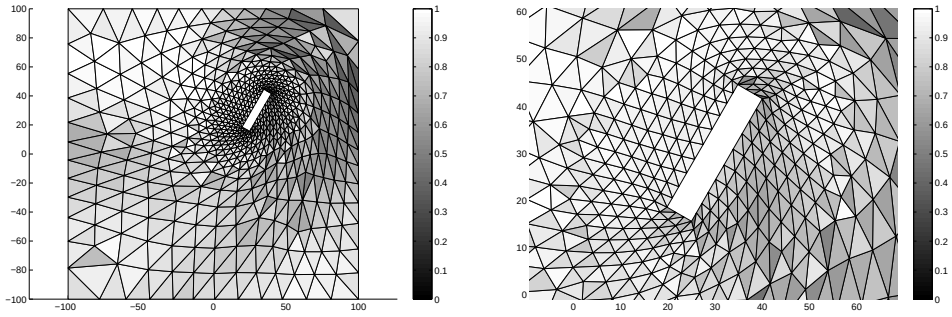
Despite the considerably larger savings obtained using only coefficient interpolation, the RBF based ROM does not sacrifice on the accuracy of the reproduced mesh. In fact, the RBF based ROM produces better quality meshes. Figure 5.17 shows the meshes for the rotation and translation problem for both ROMs and the full order elastic deformation method and Figure 5.18 shows histogram plots comparing the percentage difference of the size and shape of each mesh element using the ROMs in comparison to the full order elastic deformation method. Using RBF interpolation we find that all elements differ by less than 5% in terms of both shape and size in comparison to the full order solution. For the Galerkin model, 99.15% and 97.26% of the elements change by less than 5% in terms of size and shape.

The one advantage of using Galerkin's projection, as opposed to using simple interpolation, is that the original set of PDE is still solved, but only in a subspace of the solution space. The control offered by the elastic deformation method is therefore still maintained, in the sense that we can still change the Jacobian stiffening constant. The snapshots for the POD based ROM can for example be trained for a stiffening constant of $\chi = 1$, or a specific set of elastic properties, but the ROM can be used for a much larger range. Using RBF interpolation, the ROM can only reproduce meshes for the exact conditions used in training.

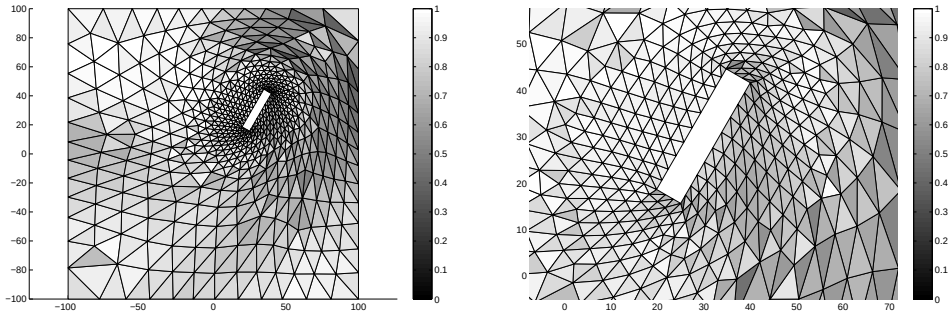
Figure 5.19 compares the mean element qualities of the mesh after rotation and translation for a varying stiffening constant χ, for both the full order elastic deformation and the Galerkin based ROM. We find that the ROM, despite being trained for $\chi = 1$, is capable of accurately reproducing meshes for the range of χ from 0.8 to 1.3, and can produce adequate meshes for the range of $\chi = [0.5 \text{ to } 2]$.

Using Galerkin projection we are capable of generating a ROM that for larger problems is capable of providing noticeable reductions in computational cost while largely maintaining the control of the original method. Using RBF interpolation to compute

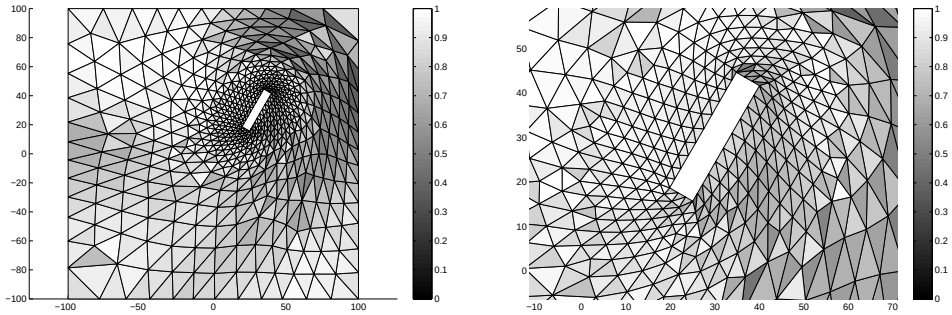
the coefficients on the other hand limits the range of application of the ROM. It does however provide better mesh approximations at significantly reduced computational costs.



(a) Full order elastic deformation.

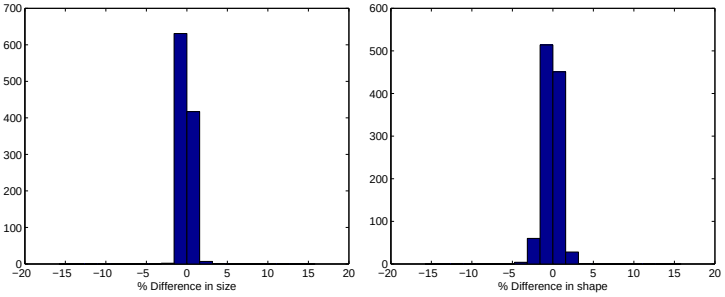


(b) POD ROM with coefficient interpolation.

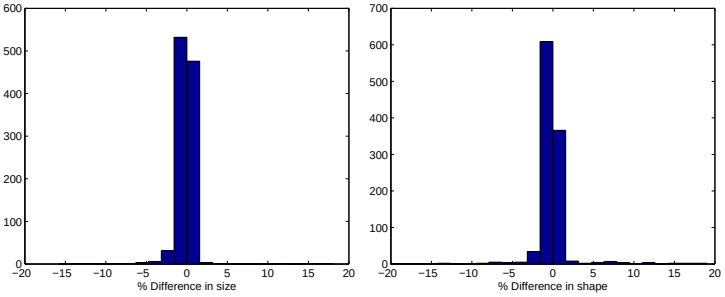


(c) POD ROM with Galerkin projection.

Figure 5.17: Comparison of final meshes after rotation and translation of elastic deformation mesh movement method and reduced order models thereof.



(a) POD ROM with coefficient interpolation.



(b) POD ROM with Galerkin projection.

Figure 5.18: Histogram plot of percentage difference in shape and size for each element between the elastic deformation method ROMs and full order solution. For $\triangle x = 30$, $\triangle y = 30$, $\phi = 60°$CCW, and $\chi = 1$.

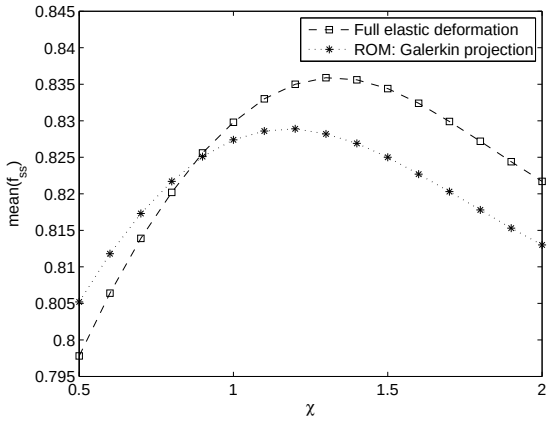


Figure 5.19: Mean element qualities using both the full elastic deformation method and Galerkin projection based ROM for changing χ.

5.5 Conclusion

In this chapter we demonstrated the ability of the method of POD to generate effective reduced order models of mesh movement algorithms. We implemented and tested the

method to optimization and elastic deformation mesh movement schemes, though the method is applicable to any movement method. We trained the POD models as a pre-processing step to a simple combined rotation and translation test case. It was found that computing the POD expansion coefficients using RBF interpolation alone resulted in comparably good quality final meshes, with CPU cost reductions in excess of 99%.

The following chapter will focus on the implementation of the proposed POD model to a common FSI simulation. An adaptive training model is introduced, whereby the POD model can be implemented without having to be pre-trained.

Chapter 6

Mesh Movement ROM Applied to a Benchmark FSI Problem and Adaptive Model Training

6.1 Introduction

In Chapter 5 we have shown that the method of proper orthogonal decomposition can be applied successfully to generate reduced order models of mesh movement algorithms. In this chapter, we aim to implement and test the method on a real FSI simulation, and introduce an adaptive training model for the POD ROM. The problem we aim to investigate is the common FSI benchmark problem [18, 42] depicted in Figure 6.1, with flow over a fixed rectangle with a flexible tail, with fluid flow from the left to right domain boundary.

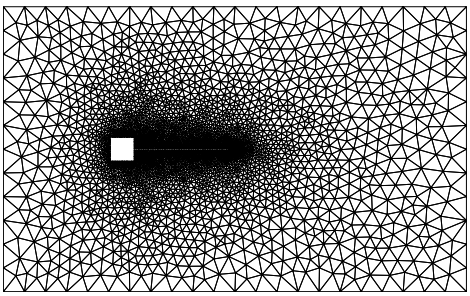


Figure 6.1: Initial mesh for oscillating tail benchmark problem.

The FSI simulation results for the benchmark problem was made available by Oxtoby and Malan [34], and is based on their unified finite-volume FSI computational code currently under development at the Council for Scientific and Industrial Research, South Africa. For the flexible tail problem they picked up two predominant boundary motion

types; the first is for the flexible tail predominantly deflecting in its first mode with minor second mode oscillations, and secondly they noticed large second mode oscillations. Pressure plots of the two boundary motion types, at selected time steps, are shown in Figures 6.2 and 6.3.

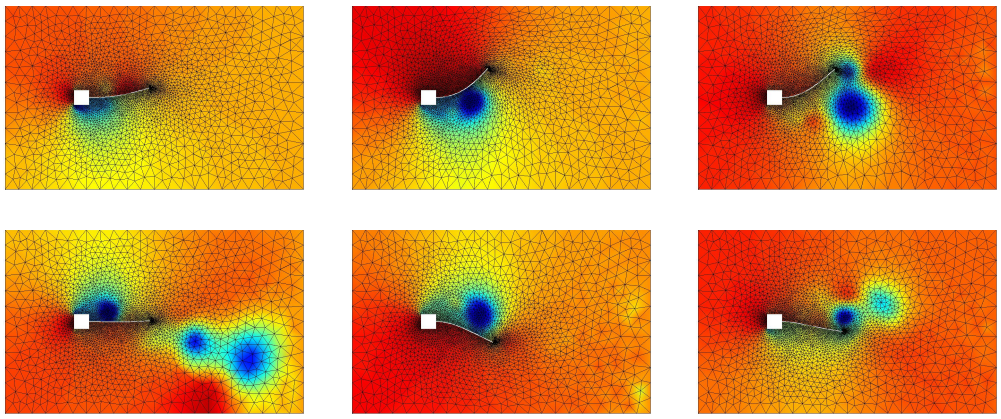


Figure 6.2: Pressure plots for first mode deformations.

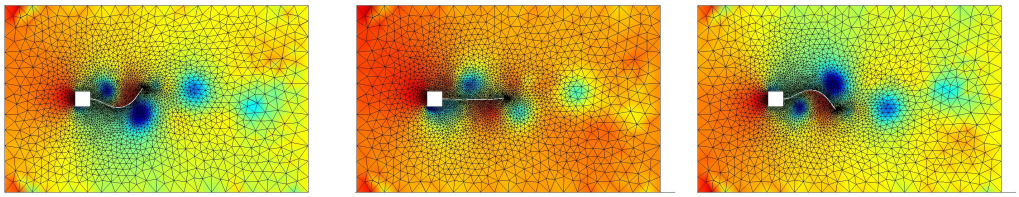


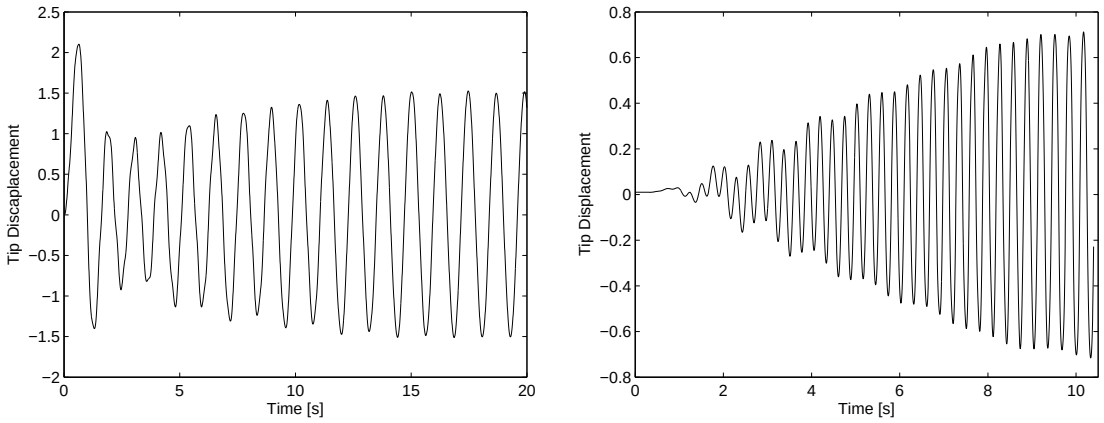
Figure 6.3: Pressure plots for second mode deformations.

For our investigation, we are not concerned with the actual fluid flow, but rather the boundary and mesh motion induced by the fluid flow. To provide an indication of the tail motion, the vertical displacement of the beam tip is plotted in Figure 6.4 for both modes of deformation. Approximately 4500 and 2000 time steps were required to ensure that limit state is achieved for the two simulations respectively.

To facilitate the discussion in the remainder of the chapter, we will refer to the boundary motion of Figure 6.2 as first mode deformation and Figure 6.3 as second mode deformation.

6.2 Analysis of Full Order Mesh Movement

For the results of Oxtoby et al. [34], the mesh movement was performed using mesh optimization, through the use of quality metric 2 introduced in Section 4.3. If we consider that for the first mode shape deformation analysis, 4500 mesh movement steps



(a) First mode deformation. 4500 time steps. **(b)** Second mode deformation. 2000 time steps.

Figure 6.4: Beam tip vertical displacement plots.

are required, at an average of 22 seconds per mesh update step, it implies a total computational time of close to 27 hours for the mesh movement alone.

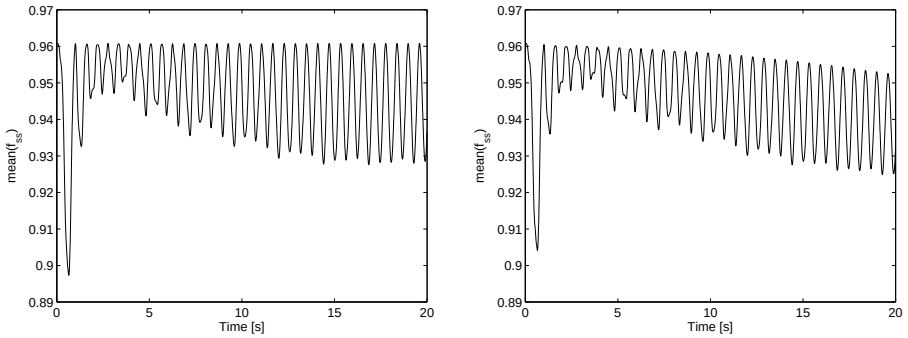
It should be noted that the times reported here are not those obtained by Oxtoby and Malan, but the times obtained by our own implementation of the same method. The implementation of the mesh movement by Oxtoby et al. is on a different computational platform in a C++ environment where we implemented our own version in MATLAB. Furthermore, their mesh movement was performed using Newton optimization, where the gradients and Hessian are computed through Finite Differencing, where we compute these analytically.

Our motives for reporting on the CPU times required by the mesh movement is not to naively provide definitive times. We realize that the computational times are dependent on a variety of factors, of which implementation and programming environment is but two; especially considering that MATLAB is primarily a convenient development environment and is not ideal for efficient number crunching. We report on the times so as to enable us to make order of magnitude estimates of the computational cost for the methods we hope to compare. Specifically in the case where we report 27 hours required for the 4500 mesh movement steps we merely try to highlight that the mesh movement, based on mesh optimization, comes at significant computational effort.

The mesh movement accounts for a significant percentage of the total CPU time required for the FSI simulation. Oxtoby et al. claim the mesh movement alone accounts for close to 60% of the full FSI solution time. It is well documented that mesh movement based on optimization is highly expensive, and the cost may be considerably reduced by implementing mesh movement methods such as RBF interpolation. On average, mesh motion through our implementation of RBF interpolation requires around 1.1 seconds for a single increment, which results in a total time required of approximately 80 minutes. On its own, this already represents a significant reduction in the mesh

movement time, from our estimate close to a factor of 20.

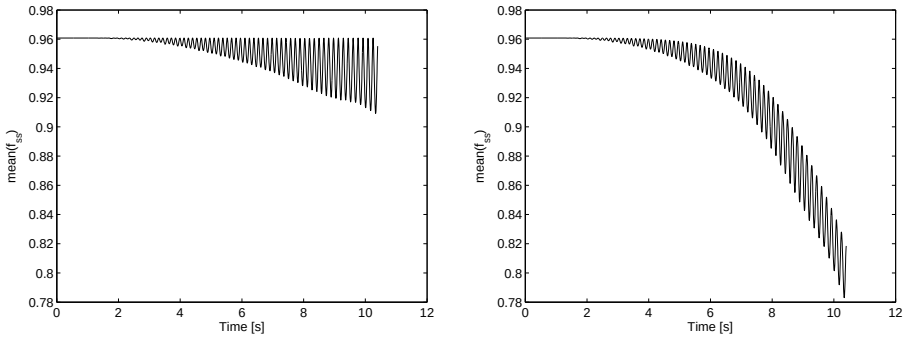
In Figures 6.5 and 6.6, we compare the mean element qualities, based on metric 1, $0 \leq f_{ss} \leq 1$ (Section 4.3.1.1), for the actual mesh movements obtained by Oxtoby et al. and through the use of RBF interpolation. The results for the RBF interpolation is obtained by using a single displacement increment between each time step. The mesh quality plots highlight the path dependency of RBF interpolation and the possibility of mesh degeneration for a transient FSI problem. The problem is especially prominent for the second mode oscillations. To remove the mesh quality deterioration more RBF displacement increments would have to be used between time steps.



(a) Full order mesh optimization. **(b)** Full order RBF interpolation.

Figure 6.5: Mean element quality plots for first mode deformations for full order mesh movement using mesh optimization and RBF interpolation based on $0 \leq f_{ss} \leq 1$.

The deterioration using a single RBF displacement increment is not critical for the first mode oscillations, but becomes a problem for the second mode deformations. It is especially a problem considering that the minimum element quality in the mesh drops to as low as 0.18, which is close to degenerate. For the full mesh optimization, the lowest element quality through all time steps is 0.43.



(a) Full order mesh optimization. **(b)** Full order RBF interpolation.

Figure 6.6: Mean element quality plots for second mode deformations for full order mesh movement using mesh optimization and RBF interpolation based on $0 \leq f_{ss} \leq 1$

This example highlights one of the advantages of using mesh optimization; for any boundary displacement, there exists a unique mesh solution and the mesh movement is not path dependent. The computational cost is however prohibitively expensive, and cannot realistically be used in any FSI simulation of a real world problem, for even moderately large meshes. Generating a POD ROM of the problem would reduce the associated cost.

6.3 POD Model and Adaptive Training

For the oscillating tail mesh movement problem, we now introduce the concept of adaptive POD training. It is, to the best of our knowledge, the first application of POD in the context of ROM, where the model is not trained as a pre-processing step. In all applications of POD ROMs in literature, a full set of simulations are performed and the results then used to train the model. The model is then capable of reproducing the full results, at significant cost reductions, and can be applied for minor problem parameter modifications. All the applications of POD to mesh movement in the previous chapters, have followed this philosophy.

It is difficult to justify the use of POD in this context. If a full FSI simulation is required to generate a POD model of the mesh movement, it would make far more sense to generate a ROM of the full FSI results rather than the mesh movement alone. It is possible to generate the POD training snapshots based on predictions of the boundary movement. There does however remain the question as to how many training snapshots are required, and how they should be selected. As an example, for the oscillating tail problem, the model can be trained by generating snapshots for the various modes of deformation. Generating snapshots of hypothetical boundary displacements, without prior knowledge of the simulation, allows for the possibility that erroneous selections are made. The snapshots might not properly define the actual motion, or computational time might be wasted on snapshots that may never feature in the full FSI simulation.

To alleviate these issues, we propose the use of an adaptively trained POD model, to facilitate the full order mesh movement, with the aim of reducing computational costs for first time simulations. The idea is fairly simple. The POD model is initialized with the starting mesh in its undeformed configuration and the first full order mesh movement step based on the first FSI time step. The model is then used for all subsequent boundary movements until the mesh quality deteriorates below some lower limit. A full order mesh optimization is then performed, with the results used as an additional training snapshot to update the POD model. When computing the POD basis modes, each of the snapshots computed to date would then be used.

For the full mesh movement, the minimum element quality across the range of time steps is 0.4276, and the lowest mean element quality is 0.8973 for the first mode deformation and for the second mode deformations the lowest minimum and mean is 0.4265 and

0.9092 respectively. Because we already have the results of the full FSI simulation, and associated meshes, we know beforehand what appropriate selections would be for the lower limits.

If the method is used for first time analysis, such knowledge is not available. If the lower limits are pre-set without such knowledge, there is the possibility that these limits might be set either too high or too low. If the lower limits are higher than what is achievable by the full order mesh movement algorithm, more updating snapshots would be generated than would be necessary. Or if the limits are too low, we might obtain acceptable meshes for a simulation, but meshes that have unnecessarily low mesh qualities than what would have been obtained by the full order mesh movement. To alleviate this uncertainty, it is possible to set these lower limits adaptively as well, by setting them as a small percentage lower than the lowest mesh qualities obtained in preceding updating snapshots.

Mesh optimization is particularly well suited to the adaptive training model because there exists a unique solution for any given boundary movement. At each instance where the mesh movement based on the ROM deteriorates below the lower limits, and an updating snapshot is required, the mesh is improved by running a full order optimization routine. This is not true for path dependent methods such as RBF interpolation or the elastic deformation method. Once the mesh has degenerated, using a full order RBF interpolation step will not improve the mesh quality, but rather result in further mesh degeneration.

Having said that, the adaptive training method is applicable to any mesh movement method, as long as sufficient precautions are taken. For example, if RBF interpolation is used to generate the updating snapshots, the boundary and mesh can be displaced from the original position to the deformed state as long as numerous incremental displacement steps are used. In essence, the adaptive procedure will save path dependent methods from the sort of mesh degeneration shown in Figure 6.6(b). Applying the adaptive POD training to path dependent methods such as RBF interpolation would essentially render them path independent.

Lastly, the number of POD modes retained for the ROM can be adapted as well. Due to the negligible cost of the ROM it was decided for the purposes of this demonstration to retain all the modes already computed at any given time. It is however possible to retain only the K^{th} most 'energetic' modes. In other words, use all the modes computed until the addition of an extra mode does not improve the accuracy of the ROM. This could be useful for problems that require a large number of updating snapshots, in which case retaining all the modes might render the computational time of the ROM noticeable.

The proposed method of POD can be applied as is to any FSI code with an already existing mesh movement algorithm. It will allow for computational cost reductions for first time simulations, without the user having to have any prior knowledge regarding system behaviors.

6.3.1 Results of Adaptive ROM

Let us first consider an adaptive ROM for the first mode deformations, by pre-setting the lower limits to 0.4 and 0.9 for minimum and mean element qualities respectively. As in Chapter 5, the ROM is generated using RBF interpolation to compute the expansion coefficients and using the interpolated results as a starting guess for full optimization. For the pre-set lower limits, the adaptive model requires a total of 28 updating snapshots. The number of POD modes retained for the ROM approximation is set equal to the total number of available snapshots at any given point.

To provide an indication as to how well the POD ROMs replicate the full order mesh movement, the normalized error function between the exact and approximate nodal coordinates is plotted in Figure 6.7, where the error function is defined by

$$||e||_2 = \frac{1}{N}\left(\frac{\sum_{i=1}^{N}\sqrt{(x_i - x_{i_{\text{exact}}})^2}}{l}\right), \tag{6.1}$$

where l is the average length of all edges connected to node i.

From the plot, it is evident that the ROM based on coefficient optimization outperforms the ROM based on coefficient interpolation. However, of far more importance than the actual differences in nodal coordinates, is whether the meshes generated by the ROMs are still appropriate for use in a simulation. To this end, the mean element qualities for the full mesh optimization and the ROM based only on coefficient interpolation is shown in Figure 6.8(a) along with the 28 updating positions.

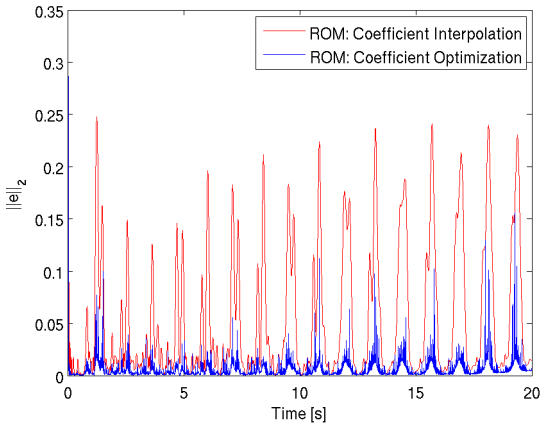
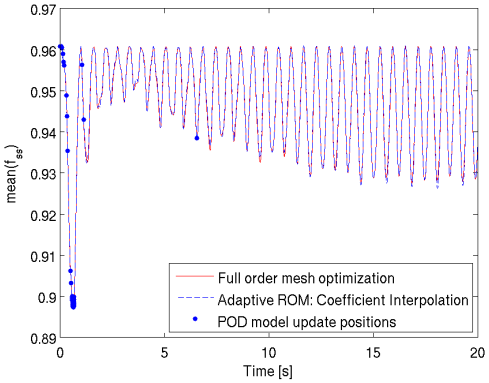


Figure 6.7: Normalized error between the nodal coordinates obtained by the POD ROMs compared to the full order mesh optimization for the first mode deformation.

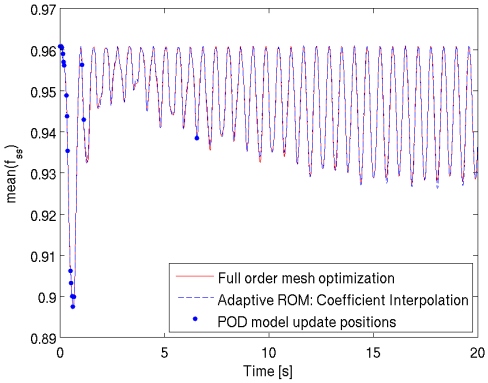
With only 28 updating snapshots, and using only coefficient interpolation, POD is capable of producing a ROM that closely replicates the full order model, and does

so at a substantial cost reduction. The full mesh movement takes approximately 27 hours to solve for the 4500 time steps, where the ROM takes 653 seconds, including the snapshots training. Of the total time, updating the POD modes and RBF model takes a total of 2.3326 seconds. The POD model itself takes an average of 0.003 seconds to compute per time step, and for all the time steps combined, about 13.5 seconds.

The POD related costs are therefore negligible in comparison to the costs of the full optimization mesh movement. In essence, the computational time for the mesh movement is reduced from 4500 full optimization problems, down to 28. The adaptive ROM significantly outperforms even the RBF interpolation model, while maintaining all the advantages offered by full optimization.



(a) Mesh quality lower limits pre-set to 0.4 and 0.9 for minimum and mean element qualities. 28 updating snapshots required.



(b) Mesh quality lower limits set to 10% lower than mesh qualities of updating snapshots. 16 updating snapshots required.

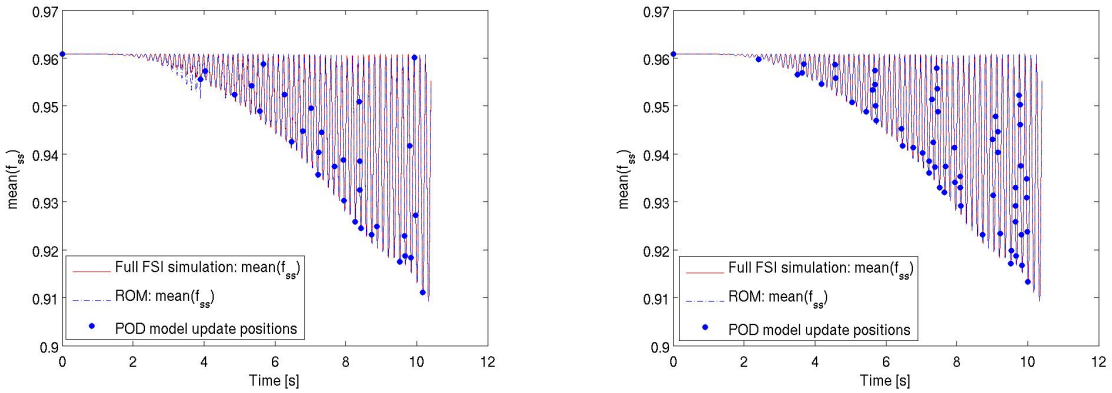
Figure 6.8: Mean element quality comparison, for first mode deformation, using full order mesh optimization and adaptively trained POD ROM based on coefficient interpolation. $0 \leq f_{ss} \leq 1$.

Despite the clear cost reductions in comparison to the full mesh optimization, the adaptive model in Figure 6.8(a) illustrates the risk if the lower quality limits are set too high. In the range of 0.6 to 0.68 seconds, the mean element qualities for the full optimization model drops below 0.9, which was the value of our pre-set lower limit. In this region, 15 updating snapshots are generated, one for every time step. If we adaptively set the lower limit as 10% lower than the lowest qualities in the updating snapshots, the 15 snapshots generated in this region is reduced to just 3, or a total of 16 updating snapshots for the full simulation. The POD model costs is now reduced from 653 to 368 seconds (Figure 6.8(b)), while not having any influence on the quality of the mesh movement model.

For the second mode oscillations, the adaptively trained ROM requires more updating snapshots than was required for first mode boundary movements. Figure 6.9 shows the mean element qualities along with update positions for the ROM using coefficient interpolation and full mesh movement. Figure 6.9(a) is for pre-set lower limits of 0.9

and 0.4 for mean and minimum element qualities respectively, and Figure 6.9(b) is for adaptively set lower limits (10% lower than updating snapshots).

The pre-set model requires a total of 32 updating snapshots. While the mesh obtained using this ROM is more than adequate for simulations, since the element qualities never drop below the pre-set limits, it does however deviate from the solution obtained by the full mesh motion for the time steps between 2 and 4 seconds. By adaptively changing the limits requires an additional 26 snapshots, but improves on the replication of the full order mesh movement model. An additional 26 updating snapshots is an expensive computational addition, yet the ROM remains cheaper than both the full optimization and RBF interpolation movement models. Furthermore, the ROM does not suffer from the path dependent mesh degeneration experienced by the full order RBF interpolation method (Figure 6.6(b)).



(a) Mesh quality lower limits pre-set to 0.4 and 0.9 for minimum and mean element qualities. 32 updating snapshots required.

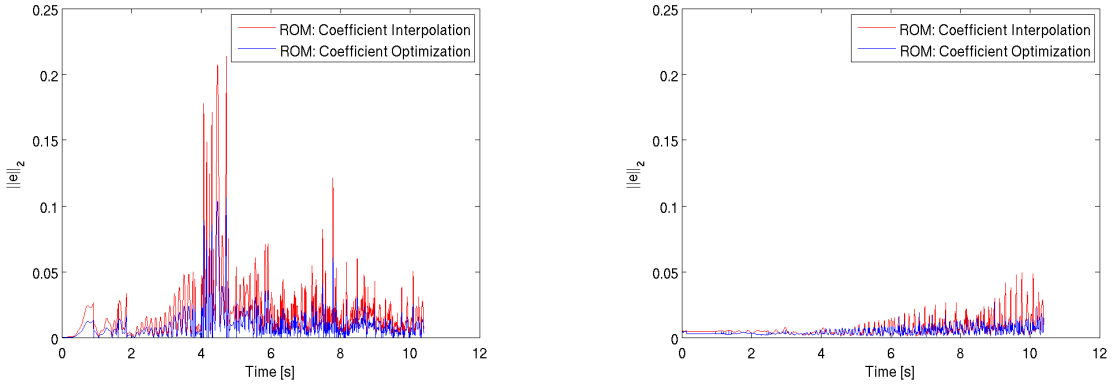
(b) Mesh quality lower limits set to 10% lower than mesh qualities of updating snapshot. 58 updating snapshots required.

Figure 6.9: Mean element quality comparison, for second mode deformation, using full order mesh optimization and adaptively trained POD ROM based on coefficient interpolation. $0 \leq f_{ss} \leq 1$.

The mean element quality plots in Figure 6.9 is for a ROM based on coefficient interpolation. The mesh qualities obtained are more than good enough to be used in any full FSI simulation. It can however be shown that using coefficient optimization to further improve on the starting guess provided by interpolation produces superior approximations. To illustrate, the normalized error between the nodal coordinates obtained by the POD ROMs compared to the full order mesh optimization is shown in Figure 6.10.

Despite the superior performance of ROM using coefficient optimization, the increased accuracy is unnecessary. While coefficient optimization ROM is more than an order of magnitude computationally cheaper, and scales better than full optimization, it is also more than an order of magnitude more expensive than using just coefficient interpolation. For this reason, and the fact that the coefficient interpolation ROM

produces sufficiently good quality meshes, it is the advised ROM technique for mesh movement.



(a) Mesh quality lower limits pre-set to 0.4 and 0.9 for minimum and mean element qualities.

(b) Mesh quality lower limits set to 10% lower than mesh qualities of updating snapshot.

Figure 6.10: Normalized error between the nodal coordinates obtained by the POD ROMs compared to the full order mesh optimization for the second mode deformation.

As a final note, POD modes are better at interpolating data contained within the training snapshots as opposed to extrapolating. The adaptive training implementation of the POD model inherently implies that the model is required to extrapolate for information outside the system observations. The number of snapshots used to fully define the model is more than if the FSI simulation was performed and a few snapshots were selected at key points. For the first mode deformations, by selecting snapshots at the deformation extremes and a few intermediate points, we require as few as 9 snapshots to generate a ROM as adept as the one in Figure 6.8, where the adaptively trained model required 16. It is the opinion of the author, that the gains incurred by not having to pre-train the ROM outweighs the additional cost incurred.

6.4 Conclusion

In this chapter, an adaptive training model was introduced. The model was implemented and tested on a common flexible beam FSI benchmark simulation. The adaptive POD model is capable of accurately replicating the full mesh movement at significant cost reductions. For the simulation, two distinct deformation modes were identified, where a total of 4500 and 2000 time steps were required for the simulation of each of the respective motion types. The adaptive model required as few as 16 and 58 updating, full order mesh movement solutions, for the first and second mode deformations respectively.

The unique advantage of the training procedure is that no pre-training is required. The POD method, as is, can be applied to any FSI code that has an existing mesh movement scheme. The model can be used to speed up first time simulations, without the end user having to have any prior knowledge of the system being analyzed. The training model introduced in this chapter, to the best of the author's knowledge, constitutes the first use of the method of POD, in the context of ROM, where the POD model is not trained via a set of pre-computed simulations.

Chapter 7

Conclusion

The research presented in this thesis seeks to test the applicability of using POD to generate reduced order models for mesh movement algorithms. The scope of the research conducted can be divided into two main categories: implementation and comparisons of three common mesh movement algorithms; and the application of POD to generate ROMs of these mesh movement algorithms.

The three mesh movement methods investigated was RBF interpolation, mesh optimization and the elastic deformation method. It was shown that mesh movement through optimization offered the highest quality mesh movement, but at the highest computational expense. Optimization is also the only mesh movement method that offers direct control over the mesh qualities via the use of quality based cost function. The optimization was performed using the Conjugate gradient method and Newton's method with a line search. With the gradients and Hessian computed analytically, and through the use of a sparse matrix solver, Newton's method scaled the better of the two (in terms of CPU times) as a function of problem size.

The elastic deformation method, with Jacobian stiffening factor, offered the second highest quality final meshes. The method proved to be computationally the cheapest (when compared to RBF interpolation with a constant large support radius). The optimization mesh movement method was demonstrated to offer the advantage that for any given boundary displacement, there exists a unique mesh. Both RBF interpolation and the elastic deformation method are path dependent, and the mesh may deteriorate through the course of a simulation. The problem is particularly acute for the RBF interpolation method, since the interpolation function is not formulated to handle rotations.

Reduced order models were generated using POD for the optimization and elastic deformation methods. In both cases it was shown that using only RBF interpolation to compute the POD expansion coefficients offered comparably good quality final meshes. Furthermore, CPU savings in excess of 99% were observed for a single displacement increment. The interpolation model was compared to models based on coefficient opti-

mization and Galerkin projection. In both cases, the RBF interpolation model offered less control over model parameter changes, but significant CPU cost reductions.

Lastly, we introduced a fully automatic adaptive training model, whereby a POD based ROM can be used without the model having to be pre-trained. The POD model is progressively trained and updated as a function of mesh quality. With the adaptive training procedure, the POD methodology can now be implemented as is, to any FSI code as a wrapper to an existing mesh movement scheme. The training model allows for significant speed gains of FSI simulations involving large complex boundary deformations.

Bibliography

[1] M.F. Barone and J.L. Payne. Methods for Simulation-based Analysis of Fluid-Structure Interaction. Technical report, NASA Center for AeroSpace Information, 7121 Standard Dr, Hanover, Maryland, 21076-1320, USA, 2005.

[2] R. Barret, M. Berry, and T.F. Chan. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, 1994.

[3] J. Batina. Unsteady euler algorithms with unstructured dynamic mesh for complex-aircraft aerodynamic analysis. *AIAA Journal-American Institute of Aeronautics and Astronautics*, 29:327–333, 1991.

[4] H. Braess and P. Wriggers. Arbitrary lagrangian eulerian finite element analysis of free surface flow. *Computer methods in applied mechanics and engineering*, 190:95–109, 2000.

[5] P.A. Cavallo, N. Sinha, and G.M. Feldman. Parallel unstructured mesh adaptation for transient moving body and aeropropulsive applications. *American Institute for Aeronautics and Astronautics*, 1057, 2004.

[6] A. Chatterjee. An introduction to the proper orthogonal decomposition. *Current science*, 78(7):808–817, 2000.

[7] R.F. Coelho, P. Breitkopf, and C. Knopf-Lenoir. Model reduction for multidisciplinary optimization - application to a 2d wing. *Strucural Multidisciplinary Optimization*, 37:29–48, 2008.

[8] R.D. Cook, D.S. Malkus, M.E. Plesha, and R.J. Witt. *Concepts and applications of finite element analysis*. John Wiley & Sons, 2007.

[9] A. de Boer, M.S. van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers and Strucutres*, 85:784–795, 2007.

[10] N. De Freitas, Y. Wang, M. Mahdaviani, and D. Lang. Fast krylov methods for n-body learning. *Advances in neural information processing systems*, 18:251, 2006.

[11] L.F. Diachin, P. Knupp, T. Munson, and S. Shontz. A comparison of two optimization methods for mesh quality improvement. *Engineering with Computers*, 22(2):61–74, 2006.

[12] C. Farhat, C. Degend, B. Koobus, and M. Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer methods in applied mechanics and engineering*, 163:231–245, 1998.

[13] A. George and J. Liu. *Computer Solutions of Large Sparse Positive Definite Systems*. Prentice-Hall, 1981.

[14] B. T. Helenbrook. Mesh deformation using the biharmonic operator. *International Journal for Numerical Methods in Engineering*, 56:1007–1021, 2003.

[15] P. Holmes, J.L. Lumley, and G. Berkooz. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge University Press, 1996.

[16] PJ Holmes, JL Lumley, G. Berkooz, JC Mattingly, and RW Wittenberg. Low-dimensional models of coherent structures in turbulence. *Physics reports*, 287(4):337–384, 1997.

[17] G.A. Holzapfel. *Nonlinear Solid Mechanics - A Continuum Approach for Engineering*. John Wiley and Sons NY, 2000.

[18] B. Hübner, E. Walhorn, and D. Dinkler. A monolithic aproach to fluid-structure interaction using space-time finite elements. *Computer Methods in Applied Mechanics and Engineering*, 193:2087–2104, 2004.

[19] T.J.R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall Englewood Cliffs, NJ, 1987.

[20] C.R. Johnson, K. Okubo, and R. Reams. Uniqueness of matrix square roots and an application. *Linear Algebra and Its Applications*, 323(1-3):51–60, 2001.

[21] K. Karhunnen. Zur spektral theorie stochastischer prozesse. *Ann Acad Sci Fennicae*, 1946.

[22] P.M. Knupp. Algebraic mesh quality metrics for unstructured initial meshes. *Finite Element in Analysis and Design*, 39:217–241, 2003.

[23] P.A. LeGresley and J.J. Alonso. Airfoil design optimization using reduced order models based on proper orthogonal decomposition. In *Fluids 2000 Conference and Exhibit, Denver, CO*, 2000.

[24] D. Levin. The approximation power of moving least squares. *Mathematics of Computation*, 67(224):1517–1531, October 1998.

[25] T. Lieu, C. Farhat, and M. Lesoinne. Reduced-order fluid/structure modeling of a complete aircraft configuration. *Computer methods in applied mechanics and engineering*, 195(41-43):5730–5742, 2006.

[26] M. Loeve. Functions aleatoire de second ordre. *C R Academie des Sciences*, 1945.

[27] J.L. Lumley. The structure of inhomogeneous turbulence. *Atmospheric Turbulence and Wave Propogation*, pages 166–178, 1967.

[28] D. Lynch. Unified approach to simulation on deforming elements with application to phase change problems. *Journal of Computational Physics*, 47:387–411, 1982.

[29] D. Lynch and K. ONeill. Elastic grid deformation for moving boundary problems in two space dimensions. *Finite Elements in Water Resources*, 1980.

[30] MD McKay, RJ Beckman, and WJ Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, pages 55–61, 2000.

[31] P. Moin and R.D. Moser. Characteristic-eddy decomposition of turbulence in a channel. *Journal of Fluid Mechanics Digital Archive*, 200:471–509, 2006.

[32] M. Murayama, K. Nakahashi, and K. Matsushima. Unstructured dynamic mesh for large movement and deformation. *AIAA Journal-American Institute of Aeronautics and Astronautics*, 122, 2002.

[33] A. Nealen. An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. *URL: http://www. nealen. com/projects*, 2004.

[34] O. Oxtoby and A.G. Malan. A unified finite-volume approach to fluid-structure interaction. In *15th International Conference on Finite Element in Flow Problems (FEF09), Japan, Tokyo*, April 2009.

[35] 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*, 74:1519–1559, 2008.

[36] J. Schöberl. NETGEN An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1(1):41–52, 1997.

[37] L. Sirovich. Turbulence and the dynamics of coherent structures. I- Coherent structures. II- Symmetries and transformations. III- Dynamics and scaling. *Quarterly of Applied mathematics*, 45:561–571, 1987.

[38] J.A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimzation Theory and Classical and New Gradient-Based Algorithms*. Springer-Verlag, New-York, 2005.

[39] K. Stein, T. Tezduyar, and R. Benney. Mesh moving techniques for fluid-structure interactions with large discplacements. *Transactions of the ASME*, 70:58–63, 2003.

[40] M. Stein. Large sample properties of simulations using Latin hypercube sampling. *Technometrics*, pages 143–151, 1987.

[41] R. van Loon, P.D. Anderson, F.N. van de Vosse, and S.J. Sherwin. Comparison of various fluid-structure interaction methods for deformable bodies. *Computers and Structures*, 85:833–843, 2007.

[42] WA Wall and E. Ramm. Fluid-structure interaction based upon a stabilized (ALE) finite element method. 1998.

[43] Z.J. Wang and A.J. Przekwas. Unsteady Flow Computation Using Moving Grid with Mesh Enrichment. *Tech Rep AIAA*, 94, 1994.

[44] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method (5th edition) Volume 2 - Solid Mechanics*. Elsevier, 2000.

Appendix A

Conjugate Gradient Method for Unconstrained Minimization

The following discussion on the Conjugate gradient method follows the discussion presented by Snyman [38].

Conjugate gradient methods are a class of first order line search methods, which will converge exactly in a finite number of iterations when applied to a positive-definite quadratic function. This property of quadratic termination is a desirable one, as the method will still perform well in the region of a local minimum when applied to a non-quadratic function. There are many methods available for determining conjugate search directions, and the method we implemented is known as Fletcher-Reeves directions.

Suppose we are trying to minimize the function $f(\boldsymbol{x})$, where $\boldsymbol{x} = [x_1, x_2, ..., x_n]$, and $\boldsymbol{u}^i$, $i = 1, 2, ..., n$ is the unit vector search direction. The Fletcher-Reeves search directions are given by

$$\boldsymbol{u}^1 = -\nabla f(\boldsymbol{x}^0) \tag{A.1}$$

where $\boldsymbol{x}^0$ is the initial starting guess, and for $i = 1, 2, ..., n-1$

$$\boldsymbol{u}^{i+1} = -\nabla f\left(\boldsymbol{x}^i\right) + \beta_i \boldsymbol{u}^i \tag{A.2}$$

where $\boldsymbol{x}^i = \boldsymbol{x}^{i-1} + \lambda_i \boldsymbol{u}^i$. λ_i corresponds to the optimal descent step in iteration i, found through some or other line search method. For the Fletcher-Reeves search direction β_i is defined as

$$\beta_i = \frac{||\nabla f\left(\boldsymbol{x}^i\right)||^2}{||\nabla f\left(\boldsymbol{x}^{i-1}\right)||^2}. \tag{A.3}$$

If the function to be minimized is quadratic, i.e. of the form

$$f(x) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c, \tag{A.4}$$

where $c \in \mathbb{R}$, $\boldsymbol{b}$ is a real n-vector and $\boldsymbol{A}$ is a positive definite $n \times n$ real symmetric metric, then the search directions $\boldsymbol{u}^i$ are mutually conjugate with respect to the matrix $\boldsymbol{A}$.

The conjugate gradient algorithm, as implemented and presented in [38] is provided below.

Given the initial starting guess of $\boldsymbol{x}^0$:

1. Compute $\nabla f(\boldsymbol{x}^0)$ and set $\boldsymbol{u}^1 = -\nabla f(\boldsymbol{x}^0)$.
2. For $i = 1, 2, ..., n$ do:
   (a) set $\boldsymbol{x}^i = \boldsymbol{x}^{i-1} + \lambda_i \boldsymbol{u}_i$ where λ is found such that
   $$f\left(\boldsymbol{x}^{i-1} + \lambda_i \boldsymbol{u}^i\right) = \min_{\lambda} f\left(\boldsymbol{x}^{i-1} + \lambda \boldsymbol{u}^i\right) \quad \text{(line search)}.$$
   (b) compute $\nabla f\left(\boldsymbol{x}^i\right)$,
   (c) if convergence criteria satisfied, then stop, and set the minimum $\boldsymbol{x}^* = \boldsymbol{x}^i$.
   (d) if $1 \leq i \leq n-1$:
   $\boldsymbol{u}^{i+1} = \nabla f\left(\boldsymbol{x}^i\right) + \beta_i \boldsymbol{u}^i$ with β_i computed using A.3.
3. Set $\boldsymbol{x}^0 = \boldsymbol{x}^n$ and go to step 2 (restart).

The line search method implemented is the standard form of the Powell quadratic interpolation method. The convergence criteria used for the mesh movement problem is $||\boldsymbol{x}^i - \boldsymbol{x}^{i-1}|| < 1 \times 10^{-5}$.

Appendix B

Newton Method With Line Search

Newton's method is a second order line search method, and minimizes a function $f(\boldsymbol{x})$, by solving for $\nabla f(\boldsymbol{x}) = 0$ iteratively. Given an initial starting guess of $\boldsymbol{x}^0$, Newton solves the updated value by:

1. $\nabla^2 f\left(\boldsymbol{x}^{i-1}\right) \triangle \boldsymbol{x} = -\nabla f\left(\boldsymbol{x}^{i-1}\right),\ i = 1, 2, ...$
   (a) solve $\triangle \boldsymbol{x}$
2. $x^i = x^{i-1} + \triangle x$
3. Repeat until $||\triangle x|| < \varepsilon$

where ε is the convergence criteria.

The method in its original form is highly dependent on starting point, and convergence is not guaranteed. The method can be made more robust, by using Newton's method to compute the search direction and using a standard line search to determine the step size in the computed direction. The modified form of Newton's method for iteration i is then given by:

1. $\nabla^2 f\left(\boldsymbol{x}^{i-1}\right) \boldsymbol{u}^i = -\nabla f\left(\boldsymbol{x}^{i-1}\right)$
   (a) Solve $\boldsymbol{u}^i$.
2. $\boldsymbol{x}^i = \boldsymbol{x}^{i-1} + \lambda_i \boldsymbol{u}_i$ such that
   $f\left(\boldsymbol{x}^{i-1} + \lambda_i \boldsymbol{u}^i\right) = \min_{\lambda} f\left(\boldsymbol{x}^{i-1} + \lambda \boldsymbol{u}^i\right)$ (line search).

Once again, Powell's quadratic interpolation method was used for the line search and the convergence criteria used for the mesh movement problem is $||\boldsymbol{x}^i - \boldsymbol{x}^{i-1}|| < 1 \times 10^{-5}$.