Model Order Reduction

using the
Discrete Empirical Interpolation Method

R.J. Dedden
Model Order Reduction

using the

Discrete Empirical Interpolation Method

MASTER OF SCIENCE THESIS

For the degree of Master of Science in Mechanical Engineering at Delft
University of Technology

R.J. Dedden

October 25, 2012

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology
Abstract

The invention of the computer opened new research fields in physics and engineering. One of the developments is the Finite Element Method (FEM). It has matured over the last decades, resulting in complex applications with many Degrees Of Freedom (dofs). The large number of dofs made the finite element problems expensive to solve. To make Finite Element Analysis (FEA) less expensive, Model Order Reduction (MOR)-techniques have been developed. These techniques approximate the original full order problem by a system of lower dimension. This is done by writing the displacement field in terms of a set of reduced coordinates. For linear problems, this can considerably reduce the amount of computations required. However, the number of computations for nonlinear problems can not be reduced in the same way.

This thesis focusses on a special type of nonlinear problems, namely geometrically nonlinear problems. In general, the internal force of such systems consists of both linear and nonlinear contributions. Whereas the linear contribution can be reduced with the reduced coordinates, the evaluation of the nonlinear term requires the full order displacement field to be known. This forms a bottleneck in the computations in terms of computation time.

A solution to the bottleneck of the nonlinear term is found in the Discrete Empirical Interpolation Method (DEIM). The DEIM is used in combination with a Proper Orthogonal Decomposition (POD), yielding the POD-DEIM reduction of the nonlinear part of the internal force. The POD-DEIM reduction approximates the space of the nonlinear part of the internal forces by a POD. The resulting reduced subspace is then interpolated with DEIM.

In practice, the POD-DEIM reduction bores down to an approximation of the nonlinear part of the internal force that requires only a few components (dofs) of the nonlinear internal force vector. Using these few components, the remaining components are approximated through interpolation.

The advantage of DEIM over other reduction methods that are able to reduce nonlinear terms in systems of equations, is that the DEIM is able to select its components or collocation points from the full set of dofs automatically. This has the advantage that no user input is required in the reduction of the nonlinear finite element model.

The DEIM has been tested on several load case examples, using mass-spring systems and geometrically nonlinear bar and beam elements. Two different approaches have been successfully
applied to bar elements. The direct approach applies the POD-DEIM reduction directly on the nonlinear term in the system of equations that results from the finite element model. The unassembled approach applies POD-DEIM to the nonlinear terms before the finite element model is assembled. Both methods showed that accurate results can be obtained while using the nonlinear response of only a limited number of dofs.

Although the potential of DEIM for geometrically nonlinear finite elements has been shown, some difficulties will have to be overcome before DEIM can be used on general problems.

First of all, both the direct and the unassembled approach have some inherent inefficiencies that limit the computational savings. The direct approach generally selects a high number of finite elements, because the selected dofs are often shared by many elements. This problem is solved using the unassembled approach. However, the subspace of unassembled nonlinear internal forces used in the unassembled approach is generally of a high dimension. This makes the POD-DEIM reduction in itself computationally intensive, which is undesired.

A second problem is that instability occurred in some of the POD-DEIM reduced responses. The reason for this instability is still unknown, but it is suspected to be related to the fact that the reductions done on the displacement field and the nonlinear part of the internal force are independent and thus have no common optimality criterion.

The third important issue involves the application of POD-DEIM on finite elements that describe a combination of different types of internal forces and or moments. Snapshots of these so-called heterogeneous internal forces will have to be weighted before they can be used as an input to the POD.
# Table of Contents

**Preface**
- xi

1 **Introduction**
  - 1-1 Computers & Numerical Simulation ................................................. 1
  - 1-1-1 The early days of Finite Element Calculations ................................. 2
  - 1-1-2 Improvements in Computers ......................................................... 2
  - 1-1-3 Software for Design Purposes ..................................................... 3
  - 1-1-4 The Future of Finite Element Simulation ........................................ 3
  - 1-2 Model Order Reduction ................................................................. 4
  - 1-3 Conclusion ....................................................................................... 6

2 **Background**
  - 2-1 A brief introduction to the Finite Element Method ............................. 9
  - 2-1-1 Discretising Continua ...................................................................... 9
  - 2-1-2 The Finite Element Method ........................................................... 11
  - 2-1-3 Remarks ....................................................................................... 15
  - 2-2 Geometrically Nonlinear Structures ................................................. 15
    - 2-2-1 A Linear-Elastic Bar with Geometrical Nonlinearity ....................... 16
    - 2-2-2 Nonlinear Equations .................................................................. 18
  - 2-3 Solving the Equations of Motion ....................................................... 18
    - 2-3-1 Static Linear Systems .................................................................. 19
    - 2-3-2 Static Nonlinear Systems .......................................................... 19
    - 2-3-3 Dynamic Linear Systems ............................................................ 20
    - 2-3-4 Dynamic Nonlinear Systems ....................................................... 22
  - 2-4 Nonlinear Equations & Computation Time ......................................... 22

---

Master of Science Thesis

R.J. Dedden
# Table of Contents

3 Overview of Reduction Methods  
3-1 Reduction by Projection ............................................. 27  
3-1-1 Interpretation ......................................................... 29  
3-2 The Bottleneck of Reduced Nonlinear Systems ......................... 29  
3-3 Reduction Methods for Nonlinear Functions ........................ 30  
3-3-1 Missing Point Estimation ............................................ 30  
3-3-2 Interpolation of Function Snapshots .............................. 31  
3-3-3 Drawbacks of MPE and IFS ........................................... 32  

4 Discrete Empirical Interpolation Method  
4-1 The DEIM Algorithm ................................................... 35  
4-1-1 Nonlinear Dynamic Systems ........................................ 35  
4-1-2 DEIM ................................................................. 37  
4-1-3 A graphical interpretation of DEIM ............................... 40  
4-1-4 Reducing degrees of freedom with DEIM ......................... 42  
4-2 Approximating the Jacobian with DEIM ............................. 48  
4-2-1 Rank deficiency ....................................................... 48  
4-3 DEIM for general nonlinear functions ............................... 49  
4-3-1 Point-to-point and general nonlinear functions ................. 49  

5 Example Load Cases  
5-1 1 DOF Problem with connected non-linear springs .................. 51  
5-1-1 Non-linear Spring Model ............................................ 51  
5-1-2 Model Geometry ..................................................... 52  
5-1-3 Equations of motion ................................................ 53  
5-2 Bar Elements: Truss frame .......................................... 55  
5-2-1 Model Geometry ..................................................... 56  
5-3 Beam Elements: Clamped-Clamped Beam ............................. 56  
5-3-1 Load case ............................................................. 57  
5-4 Remarks ................................................................. 57  

6 Implementing DEIM in the Finite Element Method  
6-1 Testing of DEIM in Finite Element Analysis ........................ 60  
6-2 Direct Application of DEIM on System Equations .................. 60  
6-2-1 1 dof Nonlinear Mass-Spring System ............................ 61  
6-2-2 Truss Frame .......................................................... 64  
6-3 Application of DEIM on Unassembled Nodal Forces ................. 70  
6-3-1 Selecting Finite Elements with the Direct Approach of DEIM ... 70  
6-3-2 Selecting Finite Elements using Unassembled Nodal Forces .... 71  
6-3-3 Truss Frame - Unassembled DEIM ................................ 72  
6-3-4 Efficiency of unassembled DEIM .................................. 73  
6-4 Conclusion ............................................................... 76
List of Figures

1-1 Two important computers of the twentieth century. ........................................ 2
1-2 Numerical model ........................................................................................................ 6
1-3 Reduced numerical model .......................................................................................... 7

2-1 A Continuum bar ......................................................................................................... 10
2-2 A linear elastic bar discretised with finite element bars. ............................................ 14
2-3 A Linear versus a nonlinear bar ................................................................................... 16
2-4 Newton-Raphson steps for a nonlinear $F, q$-function. ........................................... 20
2-5 Newton-Raphson method ......................................................................................... 21
2-6 Newmark method for linear dynamics ....................................................................... 23
2-7 Newmark method for nonlinear dynamics ................................................................. 24

4-1 Clamped-clamped beam ............................................................................................. 40
4-2 DEIM iterations 1-3 .................................................................................................... 43
4-3 DEIM iterations 4-6 .................................................................................................... 44
4-4 Examples of a componentwise and a non-componentwise function ...................... 50

5-1 Some examples of cubic nonlinear spring characteristics. ...................................... 52
5-2 Load case using four one-dimensional springs ......................................................... 53
5-3 Load case using five one-dimensional springs ......................................................... 53
5-4 Truss frame load case ............................................................................................... 56
5-5 Clamped-clamped beam load case. .......................................................................... 57

6-1 Static response of the nonlinear coupled masses ...................................................... 62
6-2 Basis vectors of the displacement and the nonlinear internal force field. .............. 63
6-3 Displacement-time responses for the mass-spring system ...................................... 64
6-4 Nonlinear Force-time responses for the mass-spring system .................................. 65
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-5</td>
<td>Results for a ramp loading on the truss frame.</td>
<td>67</td>
</tr>
<tr>
<td>6-6</td>
<td>Displacement response for node 14 of the truss frame.</td>
<td>68</td>
</tr>
<tr>
<td>6-7</td>
<td>Results for the direct approach to DEIM</td>
<td>69</td>
</tr>
<tr>
<td>6-8</td>
<td>Direct versus unassembled approach to DEIM</td>
<td>72</td>
</tr>
<tr>
<td>6-9</td>
<td>Singular values of the unassembled nonlinear part of the internal force</td>
<td>74</td>
</tr>
<tr>
<td>6-10</td>
<td>Direct application of DEIM on the Truss Frame</td>
<td>75</td>
</tr>
<tr>
<td>7-1</td>
<td>Deformed shape and responses of the beam</td>
<td>78</td>
</tr>
<tr>
<td>7-2</td>
<td>Singular values and orthonormal basis vectors of the beam load case.</td>
<td>80</td>
</tr>
<tr>
<td>7-3</td>
<td>Instability for the beam</td>
<td>81</td>
</tr>
<tr>
<td>7-4</td>
<td>Mappings between unreduced spaces and reduced subspaces</td>
<td>85</td>
</tr>
<tr>
<td>A-1</td>
<td>Bar element</td>
<td>99</td>
</tr>
</tbody>
</table>
List of Tables

6-1 Required elements for the internal forces at DEIM-selected dofs using the direct approach. .......................................................... 71

6-2 Required elements for the internal forces at DEIM-selected dofs using the unassembled approach. .................................................. 73
About a year ago, I was still undecided on what kind of thesis subject I was going to choose. During the year before, I had done an interesting internship at BMW in Germany. However, I felt that choosing a project at BMW avoided me from doing something in the field that I always liked: Computational Mechanics. With my internship and first assignment already finished, my thesis project seemed to be my last chance to do something in this field. After having told Daniel Rixen about this, he immediately hinted me to talk to Paolo Tiso for some interesting projects.

Paolo explained me that he had been studying the Discrete Empirical Interpolation Method, but he had faced some difficulties that slowed down the progress on this topic. The topic immediately appealed to me, but I was still a bit reluctant to make a decision. Would it be smart to choose a topic on which my supervisor had already been stuck for some time? What would that mean in view of the contribution I was asked to make? It all seemed a bit risky to pick a subject with absolutely no guarantees to success. However, the idea of being able to work on such a challenging subject, with proper support from experienced researchers, still sounded very attractive. After I was introduced by Paolo to the mathematics of DEIM, I became convinced that the project was well worth the risk.

During the past months, the research became kind of addictive. In fact, I almost forgot that I still needed to finish some of my courses. I particularly liked the combination of mathematics and mechanics and the challenge of translating concepts from the one field into the other. It reminded me of when I was a kid, playing with a big box of Lego bricks. Lego bricks are versatile: They can be used to build everything within a kid’s imagination. This project made me do the same thing, except that the box with bricks was replaced by a ‘toolbox’ of mathematical algorithms. Understanding the properties of these versatile mathematical ‘bricks’ allows their full potential to be unleashed. The model reduction strategy that results from combining these ‘bricks’ can thus be seen as a Lego design.

Looking back, I must admit that not all bricks are yet in the right place, and some loose bricks still need to find their place. Also, some totally new bricks might be needed to finalise the design. Still, the work-in-progress looks promising. I enjoyed working on the contributions I made, hoping that at some point in time I, or someone else, might be able to finish what has been started.
Chapter 1

Introduction

The purpose of this thesis is to investigate the potential for the application of Discrete Empirical Interpolation Method (DEIM) to nonlinear structural mechanics. This mathematical algorithm is studied with the intention to use it for Model Order Reduction (MOR). The research field of Model Order Reduction studies ways to (mathematically) simplify numerical models. The motivation for studying MOR can be found from the field of numerical simulation. This chapter introduces the subject by briefly presenting the history of numerical simulation and how the current status quo motivates both scientists and engineers to study MOR.

1-1 Computers & Numerical Simulation

One of the first digital computers that were used for computing physical phenomena was developed in Germany by Konrad Zuse. Zuse studied civil engineering and got tired of having to do great amounts of stress calculations by hand. He therefore designed his own digital, mechanically operated computer, which he constructed in his parents’ living room. By 1938, his first model, the Z1, was finished. Being a rather unreliable machine, the Z1 was modified over time by Zuse by replacing unreliable parts with enhancements [19].

In 1939, at the start of World War II, Zuse was working for the German Aircraft Research Institute where he continued to work on computers. In 1941 the Z3 was completed, which operated electromechanically by the use of over 2000 relays. During the war, use was made of the Z3 to study the aircraft wing flutter phenomenon. This proved the useful role that computers could have in science and engineering.

After the war, the pioneering companies in high performance computing were located in the United States. Most notably is the company Control Data Corporation (CDC) which coined the term supercomputer in 1964 when their CDC6600 beat competitors in terms of speed by a factor of ten [20]. At the same time, progress was made in the field of numerical calculation methods such as the Finite Element Method (FEM). Simulation techniques as FEM were developed to perform accurate calculations on structures that were unable to be
modelled purely analytically. In these early days, the numerical calculations were done by hand. However, the number of calculations quickly outgrew the capacity of engineers that had to solve problems by hand. The availability of computers made it possible to automate the solving of such calculations.

1-1-1 The early days of Finite Element Calculations

The term Finite Element Method was originally suggested by professor Clough from the University of California at Berkeley. Although developments had started already in the 1950’s, it is claimed that the first functioning FEM computer program was introduced in 1961 [16]. The aim of this program was to do stress analysis on a gravity dam using triangular elements (see Figure 1-2a).

The gravity dam is a typical example of how FEM was used in those early days: Typically a design was made of a structure. After the design was considered finished, the estimated strength was validated using the Finite Element Method. This was done by modelling the design with a mesh of elements that were subjected to realistic load cases. Hence the method was used mainly as a validation tool for already developed designs.

1-1-2 Improvements in Computers

The reason that finite element calculations were mainly done for the purpose of post-design phase validation was the limited availability of computing power in those early days. The CDC 6600 was sold for $8 million, making it a very expensive machine. Not only the hardware (electronics, semiconductors, but also cooling systems) was responsible the high cost, also the software contributed an increasing amount to the total price of supercomputers. This triggered the development of the UNIX (Unix) operating system, which was launched in 1969 by AT&T’s Bell Labs [20]. Unix was modularly designed and could be easily adopted by many different computers. This in turn triggered the development of relatively cheap computers, which eventually lead to the computer being available for professional use to large numbers of people.

R.J. Dedden Master of Science Thesis
1-1-3 Software for Design Purposes

In the 1970’s the first commercial FEM software programs came on the market. From the early 1980’s on, this type of software could not only be used on mainframes or supercomputers, but also on the smaller and cheaper workstations [18]. Next to that, computers became increasingly popular as a design tool, for example in Computer Aided Design (CAD).

At this point in time the availability of both hardware and software brought up a new approach in technical product development. The first application of the FEM had been a post-design validation, but from the 1980’s on it is used more and more as an integral tool in the design process. During the design phase of large structural systems and machinery, finite element analyses are carried out multiple times as part of the design iterations.

Nowadays, many CAD-packages include finite element toolboxes or functions to allow easy exporting of designs to file formats that can be loaded into specialised FEM software. Whereas in the past FEM calculations were carried out mainly by a small group of specialists, today most users are design engineers without specific knowledge of the FEM. Some users of FEM software do not even have a background in physics or engineering, but make use of this software in their spare time projects (see Figure 1-2b).

1-1-4 The Future of Finite Element Simulation

The ongoing popularisation of finite element analysis among a broad range of industries had a number of consequences that are undesirable from the perspective of computing, but that were inevitable:

- The merging of the design process with FEM-calculations implies that finite element calculations are done multiple times.
• Closely related is the appearance of new iterative methods that make use of FEA. One example is that of topology optimisation, in which an optimal design is generated iteratively by many successive FEM calculations.

• As the method became more and more mature, the applications became increasingly complex. This is due to both the complexity of new types of finite elements and due to the increasing mesh refinement. Both are triggered by the need for more accurate and realistic calculation results.

• Despite the enormous leaps taken in the development of computers, computing power is still expensive. The amount of money that companies are able to spend on hardware and software is limited, and so is the available computing power.

• In close relation to the above bullet, the cost is also affected by the computation time needed to solve finite element problems. The productivity of people working with FEM-software decreases as they are required to wait considerable amounts of time before their models are computed. The cost-effectiveness of the design process suffers from this.

The general observation that can be done is that:

The interest in methods that can speed up the calculations done in finite element simulations will still exist in the near future. This interest is more likely to increase, rather then to disappear.

### 1.2 Model Order Reduction

The future demands make it necessary to invest in methods that reduce the computational bottleneck. Roughly three types of approaches exist:

1. Improve the computer hardware such that it can keep up with the growing demands. Faster hardware enables shorter calculation times, which goes hand in hand with cost savings.

2. Increase the speed of simulation software by optimising the program code. Software written in a way that it maximises the utilisation of system resources per time unit, while at the same time minimising the amount of unnecessary operations is beneficially.

3. Reduce the amount of necessary calculations by the application of smart techniques that filter model properties/ parameters that are important from those that are not important to take into consideration.

The first two approaches are subject of study in the fields of informatics and electrical engineering. Of course many other scientific and engineering disciplines are involved in hardware and software development, but the bottom line is that these approaches do not require specific knowledge of the numerical simulation techniques like FEM for which the computers are used.
The subject of this thesis is found in the third approach. MOR can be viewed as a subset of the third approach, since it is one of many reduction methods. All these methods are devised under the assumption that in general, models of physical phenomena inhere a certain amount of parameters that are of little or no importance for the overall model behaviour. For the case of Model Order Reduction this assumption can be stated more specifically:

- Assume that the model can be represented as a black box. This means that the simulation model can be viewed as a transfer function $H(x)$ from the $n$ inputs $x = [x_1, \ldots, x_n]^T \in \mathbb{R}^{n \times 1}$ to the $n$ outputs $y = [y_1, \ldots, y_n]^T \in \mathbb{R}^{n \times 1}$:

$$y = H(x). \quad (1-1)$$

See also Figure 1-2 for a block diagram representation of this transfer function.

- Suppose now that not all of the inputs and outputs are of equal importance to calculate the model response. In many cases it is possible to approximate the behaviour of the inputs and outputs by linear combinations of a fewer amount $m \ll n$ of newly defined state variables. This means that the inputs and outputs can be written as:

$$x = V\tilde{x} + r_x, \quad V \in \mathbb{R}^{n \times m}; \quad \tilde{x}, r_x \in \mathbb{R}^{m \times 1} \quad (1-2)$$

$$y = U\tilde{y} + r_y, \quad U \in \mathbb{R}^{n \times m}; \quad \tilde{y}, r_y \in \mathbb{R}^{m \times 1}. \quad (1-3)$$

Here $V$ and $U$ are matrices, and $\tilde{x}$ and $\tilde{y}$ are vectors of the reduced state variables. Together, these terms approximate the inputs $x$ and outputs $y$ as a weighted sum of the columns of $V$ and $U$. The vectors $r_x$ and $r_y$ represent the errors of the respective approximations.

- Using the newly defined inputs $\tilde{x}$ and the matrices $V$ and $U$, a new model $\tilde{H}(V, U, \tilde{x})$ can be formed. This model is defined in terms of the $m \times 1$ state vectors $\tilde{x}$ and $\tilde{y}$, and thus has fewer variables than the original $n$-dimensional model. The outputs $\tilde{y}$ can be transformed back into $y$ using matrix $U$ as defined in (1-3). The block diagram Figure 1-3 schematically shows the procedure.

Note that the reduced states $\tilde{x}$ can not be found straightforwardly from (1-2), because this equation is not invertible. The derivation of $\tilde{x}$ is dependent on the column vectors $v_1, \ldots, v_m$ of $V$. The matrix $V$, as well as the matrix $U$ are derived from the original model. Many different algorithms exist to obtain these matrices. Some of them are treated in chapters 3 and 4.

The idea behind the synthesis of the reduced model $\tilde{H}(V, U, \tilde{x})$ is that the fact that a smaller amount of inputs and outputs is taken into account is reflected in a reduced amount of calculations that have to be solved by the computer. Generally, the computational effort is reduced when less state variables are part of the model. A smaller number of inputs and outputs implies a smaller number of input-output relations that have to be solved for in the model. However, care must be taken that this benefit is not outweighed by the fact that the introduction of $V$ and $U$ causes extra calculations. For example, the operation that transforms $\tilde{y}$ into $y$ involves the matrix-vector product $V\tilde{y}$ that otherwise would not have been part of the solving algorithm.
The potential for the application of MOR is always a delicate trade-off. On the one hand, a reduction from \( n \to m \) state variables or Degrees Of Freedom (dofs) can have a positive effect on the computation times. Examples exist of calculations where reduction techniques decrease the calculation times by orders of magnitude [3]. On the contrary, reduction goes hand in hand with loss of accuracy, because the residuals \( r_x \) and \( r_y \) are not taken into account by the reduced model. This may result in inaccurate results, but also in model instabilities that cause the solving process to slowly converge or even diverge. Slow convergence means that many calculation steps are needed to find the final solution, thus slowing down the solving process. Divergence is even less desirable since in the end it will produce results that are not valid.

1-3 Conclusion

Despite the apparent drawbacks (or perhaps because of the apparent drawbacks) a lot of research is dedicated to Model Order Reduction. The availability of computers has been a major incentive to this research. In the past, reduction algorithms could be sparsely tested in practice. Today the performance of any reduction method can be tested with relative ease. Together with the ‘need for speed’ in the industries, this forms a fertile soil for research to grow on.

Finite element models are a perfect subject for Model Order Reduction. From the expertise with finite elements that has been part of the mechanical engineering discipline for many years now, the focus has been shifted to applications of MOR on these finite elements. The study of the DEIM carried out in this report is a typical example of how mechanical engineers move a bit into the field of mathematics. The main interest here is not the understanding of the reduction technique of DEIM, but merely the fact that the field of mechanical engineering could benefit greatly from speed up calculations using MOR.

This thesis will describe the application of DEIM on Finite Element Analysis. Therefore, the next chapter will introduce the Finite Element Method and will identify the computational bottlenecks that occur in finite element modelling.
Figure 1-3: The reduced numerical model. Here the matrices $V$ and $U$ are derived from the original model $H(x)$ and used to form the reduced representation $\tilde{H}(\tilde{x})$. 
Chapter 2

Background

The area of focus of this thesis is Model Order Reduction (MOR) for nonlinear mechanics. More specifically, reduction of finite element models of geometrically nonlinear structures are the subject of this thesis. This chapter introduces the difficulties that are faced when geometrical nonlinear behaviour is studied using the Finite Element Method (FEM). This will give the reader a proper understanding of the type of problems that are to be solved using the DEIM algorithm.

The first part of the chapter will briefly introduce the finite element method. The second part illustrates that solving these equations can demand high computational power when nonlinearities are present. This explains the motivation for this research in more detail then the motivation in the introduction.

2-1 A brief introduction to the Finite Element Method

The purpose of the derivations in this section is purely to introduce how the FEM yields sets of matrix equations like (2-23). In the upcoming chapters of this report, the main concept of finite elements will be a starting point for the derivation of reduction strategies.

The Finite Element Method originates in the Rayleigh-Ritz method, a method that was invented in the nineteenth century by Lord Rayleigh to study vibration problems. The method was generalised in 1909 by Walther Ritz [12]. In the twentieth century, many publications appeared on the subject of FEM. The introduction presented here is based on [5, ch. 2] and will briefly explain the concept of finite elements and introduce the equations of motion. Further information can also be found in [17].

2-1-1 Discretising Continua

In most of the cases where the response of a continuum to a certain external influence (e.g. externally applied forces, but also any other type of physical phenomenon: electrical voltage,
magnetic fields and so on) is studied, differential equations will need to be solved. For example, in structural dynamics, a structure is subjected to an external force. This force causes deformations and or rigid body motion of the structure. To describe this, one or more equations of motion can be derived. In order to know the response of the structure, this equation(s) must be solved. However, exact solutions to the equation(s) of motion generally do not exist for complex structures. To overcome this problem, the FEM discretises the structure (or continuum) with a finite number of points called nodes. A set of equations of motion can be derived for these set of nodes. Solving the set of equations of motion will then yield an approximation of the (unsolvable) response of the continuous problem.

In the following subsections, a continuum formulation of a linear-elastic bar will be translated into a finite element description. This will serve as an example to illustrate the essentials of the finite element method. The example is based on [14, ch. 1].

A Continuum Linear-elastic Bar

Consider the bar in Figure 2-1, which has a length $L$ and is assumed to be made of linear-elastic material. If the Young’s modulus $E$ is assumed to be constant, then the normal force can be expressed as:

$$N(x) = A(x)\sigma(x) = E A(x) \varepsilon_x,$$

with $\varepsilon_x = \frac{\partial u}{\partial x}$, which is valid for small displacements. If the bar is loaded by external forces $g(0)$ and $g(L)$ at both ends, then the static equilibrium equation can be written as:

$$g(0) - g(L) = N.$$

Due to this external loading, an amount of strain will occur in the bar. Obviously, the stiffness of the bar will resist the external loading. This means that the bar will store an amount of strain energy. From linear continuum mechanics, the strain energy of a linear elastic body is defined as:

$$V_{int} = \int_{V_0} \int_\varepsilon \sigma \varepsilon \, dV.$$

Hence the stresses are integrated over the strains, followed by an integration that takes the contributions of the complete material domain of the body in undeformed state $V_0$ into ac-
count. Note that the choice to integrate over the undeformed state is what is referred to as the Lagrangian description.

For the bar, (2-3) can be simplified by noting that the stress can be simply expressed in scalar form as $\sigma = E\varepsilon$. Also, the volume-integral can be simplified under the assumption of a constant cross section area $A(x) = A$. Together this yields:

$$V_{int} = \int_{V_0} \int_{\varepsilon(x)} \sigma \, d\varepsilon \, dV$$

$$= \int_{V_0} \frac{1}{2} E\varepsilon^2 \, dV$$

$$= \frac{1}{2} \int_{0}^{L} EA\varepsilon^2 \, dx.$$  \hspace{1cm} (2-4)

The strain energy can thus be found by integrating over the length of the continuum bar.

The potential energy of external forces acting on the boundary of an elastic body are expressed as:

$$V_{ext} = -\int_{S_{\sigma}} \bar{t}u \, dS,$$  \hspace{1cm} (2-7)

where $\bar{t}$ represents the externally applied stresses or surface tractions, $u$ de conjugated displacements and $S_{\sigma}$ that part of the body’s surface area that is loaded by $t$. For the bar that is loaded only at its ends, this yields:

$$V_{ext} = g(0)u(0) - g(L)u(L).$$  \hspace{1cm} (2-8)

In the next section it will be shown that the formulations for the potential energies can be approximated with the descritised formulation of the finite element bar.

### 2-1-2 The Finite Element Method

The displacement of the bar has been expressed with the variable $u$, which is a continuous function of the position along the bar: $u = u(x)$. This continuum description is relatively easy to obtain for simple structures such as the bar in Figure 2-1. The function $u(x)$ describes the exact displacement field at arbitrary $x$. However, for complex structures (truss frames of multiple bars, but also different types of structural components as beams, plates and shells) it is generally impossible to find such an exact solution. This is where the Finite Element Method finds its application.

The FEM simplifies the structural mechanics problem by assuming that the displacement field $u(x)$ can be approximated by a series of functions. Notice that $u(x)$ is written in vector form, to generalise the derivations for two and three-dimensional cases. The functions are required to be continuous, such that they can be integrated over the spatial domain $x$. Each of these shape functions has a certain weight assigned to it. The approximated displacement field can then be written as:
\[ u(x) = F(x)q, \] (2-9)

where \( F(x) \) represents an interpolation matrix that contains the shape functions and \( q \) is a vector that contains the weights.

The idea of the FEM is that a certain structure is divided into a number of elements. Each of the elements has a number of nodes. The nodes are most of the time located at the boundary of the elements. Thus adjacent elements share nodes with their neighbours.

The displacement field interpolation described above will use the displacements at the nodes as collocation points. The key property of the FEM is that the functions in \( F \) are picked such that the weights \( q \) represent the displacements at these collocation points. Doing so will ensure that the neighbouring finite elements will have the same displacements at their interconnecting nodes. This thus allows the continuity of the mesh of finite elements to be guaranteed.

With the approximated displacement field (2-9), the strains in the structure can be calculated. If the stresses are also known, then the strain energy can be calculated using (2-3). In turn, the strain energy can be used to find equilibrium equations, as will be shown using the bar example.

**A Linear Elastic Finite Element Bar**

The first step in formulating a finite element model of the bar, is to define the shape functions. The derivation of these functions is not explained here, since this would compromise the brevity of this introduction to the FEM. Define the shape functions:

\[ F_e = \begin{bmatrix} 1 - \frac{x}{L} & \frac{x}{L} \end{bmatrix}. \] (2-10)

The approximated displacement field will then be:

\[ u(x) = F_e q = \begin{bmatrix} 1 - \frac{x}{L} & \frac{x}{L} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}. \] (2-11)

The shape functions express the displacement field along the bar element as a function of the displacements \( q_1 \) and \( q_2 \) at nodes 1 and 2 (see Figure 2-2a). The strain in the bar can be derived from \( u(x) \) by taking the derivative with respect to \( x \):

\[ \varepsilon = \frac{\partial u}{\partial x} = \frac{\partial F_e}{\partial x} q = \begin{bmatrix} -1/L & 1/L \end{bmatrix} q. \] (2-12)

Substituting this expression in (2-6), the internal potential energy (strain energy) can be expressed as:
\[ V_{\text{int}} = \frac{1}{2} \int_0^L EA q^T \frac{\partial F_e}{\partial x} \frac{\partial F_e}{\partial x} q \, dx \]  
\[ = \frac{1}{2} \int_0^L EA q^T \left[ \begin{array}{c} 1 - \frac{1}{L} \ \\ \frac{1}{L^2} \end{array} \right] \left[ \begin{array}{c} 1 - \frac{1}{L} \ \\ \frac{1}{L^2} \end{array} \right] q \, dx \]  
\[ = q^T \frac{1}{2} \int_0^L EA \left[ \begin{array}{c} \frac{1}{L^2} \ \\ -\frac{1}{L^2} \end{array} \right] \frac{d}{dx} q \]  
\[ = q^T EA \left[ \begin{array}{c} 1 \ \\ -1 \end{array} \right] \frac{1}{L} \qquad (2-14) \]  
\[ = q^T K_e q. \quad (2-15) \]

The potential energy of the external forces can now be written in terms of \( q_1 \) and \( q_2 \), since they represent the actual displacements at the ends of the bar:

\[ V_{\text{ext}} = - \left[ \begin{array}{c} q_1 \\ q_2 \end{array} \right] \left[ \begin{array}{c} g_1 \\ g_2 \end{array} \right] = -q^T g. \quad (2-18) \]

If it is assumed that no other potentials are present, then the sum of the internal and external potential energies is zero:

\[ V_{\text{int}} + V_{\text{ext}} = 0 \quad (2-19) \]
\[ q^T K_e q - q^T g = 0. \quad (2-20) \]

The discretised equilibrium equations now follow from the variation of this discretised expression of the potential energies (Lagrange equations):

\[ \frac{\partial V}{\partial q} = K_e q - g = 0 \Rightarrow K_e q = g. \quad (2-21) \]

This concludes the finite element discretisation of the continuum bar. The bar is modelled with one bar element. The matrix \( K_e \) is named element stiffness matrix and the vectors \( q_e \) and \( g_e \) are named element displacement and element external force vector respectively. In the following section, this description will be generalised for multiple bar elements.

### Assembly of multiple elements

The finite element bar described in the previous section consists of one bar element. Suppose now that the bar is discretised with two bar elements (Figure 2-2b). The bar elements have a normal stiffness \( k_1 = EA/L_1 \) and \( k_2 = EA/L_2 \) respectively. Referring to (2-16), the equilibrium equations thus yield:
(a) A bar discretised with one finite element bar. (b) The same bar, now discretised with two bar elements of unequal lengths.

Figure 2-2: A linear elastic bar discretised with finite element bars.

\[
\begin{align*}
    & k_1(q_1 - q_2) - k_1q_1 + (k_1 + k_2)q_2 - k_2q_3 = g_1 \\
    \Rightarrow & \quad \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} \\
    \end{align*}
\]

Hence an expression is found that describes the equilibrium equations of the system of assembled elements 1 and 2. Observe that the assembly of the elements is simply done by putting the element stiffness matrices \( K_e \) of each element in a larger global stiffness matrix \( K \). The two elements are physically linked together at node 2, which is resembled by the fact that component (2,2) of \( K \) is the sum of the lower right component (2,2) of \( K_{e,1} \) and the upper right component (1,1) of \( K_{e,2} \).

Similar expressions can be found for more complex structures with large numbers of elements. Although in this example some important aspects have been omitted (for example, how to assemble elements that are rotated with respect to each other, how to combine different types of elements), the important thing is that the assembly of multiple elements yields (large) systems of equations. The connections between the individual elements is reflected in these equations by the structure of the stiffness matrix.

**From Statics to Dynamics**

Thus far, the derivation of the finite element model of the bar considered only static equilibrium equations. The bar was considered massless and hence no inertia effects were taken into account. Solving (2-23) for the displacements \( \mathbf{q} \) then consists of a simple back substitution, as will be shown in Section 2-3-1.

Assume now that the mass of the bar is taken into account. The total mass is \( m(L_1 + L_2) \), where \( m \) is the mass per unit length of the bar. The total mass is divided among the three nodes. Using derivations based on energies analogous to that of the stiffness matrix, a mass matrix can be derived:

\[
\begin{align*}
    & M_{e,1} = \frac{m}{6} \begin{bmatrix} 2L_1 & L_1 \\ L_1 & 2L_1 \end{bmatrix} \\
    & M_{e,2} = \frac{m}{6} \begin{bmatrix} 2L_2 & L_2 \\ L_2 & 2L_2 \end{bmatrix} \\
    \Rightarrow & \quad M = \frac{m}{6} \begin{bmatrix} 2mL_1 & mL_1 & 0 \\ mL_1 & 2m(L_1 + L_2) & mL_2 \\ 0 & mL_2 & 2mL_2 \end{bmatrix}. \\
\end{align*}
\]
If the inertia is taken into account, then the displacement vector $\mathbf{q}$ becomes time-dependent. The equations of motion in matrix form then yield:

$$\mathbf{M} \ddot{\mathbf{q}}(t) + \mathbf{K}\mathbf{q}(t) = \mathbf{g}(t).$$

(2-25)

Notice that the external force $\mathbf{g}(t)$ is also considered to be a time-dependent function, although this is not necessarily the case. Observe that the dynamic equations of motion (2-25) are a system of ordinary differential equations in time $t$. Hence, the partial differential equation that was a function of (continuous) displacements $q(x)$ and (continuous) time $t$ is translated into a problem of a system of ordinary differential equations in time. The system can be solved for a finite number of time steps using time integration methods that will be briefly outlined in Section 2-3-3.

2-1-3 Remarks

The simple case shown here is easily extended to more complex cases, where larger numbers of elements with more difficult definitions are used to discretise a continuum structure. The bottom line of FEM is that a continuum which has an infinite number of spatial locations is discretised with a finite number of spatial locations or nodes. Interpolation functions (such as (2-9)) are used to approximate the displacement field. Quantities like force or strain can then be derived using this approximated displacement field. The displacement field can be used to find expressions of the discretised system energies. Using the Lagrange equations, the static (or dynamic) equilibrium equations can be derived.

In the case of statics, one or more equilibrium equations can be derived per node that describe how the node displaces under external loading. The equilibrium equations are coupled with each other and can be written in a matrix form (2-23).

In the case of dynamics, the equilibrium equations have the form of ordinary differential equations. Again, these differential equations are coupled with each other and can thus be written as a system (2-25).

As a final remark, it should be noted that thus far no damping is included in the equations. Adding a damping term such as $\mathbf{C} \dot{\mathbf{q}}(t)$, where the damping force is a linear function of the generalised velocities $\dot{\mathbf{q}}(t)$, would make the equations (2-25) more complex. The derivations in this thesis are done without damping taken into account. This does not result in any loss of generality, which explains why the damping terms will not be added to the equations.

2-2 Geometrically Nonlinear Structures

The finite element bar introduced in Section 2-1 has been derived under the assumption of linear elasticity and small displacements. This implies that the stiffness $k = EA/L$ (or the stiffness matrix $\mathbf{K}$) is independent of the deformation state of the structure. If this is the case, then the stiffness matrix of an assembly of bar elements (2-23) is a constant matrix and a linear system of equations results from the Finite Element Analysis. Solving such a linear system is relatively easy, see Section 2-3.
The relation between displacement and internal force is linear when the load is aligned parallel to the main axis. This makes the relation between displacements and internal force non-linear.

Figure 2-3: Geometrically linear and nonlinear behaviour shown for a linear-elastic bar.

However, when large displacements are present, it may not be sufficient to model structural behaviour using a constant stiffness matrix. Large displacements alter the configuration of the system in such a way that the initial stiffness (in the undeformed state) cannot accurately describe the structure’s behaviour in the deformed state. The stiffness matrix must therefore become a function of the displacements, such that it can be updated to the new deformed state.

The reorientation of the structure under loading causes the internal force to be a nonlinear function of the displacements. This means that doubling an arbitrary displacement \( u \) into \( 2u \), will not yield a corresponding internal force \( F \) that is twice as high as the one belonging to displacement \( u \): \( F(2u) \neq 2F(u) \).

If the nonlinear behaviour described above is caused by the reorientation of a structure due to large displacements, then the structure is said to be geometrically nonlinear. An example of such behaviour will be presented using the linear elastic bar element of Section 2-1-2.

### 2-2-1 A Linear-Elastic Bar with Geometrical Nonlinearity

Suppose that the linear elastic bar element shown in Figure 2-2a is pin supported on one side, and loaded in axial direction on the other side by a static force (see Figure 2-3a). The bar will deform under the loading. The normal force \( N \) and the external force \( g \) can then be written as:

\[
N = \sigma A = \varepsilon EA, \quad g = -N,
\]

and recall that the strain is:

\[
\varepsilon = \frac{\Delta L}{L_0} = \frac{u}{L_0}.
\]

Thus far, the external load and the main axis of the bar have always remained aligned with each other. The external loading did not cause any structural reorientation, which explains why the initial stiffness matrix is not a function of the deformation.
Now suppose that a new load case is defined in which the bar is oriented such that its main axis is not aligned parallel to the external force. In Figure 2-3b, the bar is rotated such that the end is displaced in the vertical direction by an amount \(z\) [4, ch. 1]. This rotation is done without stretching the bar. However, after rotating, the end is constrained in horizontal direction. To describe deformations from the new initial state, an initial displacement \(z_{\text{perpendicular}}\) is introduced, and deformations in the direction of \(z\) are denoted with \(w\). If the bar is loaded with a vertical force \(g_w\) at the end, a deformation \(w\) will occur and the relation between \(g_w\) and the normal force \(N\) will be:

\[
g_w = N \sin \theta \approx \frac{N(z + w)}{l} \quad (2-28)
\]

In the linear bar model, the strain was expressed in terms of deformation \(u(x)\) only. This simple strain measure is valid only if the displacements \(w\) remain small when compared with the length \(L\) of the bar. However, here the force \(g_w\) did cause a relatively large displacement in the bar, without any deformation components \(u\) in \(x\)-direction. Obviously, the linear model (2-1) can not describe how high the normal force due to the large vertical displacement of the end of the bar will be. The strain measure defined in (2-27) is not suited for these vertical displacements.

Using Pythagoras’ theorem, a new strain measure can be defined that is able to take the displacements \(z\) and \(w\) into account:

\[
\varepsilon = \sqrt{(z^2 + w^2) + l^2} - \sqrt{z^2 + l^2} \\
\approx z \frac{w}{l} \left( \frac{w}{l} \right) + \frac{1}{2} \left( \frac{w}{l} \right)^2. \quad (2-29)
\]

\[
\approx \frac{z^2 w}{2l^2}. \quad (2-30)
\]

The intermediate steps in this derivation have been omitted here (see [4, ch. 1], from which this example is adopted). Observe that the new strain is a quadratic function of \(w\). Thus the normal force in the bar depends quadratically on \(w\):

\[
N = EA \left( \frac{z}{l} \left( \frac{w}{l} \right) + \frac{1}{2} \left( \frac{w}{l} \right)^2 \right). \quad (2-31)
\]

Substitution of the normal force into (2-28) yields an explicit function of the vertical component of the internal force at the end node:

\[
F_z = \frac{EA}{l^3} \left( z^2 w + \frac{3}{2} w^2 + \frac{1}{2} w^3 \right). \quad (2-32)
\]

Clearly, the vertical force \(F_z\) is a nonlinear function of the displacement \(w\). The stiffness matrix in this example is a scalar, that is defined as the derivative of the vertical internal force with the vertical displacement:

\[
k = \frac{\partial F_z}{\partial w} = (z + w) \frac{dN}{dw} + \frac{N}{l} = \frac{EA}{l} \left( \frac{z}{l} \right)^2 + \frac{EA}{l} \left( \frac{2zw + w^2}{l^2} \right) + \frac{N}{l}. \quad (2-33)
\]
Substitution of (2-31) into (2-33) expresses the stiffness as a function of the initial geometry $z$, $l$ and displacement $w$. Hence both the internal force $F_y$ and the tangential stiffness $k$ are a function of the geometry.

### 2-2-2 Nonlinear Equations

The dependence of the internal force and stiffness on the geometry, as shown in the previous subsection, is a typical example of geometrically nonlinear structural behaviour. Although the material of the bar is linear elastic, the large displacements make the equilibrium equation nonlinear. This concept can be easily generalised for multiple dofs.

Suppose that the equations of motion (2-25) are modified by replacing the linear internal force $Kq(t)$ by a term $f_{int}(q(t))$ that accounts for the geometrical nonlinear behaviour. In many cases, this internal force consists of a linear and a nonlinear part, which may be separated:

$$f_{int}(q, t) = Kq(t) + f(q(t)).$$

Expressing this explicitly, the static equilibrium equations write:

$$Kq + f(q) = g$$

and the dynamic equations of motion write:

$$M\ddot{q}(t) + Kq(t) + f(q(t)) = g(t),$$

Although it is not always possible to separate a linear part of the internal forces, the choice is made to explicitly write the separate parts. One good reason for this is that DEIM will be used on the separated nonlinear term only, as will be explained in Chapter 4. As a final remark, it is pointed out that the time-dependence of $f$ is only present through the time-dependence of $q = q(t)$. Hence the nonlinear part of the internal force can be written as $f = f(q(t))$ without loss of generality.

### 2-3 Solving the Equations of Motion

The derivations in the previous sections have led to four different types of systems of equations, all derived from finite element models. The linear static equations (2-23) are the shortest. The nonlinear counterpart of these equations featured an extra stiffness term to include that the stiffness is dependent on the geometry. For the case of dynamic loadings (2-25), the nonlinear case is derived in an analogous way from the linear equations (2-36).

Solving these four types of equations requires different techniques. This section will provide an overview. The information provided is not meant to be complete, but it should provide enough details for the reader to understand the difficulties that are experienced, especially in the nonlinear cases.
2-3 Solving the Equations of Motion

2-3-1 Static Linear Systems

Solving the linear static load case (2-23) involves a ordinary back substitution:

\[ \mathbf{Kq} = \mathbf{g} \Rightarrow q = \mathbf{K}^{-1}\mathbf{g}. \]  
(2-37)

Hence no differential equations are involved and the problem can thus be solved algebraically in one step.

2-3-2 Static Nonlinear Systems

The nonlinear system (2-35) cannot be solved in one step, because the stiffness depends on the orientation. The initial internal force \( \mathbf{f}_0 = 0 \) and stiffness matrix \( \mathbf{K}_0 \) at the equilibrium position \( \mathbf{q}_0 \) are not valid to describe the model for nonzero displacements away from the initial state \( \mathbf{q}_0 \). To overcome this inaccuracy, \( \mathbf{f}(t) \) and \( \mathbf{K} \) need to be updated during solving. A common updating scheme is the Newton-Raphson method. Here a brief overview of the method is given. More details can be found in [5, p.597] and [10].

**Newton-Raphson Iterations for a Scalar Function**

The main property of the Newton-Raphson method is that it relies on successive linearisations of the nonlinear force-displacement characteristic (see Figure 2-4). The procedure involves the following steps:

1. The external load \( \mathbf{g} \) is divided in a number of load steps through multipliers \( \lambda = [\lambda_1, \ldots, \lambda_m]^T \) with \( 0 < \lambda_1 < \lambda_2 < \ldots < \lambda_m = 1 \).

2. At the initial state \( \mathbf{q}_0 = 0 \), the function \( F(q) \) is linearised. The response of this linearised function to a external load \( \lambda_1 \mathbf{g} \) is calculated. This means solving for \( q_1 \):

\[ \frac{\partial F}{\partial q} \bigg|_{\mathbf{q}_0} q_1 = \lambda_1 \mathbf{g}. \]  
(2-38)

3. The linearised internal force \( \frac{\partial F}{\partial q} \big|_{\mathbf{q}_0} q_1 = kq_1 \) and external force \( \lambda_1 \mathbf{g} \) are not in equilibrium due to the linearisation inaccuracy. This load imbalance (or residual) is:

\[ r = \lambda_1 \mathbf{g} - f(q_1), \quad f(q_1) = kq_1. \]  
(2-39)

4. The residual \( r \) is iteratively minimised by evaluating the new stiffness \( k(q_1) \) and using that stiffness to do a correction to the displacement \( q_1 \):

\[ \Delta q = r/k(q_1) \Rightarrow q_1 = q_1 + \Delta q. \]  
(2-40)

This step will be repeated several times, until the residual becomes smaller than a predefined tolerance:

\[ r < \epsilon. \]  
(2-41)
5. If the residual becomes smaller than the tolerance, the external load step is increased from $\lambda_i$ to $\lambda_{i+1}$ and the process is started over again at step 2 with a new initial state $q_{i+1}$. Final convergence is granted when the iterative procedure has converged for $\lambda_i = 1$.

The derivations above are easily generalised to multi variable problems. The algorithm is visualised for the general case of systems with multiple Degrees Of Freedom in Figure 2-5. Switching to the general case of non-scalar functions with $n$ Degrees Of Freedom (dofs), the following becomes clear:

- The Newton-Raphson method solves nonlinear problems by solving a series of linear equations.
  - For each of the $N$ load steps $\lambda_i \mathbf{g} \in \mathbb{R}^{n \times 1}$, a residual is iteratively minimised.
  - This involves successively solving $m$ linear problems.
- For each of the linear problems, a corresponding linearised stiffness $\mathbf{K}(q_i)$ needs to be evaluated. Solving for $\Delta \mathbf{q}$ requires the inverse of this stiffness matrix $\mathbf{K}(q_i)$.
- This means that a system $\Delta \mathbf{q} = \mathbf{K}(q_i)^{-1}\mathbf{r}$ must be solved.
- It can thus be concluded that solving a nonlinear problem requires an equal amount of computation steps as that are needed to solve $N \times m$ linear equations (see Section 2-3-1).

2-3-3 Dynamic Linear Systems

The most common method to solve dynamic equations (2-25) for an amount of time steps $t_1, t_2, \ldots, t_N$ is the implicit Newmark method (see Figure 2-6). The Newmark method will not be detailed here extensively, but some aspects are important to remember:

1. The Newmark time integration method calculates the responses $\mathbf{q}_{n+1}, \dot{\mathbf{q}}_{n+1}, \ddot{\mathbf{q}}_{n+1}$ at time instance $t_{n+1}$ using the responses $\mathbf{q}_n, \dot{\mathbf{q}}_n, \ddot{\mathbf{q}}_n$ at $t_n$. 

R.J. Dedden

Master of Science Thesis
Initialise:
\[ f, g, \lambda, q = q_0 \]

Load step increment:
\[ i = i + 1 \]

Calculate

**Internal force + Stiffness:**
\[ f(q_i), \quad \frac{\partial f(q_i)}{\partial q_i} = K(q_i) \]

**Residual Evaluation:**
\[ r = \lambda_i g - f(q_i) \]

**Newton-Raphson Correction**
solve for \( \Delta q_i \):
\[ K(q_i)\Delta q_i = r \]

**Update displacements:**
\[ q_i \leftarrow q_i + \Delta q_i \]

Check Convergence:
\[ \frac{|r|}{|g|} < tol \]

Store Results:
\[ F_s \leftarrow [F_s, f_i] \]
\[ Q_s \leftarrow [Q_s, q_i] \]

Figure 2-5: The Newton-Raphson method.
2. At each iteration, the accelerations $\ddot{q}_{n+1}$ are calculated using estimated displacements $\dot{q}^*_n$ and velocities $\ddot{q}^*_n$, which in turn are based on the responses at $t_n$.

3. The acceleration computation step makes use of a so-called stepping matrix $S = M + h\gamma C + h^2 \beta K$. Since the system is linear, this matrix needs to be evaluated only once. However, the system is solved for each time step. This requires $S$ to be factorised.

Hence the linear Newmark time integration method requires one linear system to be solved for each time step. Further information about the Newmark method can be found in [6]

2.3 Nonlinear Dynamic Systems

The implicit Newmark method of the previous section needs some modifications to make it applicable to nonlinear systems. The main difference is that the stepping matrix $S$ is no longer constant throughout the integration, because the stiffness $K(q_{n+1})$ is now a function of the response $q_{n+1}$ (and possibly the damping $C(\dot{q}_{n+1})$ of $\dot{q}_{n+1}$) (Figure 2-7). Because of this nonlinearity, the equilibrium cannot be satisfied in one step. Like the nonlinear static case, a Newton-Raphson iteration is needed to satisfy equilibrium by minimising a residual $r$ for every time step. This means that for each of the $N$ time steps $m$ Newton-Raphson iterations are carried out, yielding a total of $N \times m$ linear equations that must be solved.

2.4 Nonlinear Equations & Computation Time

Finding solutions to the equations that were derived with the Finite Element Method involves solving systems of algebraic equations. For static linear problems, the solution is found by solving one of such algebraic equations. If the static problem is nonlinear, a series of in total $N \times m$ linear problems must be solved to find equilibrium. Here $m$ represents the number of iterations per load step, while $N$ represents the number of load steps. Dynamic linear problems are solved using the Newmark method. For each of the $N$ time steps, an algebraic equation must be solved. Nonlinear dynamic problems solved with the nonlinear variant of the Newmark method involve a nonlinear problem that must be solved for each of the $N$ time steps. Just like in a Newton step, the nonlinear problem is solved through $m$ successive linear problems. Thus a total of $N \times m$ linear problems need to be solved. For each of the linear problems the stiffness matrix must be evaluated, because it is a function of the displacements $1$.

Depending on the numbers $(N, m)$, finding the solution of a nonlinear problem requires multiples of the computation time of a linear problem. In practice, the inevitable increase in computational workload can make the difference between a computation time in the order of minutes for linear problems and computation times in the order of week(s) or even months for nonlinear problems of equal dimension.

Often this increase in computation time cannot be easily mitigated. Obvious measures such as picking a larger time step $\Delta t$ for dynamic problems, or a smaller number $k \ll N$ of load

1In practice, this statement is a bit more nuanced, because some solvers use the same stiffness matrix in more then one of the $m$ iterations. However in most cases this will slow down the convergence, which means that more iterations are needed ($m$ becomes higher)
Initialise:
$M, C, K, g, q_0, \dot{q}_0$

Initial $\ddot{q}_0$:
$\ddot{q}_0 = M^{-1}(g_0 - C\dot{q}_0 - Kq_0)$

Time increment:
$t_{n+1} = t_n + h$

Prediction:
$\ddot{q}^*_n = \dot{q}_n + (1 - \gamma)h\ddot{q}_n$
$q^*_n = q_n + h\dot{q}_n + (0.5 - \beta)h^2\ddot{q}_n$

Acceleration Computing:
$S = M + h\gamma C + h^2\beta K$
solve for $\ddot{q}_{n+1}$:
$S\ddot{q}_{n+1} = g_{n+1} - C\ddot{q}^*_n - Kq^*_n$

Correction:
$\ddot{q}_{n+1} = \dot{q}^*_n + h\gamma\ddot{q}_{n+1}$
$q_{n+1} = q^*_n + h^2\beta\ddot{q}_{n+1}$

Store Results:
$Q_s \leftarrow [q_s, q]$

Figure 2-6: The Newmark method for linear dynamics.
Initialise:  
\[ M, f, g, q_0, \dot{q}_0 \]

Initial \( \dot{q}_0 \):  
\[ \dot{q}_0 = M^{-1}(g_0 - f(\dot{q}_0, q_0)) \]

Time increment:  
\[ t_{n+1} = t_n + h \]

Prediction:  
\[
\begin{align*}
q_{n+1} &= \ddot{q}_n + (1 - \gamma)h\dddot{q}_n \\
q_{n+1} &= q_n + h\dddot{q}_n + (0.5 - \beta)h^2\ddot{q}_n \\
\dot{q}_{n+1} &= 0
\end{align*}
\]

Residual + Error Evaluation:  
\[ r_{n+1} = M\ddot{q}_{n+1} + f_{n+1} - g_{n+1} \]
\[ \epsilon = \frac{|r|}{|r_{n+1}|} \]

Check Convergence:  
\[ \epsilon \overset{?}{\leq} \text{tol} \]

NR-sensitivity:  
\[ S = \frac{1}{(\beta h^2)M + \frac{2}{\beta h}C + K_{n+1}} \]

Solve:  
\[ S\Delta q_{n+1} = -r_{n+1} \]

Correction:  
\[
\begin{align*}
q_{n+1} &= q_{n+1} + \Delta q \\
\dot{q}_{n+1} &= \dot{q}_{n+1} + \frac{\gamma}{\beta h}\Delta q \\
\ddot{q}_{n+1} &= \ddot{q}_{n+1} + \frac{1}{\beta h^2}\Delta q
\end{align*}
\]

Store Results:  
\[ F_s \leftarrow [F_s \ f_{n+1}] \]
\[ q_s \leftarrow [q_s \ q_{n+1}] \]

Figure 2-7: The Newmark method for nonlinear dynamics.
increments $\lambda_i$ for static problems have undesired drawbacks. For example, increasing the time step means that the time resolution becomes more coarse. This limits the frequency bandwidth of the responses $q(t)$ that are solved for. Reduction of the number of load steps in the Newton-Raphson iterations of a static problem brings the risk of instability to occur. From this it can thus be concluded that more advanced methods are needed to reduce the computational workload. The next chapter will give an overview of a couple of such methods.
Solving finite element problems can require a high computation effort. For structural mechanics problems this is especially the case of nonlinear statics and dynamics. Therefore, much research is dedicated to finding robust, trustworthy techniques to reduce the computational effort of solving the systems of equations like (2-23) and (2-36). This research takes place at the intersection of the fields of (numerical) mathematics and mechanics. In the past both mathematicians and engineers have contributed to this field of research. This chapter provides an overview of a small portion of this work: Projection-based reduction methods. This is done to establish a context from within the Discrete Empirical Interpolation Method (DEIM) will be introduced in chapter Chapter 4.

The first section will introduce the concept of reduction through projection. It will be shown that projection on its own may help reduce the computational effort for linear problems. On the contrary, this method alone will not suffice for nonlinear systems, as will be shown in Section 3-2. This section is followed up by a section that briefly describes two improved reduction methods that supplement the projection method of Section 3-1.

### 3-1 Reduction by Projection

Given a system of linear equations of similar type as (2-25), with \( n \) Degrees Of Freedom:

\[
M\ddot{q}(t) + Kq(t) = g(t), \quad K, M \in \mathbb{R}^{n \times n}, \quad q(t), g(t) \in \mathbb{R}^{n \times 1}.
\]  

(3-1)

The solutions \( q \in \mathbb{R}^{n \times 1} \) of this system\(^1\) span a space \( Q \in \mathbb{R}^n \). The complete space is spanned by a set of \( n \) orthonormal basis vectors \( v_i \):

\[
Q = \text{span}(v_1, \ldots, v_n).
\]  

(3-2)

\(^1\)From here on, the explicit dependence on \( t \) is dropped: \( q(t) = q \) for reading and writing\!' convenience.
However, the solutions \( \mathbf{q} \) are often attracted to a subspace \( \mathcal{Q} \subset \mathcal{Q} \). For example, consider the case of three Degrees Of Freedom. If \( \mathbf{q} \in \mathbb{R}^{3 \times 1} \) is attracted to a subspace in \( \mathbb{R}^2 \) (which is a plane!), then the solutions \( \mathbf{q} \) can be expressed in terms of the two basis vectors \( \mathbf{v}_1, \mathbf{v}_2 \) that span the subspace (read: plane) that lies in the \( \mathbb{R}^3 \) space. This idea is generalisable for spaces of higher dimension.

The idea that \( \mathbf{q} \) is attracted to a certain subspace \( \mathcal{Q} \) implies that it may be possible to express all the solutions \( \mathbf{q} \) with reasonable accuracy while using only the \( k \) basis vectors that span \( \mathcal{Q} \). If these basis vectors are collected in a matrix:

\[
\mathbf{V}_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k] \in \mathbb{R}^{n \times k}, \tag{3-3}
\]

then the solution can be approximated by expressing it in terms of these basis vectors by stating:

\[
\mathbf{q} \approx \mathbf{V}_k \bar{\mathbf{q}}, \quad \mathbf{V}_k \in \mathbb{R}^{n \times k}, \quad \bar{\mathbf{q}} \in \mathbb{R}^{k \times 1}. \tag{3-4}
\]

Here \( \bar{\mathbf{q}} \) is a set of reduced generalised Degrees Of Freedom. Substituting (3-4) into (3-1), one obtains:

\[
\mathbf{M} \ddot{\mathbf{V}}_k \bar{\mathbf{q}}(t) + \mathbf{K} \mathbf{V}_k \bar{\mathbf{q}}(t) = \mathbf{g}(t) + \mathbf{r}, \quad \mathbf{r}, \mathbf{g}(t) \in \mathbb{R}^{n \times 1}. \tag{3-5}
\]

Notice that because (3-4) is an approximation, a residual \( \mathbf{r} \) remains after substitution. This residual represents that part of the equation that lies outside the subspace \( \mathcal{Q} \). Thus the inner product of \( \mathbf{r} \) with \textit{any} of the basis vectors that span \( \mathcal{Q} \) is zero: \( \mathbf{V}_k^T \mathbf{r} = \mathbf{0} \). In terms of mechanics, one would state that the residual produces zero work on the basis of the solutions \( \mathbf{q} \). This property can be used to simplify (3-5). If the whole equation is left-multiplied with \( \mathbf{V}_k^T \), one obtains:

\[
\mathbf{\tilde{M}} \ddot{\mathbf{\bar{q}}}(t) + \mathbf{\tilde{K}} \mathbf{\bar{q}}(t) = \mathbf{V}_k^T \mathbf{g}(t) + \mathbf{V}_k^T \mathbf{r}, \quad \mathbf{\bar{q}}, \mathbf{g}(t) \in \mathbb{R}^{n \times 1}, \tag{3-6}
\]

with \( \mathbf{\tilde{M}} = \mathbf{V}_k^T \mathbf{M} \mathbf{V}_k \in \mathbb{R}^{k \times k} \), \( \mathbf{\tilde{K}} = \mathbf{V}_k^T \mathbf{K} \mathbf{V}_k \in \mathbb{R}^{k \times k} \). \tag{3-7}

Note that the system is of dimension \( k \ll n \). The reduced stiffness and mass matrices are \( k \times k \)-matrices. Hence the matrix factorisations done in for example the Newmark method (see Section 2-3-3) require less computational power, reducing the total computation time. Projecting the equations on a subspace \( \mathcal{Q} \) thus saves computation time per iteration.

The equations (3-6) are a so-called reduced order approximation of (3-1). It must be stressed that this system is an \textit{approximation}, not a one-to-one replacement for (3-1). Although the full order solution \( \mathbf{q} \) can be obtained by substituting reduced solutions \( \bar{\mathbf{q}} \) (solved from (3-6)) into (3-4), this still remains an approximation of the full dimensional solution. The quality of the approximation strongly depends on which basis vectors are used in \( \mathbf{V}_k \). Using the right algorithm to find these basis vectors is of key importance here.
3-1-1 Interpretation

Going back to the start of chapter 2, the following steps have been taken:

- A continuum mechanics problem has been discretised using the Finite Element Method (FEM).
- This yielded a system of $n$ equations. This system of equations is an approximation of the continuum mechanics problem.
- This $n$-dimensional approximation has been discretised once more using the reduction method described above. What resulted is a $k$-dimensional approximation.

The third step suggests that one could do the audacious statement:

*A $k$-dimensional finite element model of the original $n$-dimensional finite element method problem has been made!*

Although this claim is not exactly true, the two discretisations are closely related. Both the real finite element discretisation and the model reduction method originate from the *Method of weighted residuals* often named after the Russian mathematician Boris Galerkin\(^2\). The details of the Galerkin method are too complex to elaborate in this report. Interesting and reasonably comprehensible information on the Galerkin method can be found in [5, ch. 5].

### 3-2 The Bottleneck of Reduced Nonlinear Systems

The projection method of the previous section has been successfully applied to a linear system of equations (3-6). The assumption that solutions are attracted to subspaces of lower dimension $k \ll n$ allowed the system (3-1) to be approximated by (3-6). In principle, the same assumption can be made for nonlinear systems like (2-35) and (2-36). Just like linear systems, these equations can have solutions that are attracted to subspaces \( \tilde{Q} \in \mathbb{R}^n \) of smaller dimension $k$. For a suitable set of orthonormal basis vectors \( V_k = [v_1, \ldots, v_k] \), the reduced system yields:

\[
V_k^T \tilde{M} V_k \ddot{\tilde{q}}(t) + V_k^T \tilde{K} V_k \tilde{q} + V_k^T f(V_k \tilde{q}, t) = V_k^T g(t).
\]  

Again, the reduced mass matrix can be expressed as \( \tilde{M} = V_k^T M V_k \). The same thing holds for the linear stiffness matrix: \( \tilde{K} = V_k^T K V_k \). Multiplication of \( \tilde{K} \) with the reduced generalised displacements \( \tilde{q} \) involves $k^2$ multiplications, which saves computation power when compared to the $n^2$ multiplications for the unreduced system. The same thing holds for the multiplication of \( \tilde{M} \) with the reduced accelerations \( \ddot{\tilde{q}} \). However, the internal force cannot be simplified using the same concept:

Substitution of (3-4) into $f$,
\[ f(q) = f(V_k \tilde{q}), \quad (3-9) \]

in (3-8) still requires \( n^2 \) function evaluations to yield a \( f \in \mathbb{R}^{n \times 1} \) that is to be left-multiplied with \( V_k^T \).

The matrix \( V_k \) can not be written outside of the term \( f \), because of the nonlinear dependency of \( f \) on \( q = V_k \tilde{q} \). In general, the same thing holds for the Jacobian (stiffness matrix) of (3-8), which has the form:

\[ \frac{\partial}{\partial \tilde{q}} \left( \tilde{K} \tilde{q} + V_k^T f(V_k \tilde{q}) \right) = \tilde{K} + V_k^T \frac{\partial f}{\partial \tilde{q}}(V_k \tilde{q}). \quad (3-10) \]

Again, the nonlinearity must be evaluated for all \( n \) components, after which it can be multiplied with \( V_k^T \).

Looking back at the nonlinear solving methods in Section 2-3, it becomes obvious that the drawback of the projection method has large consequences for the computational effort of a Newmark integration or Newton-Raphson iteration. Having to deal with terms that require a computational effort that is proportional to \( n \), even if the solutions to the system of equations are attracted to a subspace of lower dimension, creates a bottleneck in the solving procedure.

In the case of a linear system, the internal force could be approximated using only \( k \) function evaluations. Ideally, one wants to do the same thing for the nonlinear term of the internal force. The following section will introduce methods that are aimed at approximating the nonlinear term \( f(q) \) using only \( k \) evaluations, thus making the solving procedures of Section 2-3-2 and Section 2-3-4 independent of the system dimension \( n \).

### 3-3 Reduction Methods for Nonlinear Functions

#### 3-3-1 Missing Point Estimation

The method of Missing Point Estimation is described in [2, ch. 4]. The background of this method is found in a heat conduction problem in a furnace. The temperature inside this rectangularly shaped furnace must be controlled, but the temperature can be measured only at a small number of locations. A spatially discretised model of the rectangular heat conducting domain is made. If only a small number of outputs (measured temperatures) are obtainable from this model, then one could say that the outputs are 'masked' by a mask that limits the number of spatial output locations. Based on these few measurements, the temperature as a function of space is estimated for the entire furnace. The estimated temperature field is then used as input to a controller that controls the furnace’s temperature.

Analogue to the temperature field, a nonlinear internal force field \( f = f(q, t) \in \mathbb{R}^{n \times 1} \), \( q \in \mathbb{R}^{n \times 1} \) can be considered. For simplicity the dependence on the spatial coordinates \( q \) is not shown from here on. If it is assumed that only a small number \( k \ll n \) of components of \( f \) are calculated, then this force field is masked the same way as the temperature field in [2]. One way to express this masking is by defining a boolean matrix \( P \in \mathbb{R}^{k \times n} \) that selects \( k \) components in \( f \):

R.J. Dedden  
Master of Science Thesis
\[ \tilde{f} = Pf \in \mathbb{R}^{k \times 1}. \]  

(3-11)

The internal force field can be represented by a decomposition in basis vectors in \( \mathbb{R}^n \):

\[ f(t) = \sum_i^n a_i(t) \phi_i = \Phi a(t) \in \mathbb{R}^{n \times 1}. \]  

(3-12)

Defining the reduced internal force vector \( \tilde{f}(t) \), the masked internal force can be represented as:

\[ \tilde{f}(t) = \sum_i^k \tilde{a}_i(t) \tilde{\phi}_i = \tilde{\Phi} \tilde{a}(t) \in \mathbb{R}^{k \times 1}, \quad \tilde{\Phi} = P^T \Phi \in \mathbb{R}^{k \times k}. \]  

(3-13)

The matrices \( \Phi \) and \( \tilde{\Phi} \) are found by a so-called Proper Orthogonal Decomposition (POD) \[15\]. For the special case of Hilbert spaces \(^3\), the POD is essentially the same as a singular value decomposition \([3]\) (Appendix A-3).

Equation (3-13) can be viewed as an approximation to the original function (3-12). For the reduced (masked) representation to be an accurate approximation of the original function, the following error needs to be minimised:

\[ E(t) = \| \tilde{f}(t) - f(t) \|. \]  

(3-14)

Given a number of spatial locations \( k \), two methods are treated in \([2]\) that can be used to compare the importance of an arbitrary selected point with respect to the remaining \( k - 1 \) points, using (3-14). The number of measurement locations in the furnace is limited, for example due to constraints on the available space to place sensors. Using the two criteria, the most optimal sensor locations can be selected.

### 3-3-2 Interpolation of Function Snapshots

Missing Point Estimation made use of a Proper Orthogonal Decomposition of the full order snapshots of the nonlinear function \( f = f(q, t) \) (3-12). Calculating a POD of the original snapshots \( F \in \mathbb{R}^{n \times s} \) is considered inefficient if the indices of the \( k \) unmasked components of \( f \) are known on beforehand. This section presents an alternative to MPE, named Interpolation of Function Snapshots (IFS).

Assume that the matrix \( P \) in (3-11) is known on beforehand, thus selecting a known set of \( k \) indices. Using the assumption that the solutions \( f \) are attracted to a subspace spanned by the columns of an orthonormal matrix \( V_k \in \mathbb{R}^{n \times k} \) (see Section 3-1, where the same concept is introduced with matrix \( V_k \)), \( f \) can be approximated by:

\[ f \approx V_k \tilde{f}, \]  

(3-15)

which is equal to (3-12) when the summation in (3-12) is truncated: \( V_k = [\phi_1, \ldots, \phi_k] \), \( \tilde{f} = [a_1, \ldots, a_k]^T \). Hence the full order internal force is approximated using the reduced internal

---

\(^3\)ordinary Euclidean space \( \mathbb{R}^3 \) generalised for spaces \( \mathbb{R}^d \), \( d > 3 \) \([3, \text{ p. 4}]\)

---

Master of Science Thesis  
R.J. Dedden
force \( \hat{\mathbf{f}} \) and an unknown matrix \( \mathbf{V}_k \). Notice that the definition of \( \hat{\mathbf{f}} \) is different from (3-13), since no basis vectors \( \phi_i \) are involved! Matrix \( \mathbf{V}_k \) is again derived from snapshots, but unlike MPE not from a POD of snapshots of the full order solutions. IFS makes use of \( s \) already masked snapshots:

\[
\hat{\mathbf{F}} = \mathbf{P} \mathbf{F} \in \mathbb{R}^{k \times s}.
\] (3-16)

These masked snapshots \( \hat{\mathbf{F}} \) are used in combination with \( \mathbf{V}_k \) to calculate \( \hat{\mathbf{F}} \), which approximates the full order snapshots \( \mathbf{F} \):

\[
\mathbf{F} \approx \hat{\mathbf{F}} = \mathbf{V}_k \hat{\mathbf{F}}.
\] (3-17)

The most optimal \( \mathbf{V}_k \) (given a masked snapshot collection \( \hat{\mathbf{F}} = \mathbf{P} \mathbf{F} \)) is the one that minimises the residual:

\[
\mathbf{r} = \hat{\mathbf{F}} - \mathbf{F}.
\] (3-18)

Since the system (3-17) is overdetermined \( (k \ll n) \), the minimiser \( \mathbf{V}_k \) for (3-18) can only be solved for using a least squares approximation. This implies that \( \mathbf{V}_k \) has to satisfy:

\[
\hat{\mathbf{F}}^T \mathbf{V}_k^T \mathbf{V}_k^T = \hat{\mathbf{F}}^T \hat{\mathbf{F}} \Rightarrow
\] (3-20)

\[
\mathbf{V}_k^T = \left( \hat{\mathbf{F}}^T \hat{\mathbf{F}} \right)^{-1} \hat{\mathbf{F}}^T \Rightarrow
\] (3-21)

\[
\mathbf{V}_k = \hat{\mathbf{F}} \left( \hat{\mathbf{F}}^T \hat{\mathbf{F}} \right)^{-1} \Rightarrow
\] (3-22)

Hence the projection matrix \( \mathbf{V}_k \) can be found from a routine that does not need a singular value decomposition of the training snapshots \( \mathbf{F} \) to be calculated. Instead, matrix \( \mathbf{V}_k \) is expressed in terms of the training snapshots \( \mathbf{F} \) and the mask \( \mathbf{P} \) (3-22).

### 3-3-3 Drawbacks of MPE and IFS

Without going into all the details of the two reduction methods briefly described in the previous two sections, a couple of conclusions can be drawn:

- MPE uses rather computationally impractical algorithms to find the set of \( k \) indices that guarantee the most accurate approximation of the original full order system. The algorithms described in [2] require that each of the \( n \) possible indices are tested against some accuracy criterion.

- IFS can only be implemented if the selection of \( k \) indices is known beforehand.

R.J. Dedden

Master of Science Thesis
Especially the drawback of IFS is of importance for the application to FEM-problems. It is generally very difficult to decide which Degrees Of Freedom (dofs) are the most important for the overall model behaviour. If that would be easy, then there would be no need for reduction methods in finite element analysis, since the models could then simply be stripped of all the unnecessary dofs.

Instead of having to select components of the internal force vector $\mathbf{f}$ by hand, as is the premise in IFS, one wants to let the reduction algorithm decide which components are selected. Only when that is the case, can the solving process of complex FEM models be fully automated.

The next chapter will introduce the DEIM, which is able to construct the boolean matrix $\mathbf{P}$ on its own, without the need for user input.
Chapter 4

Discrete Empirical Interpolation Method

This chapter introduces the Discrete Empirical Interpolation Method (DEIM). The theory of DEIM involves a number of mathematical concepts, especially topics that belong to the domain of linear algebra. All mathematical steps of DEIM are treated here. This will provide a solid foundation for the application to the finite element method.

4-1 The DEIM Algorithm

Although for this thesis the DEIM-algorithm is studied only in conjunction with the Finite Element Method (FEM), it is very useful to start at the mathematical origin. This bridges the gap between the paper [3] and the implementation on finite elements in nonlinear mechanics.

4-1-1 Nonlinear Dynamic Systems

Many scientific modelling and engineering design problems involve partial differential equations (PDE’s) that have to be solved for time \( t \) and/or spatial variables \( x \). Suppose the problem is written as:

\[
\frac{dy(x,t)}{dt} = \mathcal{L}(y(x,t)) + F(y(x,t)), \tag{4-1}
\]

where the linear (\( \mathcal{L}(y(x,t)) \)) and the nonlinear part (\( F(y(x,t)) \)) of the equation can be written as separate terms. This will be of importance for the derivations in the rest of this chapter. In the ideal case, analytical solutions to (4-1) can be found. A typical solution \( y_{sol}(x,t) \) then satisfies for all points \( x \) in spatial domain \( X \) and \( t \) in temporal domain \( T \). However, only a very limited set of partial differential equations has an analytical (exact) solution. For the problems that are not analytically solvable, numerical methods are applied to approximate solutions to the PDE-problem.
A common approach is to approximate the solution \( \hat{y}_{sol} \approx y_{sol} \) by requiring that the approximation must satisfy the PDE only for a finite set of points in \( X \) and \( T \). Discretising the domains then gives a large amount of differential equations that may or may not be coupled. Discretising the spatial variable \( x \) yields:

\[
\frac{d}{dt} y(t) = Ay(t) + F(y(t)). \tag{4-2}
\]

Here \( y(t) = [y_1(t), \ldots, y_n(t)]^T \in \mathbb{R}^n \) is the vector containing the finite number \( n \) of discrete time-dependent functions \( y_i(t) \). The constant matrix \( A \in \mathbb{R}^{n \times n} \) represents the part of the equation that is linear in \( y(t) \). It is assumed that \( F(y(t)) = [F_1(y_1(t)), \ldots, F_n(y_n(t))]^T \in \mathbb{R}^{n \times 1} \), a scalar valued nonlinear function in \( y(t) \). This means that each of the components \( F_i(y_i(t)) \) depends on one corresponding component \( y_i(t) \) in \( y(t) \) only. This will be referred to as ‘point-to-point dependence’. Furthermore, it is assumed that \( A \) is diagonal and thus \( Ay(t) \) is also a point-to-point function. Problem cases where this simplification cannot be made will be discussed later on.

Just like the original continuous problem (4-1), the discretised set of equations (4-2) is in general impossible to solve analytically. In many cases, the approximate solutions are to be found by iterative solving methods that require information on the gradients of the \( n \) spatial discretised points. These gradients are found by calculating the derivative (Jacobian) of the discretised function with respect to \( y \):

\[
\frac{\partial}{\partial y(t)} (Ay(t) + F(y(t))) = J(y(t)) = A + J_F(y(t)), \tag{4-3}
\]

where in case of point-to-point functions the matrices \( A \) and \( J_F \) are diagonal. Jacobian \( J_F \) then looks like:

\[
J_F(y(t)) = \text{diag} [F'_1(y_1(t)), \ldots, F'_n(y_n(t))] \in \mathbb{R}^{n \times n}. \tag{4-4}
\]

**Galerkin Projection**

The concept of Galerkin projection has already been established in Chapter 2. Here, the projection matrix \( V_k \in \mathbb{R}^{n \times k} \) is introduced to project the discretised spatial domain onto a subspace \( \mathcal{X} \subset \mathbb{R}^k, k < n \) of lower dimension. Substituting \( y(t) = V_k \tilde{y}(t) \) and projecting (4-2) onto \( V_k \) results in a reduced system:

\[
\frac{d}{dt} \tilde{y}(t) = V_k^T AV_k \tilde{y}(t) + V_k^T F(V_k \tilde{y}(t)), \tag{4-5}
\]

and corresponding Jacobian:

\[
\tilde{J}(y(t)) = \tilde{A} + V_k^T J_F(V_k \tilde{y}(t))V_k \in \mathbb{R}^{k \times k}. \tag{4-6}
\]

\[1\] In fact, the FEM happens to be a perfect example of a case where the point-to-point assumption does not hold!

R.J. Dedden Master of Science Thesis
Note that $\mathbf{A} \in \mathbb{R}^{k \times k}$.

The set of spatial orthonormal basis vectors $\mathbf{V}_k$ can be obtained using various techniques, for example from a Proper Orthogonal Decomposition (POD) (see Appendix A-3).

In Chapter 2 it was already pointed out that the use of Galerkin projection does not provide a solution to the problem of having to compute the nonlinear force and its Jacobian for all $n$ degrees of freedom from the original problem. Here the same applies for (4-2) and (4-4):

$$F(\mathbf{V}_k \tilde{y}(t)), \quad \tilde{J}_F(\mathbf{V}_k \tilde{y}(t)), \quad \text{with } \mathbf{V}_k \tilde{y} = \mathbf{y} \in \mathbb{R}^n. \quad (4-7)$$

Both $F(\mathbf{V}_k \tilde{y}(t))$ and $\tilde{J}_F(\mathbf{V}_k \tilde{y}(t))$ are functions of $n$ spatial points in domain $X$. The DEIM algorithm aims to overcome these two bottlenecks, as will be explained in the following section.

4-1-2 DEIM

In this section, it will be shown that the two bottlenecks (4-7) can be overcome by the use of DEIM. DEIM is applicable to nonlinear functions as well as to its partial derivatives. Hence this method can be used for both $F(\mathbf{V}_k \tilde{y}(t))$ and $\tilde{J}_F(\mathbf{V}_k \tilde{y}(t))$. The main idea behind this DEIM-approach is to compute the nonlinear term only at $m$ carefully selected locations, with $m \ll n$, and interpolate elsewhere.

Interpolation on a basis

The nonlinear term of the differential equation (4-2) spans a subspace $\mathcal{F} \subset \mathbb{R}^n$. Let an orthonormal basis of this subspace be represented by:

$$\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_n]. \quad (4-8)$$

The column vector $\mathbf{F}(\mathbf{V}_k \tilde{y}(t))$ for arbitrary $\tilde{y}(t)$ can be represented by a weighted sum of these basis vectors:

$$\mathbf{F}(\mathbf{V}_k \tilde{y}(t)) = \mathbf{U} \mathbf{c}(t), \quad \mathbf{U} \in \mathbb{R}^{n \times n}, \quad \mathbf{c}(t) \in \mathbb{R}^{n \times 1}. \quad (4-9)$$

This is essentially the same concept as the concept of mode superposition known from engineering mechanics. In mode superposition, the dynamic response of a system is described using a weighted sum of modes or mode shapes (See for example [11, chap. 8] for more information about mode superposition).

As is the case with mode superposition, one may decide to reduce the number of basis vectors (or modes/mode shapes) that is taken into account, in order to approximate the response with a reduced representation. In most algorithms the basis vectors $\mathbf{u}_i$ are ordered in such a way that the first basis vector $\mathbf{u}_1$ denotes the most meaningful one, and the others follow by order of decreasing importance. The reduced set of basis vectors would thus be: $\mathbf{U}_m = [\mathbf{u}_1, \ldots, \mathbf{u}_m]$. Suppose that the same assumption holds for the function $\mathbf{F}(\mathbf{V}_k \tilde{y}(t))$:

$$\mathbf{F}(\mathbf{V}_k \tilde{y}(t)) = \mathbf{f}(t) \approx \mathbf{U}_m \mathbf{c}(t), \quad \mathbf{U}_m = [\mathbf{u}_1, \ldots, \mathbf{u}_m] \in \mathbb{R}^{n \times m} \quad \mathbf{c}(t) = (c_1, \ldots, c_m)^T \in \mathbb{R}^{m \times 1}. \quad (4-10)$$
Here we introduced the shorthand \( f(t) = F(V_k \tilde{y}(t)) \) which will be used from now on to simplify notation.

The system (4-10) needs to be solved for the unknown \( c_i \)'s, but since it is an overdetermined system, \( n - m \) rows in \( U_m \) must be purged out of the equation. This is done by introducing a boolean matrix \( P \) that selects the \( m \) rows that are necessary to make the system invertible:

\[
P = [e_{\phi_1}, \ldots, e_{\phi_m}] \in \mathbb{R}^{n \times m}.
\]  

Using this yet to be determined matrix (4-11), the basis interpolation of (4-10) can be made invertible and thus solvable for \( c(t) \):

\[
P^T f(t) = (P^T U_m) c(t) \Rightarrow c(t) = (P^T U_m)^{-1} P^T f(t).
\]  

With the expression for \( c(t) \) obtained, the approximation of \( F(V_k \tilde{y}(t)) \) can be expressed as a function of \( U_m, P \) and \( f(t) \) itself:

\[
f(t) = U_m (P^T U_m)^{-1} P^T f(t),
\]  

where \( D \) is referred to as the DEIM-matrix. The exact shape of \( D \) is still unknown, but due to the selection by \( P \), only \( m \) components of the right side \( f(t) \) are needed. The remaining \( n - m \) components will be interpolated (see Section 4-1-4). Our primary interest is how \( P \) can be found. This matrix results from the DEIM-algorithm, which will be explained first.

**DEIM: Algorithm for Interpolation Indices**

As a starting point, the overdetermined equation (4-10) is considered again. Now the idea behind DEIM is to make an optimal selection of \( m \) rows in \( U_m \). Here the word optimal should be thought of as in:

*The \( m \) rows of \( U_m \) must be selected such that the original overdetermined system (4-10) is approximated by the invertible system (4-12) with an error as small as possible.*

The selection procedure is performed using the boolean matrix \( P \), of which the columns are to be determined by DEIM, while making use of the orthonormal basis vectors \( U_m = \{u_1, \ldots, u_m\} \). DEIM will select for each column in \( P \) an index that will be unity. A couple of things should be noted:

- This selection procedure is done while taking care of the fact that no index that already has been selected, will be selected again. This is done by means of an inductive iteration that involves the indices \( \phi_{i-1}, \phi_{i-2}, \ldots, \phi_1 \) selected in the previous iterations \( i = 1, i - 2, \ldots, 1 \). Hence all selected indices are unique!

- Moreover, the selection is performed in such a way that the growth of an error bound is limited. This relates to the requirement that the \( m \) rows in (4-10) to be selected are optimally selected.
Algorithm 1 DEIM

Input: \{u_i\}_{i=1}^m \subset \mathbb{R}^n \text{ linearly independent}
Output: \phi = [\phi_1, \ldots, \phi_m]^T \in \mathbb{N}^m

1: \[\rho \phi_1 = \max \{|u_1|\}\]
2: U = [u_1], P = [e_{\phi_1}], \phi = [\phi_1]
3: for i = 2 to m do
4: Solve \((P^T U)c = P^T u_i\) for c
5: \(r = u_i - Uc\)
6: \[\rho \phi_i = \max \{|r|\}\]
7: \(U \leftarrow [U \ u_i], \ P \leftarrow [P \ e_{\phi_i}], \ \phi \leftarrow \left[\begin{array}{c} \phi \\ \phi_i \end{array}\right]\)
8: end for

- The fact that the basis \(U_m\) originates from a POD ensures that the DEIM algorithm will always converge and that the error growth will be limited.

The inputs to algorithm 1 are the \(m\) basis vectors of the basis \(U\) that were found by POD. The output is an array \(\phi\) that contains the indices for the unity components in boolean matrix \(P\). Matrix \(P\) can thus be constructed from \(\phi\).

Lines 1-2 are meant to initialise the algorithm. In line 1, the value \(\rho\) and the index \(\phi_1\) of the element in \(u_1\) which has the largest absolute value is determined. Note that the \(\max\) function used here adopts the same syntax as used by Matlab. Line 2 uses this index to initialise \(U = u_1 \in \mathbb{R}^{n \times 1}\) and \(P = e_{\phi_1} \in \mathbb{R}^{n \times 1}\). The identified index is stored in \(\phi \in \mathbb{R}\).

From line 3 on, the iterative procedure is started. On line 4, the interpolation constants in \(c\) are solved for. For \(i = 2\), this is a scalar value, but in further iterations it will be an \((i-1) \times 1\)-array. The equation on line 4 can be viewed as an interpolation, in which the columns in matrix \((P^T U) \in \mathbb{R}^{(i-1) \times (i-1)}\) are multiplied with the vector \(c \in \mathbb{R}^{(i-1) \times 1}\) containing the weights \(\{c_2, \ldots, c_{i-1}\}\). This interpolation equals \(P^T u_i\), which is a \((i-1) \times 1\) vector of components in \(u_i\) that have the indices selected by \(P\). The existence of the interpolation is guaranteed by the fact that \((P^T U)\) is always full rank, see Appendix A-1.

Although the interpolation is exact for the rows corresponding to indices \(\phi\), the interpolation never gives exact results for all rows in the equation on line 4. If \(Uc\) is subtracted from \(u_i\) (line 5), the residual \(r\) will always be nonzero. This can be easily understood since the vectors \(\{u_1, \ldots, u_{i-1}\}\) in \(U\) are independent of the newly selected or current vector \(u_i\). This prevents the algorithm from stalling on line 6.

As was pointed out in the bullets above, the DEIM-algorithm should select \(m\) rows in (4-10) in such a way that the error in the interpolation is bounded. This explains why the newly selected index \(\phi_i\) is picked by finding the largest component in \(|r|\). This index is included in the boolean matrix \(P\) by updating it with a new unit vector \(e_{\phi_i}\) (line 7). Doing so makes sure that in the next iteration loop \(i + 1\) the interpolation on line 4 will produce an exact fit for this index. Hence the algorithm selects indices according to the largest error found in iteration \(i\) and then the next iteration \(i + 1\) will make the interpolation exact at this index. This iterative error selection also has the property that \(P^T U\) is always full rank. Matrix \(P^T U\) is thus invertible and the existence of the interpolation on line 4 is guaranteed.
To conclude the explanation, the remaining steps of line 7 are pointed out. The first operation is the updating of $U$ with the new basis vector $u_i$, which will be needed for the next iteration. The last operation on line 7 appends the newly found index $\phi_i$ to the array of indices. When the for-loop ends, this array will be returned as an output.

### 4-1-3 A graphical interpretation of DEIM

The DEIM-algorithm described in the previous section can be visualised with a series of plotted basis vectors $u_i$, the approximation $U_c$ of $u_i$ and the residual $r$. This will greatly improve the understanding, especially for those who are more visually oriented. To make the visualisation even more intuitive, an application on an ordinary mechanics problem is used for illustration.

#### A clamped-clamped beam with a step load

Suppose that a beam is clamped between two vertical walls, and a vertical load is exerted at the middle. The beam is discretised in 10 elements of equal length (see Figure 4-1). All segments have the same mass, for each segment equally divided among two nodes. The discretisation is done using the FEM. However, the details of this technique are not of importance for the discussion here. Here the goal is to visualise the DEIM-algorithm, not to do an actual implementation of DEIM on FEM. In Section 5-3 the finite element context of the load case is detailed.

A second simplification is that only shear (vertical) forces are considered. Since both ends are clamped in the walls, a total of 9 Degrees Of Freedom remain. It is assumed that these shear forces $f(q) \in \mathbb{R}^{9 \times 1}$ are a nonlinear function of the vertical deflections $q \in \mathbb{R}^{9 \times 1}$. More detail about the nature of the nonlinearity can be found in [4, ch.7] and in the Matlab implementation in Appendix B-3-2, but can be disregarded for the present discussion.

The external force $g(t)$ is applied to the beam at $t = 0$ (step load). The beam will start oscillating, generating time-varying shear forces at each of the nodes. Suppose that for each time increment a snapshot is taken of the shear forces. Hence, if there are $N$ time steps, a $9 \times N$ snapshot matrix $F_s$ can be made. The columns of this snapshot matrix span a subspace $\mathcal{F} \subset \mathbb{R}^9$. A basis can be calculated for $\mathcal{F}$ by means of a Singular Value Decomposition (SVD) (see Chapter 2):

$$F_s = U \Sigma V, \quad U \in \mathbb{R}^{9 \times 9}, \quad \Sigma \in \mathbb{R}^{9 \times N}, \quad V \in \mathbb{R}^{N \times N}. \quad (4-14)$$

---

*The author counts himself to this group :)*
The left-hand matrix $\mathbf{U}$ contains the basis vectors $\mathbf{u}_1, \ldots, \mathbf{u}_9$ that belong to the subspace $\mathcal{F}$. Matrix $\mathbf{\Sigma}$ contains a diagonal with the singular values, with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_9$, indicating the order of decreasing dominance of basis vectors $\mathbf{u}_i$. Matrix $\mathbf{V}$ contains orthonormal basis vectors $\mathbf{v}_1, \ldots, \mathbf{v}_9, \ldots, \mathbf{v}_N$ that span a subspace $\mathcal{F} \subset \mathbb{R}^N$. The basis vectors $\mathbf{v}_1, \ldots, \mathbf{v}_9$ can be interpreted as the time history of the basis vectors $\mathbf{u}_1, \ldots, \mathbf{u}_9$. However, $\mathbf{V}$ is not of importance here.

The basis vectors $\mathbf{u}_i$, $i = 1, \ldots, 9$ can be viewed as force mode shapes or *force modes* that give a normalised distribution of the internal forces at the 9 nodes. This concept is similar to the displacement mode shapes that result from linear vibration analysis, with the difference that the force modes are *empirically derived from the snapshots*, whereas linear vibration modes are derived from the linear stiffness matrix $\mathbf{K}$ and mass matrix $\mathbf{M}$. Would the external load have been different, than the snapshot matrix would have been different, which generally would have resulted in a different basis $\mathbf{U}_x$ for a different subspace $\mathcal{F}_x$.

Using the force eigenmodes derived from POD, any possible shear force within subspace $\mathcal{F}$ can be written as a superposition of these eigenmodes:

$$f(t) = \mathbf{U}c(t)$$

This superposition of force modes can be truncated, considering a smaller number of modes $m$ than the total of 9. However, if $m \leq 9$, the matrix $\mathbf{U}_m \in \mathbb{R}^{9 \times m}$ becomes rectangular. This brings back the problem of selecting $m$ rows in $\mathbf{U}_m$ described in the previous section. Using only $m$ rows, the system (4-15) will become solvable for $c(t)$ and thus the interpolation weights $c_1, c_2, \ldots, c_m$ can be determined. The process of selecting the $m$ rows is visualised by Figure 4-2 and Figure 4-3, which show plots of basis vectors $\mathbf{u}_i$ (solid blue), interpolations $\mathbf{U}c$ (dashed black) and their remainder $\mathbf{r} = \mathbf{u}_i - \mathbf{U}c$ (dash-dotted magenta) for 6 iterations of algorithm 1.

Step #1 of DEIM initialises algorithm 1 (lines 1-2). Matrix $\mathbf{U} = \mathbf{u}_1$ is a $n \times 1$-matrix that will grow with one column at each of the upcoming steps. $\mathbf{P} = [0, 0, 0, 0, 0, 0, 1, 0, 0]^T$ is also initialised as a column vector, but will also grow one column at each step. In step #1, its only nonzero component is component number 7. Slot 7 is selected by the Matlab function $\text{max} \{\}$ as the index containing the maximum absolute value in $\mathbf{u}_1$. This index serves to initialise the row-selection process with a sensible first choice: In general, the larger the magnitude of a component, the more important it will be.

In step #2, $\mathbf{P}^T \mathbf{u}_2 = \mathbf{P}^T \mathbf{U}c$ is solved for $c$, such that $\mathbf{U}c$ equals $\mathbf{u}_2$ *in component 7*. This can be verified from the fact that the residual $\mathbf{r}$ is zero in component 7. As can be seen, the interpolation is accurate only for component 7 and component 3. Slot 3 is satisfied automatically because the beam load case is symmetrical, which reduces the number of independent degrees of freedom to 5. The calculated residual indicates at which component the error in the interpolation is the largest: component 5. This point is then selected and a column vector $\mathbf{e}_5 = [0, 0, 0, 0, 1, 0, 0, 0, 0]^T$ is appended to $\mathbf{P}$.

In step #3, the interpolation process uses the newly obtained point to make an approximation of $\mathbf{u}_3$, now using two interpolation points (components 5 and 7). As long as there are unselected components left, the residual will be nonzero. This due to the fact that the basis vectors $\mathbf{U}$ are linearly independent, which guarantees that no linear combination of the vectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{i-1}$ can be made that *exactly* matches the newly selected vector $\mathbf{u}_i$.

Master of Science Thesis

R.J. Dedden
\[ U_{i-1} c \neq u_i, \quad U_{i-1} = [u_1, u_2, \ldots, u_{i-1}] . \]  

(4-16)

Hence a maximum in |r| can always be found, which ensures that the process will never stop before all required \( m \) iteration steps are carried out.

The remaining steps (Figure 4-3) show how the DEIM-process will obtain more and more interpolation points, making the interpolation fit at an increasing number of components. After step \#5, all linearly independent indices have been selected. This explains why the interpolation exactly fits \( u_6 \). The calculations done in step \#6 are obsolete, because 5 iterations already represent the full order system (which has 5 independent dofs!). Nevertheless, the selected component 8 shows that the algorithm is still able to continue in a stable manner.

4-1-4 Reducing degrees of freedom with DEIM

At the end of Section 4-1-2 the equation that results from DEIM (4-13) has been briefly pointed out, without giving any details on the structure of the DEIM-matrix \( D \). This will be the topic of this section. As soon as the structure of \( D \) is known, it will become obvious how DEIM can reduce the computational effort of evaluating the nonlinear function \( f(t) \).

As a first step, it is useful to remember that DEIM produced a matrix \( P \in \mathbb{R}^{n \times m} \) accompanied by the set of \( m \) basis vectors \( U_m \in \mathbb{R}^{n \times m} \), with \( m \ll n \). Hence both \( P \) and \( U \) are rectangular matrices. Matrix \( D \) is defined as:

\[
D = U_m (P^T U_m)^{-1} P^T \in \mathbb{R}^{n \times n}.
\]

(4-17)

This is a \( n \times n \)-matrix of rank \( m \ll n \). This can be shown as follows:

- The product \((P^T U_m)\) is a \( m \times m \) square matrix (\( m \) rows selected from \( U_m \)). It is constructed [3, p. 5] to be a nonsingular, thus invertible matrix.
- The inverted matrix \((P^T U_m)^{-1}\) is premultiplied with \( U_m \), yielding a rank(\( m \)), \( n \times m \)-matrix: \( D_R = U_m (P^T U_m)^{-1} \). Here the subscript \( _R \) is used to point out that \( D_R \) is rectangular.
- Right-multiplying \( D_R \) with \( P^T \) expands the matrix into a \( n \times n \)-matrix of rank \( m \), by placing the linearly independent columns of \( D_R \) in an empty \( n \times n \) matrix \( D \). The order and position of the columns is determined by \( P^T \).

The shape of matrix \( D \) looks like the bar-codes at the back of products one buys at the supermarket: Black stripes denote the columns where nonzero data is stored, white spaces are (empty) zero columns. This will be illustrated using an example with 5 dofs.

An example with 5 degrees of freedom

Suppose that a nonlinear dynamic equation like (4-2) is to be solved for \( N \) time steps. The equation is discretised in 5 spatial coordinates \((n = 5)\). Suppose that a snapshot matrix
Figure 4-2: DEIM steps 1-3. In step #1, the algorithm is initialised by finding the largest element in $|u_1|$. In this case, there are two equally large points due to symmetry. The choice between these two points is made by picking the one with the lowest index. Here this did not happen, probably because of small roundoff errors that made component 7 slightly larger than component 3. In step 2, index 7 is interpolated exact: The lines of $u_2$ and $Uc$ lie on top of each other, and the residual for component 7 is zero. Now DEIM selects index 5, since the residual is the largest there. Again, the interpolation at this component will be exact in the next step, as can be seen from the third picture. Notice that the interpolation at index 7 remains exact. Hence at each iteration the number of exact points will increase by one.
Figure 4-3: DEIM steps 4-6. The process of finding the largest absolute error, followed by an exact interpolation continues. Step-by-step, the interpolation fits the newly selected basis vector $u_i$ for an increasing amount of indices. After 5 steps, all *independent* (because of the symmetry in the beam, only 5 components are truly independent!) indices are used. Hence in step 6, total convergence is achieved.

Legend: $\mathbf{u}_i$, $\mathbf{U}_c$, $\mathbf{r}$. * = newly selected DEIM index, * = previously selected DEIM index.
$F_s \in \mathbb{R}^{5 \times N}, N \gg 5$ is obtained from the nonlinear function $f(t) \in \mathbb{R}^{5 \times 1}$ (see (4-2)). A basis $U_m$ for the subspace $\mathcal{F} \subset \mathbb{R}^5$ is found from POD:

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\ u_{21} & u_{22} & u_{23} & u_{24} & u_{25} \\ u_{31} & u_{32} & u_{33} & u_{34} & u_{35} \\ u_{41} & u_{42} & u_{43} & u_{44} & u_{45} \\ u_{51} & u_{52} & u_{53} & u_{54} & u_{55} \end{bmatrix}. \quad (4-18)$$

Here the choice is made (based on the singular values, see Appendix A-3) to only include the first 3 basis vectors ($m = 3$). This gives a rectangular matrix $U_m$, containing the reduced basis:

$$U_m = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \\ u_{41} & u_{42} & u_{43} \\ u_{51} & u_{52} & u_{53} \end{bmatrix}. \quad (4-19)$$

The columns of this matrix are used as input for DEIM. DEIM will select $m = 3$ rows in $U_m$ and return a set of indices $\phi \in \mathbb{R}^{3 \times 1}$ that identify these rows. Suppose that the boolean matrix $P$ constructed from $\phi$ yields:

$$P^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \quad (4-20)$$

Hence rows 1, 4 and 3 are selected by DEIM. With $P$ known, it is easy to see that the DEIM reduces the number of function evaluations needed from the nonlinear function $f$. If (4-13) is rewritten as:

$$f(t) = (U_m(P^T U_m)^{-1}) P^T f(t), \quad (4-21)$$

thus separating the product of $P^T f(t)$. This product yields:

$$P^T f(t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_1(t) \\ f_2(t) \\ f_3(t) \\ f_4(t) \\ f_5(t) \end{bmatrix} = \begin{bmatrix} f_1(t) \\ f_4(t) \\ f_3(t) \end{bmatrix}. \quad (4-22)$$

From this it is observed that **DEIM will only use a selection of $m$ components in $f(t)$, possibly in a changed order, to calculate the interpolation.** Hence only $m$ components in $f(t)$ need to be calculated. This shows that the vector $f(t)$ on the right-hand side of equation (4-13) can be replaced by a sparse variant of the same vector, for example:
46 Discrete Empirical Interpolation Method

\[ f(t) = \begin{bmatrix} f_1(t) \\ f_2(t) \\ f_3(t) \\ f_4(t) \\ f_5(t) \end{bmatrix} \leftarrow \begin{bmatrix} f_1(t) \\ 0 \\ f_3(t) \\ f_4(t) \\ 0 \end{bmatrix}. \quad (4-23) \]

Hence, the total number of function evaluations that are needed is reduced from \( n = 5 \) to \( m = 3 \). The DEIM-matrix \( D \) will show how the interpolation will produce an approximated \( f(t) \approx \hat{f}(t) \) of which all components are populated with nonzero numbers.

The first step is to determine the product \( P^T U_m \):

\[ P^T U_m = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \\ u_{41} & u_{42} & u_{43} \\ u_{51} & u_{52} & u_{53} \end{bmatrix} = \begin{bmatrix} u_{41} & u_{42} & u_{43} \\ u_{31} & u_{32} & u_{33} \end{bmatrix}, \quad (4-24) \]

The inverse of \( P^T U_m \) is analytically solvable, see Appendix A-2 for details about this specific example. A proof of the general invertability of \( P^T U_m \) is given in Appendix A-1. The columns of this inverse can be given a shorthand notation \( \lambda_i \), with subscript \( i \) referring back to the row index of the original \( P^T U_m \), expressed in \( u_{ij} \). This yields:

\[ (P^T U_m)^{-1} = \Lambda = \begin{bmatrix} \lambda_1 & \lambda_4 & \lambda_3 \end{bmatrix}, \quad \Lambda \in \mathbb{R}^{m \times m}, \quad \lambda_i \in \mathbb{R}^{m \times 1}. \quad (4-25) \]

Matrix \( D_R \) (4-26) can be found from left-multiplication with \( U_m \):

\[ D_R = U_m (P^T U_m)^{-1} = \begin{bmatrix} \lambda_1(U_m) & \lambda_4(U_m) & \lambda_3(U_m) \end{bmatrix}. \quad (4-26) \]

The exact description of the components in this matrix is omitted, because it is not useful here. Important is what happens if the last step, postmultiplication with \( P^T \), is carried out:

\[ D = D_R P^T = U_m (P^T U_m)^{-1} P^T = \begin{bmatrix} 1 & 0 & 0.0 & 0.0 & 0 \\ 0.0 & 0 & 1 & 0.0 & 0 \\ 0.0 & 0.0 & 0 & 1 & 0 \\ \lambda_{51}(U_{m5}) & 0 & \lambda_{53}(U_{m5}) & \lambda_{54}(U_{m5}) & 0 \end{bmatrix}. \quad (4-27) \]

Observe that \( D \) indeed has a shape similar to a bar code, as has been pointed at the begin of this section. Notice that a distinction is made between components that are numerically zero \((0.0)\), and components belonging to the empty columns that correspond to the \( n - m \) variables that are reduced out of the equation \((0)\). Furthermore, it can be easily understood that the nonzero columns in \( D \) are linearly independent. This can be seen from the fact that the unity entries on the diagonal of \( D \) appear in rows that have only one nonzero component. The interpolation yields a fully populated vector \( \hat{f}(t) \) in which the components are a function of the three components \( f_1, f_3 \) and \( f_4 \):
\[ \hat{f}(t) = Df(t) = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
\lambda_{21} & 0 & \lambda_{23} & \lambda_{24} & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
\lambda_{51} & 0 & \lambda_{53} & \lambda_{54} & 0
\end{bmatrix} \begin{bmatrix}
f_1(t) \\
f_2(t) \\
f_3(t) \\
f_4(t) \\
0
\end{bmatrix} = \begin{bmatrix}
f_1 \\
\lambda_{21} f_1 + \lambda_{23} f_3 + \lambda_{24} f_4 \\
\lambda_{51} f_1 + \lambda_{53} f_3 + \lambda_{54} f_4
\end{bmatrix}. \]
4-2 Approximating the Jacobian with DEIM

The previous sections were entirely focussed on the use of DEIM for approximation of the nonlinear function \( f(t) \). No attention has been paid to the fact that DEIM is also applicable to the Jacobian of function \( f(t) \). In this section it will be shown that the Jacobian has some important properties that should be kept in mind when DEIM is applied.

4-2-1 Rank deficiency

Suppose that the Jacobian of \( f(t) \) in the 5 degrees of freedom example from the previous section is defined as:

\[
\begin{bmatrix}
\frac{\partial f_1}{\partial q_1} & 0 & 0 & 0 & 0 \\
0 & \frac{\partial f_2}{\partial q_2} & 0 & 0 & 0 \\
0 & 0 & \frac{\partial f_3}{\partial q_3} & 0 & 0 \\
0 & 0 & 0 & \frac{\partial f_4}{\partial q_4} & 0 \\
0 & 0 & 0 & 0 & \frac{\partial f_5}{\partial q_5}
\end{bmatrix}
\]  

(4-29)

Where one may notice that a diagonal Jacobian (stiffness matrix) is not realistic for finite elements, but for now this simplification is assumed to be valid. Applying DEIM to the Jacobian yields:

\[
\hat{J}_f = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
\lambda_{21} & 0 & \lambda_{23} & \lambda_{24} & 0 \\
0.0 & 0 & 1.0 & 0 & 0 \\
0.0 & 0 & 0.0 & 1.0 & 0 \\
\lambda_{51} & 0 & \lambda_{53} & \lambda_{54} & 0
\end{bmatrix} \begin{bmatrix}
\frac{\partial f_1}{\partial q_1} & 0 & 0 & 0 & 0 \\
0 & \frac{\partial f_2}{\partial q_2} & 0 & 0 & 0 \\
0 & 0 & \frac{\partial f_3}{\partial q_3} & 0 & 0 \\
0 & 0 & 0 & \frac{\partial f_4}{\partial q_4} & 0 \\
0 & 0 & 0 & 0 & \frac{\partial f_5}{\partial q_5}
\end{bmatrix} = \begin{bmatrix}
\frac{\partial f_1}{\partial q_1} & 0 & 0 & 0 & 0 \\
\lambda_{21} \frac{\partial f_2}{\partial q_2} & \lambda_{23} \frac{\partial f_2}{\partial q_3} & \lambda_{24} \frac{\partial f_4}{\partial q_4} & 0 \\
0.0 & 0 & \frac{\partial f_3}{\partial q_3} & 0 & 0 \\
0.0 & 0 & 0.0 & \frac{\partial f_4}{\partial q_4} & 0 \\
\lambda_{51} \frac{\partial f_1}{\partial q_1} & \lambda_{53} \frac{\partial f_2}{\partial q_3} & \lambda_{54} \frac{\partial f_4}{\partial q_4} & 0
\end{bmatrix}
\]

(4-30)

Hence \( \hat{J}_f \) is a rank 3 matrix and the rank deficiency is 5-3=2. In general, the approximated Jacobian will have rank \( m \) after DEIM is applied, since the DEIM-matrix only has \( m \) independent columns. This property reflects the true nature of DEIM: DEIM uses only \( m \) degrees of freedom to approximate a \( n \gg m \)-dof system of equations! The consequence of this property is that the Jacobian is not invertible, making it unsuitable for iterative solvers based on the Newton-Raphson method (Section 2-3-2).

Recovering the rank deficiency

The solution to the rank deficiency in \( \hat{J}_f \) is to remember that DEIM was originally intended to be an extension to the Galerkin projection, in order to overcome the computational bottlenecks (3-9) and (3-10) in discretised nonlinear systems. Projecting \( \hat{J}_f \) onto the reduced basis \( V_k \) results in the \( m \times m \) matrix \( \hat{J}_f \):

\[
V_k^T \hat{J}_f V_k = \tilde{J}_f \in \mathbb{R}^{m \times m}.
\]

(4-31)
If $m = k$, then this reduced Jacobian will generally be full rank again, and the total Jacobian will be (4-4).

In some cases, the Jacobian $\tilde{J}_f$ can still be singular. It is difficult to predict when this happens, since the basis vectors in $U_m$ used for DEIM are calculated independent from the displacement basis $V_k$. More on this can be found in Chapter 7.

4-3 DEIM for general nonlinear functions

In the previous sections it has been assumed that the nonlinear functions on which DEIM is applied are point-to-point. In this section DEIM is generalised to nonlinear functions that for example may depend on multiple spatial variables at component $f_i(t_i)$.

4-3-1 Point-to-point and general nonlinear functions

Until now, the DEIM has been introduced as a method that reduces the number of components that need to be calculated in the nonlinear function $f(t)$. Each of these $m$ selected components was assumed to be a function of a corresponding component in the spatial domain:

$$f_i(t) = f_i(y_i, t)$$

This componentwise or point-to-point dependency can be illustrated using an arbitrary nonlinear function $F(x, y)$, $x, y \in \mathbb{R}$. This nonlinear function can be discretised by a finite number of points $x_i, y_j$. In a point-to-point function, the function value $F_{rs}$ at $x_r, y_s$ is calculated from $x_r$ and $y_s$ only:

$$F_{rs} = F_{rs}(x_r, y_s).$$

This componentwise dependency is shown in figure 4-4.

The great advantage of point-to-point functions is that the selection of components in $f(t)$ straightforwardly indicates which variables are needed to evaluate the selected components. In fact, boolean matrix $P$ can be substituted inside $f$:

$$f(V_k\tilde{y}(t)) \approx U(P^T U)^{-1} P^T f(V_k\tilde{y}(t)) = U(P^T U)^{-1} f(P^T V_h\tilde{y}).$$

Hence for componentwise nonlinear functions, no further measures are needed to find the inputs that belong to the selected components in $f(t)$.

Unfortunately, only a limited number of interesting physical or mathematical problems are point-to-point. In many cases, the components in a general nonlinear function $F(x, y)$ depend on a sequence of inputs, for example:

$$F_{rs} = F_{rs}(x_{r-m}, \ldots, x_r, \ldots, x_{r+m}; y_{s-k}, \ldots, y_s, \ldots, y_{s+m}).$$

Figure 4-4b this type of dependency. The consequence of this multipoint nature is that matrix $P$ does not indicate which input variables $(x_i, y_i)$ are needed to calculate the component.
A componentwise or point-to-point dependent function. Each of the points in the $(x,y)$-function surface depends on the input of multiple plane has a single counterpart on the function points in the $(x,y)$-plane.

**Figure 4-4:** An example of a point-to-point or componentwise function (a) and the general case where a function evaluation depends on multiple components (b).

selected by $P$. This will be of importance for the application of DEIM on FEM, because FEM in nonlinear mechanics is a typical example in which multipoint nonlinear functions occur. The DEIM-interpolated nonlinear function can thus only be written as (compare with (4-34)):

$$f(V_k\tilde{y}(t)) \approx U(P^TU)^{-1}P^Tf(V_k\tilde{y}(t)).$$

(4-36)

Finding the input components that correspond with the $m$ selected components in $f$ may not always be straightforward. Fortunately, it will turn out to be relatively easy for the Finite Element Method. The reason for this is that the model topology (in terms of which elements are connected to each other) gives away the inputs required for an arbitrary internal force component. Chapter 6 will provide more detail on this.
This chapter details the load cases used throughout the project. They have been put in a separate chapter because their details may distract the reader from the essentials of how DEIM is implemented into FEM, which is the subject of Chapter 6. The finite element models of a bar (Section 5-2) and a beam element (Section 5-3) that are used throughout this report are detailed in this chapter. However, the chapter starts with a 1 dof problem of connected nonlinear springs, which serves as a step stone between the mathematics of DEIM and the finite element context.

5-1 1 DOF Problem with connected non-linear springs

The first and most basic example is a mechanical system build of nodes or point masses connected with nonlinear springs. Although this test case does not make use of the finite element method, it will be shown that it shares important properties with the finite element method. These shared properties enable us to study DEIM on this simplified test case first, and then generalise it to real finite elements. The model described in this section is analytically derived using Lagrange equations [13] and the Matlab symbolic toolbox. See Appendix B-2.

5-1-1 Non-linear Spring Model

The non-linear spring characteristic consists of a linear stiffness $k_1$ and a cubic term $k_3$. The internal spring force $f$ is thus defined as:

$$f = k_1 u + k_3 u^3.$$  \hspace{1cm} (5-1)

The spring has stress-stiffening behaviour for values $k_1$ and $k_3$ larger than 0. The load case described in the next Section 5-1-2 will eventually be solved for positive nonzero values for $k_1$ and $k_3$. The choice for a cubic term instead to make the spring non-linear may seem a
bit awkward, since a quadratic term $k_2u^2$ would make the spring non-linear as well. However, a cubic term is chosen because the third power ensures that the spring is stable. In a conservative system, a stable force always restores the system to its initial position after a small perturbation is experienced. The cubic term (and also the linear term) ensures that the spring force will always oppose the direction of motion of the disturbance. In case the added term were quadratic, the resulting spring characteristic would not be stable, because the spring force would not always oppose the disturbance. The instability that is caused by such type of springs is interesting to study on its own, but complicates the numerical stability of the model. Figure 5-1 shows the linear spring characteristic, together with an example of a stiffening and a softening spring. Note that all of these lines have a nonzero derivative at $x = 0$. This is a necessary property for a Newton-Raphson iteration to yield good results. During the iterative process of solving the equilibrium for a certain load step, the derivative is used to 'steer' the solver to a converging solution. A derivative $\frac{df}{dx} = 0$ will result in a singular system when the displacement correction is solved in the iterative loop (see 2-3).

5-1-2 Model Geometry

The load case of the nonlinear 1 dof springs has been defined while keeping in mind that it should allow all features of DEIM to be studied and at the same time be as simple as possible.

The first proposed test case consisted of three nodes that were connected by springs between two walls. The leftmost node was subjected to a force. In the full order solution the external load is transferred from the first node to the second, and from the second to the third. If DEIM is used with only one out of three dofs selected, the first node would become selected as a collocation point. The third point would then be completely independent. The second point would not be selected directly, but its displacement must be known in order to find the internal force at point 1. This is caused by the coupling of node 1 and 2 by the spring, which also appears in the equations of motion.

Figure 5-2 shows the three dofs model and how the connections make the internal forces
dependant on the adjacent nodes. Clearly, the middle node is connected to all nodes and the internal force depends on contributions from all of them. If, in stead of one, two DEIM-vectors were selected, the second node could be selected as a collocation point for the second DEIM-vector. Since the collocation points have to be calculated exact, the displacement of node three must be known. The result is that all displacements have to be known, which thus permits to study the effect of completely discarding both forces and displacements of nodes.

The easiest solution to the problem described above, is to simply add a fourth degree of freedom by adding a new node between node 3 and the right wall. The resulting load case provides a much richer model, without any sacrifice in terms of complexity. With four nodes, both approximations with one or two DEIM-vectors can be studied, as long as the collocation points are located at nodes 1 and 2, or 3 and 4.

5-1-3 Equations of motion

In this subsection the equations of motion are derived using the Lagrange approach. The derivations are limited to the static load case, in which inertia and damping effects play no role. However, this static load case can be easily generalised to the dynamic load case that will also be used in Chapter 6. It is left to the reader to study the Matlab code that implements the dynamics. See Appendix B-2 for the Matlab code.

Energies

First step is to derive the system energies. In the static load case, kinetic energy plays no role. Hence the potential energies of the springs are the only energies to derive. Going along the nodes from left to right and bottom to top, the potentials are:
\[ V_1 = \frac{1}{2}k_1 u_1^2 + \frac{1}{4}k_3 u_1^4 \]  
\[ V_2 = \frac{1}{2}k_1(u_2 - u_1)^2 + \frac{1}{4}k_3(u_2 - u_1)^4 \]  
\[ V_3 = \frac{1}{2}k_1(u_3 - u_2)^2 + \frac{1}{4}k_3(u_3 - u_2)^4 \]  
\[ V_4 = \frac{1}{2}k_1(u_4 - u_3)^2 + \frac{1}{4}k_3(u_4 - u_3)^4 \]  
\[ V_5 = \frac{1}{2}k_1 u_4^2 + \frac{1}{4}k_4 u_4^4 \]  

Summing up all contributions yields the total system energy:

\[ V = V_1 + V_2 + V_3 + V_4 + V_5 \]  

**Lagrange Equations**

For a system in which no forces due to kinetic energy, and no nonconservative forces are present, the Lagrange equations bore down to the derivative of the potential with the generalised coordinates. Calling \( \mathbf{u}^T = [u_1 \ u_2 \ u_3 \ u_4]^T \) the generalised coordinates, the Lagrange equations become:

\[ \frac{\partial V}{\partial \mathbf{u}} - \mathbf{Q} = 0, \]  

where \( \mathbf{Q} \) denotes the external force. In this case \( \mathbf{Q} = [g \ 0 \ 0 \ 0]^T \).

The analytical expression for the Lagrange equations is rather lengthy and the value of showing it here is limited. On the contrary, the nonlinear stiffness matrix \( \mathbf{K} = \mathbf{K}(k_1, k_3, u) \) (a function of \( k_1, k_3 \) and \( u \)), can be derived from these equations:

\[ \mathbf{K}(k_1, k_3, u) = \frac{\partial^2 V}{\partial \mathbf{u}^2} \]  

The stiffness matrix can be split in a linear and a nonlinear term:

\[ \mathbf{K} = \mathbf{K}_{lin} + \mathbf{K}_{nl}, \]  

with the following definitions:

\[ \mathbf{K}_{lin} = \mathbf{K}(k_1, k_3, u)|_{k_3=0} \]  
\[ \mathbf{K}_{nl} = \mathbf{K} - \mathbf{K}_{lin}. \]  

Note that the nonlinear part of \( \mathbf{K} \) could also be defined as:

R.J. Dedden Master of Science Thesis
\[ K_{nl} = K\big|_{k_1=0}. \] (5-13)

However, the definition of (5-11) is preferred, since it establishes the distinction between linear and nonlinear information more clearly. If the sixth spring potential is taken into account, the linear stiffness term yields:

\[
K_{lin} = \begin{bmatrix}
2k_1 & -k_1 & 0 & 0 \\
-k_1 & 2k_1 & -k_1 & 0 \\
0 & -k_1 & 2k_1 & -k_1 \\
0 & 0 & -k_1 & 2k_1
\end{bmatrix},
\] (5-14)

and the nonlinear term (Using Matlab notation) yields:

\[
K_{nl} = \begin{bmatrix}
K_{nl}(:,1) & K_{nl}(:,2) & K_{nl}(:,3) & K_{nl}(:,4)
\end{bmatrix}
\] (5-15)

with

\[
K_{nl}(:,1) = \begin{bmatrix}
3k_3(2u_1^2 - 2u_1u_2 + u_2^2) \\
-3k_3(u_1 - u_2)^2 \\
0 \\
0
\end{bmatrix}
\] (5-16)

\[
K_{nl}(:,2) = \begin{bmatrix}
-3k_3(u_1 - u_2)^2 \\
3k_3(u_1 - u_2)^2 + 3k_3(u_2 - u_3)^2 \\
-3k_3(u_2 - u_3)^2 \\
0
\end{bmatrix}
\] (5-17)

\[
K_{nl}(:,3) = \begin{bmatrix}
0 \\
-3k_3(u_2 - u_3)^2 \\
3k_3(u_2 - u_3)^2 + 3k_3(u_3 - u_4)^2 \\
-3k_3(u_3 - u_4)^2
\end{bmatrix}
\] (5-18)

\[
K_{nl}(:,4) = \begin{bmatrix}
0 \\
0 \\
-3k_3(u_3 - u_4)^2 \\
3k_3(u_3^2 - 2u_3u_4 + 2u_4^2)
\end{bmatrix}
\] (5-19)

Note that the nonlinear matrix \( K_{nl} \) is structured the same way as \( K_{lin} \), with components \( ij \) that are zero imply that dof \( j \) does not generate a nonlinear force at node \( i \).

### 5-2 Bar Elements: Truss frame

One of the more simple types of finite elements is the bar element (sometimes called truss element), of which several descriptions (both linear and nonlinear) exist. The bar element presented in this section is based on the bar element of Crisfield [4, chap. 3] (see Appendix A-5). Using this element definition, a truss frame has been defined that will be used to study DEIM.
5-2-1 Model Geometry

The truss frame is shown in Figure 5-4. The frame consists of 14 nodes, connected by a total of 20 bar elements. Nodes 1 and 8 are clamped in both horizontal and vertical directions. The structure is loaded at the top end on the right side (node 14).

5-3 Beam Elements: Clamped-Clamped Beam

Next to the geometrically nonlinear bar element, a beam element load case has been defined. The goal of this is to further generalise the implementation of DEIM by testing it on a more complex element type than the bar element. The beam element that is used in this load case is again geometrically nonlinear. The main difference between the bar and the beam element is that the internal force vector of the beam element contains shear forces and bending moments, apart from the normal forces that are also present in the bar element.

The derivations of the beam element are not considered to be within the scope of this report. The element as implemented in Matlab (see Appendix B-3-2) is based on [4, ch.7]. The derivations are more or less an extension of the bar element. The reader is referred to [4] for further information.
5-3-1 Load case

Figure 5-5 shows a beam that is discretised with 10 beam elements. Both ends of the beam are clamped, thus constraining the horizontal and vertical translations, as well as the rotations. Although the beam features linear elastic material behaviour, the geometrically nonlinear effect of reorientation under loading is incorporated. This means that the beam will behave as a normal linear beam element under axial loading, but will show nonlinear behaviour when subjected to out-of-plane loading, as shown in Figure 5-5.

5-4 Remarks

The three problem types defined in this chapter all show nonlinear behaviour where an internal force consists of a combination of a linear and a nonlinear part. This is in accordance with the general equation type for which DEIM has been derived in Chapter 4. Although there is no reason why DEIM cannot be used on purely nonlinear functions (hence, where no linear part is present), the emphasis has been laid on structural problems that combine linear with nonlinear behaviour.
The combination of linear and nonlinear contributions is typical for many geometrically nonlinear problems in structural mechanics. This motivates the choice for the three problems defined in this chapter.

The coupled point mass example was perhaps a bit artificial in the sense that the nonlinear spring characteristic is in fact closer related to materially nonlinear behaviour, rather than geometrical behaviour. Nevertheless, this makes no difference, because no geometric reorientation of the coupled point mass ‘structure’ can take place in this 1D problem.

The bar element was defined in a 2-dimensional problem setting, thus allowing for rotations of the elements that make it reorient with respect to the external loading (see also the bar example in Section 2-2-1). This makes the bar a good example of geometrical nonlinearities.

The beam element is chosen to further generalise the application of DEIM in the finite element context. This will be the treated in Chapter 7, but first of all the point masses and the bar element examples are tested in Chapter 6.
Implementing DEIM in the Finite Element Method

This chapter connects chapter 2 with chapter 4. The obvious goal is to implement DEIM on the finite element models obtained for nonlinear systems. This should be accomplished by using DEIM to reduce the evaluation cost of equations of the types (2-23) and (2-36). Although the motivation for this research stems from an interest of reducing calculation time, this will not be the focus of this chapter. First of all, the programming languages of Matlab that is used is not suited to make a fair comparison between reduced and unreduced systems. Too many functions and methods are concealed for the user, which does not allow a fair, objective comparison to be made. Moreover, time savings are only of importance when the reduction has proven to yield accurate results. Therefore this chapter will focus on the possibilities for implementation of DEIM and how these can be tested for their accuracy.

First of all, DEIM is used straightforwardly by direct application to (2-23) and (2-36). See Section 6-2, where this is tested using the example load cases described in Section 5-1 and Section 5-2. Although the described method gave good results, the real computational saving turned out to be limited. Therefore, a second method, described in Section 6-3, has been tried. This approach turned out to be more optimal, although there are still some issues that remain unsolved. Nevertheless, the potential of DEIM on finite elements has been proved by this application on bar elements.

Before moving on to Section 6-2, one last remark must be made: At the beginning of this project, there seemed to be no need to go into all the (mathematical) details of DEIM. The implementation seemed quite straightforward, allowing for load cases with complex elements such as beams and shells to be tested with relative ease. However, it turned out that the complexity of such elements tends to mask a lot of important details on how DEIM operates. This forced us to go back to less complex elements such as the nonlinear bar element and mechanical systems consisting of point masses. An example with the more complex beam element will be presented in Chapter 7.
6-1 Testing of DEIM in Finite Element Analysis

Application of DEIM on finite element test cases should answer the question whether or not DEIM can be used to reduce such models. Along with this first example a step-by-step procedure is introduced, which will be used throughout this chapter to study the results of DEIM implementations:

1. The first step is to verify that the given load case shows nonlinear behaviour. This is done by application of a static loading. A force-deflection curve can be obtained from this loading, which should indicate if nonlinearities are present. If not, then the load case must be modified to yield nonlinear behaviour.

2. If the load case is suited for studying nonlinear responses, then a dynamic loading is applied. Depending on the example, step or ramp loadings are used. The (maximum) amplitude of this loading equals the static loading. The dynamic loading is time-integrated using the Newmark method (Section 2-3-4) both for the fully nonlinear and a linearised system. For each of the \( N \) time steps, samples of the nonlinear internal force \( f(q(t_i)) \) and the corresponding displacements \( q(t_i) \) are taken and stored in snapshot matrices \( F_s \) and \( Q_s \).

3. The snapshot matrices are used to characterise the subspaces of the internal nonlinear forces and the displacements. This is done by computing their POD with the singular value decomposition (Appendix A-3):

\[
Q_s = V\Sigma_q W_q^T, \quad V \in \mathbb{R}^{n \times n}, \quad \Sigma_q \in \mathbb{R}^{n \times N_s}, \quad W_q \in \mathbb{R}^{N_s \times N_s}, \quad (6-1)
\]

\[
F_s = U\Sigma_f W_f^T, \quad U \in \mathbb{R}^{n \times n}, \quad \Sigma_f \in \mathbb{R}^{n \times N_s}, \quad W_f \in \mathbb{R}^{N_s \times N_s}. \quad (6-2)
\]

The space of nonlinear forces \( F \) is spanned by the orthonormal column vectors in matrix \( U \), and the space of displacements \( Q \) is spanned by orthonormal column vectors of matrix \( V \).

4. A reduced orthonormal basis for the displacements is made by selecting \( k \) basis vectors of \( V \). This reduced set of basis vectors is stored in matrix \( V_k \) and used to reduce (3-8). This yields the POD-reduced responses which will serve as a reference for DEIM.

5. The bottleneck in the POD-reduced system is further reduced using DEIM, with the first \( m \) basis vectors of \( U \), collected in matrix \( U_m \). The obtained POD-DEIM results can be compared with the POD-reduced responses.

6. To validate the accuracy of both the POD- and POD-DEIM reduced systems, a comparison is made with the original full order solution obtained in step 2.

6-2 Direct Application of DEIM on System Equations

The most straightforward application of DEIM is by direct substitution in the equilibrium equations (statics) and equations of motion (dynamics). Taking (2-36) as an example, the reduced form according to Section 3-2 can be written as follows:
\[ V_k^T M V_k \ddot{q}(t) + V_k^T K V_k \dot{q} + V_k^T f(V_k \dot{q}, t) = V_k^T g(t), \quad (3-8) \]

where \( V_k^T f(V_k \dot{q}, t) \) forms the bottleneck that has to be reduced with DEIM. In chapter 4 it was found that this reduction is done by an interpolation of \( m \) components in \( f \in \mathbb{R}^{n \times 1} \). The interpolation matrix or \textit{DEIM-matrix} \( D \) is structured in such a way that only \( m \) selected components in \( f \) are involved in the matrix-vector multiplication:

\[
\hat{f} = D f, \quad D = U (U P)^{-1} P^T. \quad (6-3)
\]

This result can then be left-multiplied with \( V_k \):

\[
\hat{f} = V_k^T \hat{f}. \quad (6-4)
\]

The same applies for the Jacobian of \( f \) (which is used in the Newmark method solver, see Section 2-3-4):

\[
\hat{J} = V_k^T \hat{J} f V_k = V_k^T D \hat{J} f V_k. \quad (6-5)
\]

Note that to avoid a singular stiffness matrix, \( k \leq m \) must be selected. This can be explained from the fact that \( D \) contains \( m \) independent columns (see for example Section 4-1-4). The following examples will use \( k = m \).

### 6-2-1 1 dof Nonlinear Mass-Spring System

The first example is the 4 dof problem (Section 5-1). The first step is to verify the nonlinear behaviour using a static load case. A load \( g = [g \ 0 \ 0 \ 0]^T \) is applied. The response is shown in Figure 6-1. Observe that the nonlinear response is highly stress stiffening, thus showing the nonlinear behaviour of the system.

The dynamic response is evaluated for both the linearised and the nonlinear system. The apparent difference between the static linearised and the fully nonlinear equations is reflected in the dynamic responses (Figure 6-3 and Figure 6-4). The linear response shows larger displacement amplitudes then the nonlinear response.

With the snapshots that were obtained from the nonlinear dynamic system, the \( 4 \times 4 \) matrices \( V \) and \( U \) are calculated through an SVD:

\[
V = \begin{bmatrix}
-0.6904 & 0.6143 & -0.3614 & -0.1235 \\
-0.5640 & -0.1447 & 0.6786 & 0.4477 \\
-0.4021 & -0.5923 & 0.0001 & -0.6982 \\
-0.2084 & -0.5009 & -0.6394 & 0.5448 \\
\end{bmatrix}, \quad (6-6)
\]

\[
U = \begin{bmatrix}
-0.9993 & -0.0362 & -0.0080 & 0.0061 \\
-0.0015 & -0.1569 & 0.9811 & 0.1131 \\
0.0014 & 0.1468 & -0.0901 & 0.9851 \\
-0.0376 & 0.9760 & 0.1710 & -0.1298 \\
\end{bmatrix}. \quad (6-7)
\]
Figure 6-1: Static response of the nonlinear coupled masses. Load amplitude $\lambda$ is scaled between 0 and 1. Green shows the linearised response, blue is the nonlinear response.
These basis vectors are also plotted in Figure 6-2. Two POD-DEIM reductions are tested: One that uses only one dof \((m = 1)\) and one that uses two dofs \((m = 2)\). These column vectors are selected by the corresponding singular values, as is explained in Appendix A-3.

The results for \(m = 1\) in Figure 6-3 show an increased amplitude for \(q_1\), whereas the other dofs are smaller then the linear response and even smaller then the nonlinear one in case of \(q_4\). This can be explained from the fact that the first displacement vector \(q_1\) has a linear distribution of components that puts the highest bias on \(q_1\) and the lowest on \(q_4\). The system of 4 dofs is reduced to a one dof problem, putting constraints on the solution. This artificially increases the stiffness, which is also reflected by the total internal forces Figure 6-4. Adding a second pair of vectors \((v_2, u_2)\) enables the reduced system to approximate the full order displacement solution much better: The red line ‘follows’ the blue one much better then the linearised system does. On the contrary, the internal forces are not well represented for \(m = 2\), although it has improved when compared with the case of \(m = 1\).

The overall impression from the POD-DEIM reduction on the mass-spring system does not provide conclusive information on the potential of DEIM. Although some responses seem to be improved in comparison with the computationally cheap linearisation, still others are worse. The overall picture is therefore still a bit unclear.

The aim of Model Order Reduction is to filter out any redundant information in (FEM) models. With this concept in mind, the question rises: How much redundancy is available in the load case considered here? The system consisting of four equal coupled masses might be too simple for reduction techniques to have potential. The main advantage of such simple load cases is that the results can be interpreted with relative ease when compared to more complex load cases. It has served as an important step-stone in between the theory and the application. However, for a good testing of the used reduction methods more complex models must be considered.
Implementing DEIM in the Finite Element Method

Figure 6-3: Displacement-time responses for the nonlinear equations, its linearisation and the DEIM-approximation using two sets of two basis vectors (one for \( m = 1 \), one for \( m = 2 \)) of \( V \) and \( U \).

6-2-2 Truss Frame

The limited number of Degrees Of Freedom and the fact that no real finite elements are considered make the mass-spring example more of a step-stone rather than an ultimate goal. Therefore, the steps of the previous section are repeated for the truss frame. With a larger number of dofs, the chances are higher that some elements will be mildly stressed when compared to the others. The behaviour of these elements might be described accurately using only the linear stiffness information that was already known from linear analyses. The nonlinear part of the stiffness can then be represented by a contribution that is found through interpolation of the exact responses of a small number of elements in the structure.

Just like was done for the mass-spring system, the first step is to do a static loading to verify that the structure is indeed showing nonlinear behaviour (Figure 6-5a). The dynamic response is solved for a ramp load at the tip of the truss at node 14. After \( 1/3 \)rd of the time steps, the load reaches its maximum amplitude, at which the load level is maintained throughout the remaining time steps. Again, the difference between the linearised and the full system of equations
Figure 6-4: Responses of the nonlinear internal force as a function of time. The full nonlinear equations, its linearisation and the POD-DEIM approximation for $m = 1$ and $m = 2$ are shown.
is of size $q$, $f \in \mathbb{R}^{20}$. This is since there are 14 nodes that each have 2 dofs. Nodes 1 and 8 are constrained at the fixed world at the left side. The remaining $14 \times 2 - 2 \times 2 = 20$ dofs are free.

The snapshots obtained for the nonlinear part of the internal forces yield a matrix of basis vectors $U \in \mathbb{R}^{20 \times 20}$. The corresponding singular values show a distribution (Figure 6-5d) that indicates that much of the nonlinear internal force dynamics is contained in the subspace spanned by the first four basis vectors. The first four singular values have a total sum of $2.8081 \times 10^9$. The sum of all singular values is $2.8098 \times 10^9$. Hence the total ‘energy’$^1$ is for 99.9% contained in the subspace spanned by the basis vectors $U_4 = [u_1 \ldots u_4]$. This determined the number of dofs to be selected for testing the POD-DEIM reduction.

The POD-DEIM reduction has been tested with 1, 2, 3 and 4 dofs. The results for the displacements are given in Figure 6-7. As can be seen, the responses for 2, 3 and 4 dofs are able to approximate the full 20-dof system with reasonable accuracy during the first $1/3^{rd}$ of the total time. After that, the ramp loading reaches its ceiling and remains at that load level. The 4-dof system is able to track the original response, for the POD- as well as for the POD-DEIM approximation. The approximation with 3 dofs is accurate for the POD-reduction, but becomes unstable whenever it is further reduced using DEIM. The POD-DEIM response for 2 dofs also deviates from the POD-response, but it does not become unstable.

Although the first singular value $\sigma_{f,1} = 2.7420 \times 10^9$ of $\Sigma_f$ contains $2.7420/2.8098 = 97.5\%$ of the ‘energy’, one dof is not enough to describe the 20-dof dynamic system. The inaccuracy in the response is however not caused by the DEIM-interpolation, but by the limitations of the displacement reduction. Indeed, the reference POD-reduced system is also not able to track the full order response. One thus has to be careful with selecting the number of reduced dofs, since they do not depend on $\Sigma_f$ alone, but also on $\Sigma_q$.

**Remarks**

The results obtained for the geometrically nonlinear truss frame show that the application of DEIM to a POD-reduced system can provide good results. For $m = 4$ dofs, the POD-DEIM response could accurately track both POD-reduced and the unreduced system. This POD-DEIM approximation required only information from 4 out of 20 components from the nonlinear internal force vector $f \in \mathbb{R}^{20 \times 1}$, together with $4 \times 4 = 16$ corresponding components of the Jacobian matrix. Without the additional reduction of DEIM, the POD-reduced system would theoretically have required $20 \times 20 = 400$ components$^2$ to be known from the Jacobian. The fact that the responses are accurate and stable while using a much smaller number of components implies that the computational cost can be reduced. Care must however be taken with this statement, as in the next section it will turn out that the real saving is limited.

---

$^1$The term ‘energy’ might be confusing here. It is used in many papers as a measure to denote the relative importance of singular values. A high singular value indicates that the solutions to a system of equations is attracted to the corresponding orthonormal vector. It has no one-to-one relation with the physical energy as in *mechanical energy* or *thermal energy*.

$^2$Of course in many cases the stiffness matrix is *sparse*. The number of components thus is a theoretical upper bound.

R.J. Dedden

Master of Science Thesis
(a) Deformed shape for the truss frame under static loading. (b) Displacement-load curve for node 14 (tip). The external load has been scaled: $\lambda \in [0, 1]$. Clearly, the relation is nonlinear.

(c) Singular values corresponding to the basis vectors $V = [v_1 \ldots v_n]$ of the displacement vectors $U = [u_1, \ldots, u_n]$ of the space of the nonlinear part of the internal force.

Figure 6-5: Results for a ramp loading on the truss frame.
Figure 6-6: The displacement response for node 14 of the truss frame. The exact, fully nonlinear response (blue) clearly differs from the linearised one (green).
Figure 6-7: Results obtained by direct application of DEIM on the equations of motion of the truss frame.
6-3 Application of DEIM on Unassembled Nodal Forces

The successful reduction of the truss load case has been implemented with relative ease, since the reduction steps are performed on already assembled nodal force vectors. The accuracy of the POD-DEIM reduction could be tested simply by solving the original full order system, followed by a filtering operation. This filtering is done through left-multiplication of the internal force and stiffness matrix with the DEIM-matrix $D$, as explained in Chapter 4. Of course, this approach does not provide a computational time saving because all elements are still processed by the FEM-code.

6-3-1 Selecting Finite Elements with the Direct Approach of DEIM

The accurate results from the previous section triggered the next step: A study of the efficiency of the direct approach for POD-DEIM on equations such as (3-8). Two aspects are important for the solving of nonlinear system equations:

- The cost of obtaining the nonlinear part of the internal force $f$ and the nonlinear part of the Jacobian $K$.
- The cost of factorising the Jacobian in the Newmark time integration (dynamics) or Newton-Raphson iteration (statics).

The second aspect has been mentioned in Section 2-3. Roughly speaking, the computational effort required to factorise (numerically invert) the Jacobian is proportional to the density of this matrix. Hence more components in $K$ generally means that more computations have to be done. The direct approach of POD-DEIM contributes to a reduction of this amount of computations by reducing the amount of components that are taken into account. On the contrary, the first aspect is not addressed by this direct approach, since all elements are first processed and then filtered afterwards.

The inefficiency of the direct approach gave rise to the question whether it is possible to also reduce the amount of finite elements that are processed by the nonlinear finite element code. Here it is important to realise that the reduction from 20 to 4 dofs that was made does not imply that only 4 of the finite elements have to be processed! This can be explained as follows:

- Suppose that one of the two dofs of node 7 in Figure 6-5a is selected as a DEIM-point. Hence the internal force is to be calculated exact for this dof.
- The internal force in node 7 is the sum of contributions from elements 6, 18 and 24.
- Thus individual (local) internal forces in these three elements have to be solved for and are then summed up to yield the global internal force.
- Hence the total amount of finite elements that is processed is higher than the number $m$ of dofs used in the assembled set of equations (3-8).

For the truss frame, the elements that have to be processed in the direct approach are listed in Table 6-1. Clearly, the amount of elements increases quickly. For $m = 3$ and $m = 4$, already 10 out of 20 elements are needed! This greatly reduces the efficiency of the chosen POD-DEIM implementation.
6-3 Application of DEIM on Unassembled Nodal Forces

Table 6-1: Required elements for the internal forces at DEIM-selected dofs using the direct approach.

<table>
<thead>
<tr>
<th>$m$</th>
<th>Element numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6,18,24</td>
</tr>
<tr>
<td>2</td>
<td>6,7,8,14,18,19,24</td>
</tr>
<tr>
<td>3</td>
<td>5,6,7,8,14,17,18,19,23,24</td>
</tr>
<tr>
<td>4</td>
<td>5,6,7,8,14,17,18,19,23,24</td>
</tr>
</tbody>
</table>

6-3-2 Selecting Finite Elements using Unassembled Nodal Forces

The efficiency problem that appeared with the direct approach is not of importance for truss frames of bar elements only. In general, this problem will appear for all types of elements and model topologies. It is inherent to the Finite Element Method that nodes are shared by multiple elements. To generalise this problem, an arbitrary mesh of a plate structure is considered in Figure 6-8a. Here edge nodes are shared by one or two elements (one at the vertices and two at the edges). Nodes that lie inwards from the edges even share four elements. If an arbitrary number of dofs is picked by DEIM, then the number of elements quickly increases.

A possible solution to the selection problem is to allow DEIM to operate on the nonlinear internal force data before the element assembly is done. Unassembled nodal forces obviously belong to single elements, instead of them being a sum of contributions from multiple elements. As can be seen from Figure 6-8b, this results in a smaller amount of selected elements.

For $j$ elements that each have $i$ dofs (for the plate $i = 8$ (2 per node) and $j = 20$), the unassembled nonlinear force $f^u$ and the stiffness matrix $K^u$ have the following shape:

$$f^u = \begin{bmatrix} f_1^u \\ \vdots \\ f_j^u \end{bmatrix} \in \mathbb{R}^{(j \cdot i) \times 1}, \quad K^u = \begin{bmatrix} k_{i1}^{11} & \cdots & k_{i1}^{1i} \\ \vdots & \ddots & \vdots \\ k_{i1}^{j1} & \cdots & k_{i1}^{ji} \end{bmatrix} \quad \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} k_{i1}^{ij} & \cdots & k_{i1}^{ji} \\ \vdots & \ddots & \vdots \\ k_{ij}^{ij} & \cdots & k_{ij}^{ji} \end{bmatrix} \in \mathbb{R}^{(j \cdot i) \times (j \cdot i)}. \quad (6-8)$$

Using unassembled nodal forces implies that a different set of basis vectors is calculated from unassembled snapshots of internal forces. Taking a total of $N_s$ snapshots from $f^u$ yields the matrix of unassembled nonlinear internal force snapshots $F_s^u$ and the corresponding singular value decomposition (POD):

$$F_s^u = \begin{bmatrix} f^u_1 \\ \vdots \\ f^u_{N_s} \end{bmatrix} \in \mathbb{R}^{(j \cdot i) \times N_s}, \quad U^u_j \Sigma^u_j W^u_j \quad U^u_j \in \mathbb{R}^{(j \cdot i) \times (j \cdot i)}, \quad \Sigma^u_j \in \mathbb{R}^{(j \cdot i) \times N_s}, \quad W^u_j \in \mathbb{R}^{N_s \times N_s}. \quad (6-9)$$

Master of Science Thesis  R.J. Dedden
72 Implementing DEIM in the Finite Element Method

(a) The direct approach of DEIM is inefficient because selected dofs can be shared by many elements before they are assembled. This avoids many elements being selected by a small number of DEIM-dofs.

(b) The unassembled approach selects dofs of elements before they are assembled. This avoids many elements being selected by a small number of DEIM-dofs.

Figure 6-8: The direct approach compared with the unassembled approach on a meshed plate structure.

The DEIM interpolation of the nonlinear part of the internal force for \( m \) dofs yields:

\[
\hat{\mathbf{f}}^u = \mathbf{D}^u \mathbf{f}^u, \quad \hat{\mathbf{K}}^u = \mathbf{D} \mathbf{K}^u.
\]  

(6-11)

This approximated quantities are still unassembled and thus have to be reassembled again. The reassembly process takes the blocks of the elementary nonlinear nodal forces and stiffness matrix and stores them in the corresponding components of an empty assembled vector and matrix respectively. Multiple contributions on individual components of these assembled force vector \( \mathbf{f} \) and stiffness matrix \( \mathbf{K} \) are summed up, just like is done in the element assembly process of the finite element code. The Matlab code that performs this operation is not detailed here, but can be found in Appendix B-4-1.

6-3-3 Truss Frame - Unassembled DEIM

The load case that was studied with the direct approach of DEIM in Section 6-2 is now repeated for the unassembled approach. The nodal forces are now stored per element, which results in a total number of \( 24 \times 4 - 6 = 90 \) dofs. The singular values corresponding to the basis vectors of the unassembled snapshots can be found in Figure 6-9. Observe that the distribution shows a large number of singular values that are approximately zero. This indicates that the snapshots are all attracted to a subspace \( \tilde{\mathbf{F}}^u \) that is spanned by the first 48 basis vectors of \( \mathbf{U}^u \in \mathbb{R}^{90} \). This can be explained from the fact that the elementary nodal forces are physically coupled with each other. Although treated as independent quantities, the physical coupling between them could make them algebraically dependent upon each other. This is then reflected in the rank-condition of \( \mathbf{F}^u \) and \( \mathbf{U}^u \).

The results for \( m = 1, \ldots, 4 \) show that the approximations are of similar quality when compared with the direct approach (compare Figure 6-5 with Figure 6-10). However, looking at the number of assembled elements, the unassembled approach turns out to be far more efficient: Comparing Table 6-1 with Table 6-2 indeed shows that the number of selected elements can never become higher than the number of dofs.
Furthermore, it can be observed that the unassembled approach also yields unstable results for $m = 3$. The response for $m = 5$ is also tested, which is also unstable. Because $m = 4$ gives good results, this cannot be related to the amount of 'energy'.

It was suggested that the instability of some solutions might be due to the fact that the approximated nonlinear part of the Jacobian $\tilde{J}$ is not symmetric. Indeed, $\tilde{J}$ has the structure of the matrix in the example of Section 4-2, which is bar code like and thus not symmetric. From a physical point of view this is unnatural, since it means that couplings between different dofs are not always bi-directional. This would mean, for example, that dof $i$ can exert a force on dof $j$, but $j$ cannot exert a force on dof $i$.

The symmetry observed in stiffness matrices of structural finite element models guarantees that the couplings are always equal. This property is satisfied by the derivation of the stiffness matrix. It is always symmetric, because it is derived from a potential, involving the product of the shape function matrix with its own transpose (see (2-13)-(2-17)).

To verify whether the loss of symmetry makes the system unstable, the trajectories of Figure 6-10 have been solved once more, but without reduction of the Jacobian. DEIM was thus applied only on the nonlinear part of the internal force. Surprisingly, this made no difference. The instability is thus not caused by the approximation of the Jacobian.

### 6-3-4 Efficiency of unassembled DEIM

Thus far, the unassembled DEIM approach proved to be effective in reducing the amount of finite elements that have to be processed. However, this is not the only aspect of this implementation that influences the efficiency. Roughly speaking, the computational cost consists of three contributions:

1. The computational cost of having to process a number $N_{\text{elem}}$ of finite elements. This is done for every Newmark or Newton-Raphson iteration (see Section 2-3-4).

2. The cost of reassembling the DEIM-approximated nonlinear forces and local stiffness matrices into assembled force vectors and stiffness matrices. This step also has to be performed for every iteration.

3. The cost of obtaining the basis $U^n$ from the unassembled snapshots. This has to be done only once.
Figure 6-9: The singular values that belong to the orthonormal basis vectors $u_1, \ldots, u_{90}$ that were derived from the $N_s$ unassembled nonlinear internal force snapshots.
Figure 6-10: Results obtained by direct application of DEIM on the Truss Frame. Note that unstable responses are detected by the solver. If the stepping matrix becomes badly conditioned (e.g. cond < $1e^{-18}$), the solving is stopped. The responses for the remaining time steps are zero-padded, which explains why the plots for $m = 3$ and $m = 5$ become zero.
Step 1 generally is improved when compared to the direct approach. This is because no more then \( m \) elements are involved, whereas the direct approach selects a much higher number of elements then the number of selected dofs.

Step 2 differs from the direct approach to DEIM since here the data is manipulated by DEIM before assembly takes place. This requires that the elementary internal nonlinear forces and stiffness matrices are all stored in unassembled force vector \( f_u \) and stiffness matrix \( K_u \). The size of these arrays can become a storage problem, since they are much larger then their assembled counterparts. Moreover, as the unassembled stiffness matrix \( K_u \) get multiplied with the DEIM-matrix \( D \), a bar code shaped matrix results that is generally non-sparse (see the example in Section 4-2). This means that the columns with indices \( \phi \) are generally completely filled with nonzero numbers. These numbers represent artificial coupling between elements that may be physically uncoupled. These couplings obviously have to be taken into account during the reassembly process. Thus much more components of the unassembled stiffness matrix are involved in the reassembly process, then would normally be the case when reassembly is done for elements that are physically coupled.

Step 3 involves the SVD (see Appendix A-3). Without going into detail about the exact computational cost, it is obvious that the unassembled approach requires a much larger number of computations. The direct approach is much more efficient here.

6-4 Conclusion

It has been shown that the application of DEIM can potentially reduce the number of computational steps needed to do a time integration of a geometrically nonlinear finite element truss frame. Although both the direct and the unassembled implementations have not yet been thoroughly investigated for their true computational savings, the potential of DEIM has been shown. The possibility of doing a time integration while taking the nonlinear information of only a few elements into account has been proven to yield accurate results when compared with the full order solution. On the contrary, the DEIM applications also showed that stable results are not always guaranteed. In some cases the POD reduced system gave stable results, whereas the application of the DEIM reduction (POD-DEIM) destabilised the solving process (for example: \( m = 3 \) and \( m = 5 \) in Figure 6-10). For the bar load case, it turned out that the instability is not caused by the fact that the POD-DEIM reduced Jacobian is not symmetric. The true reason for the instability is still unclear. Chapter 7 will try to address this issue.

Of the two approaches studied, the unassembled approach is judged to be the most promising, because it grants the user control over the maximum amount of elements that have to be assembled. With the direct approach, selected dofs can be shared by many elements. Depending on the type(s) of elements used and the model topology, selection of a small fraction of elements could still require a large number of elements to be assembled. The unassembled approach solves this inefficiency by doing the selection of dofs before the elements are assembled, thus allowing for individual elements to be pinpointed. The drawback however is that the size of the snapshot matrices is generally much larger then with the direct approach.
Chapter 7

Suggestions for Further Research on DEIM

Although Chapter 6 showed that DEIM can be applied with success on geometrically nonlinear finite element problems, there are still many things that need to be further studied. This chapter is dedicated to those who plan to do further research on DEIM in the Finite Element Method. Three important problems that were observed are detailed in this chapter:

1. DEIM cannot be directly applied on snapshots of finite element models that concern a combination of different forces and moments. The application of DEIM on bar elements was straightforward, because only nonlinear axial forces are present in bar elements. More complicated elements such as beams and shells output a combination of forces and moments. It will turn out that basis vectors \( u_1, \ldots, u_m \) obtained from a POD of these snapshots is unsuited to be used as an input to DEIM.

2. The unassembled approach generally yields snapshot-matrices that belong to a high-dimensional space (many rows in the snapshot matrix), as was already pointed out in the previous chapter. Something must be done to avoid this problem, but without compromising the beneficial property that unassembled DEIM needs only a limited amount of elements to be processed by the finite element code.

3. The instability that occurred in some of the simulations done in the previous chapter. A possible origin of this problem may lie in the fact that the basis vectors for the displacement reduction (matrix \( V_m \)) and the nonlinear force reduction (matrix \( U_m \)) are derived independently from each other.

It must be stressed that the suggested solutions or solution directions presented here reflect the status quo of the research. None of the concepts that are described here have been extensively tested, nor have they been theoretically validated. The suggestions done are to convey a way of thinking and to encourage the reader to evaluate on the statements made.

Master of Science Thesis

R.J. Dedden
7-1 DEIM on Heterogeneous Elements

Thus far, DEIM has been tested on nonlinear coupled mass-spring systems and nonlinear finite element bars. This section will show that application of DEIM on element types that combine different types of internal forces and even moments is not straightforward.

7-1-1 Application of DEIM on Nonlinear Beam Elements

The beam load case presented in Section 5-3 uses 2D nonlinear beam elements. This type of element has six dofs, three for each node. This yields the following internal force for element $j$:

$$
f^j = \begin{bmatrix} N_1 \\ V_1 \\ M_1 \\ N_2 \\ V_2 \\ M_2 \end{bmatrix}. \tag{7-1}$$

Here $N_i$, $i = 1, 2$ denotes the nodal axial or normal force, $V_i$ is the shear force and $M_i$ the bending moment. These physical quantities always remain uncoupled from each other during assembly, e.g. no normal forces are summed up with bending moments or shear forces and vice versa. Hence there is no loss of generality if the direct approach to DEIM is assumed.

The load case in Section 5-3 is studied using the steps described in Section 6-1. The beam is subjected to an out-of-plane step loading acting on node 8. As was mentioned in Section 5-3, this loading triggers the nonlinear behaviour of the beam. The snapshots that were collected from the full order solution were used as input for the singular value decomposition (POD,
The singular values that belong to the orthonormal basis vectors are shown in Figure 7-2a. As one can observe from the figure, the 27 singular values can be divided into three groups of 9. The corresponding orthonormal basis vectors \( u_1, \ldots, u_{27} \) reflect that there exist three groups of singular values. The absolute values of the components of \( u_1, \ldots, u_9; u_{10}, \ldots, u_{18} \) and \( u_{19}, \ldots, u_{27} \) are plotted in Figure 7-2b, Figure 7-2c and Figure 7-2d respectively. Clearly, each of the groups of vectors shows its own characteristic behaviour:

- Vectors \( u_1, \ldots, u_9 \) show large values in every \( i = 1 + 3n, n \in \mathbb{N} \) component. This means that these vectors are biased to the normal forces \( N_i \) (7-1).
- Vectors \( u_{10}, \ldots, u_{18} \) have a similar bias, but then to the shear forces \( V_i \).
- The bias of \( u_{19}, \ldots, u_{27} \) is on the bending moments \( M_i \). This bias seems to be stronger then the biases of the foregoing groups. This is related to the fact that the jump in the singular values from \( \sigma_{18} \) to \( \sigma_{19} \) is larger then the one from \( \sigma_9 \) to \( \sigma_{10} \).

The clear distinction that is made by the SVD is caused by the fact that the snapshot components that belong to the bending moments have numerical values that have a much smaller absolute value then those belonging to the shear forces. In turn, the shear forces have components of smaller magnitude then the normal forces. This has an important consequence for the POD-DEIM approximation:

The system of equations is reduced by selecting the first \( m \) basis vectors of \( U \), resulting in a set of orthonormal basis vectors \( U_m \). Since \( m \) is typically chosen much smaller then the total number of Degrees Of Freedom (dofs) \( (m \ll n) \), the reduced basis for the nonlinear part of the internal forces will consist of basis vectors that put more emphasis on the axial forces then on the shear forces and bending moments. Snapshot data from the bending moments and/or shear forces is then disregarded in the POD-DEIM approximation.

For the beam load case it can be shown that this gives poor results. Using the direct approach, the responses deteriorate even for \( m = 18 \) (see Figure 7-3). The reason is that the bending moments play an important role in the out-of-plane load case. However, the basis vectors that contain the information needed to accurately describe the out-of-plane deformation in terms of bending moments, are missing in the reduction whenever \( m < 19 \). The selected subspace \( \tilde{F} \) is thus not suitable to describe the out-of-plane load case. This results in instability after only a small number of time steps. Even \( m = 25 \) already gives results that deviate clearly from the full order solution.

### 7-1-2 Possible Solutions for Heterogeneous Snapshots

The cause of the problem that occurred with the beam load case could be easily found from the large jumps in the singular values. The bending moments were clearly disregarded in the reduction, even though the load case showed out-of-plane bending. This contradiction between expected system behaviour (bending of the beam, thus bending moments are of importance) and the behaviour of the POD-DEIM reduction (disregard of the bending components in the basis vectors) can be generalised for any kind of snapshot matrix that stores quantities of different physical properties. For now, these kind of snapshots are called heterogeneous snapshots.

Master of Science Thesis

R.J. Dedden
The singular values are clearly divided into three groups.

(b) $|u_1|, \ldots, |u_9|$: Dominance of $N$.

(c) $|u_{10}|, \ldots, |u_{18}|$: Dominance of $V$.

(d) $|u_{19}|, \ldots, |u_{27}|$: Dominance of $M$.

Figure 7-2: Singular values and corresponding orthonormal basis vectors (left-hand singular vectors) of the space of the nonlinear part of the internal force $f$. Absolute values of the components of the basis vectors are shown, and colors are used to make components of different vectors distinguishable. The basis vectors that belong to each of the three groups in (a) are plotted in (b), (c) and (d). Each of the groups has a different component of the internal force ($N$, $V$ or $M$) that is dominant.
Figure 7-3: Instability occurs after a small number of time steps for the beam load case.
POD with Weighted Inner Product

The solution might be to weigh the snapshot components. The phenomenon of having heterogeneous snapshots as an input to a POD is not limited to problems in the field of structural mechanics. In fact, in the development of POD this problem has already been addressed [15], though be it in an abstract manner.

The concept of [15] is to define a matrix $W$ that can be used to transform the inner product of two orthonormal basis vectors $u^T u$ into $u^T W u$, with $W$ a symmetric, positive definite matrix. The matrix $W$ can then be used to express a new set of basis vectors:

$$\bar{u} = W^{1/2} u \in \mathbb{R}^n,$$  \hspace{1cm} (7-2)

These basis vectors form the left-hand matrix of a new SVD for weighted snapshots $\bar{F}_s \in \mathbb{R}^{n \times N_s}$:

$$\bar{F}_s = \bar{U} \Sigma \bar{V}^T, \quad \bar{U} \in \mathbb{R}^{n \times n}, \quad \bar{V} \in \mathbb{R}^{N_s \times N_s}. \hspace{1cm} (7-3)$$

The weighted snapshots turn out to be expressible as a function of the original snapshots $[f_1, \ldots, f_{N_s}]$:

$$\bar{F}_s = W^{1/2} F_s.$$  \hspace{1cm} (7-4)

The weighting of the snapshots (7-4) effectively modify the relative importance of individual components of the snapshot vectors. In case of the beam example, this would mean that the bending moments receive higher weights then the normal forces, thus correcting the bias that the SVD puts on the normal forces. The question is now how to find a weighting matrix that properly weights the snapshot components.

Weighting with Nodal Displacements

The first solution that has been suggested is to multiply the nonlinear internal force components with the displacements that belong to the same nodes. This idea makes sense from a physical point of view, because it can be related to the concept of Work. By weighting the internal nonlinear forces and moments with their corresponding displacements/rotations, the total work per node can be calculated. All snapshot components then have the same unit [J]. For an individual snapshot component $f_j$ this nonlinear work can be expressed as:

$$W^{nl}_j(t) = \int_0^{q_{ij}} f_j(t)q_j(t) \, dq_j$$  \hspace{1cm} (7-5)

The nonlinear part of the work requires that the product $f_j q_j$ is integrated over the complete displacement trajectory, starting from the equilibrium position $q_{ij} = 0$. This requires many extra calculation steps. From this it is concluded that the use of work to weigh the snapshots is not feasible.

R.J. Dedden \hspace{1cm} Master of Science Thesis
Weighting with Nodal Velocities

One way to overcome the problem with displacement-weighted snapshots is to choose a physical quantity different then work. A logical choice is to weigh the snapshots with an expression of power, as will be explained here.

The main difference between work and power, is that the work requires the time-history of the displacements to be known. Power is an instantaneous physical quantity, that depends only on the current forces and displacements. Like the work, the nonlinear part of the power can be expressed as:

\[ P_{nl}^j(t) = f_j \dot{q}_j. \]  (7-6)

Hence power is a physical quantity that is easily obtainable and provides a weighting that converts all snapshot components into the same physical unit. The only problem is that this approach is not applicable to static problems, since no velocity information is present. On the contrary, dynamic problems that are solved with the Newmark method always have the velocity information available, because it is used in the solving scheme (see Section 2-3-4).

7-2 Large Snapshot Matrices

Section 6-3 proved that the unassembled approach to DEIM is able to select a number of elements that is smaller than or equal to the number of selected dofs \( m \). However, the drawback of this method is that the snapshot matrices of the nonlinear part of the internal force can have a very large dimension. The theoretical maximum number of rows of such snapshot matrices is equal to the number of elements \( N_{elem} \) multiplied with the number of dofs per element \( N_{dof}^e \).

For the nonlinear bar elements, \( N_{dof}^e \) was only 4. However, more complex types of (3D) elements can have multiple times the amount of dofs. In practice, this puts considerable limitations on the use of the unassembled approach. The idea is to come up with different methods that share the beneficial property of unassembled DEIM to select a limited amount of elements, but without the drawback of having to deal with large snapshot matrices and corresponding basis vectors.

7-2-1 Snapshots of Elementary quantities

A possible solution to the described problem can be found by realising that DEIM may be applied to a different quantity that can be derived from the finite element model. Thus far, the interpolation was done using basis vectors that were based on internal forces at the nodes of the finite elements. However, if it is assumed that the snapshots of the nonlinear (part of the) internal forces at the nodes can be averaged per element, then the snapshots that are used as input for the POD can be of much smaller dimension.

The idea stems from the application of DEIM to FEM, rather then from the mathematics of the algorithm. As has been mentioned in Section 6-3-4, the underlying idea behind the application of DEIM on finite elements is to reduce the amount of elements that have to
be processed by the finite element code. As described, this is not the only aspect on which computational savings can be achieved. However, the apparent problem of the unassembled approach to DEIM opened a way of looking at DEIM from the perspective of the finite element context. The main question is:

Can DEIM be used on FEM, such that the selected indices \( \phi \) (Section 4-1-2) refer directly to elements of the finite element model?

Obviously, if there is only one snapshot component per element, then the size of the snapshots is much smaller then in the unassembled approach. At the same time, the beneficial property that individual elements are selected for processing remains untouched. The trick is to find a physical quantity that describes the global model behaviour accurate enough to use it for the POD that is part of the POD-DEIM reduction.

7-2-2 Element Averages of Nodal Nonlinear Forces

The most straightforward solution is to define a nonlinear force quantity that is found by summing up nodal internal forces of unassembled elements. For an \( n_e \times 1 \) elementary nonlinear internal force vector \( \mathbf{f}^j \) of element \( j \), this can be written as:

\[
\mathbf{f}^j = \mathbf{a}^T \mathbf{f}^j, \quad \mathbf{a} = [a_1, \ldots, a_{n_e}], \quad \mathbf{f}^j \in \mathbb{R}^{n_e \times 1}.
\]  

(7-7)

The choice of the coefficients \( a_i \) is still not discussed. It is possible to normalise these components by defining some norm on the global ensemble of the coefficients of all elements. Another possibility would be to combine this element average concept with the weighted snapshots techniques that were discussed in Section 7-1-2. This would then result in an element-averaged physical quantity such as power that properly accounts for the various heterogeneous contributions from the nonlinear internal force vector \( \mathbf{f}^j \).

7-3 Instability of POD-DEIM Reduced Solutions

The results that were obtained in Chapter 6 showed that sometimes the POD-DEIM reduction becomes unstable, although the corresponding solution of the POD-reduced system could accurately track the full order solution. This happened for example in the POD-DEIM reduction of the truss frame when three or five \((m - 3 \text{ or } m = 5)\ dend only the index \( m = 4 \) is not unstable!

The instability was suggested to be related to the loss of symmetry in the stiffness matrix or Jacobian, but this turned out not to be the case for the bar element load case.

7-3-1 Mappings between Displacement Space and Nonlinear Force Space

The reason that instability occurs for some reductions can perhaps be explained from the fact that the two reductions (a reduction of the displacement field and a reduction of the nonlinear part of the internal force) are done independent from each other.

This can be explained with the aid of some pictures that visualise the various subspaces.
Figure 7-4: The full order spaces $Q$ and $F$ are shown in gray. The mappings $f$ and $f^{-1}$ always exist. The second picture shows the subspaces $\tilde{Q} \subset Q$ and $\tilde{F} \subset F$. These are found from a POD. Because the only criterion on these subspaces is that they are optimal with respect to their respective full dimensional spaces $Q$ and $F$, the existence of the mappings is not guaranteed. A different set of subspaces $\tilde{Q}^{*}$ and $\tilde{F}^{*}$ might exist that does guarantee the existence of the mappings. The GSVD might be a suited algorithm to find these new subspaces.
Suppose that from an existing nonlinear finite element model a discretised nonlinear internal force function \( f \in \mathbb{R}^{n \times 1} \) can be derived. Here \( n \) denotes the number of dofs.

The set of possible displacements \( q \in \mathbb{R}^{n \times 1} \) of the finite element model are represented by a space that is denoted with \( \mathcal{Q} \subset \mathbb{R}^n \).

Similarly, the set of possible nonlinear internal force vectors can be represented by \( \mathcal{F} \subset \mathbb{R}^n \).

The function \( f(q) \) does a mapping for any given displacement vector \( q \in \mathcal{Q} \). For a general mapping function \( f \), this can be expressed as:

\[
f : \mathcal{Q} \rightarrow \mathcal{F}.
\] (7-8)

If the mapping done by function \( f \) is assumed to be reversible, then the displacements \( q \) corresponding to a given internal force \( f \) can always be found: \( f^{-1} : \mathcal{F} \rightarrow \mathcal{Q} \). This is shown in Figure 7-4.

Suppose that the \( q \) and \( f \) are reduced by projecting their spaces onto subspaces \( \tilde{\mathcal{Q}} \subset \mathcal{Q} \) and \( \tilde{\mathcal{F}} \subset \mathcal{F} \) respectively.

The column vectors of \( V_m \in \mathbb{R}^{n \times m} \) and \( U_m \in \mathbb{R}^{n \times m} \) hold the orthonormal basis vectors that span these subspaces.

Remember that in the POD-DEIM reduction, the matrix \( V_m \) was used to project the displacements onto, and the matrix \( U_m \) was used to do the combined projection and interpolation of DEIM. The DEIM-approximation approximated the nonlinear part of the internal force as follows: \( \hat{f} = U_m c \) (Section 4-1-2).

Both sets of basis vectors that span these subspaces are calculated independently: \( V_m \) is found from displacement snapshots \( \mathcal{Q}_s \) through POD, and \( U_m \) is found by a POD of snapshots of the nonlinear part of the internal force \( \mathcal{F}_s \).

The reduced system now consists of a mapping between the subspace of reduced displacements \( \tilde{\mathcal{Q}} \) to the subspace of reduced, interpolated nonlinear part of the forces \( \tilde{\mathcal{F}} \).

In the ideal case, mappings \( \tilde{f} : \tilde{\mathcal{Q}} \rightarrow \tilde{\mathcal{F}} \) and \( \tilde{f}^{-1} : \tilde{\mathcal{F}} \rightarrow \tilde{\mathcal{Q}} \) are possible. However, there is no guarantee that this is the case, as could be observed from the results for \( m = 3 \) and \( m = 5 \) (see Figure 6-7). The two subspaces \( \tilde{\mathcal{Q}} \) and \( \tilde{\mathcal{F}} \) were derived separately from each other, each based on their own set of snapshots. Although these snapshots have a physical relation with each other, the POD’s that yield \( V_m \) and \( U_m \) are calculated independently. Put in different words, this means:

The POD calculates the orthonormal basis vectors \( V_m \) in an optimal sense. The optimality criterion is to find the set of basis vectors who’s span \( \tilde{\mathcal{Q}} = \text{span}(v_1, \ldots, v_m) \) best approximates the displacement space \( \mathcal{Q} \) spanned by the snapshots. Next to that, the same is done for the nonlinear internal force snapshots, yielding subspace \( \tilde{\mathcal{F}} = \text{span}(u_1, \ldots, u_m) \), again an optimal reduced representation of its full order counterpart \( \mathcal{F} \). Both PODs are independent from each other.
Hence the existence of the mappings $\tilde{f}: \tilde{Q} \to \tilde{F}$ and $\tilde{f}^{-1}: \tilde{F} \to \tilde{Q}$ are not taken into account by the two separate PODs. This may be the reason for the apparent instability for $m = 3$ and $m = 5$.

Granted that the instability is caused by the independency of the PODs, the solution might be found from an algorithm that is able to constrain the optimality criteria that are used by the POD with the extra condition that the mappings $\tilde{f}: \tilde{Q} \to \tilde{F}$ and $\tilde{f}^{-1}: \tilde{F} \to \tilde{Q}$ exist. Such a new algorithm thus yields matrices $V^*_m$ and $U^*_m$ different from $V_m$ and $U_m$.

### 7-3-2 Optimal Bases Using the GSVD

The existence constraint described in the previous section suggests the use of an algorithm that \textit{integrally acts on the snapshots of both the displacements and the nonlinear part of the internal force}. One possible algorithm is the Generalised Singular Value Decomposition (GSVD). The GSVD is described in Appendix A-4. Here only the main point of interest is presented.

**Definition**

Given two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times n}$. Let $q = \min(p, n)$. Then there exist:

- Orthogonal matrices $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{p \times p}$.

- A square non-singular matrix $X \in \mathbb{R}^{n \times n}$.

- Diagonal matrices $C = \text{diag}(c_1, \ldots, c_n)$, $S = \text{diag}(s_1, \ldots, s_q)$

such that:

$$A = UCX^T,$$

$$B = VSX^T,$$

$$C^2 + S^2 = I.$$  \hspace{1cm} (7-9) \hspace{1cm} (7-10) \hspace{1cm} (7-11)

**Application**

Suppose that the matrices $A = F_s$ and $B = Q_s$ are snapshot matrices of the nonlinear part of the internal forces and of the displacements. The matrices $U$ and $V$ then span the spaces $Q$ and $F$. The matrix $X$ contains information on the time history of both the snapshot sets in $A$ and $B$. This matrix is part of the decompositions of both $A$ and $B$, and thus couples the decompositions. This might provide a framework to find bases $V^*_m$ and $U^*_m$.

The application of the GSVD is not as straightforward as it may seem. Although it shares many properties with the SVD, a couple of remarks have to be made for the sake of clarity:

- The definition presented here is one of the possible definitions. Other alternative definitions exist, which may have (slightly) different results. The implementations in numerical software packages can therefore be different.
• The matrices $U$ and $V$ have independent, but not necessarily orthonormal columns. Direct substitution of $U_m^* = U_m$ and $V_m^* = V_m$ is therefore not possible.

Thus far, it remains to be investigated whether the GSVD can solve the instability problem. Currently no conclusive information is available, neither from theory nor from testing.
The investigations on the use of Discrete Empirical Interpolation Method (DEIM) for the reduction of geometrically nonlinear finite element models have provided a number of results. First of all, it has been shown that DEIM is able to approximate fully nonlinear behaviour of geometrically nonlinear finite element models with good accuracy. This is done by using the nonlinear force information of only a small number of finite elements. The nonlinear part of the internal forces is calculated exact for these DEIM-selected elements. The nonlinear part of the internal force of the remaining elements is interpolated with the exact responses of the DEIM-selected elements. The DEIM-algorithm thus reduces the number of components of the global internal force that have to be calculated. DEIM is to be combined with a reduction on the displacement field, through a Proper Orthogonal Decomposition (POD). With this so-called POD-DEIM approximation, it was possible to reduce the nonlinear contributions in a truss frame from 20 elements back to just 4 elements.

Two different implementations of the POD-DEIM approximation have been tested: the direct approach and the unassembled approach. Both methods have their respective inefficiencies: The direct approach tends to use large amounts of elements to process only a few DEIM-selected nonlinear internal forces. This is because the selected nodal nonlinear internal forces require contributions from all the adjacent elements. On the contrary, the unassembled approach operates on internal force snapshots before the model is assembled. This limits the amount of selected elements to a number smaller then or equal to the number of DEIM-selected Degrees Of Freedom. However, the unassembled approach suffers from the fact that the snapshot-matrices have an undesirably large size. Further research on taking snapshots of element averaged quantities may overcome this problem.

The POD-DEIM reduction has not yet been successfully tested in combination with heterogeneous snapshots. Such snapshots occur when the finite elements inhere various types of forces and or moments. Then the numerical values of these snapshots do not straightforwardly provide information on their relative importance. To overcome this problem, a method that applies weights to the snapshot components is suggested, but not implemented yet.

Unfortunately, the POD-DEIM approximation does not always remain stable during time integration with the Newmark method. For example, the direct approach became unstable
for 3 selected dofs. The same was observed from the unassembled responses for 3 and 5 dofs. The reason may be that a basis for the subspace $\mathcal{Q}$ of the reduced displacement field is found through POD without taking into account the existence of the corresponding reduced nonlinear internal force subspace $\mathcal{F}$. In turn, the POD that finds the basis for $\mathcal{F}$ does not take the existence of $\mathcal{Q}$ into account. Hence the existence of the mappings $f : \mathcal{Q} \rightarrow \mathcal{F}$ and $f^{-1} : \mathcal{F} \rightarrow \mathcal{Q}$ is not part of the optimality criteria for the two PODs. It is suggested that the GSVD is able to take this requirement into account. Further research may show whether this statement is correct.

Based on the observations summed up above, the conclusion is drawn that the DEIM has the potential to improve the Model Order Reduction (MOR) techniques that are now common in structural mechanics. Although a number of difficulties remain to be overcome, the POD-DEIM reduction showed that accurate results can be obtained using only a fraction of the nonlinear information of a finite element model.
Appendix A

Theory Reference
**A-1 Proof of the Existence of** \((P^T U)^{-1}\)

This proof is adopted from [3]. Here it is presented with some comments. It is proven that the invertible matrix \((P^T U)^{-1}\) of DEIM iteration \(l - 1\) can be expanded with a new column and row as a result of iteration \(l\), yielding an expanded matrix that is also invertible.

### A-1-1 Proof

Given \(m\) orthonormal basis vectors \(u_1, \ldots, u_m\). The first step is to introduce a simplified notation for various quantities:

- \(\bar{U} = [u_1, \ldots, u_{l-1}] \in \mathbb{R}^{n \times (l-1)}\), \(\bar{P} = [e_{\phi_1}, \ldots, e_{\phi_{l-1}}]\),
- \(u = u_l \in \mathbb{R}^n\), \(p = e_{\phi_l} \in \mathbb{R}^n\),
- \(U = [\bar{U} \ u] \in \mathbb{R}^{n \times l}\), \(P = [P \ p] \in \mathbb{R}^{n \times l}\).

Hence the quantities with a bar ‘\(\bar{\ }\)’ denote the terms from iteration \(l - 1\). Suppose that \(\bar{M} = \bar{P}^T \bar{U}\) is an invertible matrix. For step \(l\), this matrix is appended as follows:

\[
M = \begin{bmatrix}
\bar{M} & \bar{P}^T u \\
\bar{P}^T \bar{U} & \bar{P}^T u
\end{bmatrix} \tag{A-2}
\]

Setting \(a^T = p^T \bar{U}\), \(c = \bar{M}^{-1} \bar{P}^T u\), and \(\rho = p^T u - a^T c = p^T \left(u - UM^{-1}P^Tu\right)\), this matrix can be factorised in a product of two matrices:

\[
M = \begin{bmatrix}
\bar{M} & 0 \\
a^T & c
\end{bmatrix}
\begin{bmatrix}
I & 0 \\
0 & 1
\end{bmatrix}, \tag{A-3}
\]

which clearly shows that the second matrix is full rank, and thus invertible. Scalar \(\rho\) is guaranteed to be nonzero, due to the fact that it is the largest absolute component in \(r\):

\[
r = u - \bar{U}c \neq 0, \quad \rho = \|r_{\infty}\| = \max(|r|). \tag{A-4}
\]

Hence, since it was assumed that \(\bar{M}\) is invertible, the first matrix in (A-3) is also invertible. Recalling that the product of two invertible matrices is always invertible, this proofs that the expanded matrix \(M\) is invertible.

In the initialisation step \((l = 1)\), the DEIM-algorithm prescribed that the first selected index \(\phi_1\) corresponds to the maximum absolute component in \(u_1 \neq 0\). Hence, this defines \(M = e_{\phi_1}^T u_1\) to be a nonzero, thus invertible, scalar. In the next iteration \((l = 2)\), this scalar is identified by \(\bar{M}\) in (A-2). It can thus be shown through induction that if \(M\) is invertible at step \(l - 1\), then it will also be at step \(l\). QED.

R.J. Dedden

Master of Science Thesis
A-1 Proof of the Existence of \((P^T U)^{-1}\)

A-1-2 Inverting \(M\)

To finalise, recall that for a product of any two invertible matrices \(A\) and \(B\), the inverse is defined by:

\[
(AB)^{-1} = B^{-1}A^{-1}.
\]  

(A-5)

The inverse \(M^{-1} = (P^T U)^{-1}\) is thus found to be:

\[
M^{-1} = \begin{bmatrix}
1 & -c \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
M^{-1} & 0 \\
-\rho^{-1}a^TM^{-1} & \rho^{-1}
\end{bmatrix}
\]  

(A-6)

\[
M^{-1} = \begin{bmatrix}
1 & -c \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
I & 0 \\
-\rho^{-1}a^T & \rho^{-1}
\end{bmatrix}
\begin{bmatrix}
M^{-1} & 0 \\
0 & 1
\end{bmatrix}
\]  

(A-7)

\[
M^{-1} = \left\{ \begin{bmatrix}
I & 0 \\
0 & 0
\end{bmatrix} + \rho^{-1}\begin{bmatrix}
c \\
-1
\end{bmatrix} \begin{bmatrix}
a^T \\
-1
\end{bmatrix} \right\}
\begin{bmatrix}
M^{-1} & 0 \\
0 & 1
\end{bmatrix}
\]  

(A-8)
A-2 The Inverse of $P^T U$ for a $5 \times 3$ example

In Section 4-1-4 the inverse of $P^T U$ is used for the derivation of the DEIM-matrix $D$. This inverse can be calculated analytically, which is the subject of this appendix.

The first step is to recall (4-24):

$$
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
u_{11} & u_{12} & u_{13} \\
u_{21} & u_{22} & u_{23} \\
u_{31} & u_{32} & u_{33} \\
u_{41} & u_{42} & u_{43} \\
u_{51} & u_{52} & u_{53}
\end{bmatrix}
\begin{bmatrix}
u_{11} & u_{12} & u_{13} \\
u_{21} & u_{22} & u_{23} \\
u_{31} & u_{32} & u_{33} \\
u_{41} & u_{42} & u_{43} \\
u_{51} & u_{52} & u_{53}
\end{bmatrix}
= B
= \begin{bmatrix}
b_{11} & b_{12} & b_{13} \\
b_{21} & b_{22} & b_{23} \\
b_{31} & b_{32} & b_{33}
\end{bmatrix},
$$

where now a new notation $P^T U_m = B$ (with a new, consistent indexing) is introduced to avoid confusion in the next step. The inverse of $B$ is analytically solvable, yielding:

$$
B^{-1} = \frac{1}{\det(B)}
\begin{bmatrix}
b_{22}b_{33} - b_{23}b_{32} & b_{13}b_{32} - b_{12}b_{33} & b_{12}b_{23} - b_{13}b_{22} \\
b_{23}b_{31} - b_{21}b_{33} & b_{11}b_{33} - b_{13}b_{13} & b_{13}b_{21} - b_{11}b_{23} \\
b_{21}b_{22} - b_{22}b_{21} & b_{12}b_{31} - b_{11}b_{22} & b_{11}b_{22} - b_{12}b_{21}
\end{bmatrix},
$$

$$
\det(B) = b_{11}(b_{22}b_{33} - b_{23}b_{32}) - b_{12}(b_{23}b_{21} - b_{22}b_{31}) + b_{13}(b_{21}b_{32} - b_{22}b_{31}).
$$

Back substituting $b_{11} = u_{11}, b_{21} = u_{24}, \ldots, b_{33} = u_{33}$ results in:

$$
(P^T U_m)^{-1} = \frac{1}{\det(P^T U_m)}
\begin{bmatrix}
u_{33}u_{42} - u_{32}u_{43} & u_{13}u_{32} - u_{12}u_{33} & u_{12}u_{43} - u_{13}u_{42} \\
u_{31}u_{43} - u_{33}u_{41} & u_{11}u_{33} - u_{13}u_{31} & u_{13}u_{41} - u_{11}u_{43} \\
u_{32}u_{41} - u_{31}u_{42} & u_{12}u_{31} - u_{11}u_{32} & u_{11}u_{42} - u_{12}u_{41}
\end{bmatrix},
$$

$$
\det(P^T U_m) = u_{11}(u_{33}u_{42} - u_{32}u_{43}) - u_{12}(u_{33}u_{41} - u_{31}u_{43}) + u_{13}(u_{32}u_{41} - u_{31}u_{42}).
$$

Observe that the columns of (A-12) can be given a shorthand notation $\lambda_i$, with subscript $i$ referring back to the row index of the original $P^T U_m$ (4-24), expressed in $u_{ij}$:

$$
(P^T U_m)^{-1} = \Lambda = [\lambda_1 \lambda_4 \lambda_3], \quad \Lambda \in \mathbb{R}^{m \times m}, \quad \lambda_i \in \mathbb{R}^{m \times 1}.
$$

This finalises the derivation of $(P^T U)^{-1}$. 

---

R.J. Dedden  
Master of Science Thesis
A-3  The Proper Orthogonal Decomposition and the Singular Value Decomposition

According to [3], the Proper Orthogonal Decomposition (POD) is ‘a method for constructing a low-dimensional approximation representation of a subspace in Hilbert space. It is essentially the same as the Singular Value Decomposition (SVD) in a finite dimensional space or in Euclidean space.’

The idea behind POD is that the column vectors of a data set $Y_s = (y_1, \ldots, y_{N_s}) \subset \mathbb{R}^n$ span a space $\mathcal{Y} = \text{span}(y_1, \ldots, y_{N_s}) \subset \mathbb{R}^n$ of dimension $r$. The space $\mathcal{Y}$ contains a finite number of subspaces, that contain a part of the space $\mathcal{Y}$. Let the $k$-dimensional subspace $\tilde{\mathcal{Y}} \subset \mathcal{Y}$ for $k < r$ denote such a part of the space. The central question in POD is:

Which $k$-dimensional subspace is the best representation of the original space $\mathcal{Y}$?

The idea is that an optimal $k$-dimensional subspace $\tilde{\mathcal{Y}}^*$ may describe the space $\mathcal{Y}$ better than an arbitrary $k$-dimensional subspace $\tilde{\mathcal{Y}}$. The reasoning behind this is that the snapshot vectors $y_i$ have relatively small components outside the $k$-dimensional subspace. A suitable algorithm will be needed to find the optimal subspace $\tilde{\mathcal{Y}}^*$. However, the first step is to define an optimisation problem that quantifies the ‘optimality’ of arbitrary subspaces $\tilde{\mathcal{Y}}$.

A-3-1  Optimisation Problem Formulation

An arbitrary $k$-dimensional subspace $\tilde{\mathcal{Y}}$ can be described by a set of $k$ orthonormal basis vectors that are selected from a set of $r$ basis vectors that span $\mathcal{Y}$:

$$U = [u_1, \ldots, u_r] \in \mathbb{R}^{n \times r}. \quad (A-15)$$

The selection of $k$ basis vectors can be done using the optimisation problem:

$$\min_{\{u_i\}_{i=1}^k} \sum_{j=1}^{N_s} \|y_j - \sum_{i=1}^k (y_j^T u_i) u_i\|_2^2, \quad (A-16)$$

subject to $u_i^T u_j = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$ \quad (A-17)

which expresses the optimality of an arbitrary subspace spanned by the $k$ basis vectors $u_i$. It calculates an objective for the optimal subspace using the following:

- Snapshot vector $y_j$ is projected onto the subspace $\tilde{\mathcal{Y}}$, that is spanned by an arbitrary set of $k$ basis vectors $u_i$: $\tilde{y}_j = \sum_{i=1}^k (y_j^T u_i) u_i$.

- The projected vector is subtracted from its original, unprojected counterpart. This yields the residual $r = y_j - \tilde{y}_j$, which can be interpreted as the part of $y_j$ that lies outside $\tilde{\mathcal{Y}}$. Taking the 2-norm (the vector length of $r$) yields an absolute error measure: $\|r\|_2 = \sqrt{r^T r}$.

- The procedure is done for all $N_s$ snapshot vectors, and the resulting absolute errors are summed up.
The minimiser of the sum above yields the set of basis vectors that span the optimal subspace $\hat{Q}_s$. This solution to the optimisation problem is found from the SVD of $Y_s$.

### A-3-2 The Singular Value Decomposition

The SVD of the snapshot matrix $Y_s$ is defined as:

$$Y_s = U\Sigma W^T,$$  \hspace{1cm} (A-18)

where the \textit{left-hand matrix} $U = [u_1, \ldots, u_n] \in \mathbb{R}^{n \times n}$ contains orthonormal $n \times 1$ vectors, the \textit{right-hand matrix} $W = [w_1, \ldots, w_{N_s}] \in \mathbb{R}^{N_s \times N_s}$ are orthogonal $N_s \times 1$ vectors and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n \times N_s}$, a rectangular matrix.

The \textit{singular values} $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n > 0$ govern the order of the basis vectors in $U$. If the first $k$ basis vectors in $U$ are selected, then it can be shown that the minimum 2-norm error from approximating the snapshots using the POD basis is given by:

$$\sum_{j=1}^{N_s} \|y_i - \sum_{i=1}^{k}(y_j^T u_i)u_i\|^2 = \sum_{i=k+1}^{r} \sigma_i^2.$$  \hspace{1cm} (A-19)

Hence the sum of the remaining $n - k$ singular values gives an indication of the accuracy of the approximation $\hat{Q}$ of $Q$. Minimising this sum thus yields the solution to the optimisation problem (A-16). Calculating the SVD by hand is possible, but not treated here. See [8, ch. 7] for more details.
The Generalised Singular Value Decomposition

The GSVD as presented here is adopted from [9, ch. 7]. Note that other definitions exist as well, for example [1].

Given two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times n}$. Let $q = \min(p, n)$. Then there exist:

- Orthogonal matrices $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{p \times p}$.
- A square non-singular matrix $X \in \mathbb{R}^{n \times n}$.
- Diagonal matrices $C = \text{diag}(c_1, \ldots, c_n)$, $S = \text{diag}(s_1, \ldots, s_q)$

such that:

\[
A = UCX^T, \quad \text{(A-20)}
\]
\[
B = VSX^T, \quad \text{(A-21)}
\]
\[
C^T C + S^T S = I. \quad \text{(A-22)}
\]

A proof is given in [7, p. 466]. This proof does not mention that if rank($[A; B]$) $> m$ then $C$ is not necessarily a diagonal matrix, but has a non-zero upper diagonal $p = \text{rank}([A; B]) - m$

This means that the nonzero components of $C$ are located at indices $c_{ij}: j - i = p$. Hence the coefficients can possibly lie on one of the upper diagonals of $C$ [9].

In most implementations, the nonzero diagonal $\text{diag}(c_1, \ldots, c_n)$ are in decreasing order ($1 > c_1 \geq c_2 \geq \ldots \geq c_n$) and the nonzero diagonal $\text{diag}(s_1, \ldots, s_q)$ in increasing order ($0 < s_1 \leq s_2 \leq \ldots \leq s_q$), which follows from (A-22).
A-5 2D Bar Element using Engineering Strain

It is important to make a distinction between \textit{geometrical} and \textit{material} nonlinear behaviour. In the bar element in this section the nonlinearity arises only from the geometry! Hence the material parameters (Young’s modulus $E$, cross section area $A_0$) are constants. Moreover, the strain measure $\varepsilon$ is a linear function of the elongation, as will be pointed out below. The focus of the first part of this section will be on the properties that make the geometrically nonlinear bar differ from an ordinary linear bar element. The second part will then show the bar element description accompanied with some comments on how it inherits the nonlinear properties identified in the first part.

\textbf{Material Properties}

The material properties of the bar element are assumed to be linear. Thus the Young’s modulus $E$ and the cross section area $A_0$ are constants, as mentioned above. With the initial length defined as $L_0$, this yields a constant (linear) normal stiffness:

$$k = \frac{EA_0}{L_0} \quad (A-23)$$

If the bar element is loaded in axial direction by nodal forces $F_1$ and $F_2$, the static equations of motion would be:

$$\frac{EA_0}{L_0}(u_1 - u_2) = F_1 \quad (A-24)$$
$$\frac{EA_0}{L_0}(u_2 - u_1) = F_2 \quad (A-25)$$

See [5, chap. 2.2] for more details about a linear bar. Obviously, this is exactly the same as for a linear bar element as treated in Section 2-1-2.

\textbf{Geometrical Properties}

The linear bar element is in essence a 1 dof model. If local coordinates are considered, the axial displacements $u_1$ and $u_2$ are the only degrees of freedom, acting along \textit{local coordinate} $x_l$. When the element needs to be expressed in terms of global coordinates, the original $2 \times 2$-stiffness matrix is usually expanded into a $4 \times 4$ stiffness matrix $K_g$ by multiplication with a transformation matrix $T$, which is a function of the relative rotation $\beta$ of the local element coordinates with respect to the global coordinate system. The result is a $4 \times 4$ stiffness matrix:

$$K_l = \frac{EA}{L_0} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad T = \begin{bmatrix} c & s & 0 \\ s & 0 & 0 \\ 0 & 0 & c & s \end{bmatrix}, \quad c = \cos \beta, s = \sin \beta \quad (A-26)$$

$$\Rightarrow K_g = T^T K_l T = \frac{EA}{L_0} \begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix} \quad (A-27)$$
For a linear analysis, the angle $\beta$ is evaluated once, thus fixing the coordinates in which the local deformations are described. This means that every deformation/strain in the element will be aligned with the originally evaluated coordinate direction! Although the nodes can move freely inside the global coordinates $(x_g, z_g)$, no updating of $\beta$ as a function of these global displacements takes place.

The statement above is not always realised by engineers that use FEM-software.\(^1\) It is one of the core ingredients that makes linear analysis such easy, both from computational perspective and for comprehension. However, the consequence is that it is impossible to include the effect of geometrical reorientation due to loading into the FEM-model.

The essence of geometrical nonlinear analysis is that the effect of reorientation is taken into account. For the bar element, there are a couple of methods possible, including the use of co-rotational elements and rotated strain descriptions. Crisfield [4, chap. 3] treats these approaches into detail. Here the rotated engineering strain approach is used.

**Rotated Engineering Strain Bar Element**

The engineering strain in the linear (1 dof) bar model is defined as:

$$\varepsilon = \frac{\partial u}{\partial x}.$$  \hspace{1cm} (A-28)

Let us now introduce a second local displacement direction $w = [w_1 \ w_2]^\text{T}$, orthogonal to $u$. If an arbitrary displacement would have a component in $w$, the original strain description (A-28) would no longer be valid. Instead, the strain would include a term in $w$. For example:

$$\varepsilon_G = \frac{\partial u}{\partial x} + \frac{1}{2} \left( \frac{\partial w}{\partial x} \right)^2$$  \hspace{1cm} (A-29)

The derivation of this second order strain term will not be treated here. It is based on Green strain (See [4, eq 4.84, p. 119]). For now, the somewhat easier engineering strain will be used. Expressing the engineering strain in terms of the bar’s initial and final length:

\(^1\)For me, it took some time to realise the consequences in its full extend.
\[ \varepsilon_E = \frac{L_n - L_0}{L_0} = \frac{a_n - a_0}{a_0}, \tag{A-30} \]

where \( L_n \) can be written in terms of \( u \) and \( w \), the strain is now also a function of displacements orthogonal to its normal axis. \( a_0 = L_0/2 \) And \( a_n = L_n/2 \), Denoting half the length of the bar in its initial and final state. The terms in (A-30) now have to be expressed in terms of the nodal displacements and the initial positions. First the initial position \( \mathbf{x} \) and the displacement vector \( \mathbf{p} \) are defined:

\[
\mathbf{x} = \begin{bmatrix} x_1 \\ z_1 \\ x_2 \\ z_2 \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} u_1 \\ w_1 \\ u_2 \\ w_2 \end{bmatrix} \tag{A-31}
\]

Note that these definitions differ from [4], but only because \( z_1 \) and \( x_2 \) \((w_1 \text{ and } u_2)\) have changed positions.

Next, the element length is written in terms of its components in the local coordinates. Consider the initial length:

\[
L_0^2 = 4a_0^2 = \mathbf{x}_1^T \mathbf{x}_2, \quad \mathbf{x}_1 = [x_2 - x_1 \ z_2 - z_1]^T
\]

The length \( a_0 \) can be written in terms of absolute coordinates using the following matrix equation:

\[
a_0^2 = \frac{1}{4} \begin{bmatrix} x_2 - x_1 \\ z_2 - z_1 \end{bmatrix}^T \begin{bmatrix} x_2 - x_1 \\ z_2 - z_1 \end{bmatrix} = \begin{bmatrix} x_1 \\ z_1 \\ x_2 \\ z_2 \end{bmatrix}^T \begin{bmatrix} -1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ z_1 \\ x_2 \\ z_2 \end{bmatrix}
\]

\[
= \frac{1}{4} \mathbf{x}^T \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \mathbf{x}. \tag{A-33}
\]

The same reasoning yields a similar expression for the final length in terms of \( \mathbf{x}' = \mathbf{x} + \mathbf{p} \):

\[
\frac{1}{4} \mathbf{x}'^T \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \mathbf{x}' = \mathbf{x}^T \mathbf{A} \mathbf{x} \tag{A-35}
\]

With the known equations for \( a_0 \) and \( a_n \), the strain is determined. The next step is to obtain the internal force vector \( \mathbf{q} \). From the principle of virtual work, the internal force is defined as:

\[ ^2 \text{Note that here } \mathbf{q} \text{ denotes force, whereas throughout the rest of the report it is used to denote generalised displacements. The reason here is to maintain maximum ‘symbolic compatibility’ with [4], in case the reader wants more detail. (The book [4] is complex enough in its own right; making things even more complicated would be unwise!)} \]

R.J. Dedden  
Master of Science Thesis
\[ q = \int \sigma \frac{\partial \varepsilon}{\partial p} \, dV_0, \quad (A-36) \]

with \( V_0 \) the initial volume. The equation immediately calls for \( \frac{\partial \varepsilon}{\partial p} \) to be determined:

\[ \frac{\partial \varepsilon}{\partial p} = \frac{1}{a_0} \frac{\partial a_n}{\partial p} = \frac{1}{a_0} \frac{\partial x'^T A x'}{\partial p} = \frac{1}{a_0 a_n} Ax' = \frac{1}{4a_0 a_n} c(x') \quad (A-37) \]

Introducing the inverse of the elongation \( \lambda = a_0/a_n \), the internal force is written concisely as:

\[ q = 2a_0 A_0 \sigma \frac{\partial \varepsilon}{\partial p} = \lambda \sigma A_0 \frac{\partial a_n}{\partial p} c(x'). \quad (A-38) \]

The tangential stiffness matrix is obtained through differentiation of \( q \) with respect to the displacements:

\[ \tilde{K} = \frac{\partial q}{\partial p} = \frac{A_0}{2a_n} c(x') \frac{\partial \sigma}{\partial p} + \frac{\sigma A_0}{2a_n} \frac{\partial c(x')}{\partial p} - \frac{\sigma A_0}{2a_n^2} c(x') \frac{\partial a_n}{\partial p}, \quad (A-39) \]

The first term is found by noting that \( \frac{\partial \sigma}{\partial p} = E \frac{\partial \varepsilon}{\partial p} \):

\[ K_{\sigma 1} = \frac{E A_0}{8a_n^2} c(x') c(x')^T = \frac{E A_0 \lambda^2}{8a_n^2} c(x') c(x')^T. \quad (A-40) \]

The second stiffness term follows from the fact that \( \frac{\partial c(x')}{\partial p} = 4A \):

\[ K_{\sigma 2} = \frac{2\sigma A_0 \lambda}{a_0} A \quad (A-41) \]

The last term is easily obtained using (A-37):

\[ K_{\sigma 2} = -\frac{\sigma A_0 \lambda}{8a_n^2} c(x') c(x')^T \quad (A-42) \]

To finalise the finite element description, a few remarks are made:

- The second and third term in the stiffness matrix depend the stress \( \sigma \). If \( \sigma = 0 \), these two terms are also 0. Thus these two matrices are involved in the geometrical ‘updating’ process described at the beginning of this section.
- Although the second term depends on \( \sigma \) and \( \lambda \), it is independent on the orientation: It is not a function of \( c(x') \).
- The first term is also dependent on the displacements (through \( \lambda \) and \( c(x') \)), but it is not zero for rigid body modes/zero displacements. This first term should be isolated in order to extract the linear stiffness content from the bar element.
- The Matlab code for the bar element can be found in Appendix B-3-1. This code will not be detailed here, since it straightforwardly implements the steps of this section. The only thing one should notice, is that \( \sigma \) is not defined explicitly, but written as \( \varepsilon E \) instead.
Appendix B

Matlab Code
Listings

B.1 deim.m .................................................. 107
B.2 dynamic_example_4dof.m (I) .................................. 108
B.3 dynamic_example_4dof.m (II) .................................. 115
B.4 bar.m .................................................. 117
B.5 beam1_int_forces.m ....................................... 118
B.6 beam1_mass_matrix.m .................................... 120
B.7 reassemble.m ............................................ 121
B.8 find_elements.m ......................................... 123
B.9 find_elements_as.m ....................................... 123
B-1 DEIM Algorithm Implementation

Listing B.1: deim.m

1 function [phi,Uu,P] = deim(U)
2     n = size(U,1);
3     m = size(U,2);
4
5     [rho phi_1] = max(abs(U(:,1)));
6     Uu = U(:,1);
7     P = zeros(n,1); P(phi_1,1) = 1;
8     phi = phi_1;
9
10    for l = 2:m
11       c = (P.'*Uu)\(P.'*U(:,l));
12       r = U(:,l) - Uu*c;
13       [rho phi_l] = max(abs(r));
14       Uu = [Uu U(:,l)];
15       P_l = zeros(n,1); P_l(phi_l,1) = 1;
16       P = [P P_l];
17       phi = [phi; phi_l];
18    end
19 end
B-2 1 D Mass-spring Example

Some parts of this file are not shown. Only the parts used for the results in Section 6-2-1 are shown.

**Listing B.2:** dynamic_example_4dof.m (1)

```matlab
1 close all;
2 clear all
3 H = struct;  % Plot & Figure handle struct
4 LW = 0.5;
5 MK = {'-o'};
6
7 analysis = [...
8 1;  % linear system
9 1;  % full NL system
10 0;  % DEIM on [LIN+NL]
11 0;  % DEIM on [NL]
12 0;  % DEIM on [LIN+NL], q-red
13 0;  % DEIM on [NL], q-red
14 0;  % DEIM on [LIN+NL], q-red, GSVD(F_s,q_s)
15 0;  % DEIM on [LIN+NL], no K-red
16 ];
17
18 syms m k1 k3 real
19 syms u u1 u2 u3 u4 real
20 syms ud ud1 ud2 ud3 ud4 real
21 syms udd udd1 udd2 udd3 udd4 real
22 syms g t c
23
24 % no. basis vectors in U and V:
25 n_U = 1;
26 n_q = n_U;
27 %
28 %%% I. Set up energies
29 %
30 %%%=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
31
32 u = [u1; u2; u3; u4];
33 ud = [ud1; ud2; ud3; ud4];
34 udd = [udd1; udd2; udd3; udd4];
35
36 %---- write u_ik = U_ik(qs,t) defining the relation between absolute displacement and d.o.f.
37 % then compute absolute velocities
38 % Note: we assume absolute angles here for the second pendulum
39
R.J. Dedden
Master of Science Thesis
% ud = jacobian(u,thet)*thetd ;

% compute energies according to their definitions

V1 = 1/2 * k1 * u1^2 + 1/4 * k3 * u1^4;
V2 = 1/2 * k1 * (u2-u1)^2 + 1/4 * k3 * (u2-u1)^4;
V3 = 1/2 * k1 * (u3-u2)^2 + 1/4 * k3 * (u3-u2)^4;
V4 = 1/2 * k1 * (u4-u3)^2 + 1/4 * k3 * (u4-u3)^4;
V5 = 1/2 * k1 * u4^2 + 1/4 * k3 * u4^4;
V6 = 1/2 * k1 * (u4-u1)^2 + 1/4 * k3 * (u4-u1)^4;

V = V1 + V2 + V3 + V4 + V5 + 0 * V6;

T1 = 1/2 * m * u1^2;
T2 = 1/2 * m * u2^2;
T3 = 1/2 * m * u3^2;
T4 = 1/2 * m * u4^2;

T = T1 + T2 + T3 + T4;

% II. Build Lagrange equations

%%% D(dT/dqd)/Dt - dT/dq + dV/dq - Q = 0

-% compute dV/dq

dV_du = simple(jacobian(V,u)).';

-% compute d/dt(dT/dqd)

dT_dud = simple(jacobian(T,ud)).';

DdT_Dtdud = simple(jacobian(dT_dud,t)) +
    simple(jacobian(dT_dud,u))*ud +
    simple(jacobian(dT_dud,ud))*udd;

-% compute dT/dq

dT_du = simple(jacobian(T,u)).';

-% Set up equations

Q = [g; 0; 0; 0];
Qd = c*[ 1 -1 0 0; -1 2 -1 0; 0 -1 2 -1; 0 0 -1 1]*ud;

Equations = DdT_Dtdud - dT_du + dV_du - Q + Qd ;

%%%
II. Computing linearized mass and stiffness matrices

Note: this is only correct for systems WITH NO OVERALL MOTION

\[
M = \text{simple}(\text{jacobian}(dT_dud, ud)); \quad \text{Mass matrix}
\]
\[
K = \text{simple}(\text{jacobian}(dV_du, u)); \quad \text{Stiffness matrix}
\]
\[
C = c \ast \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}; \quad \text{damping matrix}
\]
\[
K_{\text{eq}} = \text{my_subs}(K, [u1 u2 u3 u4], [0 0 0 0])
\]
\[
K_{\text{lin}} = \text{my_subs}(K, k3, 0)
\]
\[
K_{\text{nl}} = \text{simple}(K - K_{\text{lin}})
\]

% == Linear analysis ==

% Done to find the (linear) eigenfrequencies. Based on this, the time step is chosen

M_num = double(\text{my_subs}(M, m, m_num));
K_{\text{lin num}} = double(\text{my_subs}(K_{\text{lin}}, k1, k_1));
[X, Omega2] = \text{eig}(K_{\text{lin num}}, M_{\text{num}});
Omega2, k] = \text{sort}(\text{real}(\text{diag}(\text{Omega2})))
X = \text{real}(X(:,k));
clear k

freq = sqrt(Omega2) / (2 * pi);
T_{\text{vib}} = 1 / freq;
%% Time Integration Parameters
beta = 1/4;
gamma = 1/2;
h = 3*10^(floor(log10(min(T_vib)/10)))
tol = 1e-9;
tol_S = 1e-18;
lambda = 0:h:t_end;
N = length(lambda);
n_max = 50;

%% Linear Dynamics Analysis
if analysis(1)
disp('Linear Analysis: q_lin');
q_lin = zeros(n_dof,N);
F_lin = zeros(n_dof,N);

% First estimate (Initialisation)
qdd_n = M_num\(Q-K_lin_num*q_0);
qd_n = qd_0;
q_n = q_0;
for i = 1:length(lambda)
    % G = lambda(i)*Q;
    % Predictor
    qd_n1 = qd_n + (1-gamma)*h*qdd_n;
    q_n1 = q_n + h*qd_n + (0.5-beta)*h^2*qdd_n;
    % Stepping matrix
    S = M_num + h^2*beta*K_lin_num + h*gamma*C;
    qdd_n1 = S\(Q-K_lin_num*q_n1 - C*qd_n1);
    % Correction
    qd_n1 = qd_n1 + h*gamma*qdd_n1;
    q_n1 = q_n1 + h^2*beta*qdd_n1;
    q_n = q_n1;
    qd_n = qd_n1;
    qdd_n = qdd_n1;
    q_lin(:,i) = q_n1;
    F_lin(:,i) = K_lin_num*q_n1;
end

clear q qd qdd S
grid on
plot(lambda,q_lin(2,:),strcat(MK{:},'g'),'LineWidth',LW);
subplot(223), hold on
xlabel('$\lambda$','interpreter','none')
ylabel('$q_3$','interpreter','none')
grid on
plot(lambda,q_lin(3,:),strcat(MK{:},'g'),'LineWidth',LW);
subplot(224), hold on
xlabel('$\lambda$','interpreter','none')
ylabel('$q_4$','interpreter','none')
grid on
plot(lambda,q_lin(4,:),strcat(MK{:},'g'),'LineWidth',LW);
figure(2); hold on
subplot(221), hold on
xlabel('$\lambda$','interpreter','none')
ylabel('$f_1$','interpreter','none')
grid on
plot(lambda,F_lin(1,:),strcat(MK{:},'g'),'LineWidth',LW);
subplot(222), hold on
xlabel('$\lambda$','interpreter','none')
ylabel('$f_2$','interpreter','none')
grid on
plot(lambda,F_lin(2,:),strcat(MK{:},'g'),'LineWidth',LW);
subplot(223), hold on
xlabel('$\lambda$','interpreter','none')
ylabel('$f_3$','interpreter','none')
grid on
plot(lambda,F_lin(3,:),strcat(MK{:},'g'),'LineWidth',LW);
subplot(224), hold on
xlabel('$\lambda$','interpreter','none')
ylabel('$f_4$','interpreter','none')
grid on
plot(lambda,F_lin(4,:),strcat(MK{:},'g'),'LineWidth',LW);
leg_conts = {'linear'};
end

%% ======================================================================
%% % Full Non-linear Analysis %
if analysis
    disp('Full NL Analysis: q_s , F_s');
    q_s = zeros(n_dof,N);
    F_s = zeros(n_dof,N);
    % First estimate (Initialisation)
    qdd_n = M_num\(Q-f_0);
    qd_n = qd_0;
    q_n = q_0;
    for i = 1:length(lambda)
        % Define "xx_n" and "xx_n1" in order to prevent recursive evaluations
        qd_n1 = qd_n + (1-gamma)*h*qdd_n;
        ...
q_n1 = q_n + h*qd_n + (0.5-beta)*h^2*qdd_n;
qdd_n1 = zeros(n_dof,1);

% Initialise for each i
n = 0;
err = 1/tol;
f = double(my_subs(F,u,q_n1));
Kt_num = double(my_subs(Kt,u,q_n1));
r = M_num*qdd_n1 + C*qd_n1 + f - Q;
while norm(err) > tol
    n = n + 1;
    % Stepping matrix
    S = 1/(h^2*beta)*M_num + Kt_num + gamma/(h*beta)*C;
    Dq = S\-r;
    q_n1 = q_n1 + Dq;
    qd_n1 = qd_n1 + (gamma/(beta*h))*Dq;
    qdd_n1 = qdd_n1 + (1/(beta*h.^2))*Dq;
    c = rcond(S);
    if c < tol_S
        disp('## Badly conditioned stepping matrix:');
        disp('');
        disp(['rcond = ',num2str(c),',']);
        disp('Aborting...');
        break
    end
    if n > n_max
        disp('## Bad convergence!');
        disp('Consider smaller time step!');
        break
    end
    f = double(my_subs(F,u,q_n1));
    Kt_num = double(my_subs(Kt,u,q_n1));
    r = M_num*qdd_n1 + C*qd_n1 + f - Q;
    err = norm(r) / norm(f);
end
q_s(:,i) = q_n1;
F_s(:,i) = f;

% Push states forward in shift registry
q_n = q_n1;
qd_n = qd_n1;
qdd_n = qdd_n1;
end
clear q_n qd_n qdd_n q_n1 qd_n1 qdd_n1 S

figure(1); hold on
subplot(221), hold on
plot(lambda,q_s(:,1,:),strcat(MK{:},'b'),'LineWidth',LW);
subplot(222), hold on
plot(lambda,q_s(:,2,:),strcat(MK{:},'b'),'LineWidth',LW);
subplot(223), hold on
plot(lambda,q_s(3,:),strcat(MK{:},'b'),'LineWidth',LW);

subplot(224), hold on
plot(lambda,q_s(4,:),strcat(MK{:},'b'),'LineWidth',LW);

figure(2); hold on
subplot(221), hold on
plot(lambda,F_s(1,:),strcat(MK{:},'b'),'LineWidth',LW);
subplot(222), hold on
plot(lambda,F_s(2,:),strcat(MK{:},'b'),'LineWidth',LW);
subplot(223), hold on
plot(lambda,F_s(3,:),strcat(MK{:},'b'),'LineWidth',LW);
subplot(224), hold on
plot(lambda,F_s(4,:),strcat(MK{:},'b'),'LineWidth',LW);

leg_conts = [leg_conts 'full dof [L+NL]'];

%% ======================================================================
%% % DEIM on snapshots of forces
%% % For analysis 3,
316 [U_f,S_f,~] = svd(F_s);
317 [phi,U_f_red,P] = deim(U_f(:,1:n_U));
A = U_f_red*(P.'*U_f_red)

%% For analysis 4,6
321 F_s_nl = F_s - K_lin_num*q_s;
322 [U_f_nl,S_f_nl,~] = svd(F_s_nl);
323 [phi_nl,U_f_red_nl,P_nl] = deim(U_f_nl(:,1:n_U));
A_nl = U_f_red_nl*(P_nl.*U_f_red_nl)

%% For analysis 5,
329 [U_q,S_q,~] = svd(q_s);
330 U_q_red = U_q(:,1:n_q);

%% For analysis 7,
333 [UG_f,UG_q,XG,CG,SG] = gsvd(F_s,q_s);
334 [phiG,UG_f_red,PG] = deim(UG_f(:,1:n_U));
A_G = UG_f_red*(PG.'*UG_f_red)

figure(3);
semilogy(diag(S_f),'o');
title('Singular Values of F_s')
xlabel('$m$','Interpreter','none')
ylabel('$\sigma_{m}$','Interpreter','none')
grid on

figure(4);
semilogy(diag(S_q),'o');
title('Singular Values of Q_s')
xlabel('$m$','Interpreter','none')
Listing B.3: dynamic_example_4dof.m (II)

618  %%=====================================================================
619  %%
620  %% DEIM with separated LIN and NL part, with DISPLACEMENT REDUCTION
621  if analysis
622     disp('DEIM on [NL], q-red: q_s_d4, F_s_d4');
623     q_s_d4 = zeros(n_dof,N);
624     F_s_d4 = zeros(n_dof,N);
625     M_red_num = U_q_red.'*M_num*U_q_red;
626     C_red = U_q_red.'*C*U_q_red;
627     Q_red = U_q_red.'*Q;
628     %Qd_red = U_q_red.'*Qd;
629     % First estimate (Initialisation)
630     qdd_n = M_red_num\(Q_red-U_q_red.'*f_0);
631     %qdd_n = U_q_red.'*(M_num\(Q - f_0));
632     qd_n = U_q_red.'*q_0;
633     q_n = U_q_red.'*q_0;
634     for i = 1:length(lambda)
635         % Define "xx_n" and "xx_n1" in order to prevent recursive evaluations
636         qd_n1 = qd_n + (1-gamma)*h*qdd_n;
637         q_n1 = q_n + h*qd_n + (0.5-beta)*h^2*qdd_n;
638         qdd_n1 = zeros(n_q,1);
639         n = 0;
640         q_n1_f = U_q_red*q_n1;
641         Kt_num = U_q_red.'*(K_lin_num + A_nl*double(my_subs(K_nl,[k1 k3 u.'],[k_1 k_3 q_n1_f.'])))*U_q_red;
642         f = U_q_red.'*(K_lin_num*q_n1_f + A_nl*double(my_subs(F_nl,[u,q_n1_f])));
643         %C_num = U_q_red.'*double(my_subs(C,u,q_n1))*U_q_red;
644         r = M_red_num*qdd_n1 + C_red*qd_n1 + f - Q_red;
645         err = norm( r ) / norm( f );
646         while norm( err ) > tol
647             n = n + 1;
648             % Stepping matrix
649             S = 1/(h^2*beta)*M_red_num + Kt_num + gamma/(h*beta)*C_red;
650             Dq = S\-r;
651             q_n1 = q_n1 + Dq;
652         end
653     end
654  end
\[ q_{d1} = q_{d1} + \left( \frac{\gamma}{\beta \cdot h} \right) \cdot Dq; \]
\[ q_{dd1} = q_{dd1} + \left( \frac{1}{\beta \cdot h^2} \right) \cdot Dq; \]
\[ c = rcond( S ); \]
\[ \text{if } c < \text{tol}_S; \]
\[ \text{disp('## Badly conditioned stepping matrix:')} \]
\[ \text{disp(['rcond = ', num2str(c),'],')} \]
\[ \text{disp(' Aborting...')} \]
\[ \text{break} \]
\[ \text{end} \]
\[ \text{if } n > n_{\text{max}}; \]
\[ \text{disp('## Bad convergence!')} \]
\[ \text{disp('Consider smaller time step!')} \]
\[ \text{break} \]
\[ \text{end} \]
\[ q_{n1,f} = U_{q_red} \cdot q_{n1}; \]
\[ K_{t_num} = U_{q_red} \cdot (K_{\text{lin}_num} + A_{nl} \cdot \text{double(my_subs(K_{nl}, [k1 k3 u] \cdot [k_{1,3} q_{n1,f}.]))}) \cdot U_{q_red}; \]
\[ f = U_{q_red} \cdot (K_{\text{lin}_num} \cdot q_{n1,f} + A_{nl} \cdot \text{double(my_subs(F_{nl}, u, q_{n1,f}.))))}; \]
\[ \%C_{num} = U_{q_red} \cdot \text{double(my_subs(C, u, q_{n1}))} \cdot U_{q_red}; \]
\[ r = M_{\text{red}_num} \cdot q_{dd1} + C_{\text{red}} \cdot q_{d1} + f - Q_{red}; \]
\[ \text{err} = \text{norm}(r) / \text{norm}(f); \]
\[ \text{end} \]
\[ q_{s_d4}(:, i) = U_{q_red} \cdot q_{n1}; \]
\[ F_{s_d4}(:, i) = U_{q_red} \cdot f; \]
\[ \% Push states forward in shift registry \]
\[ q_n = q_{n1}; \]
\[ q_{d1} = q_{d1}; \]
\[ q_{dd1} = q_{dd1}; \]
\[ \text{end} \]
\[ \text{clear } q_n q_{d1} q_{dd1} q_{n1} q_{dd1} S K_{t_num} q_{n1,f} \]
\[ \text{figure}(1); \text{hold on} \]
\[ \text{subplot}(221), \text{hold on} \]
\[ \text{plot}(\lambda, q_{s_d4}(1,:), \text{strcat(MK{:}, 'r'), 'LineWidth', LW}); \]
\[ \text{subplot}(222), \text{hold on} \]
\[ \text{plot}(\lambda, q_{s_d4}(2,:), \text{strcat(MK{:}, 'r'), 'LineWidth', LW}); \]
\[ \text{subplot}(223), \text{hold on} \]
\[ \text{plot}(\lambda, q_{s_d4}(3,:), \text{strcat(MK{:}, 'r'), 'LineWidth', LW}); \]
\[ \text{subplot}(224), \text{hold on} \]
\[ \text{plot}(\lambda, q_{s_d4}(4,:), \text{strcat(MK{:}, 'r'), 'LineWidth', LW}); \]
\[ \text{figure}(2); \text{hold on} \]
\[ \text{subplot}(221), \text{hold on} \]
\[ \text{plot}(\lambda, F_{s_d4}(1,:), \text{strcat(MK{:}, 'r'), 'LineWidth', LW}); \]
\[ \text{subplot}(222), \text{hold on} \]
\[ \text{plot}(\lambda, F_{s_d4}(2,:), \text{strcat(MK{:}, 'r'), 'LineWidth', LW}); \]
\[ \text{subplot}(223), \text{hold on} \]
\[ \text{plot}(\lambda, F_{s_d4}(3,:), \text{strcat(MK{:}, 'r'), 'LineWidth', LW}); \]
\[ \text{subplot}(224), \text{hold on} \]
\[ \text{plot}(\lambda, F_{s_d4}(4,:), \text{strcat(MK{:}, 'r'), 'LineWidth', LW}); \]
\[ \text{leg_conts} = [\text{leg_conts} 'DEIM on [NL], q-red']; \]
B-3 Finite Element Definitions

B-3-1 2D Bar with Rotated Engineering Strain

Listing B.4: bar.m

```matlab
function [q,Kt,q_nl] = ES_bar(mode,node1,node2,displ,prop,n)

EA = prop(1); % Amat = 1/4*[ 1 -1 0 0; -1 1 0 0; 0 0 1 -1; 0 0 -1 1];
Amat = 1/4*[ 1 0 -1 0; 0 1 0 -1; -1 0 1 0; 0 -1 0 1];
x21 = node2(1) - node1(1);
z21 = node2(2) - node1(2);
l0 = sqrt((node1(1,1)-node2(1,1))^2+(node1(1,2)-node2(1,2))^2);

%x0 = [node1 node2].';
%p = [displ(3) - displ(1) displ(4) - displ(2)].';
%xu = [x21; z21] + p;

ln = sqrt(xu.'*xu);

a0 = 10/2;
an = ln/2;
%c = [-xu(1); -xu(2); xu(1); xu(2)];
c = 4*Amat*x;

%epsilon = (an-a0)/a0;
epsilon = 2*a0/(a0+an) + (an.^2-a0.^2)/(2*a0.^2);
%sigma = epsilon*E;

lambda = a0/an;

q = lambda*(EA*epsilon)/(2*a0)*c;

Kt1 = (EA+lambda.^2)/(8*a0.^3)*((c*c.'));
Kt_s1 = ((2*epsilon*EA+lambda)/a0)*Amat;
Kt_s2 = -(epsilon*EA+lambda.^3)/(8*a0.^3)+(c*c.');
Kt = Kt1 + Kt_s1 + Kt_s2;
Kt = norm(Kt)/4*diag(ones(4,1));

q_nl = zeros(4,1);
```

Master of Science Thesis

R.J. Dedden
Listing B.5: beam1_int_forces.m

function varargout = beam1_int_forces(mode, node1, node2, displ, prop, n)

% input: node1=[x1 y1]
% node21=[x2 y2]
% displ= element displacement vector in global coordinates
% prop=[EA; EI] beam sectional properties
% n=number of Gauss integration points

% output: f= beam internal forces

% references: Crisfield pag 204-5

% Gauss integration points and weights (6 points for an exact integration)

switch lower(mode)
    case {0, 'def', 'default'}
        mode = 0;
    case {1, 'deim'}
        mode = 1;
end

U = 0; W = 0; N = 0; M = 0;
U_nl = 0; W_nl = 0; N_nl = 0; N_lin = 0;

f = zeros(6,1);
f_nl = zeros(6,1);

Kuu = zeros(2,2); Kuw = zeros(2,4); Kw = zeros(4,4);
Kuu_nl = zeros(2); Kuw_nl = zeros(2,4); Kw_nl = zeros(4);

if mode == 0
    K = zeros(6);
else
    K_nl = zeros(6);
end

[v(:,1),v(:,2)] = gaussint(n,−1,1,1);

% rotate the displacement vector in the local coordinate frame
l = sqrt((node1(1,1)−node2(1,1))^2+(node1(1,2)−node2(1,2))^2);
dloc = displ;
u = [dloc(1,1);dloc(4,1)];
w = [dloc(2:3,1);dloc(5:6,1)];
for j = 1:size(v,1)
    xi = v(j,1);
    % scaled shape functions:
    hu = 1/2*|1−xi|;

R.J. Dedden Master of Science Thesis
\[ h_w = \frac{1}{8} [4 - 6 + 2xi \cdot 3; \\
    1 + (xi^2 - 1) \cdot (xi - 1); \\
    4 + 6xi^2 \cdot 3; \\
    1 + (xi^2 - 1) \cdot (xi + 1)]; \]

\[ b_u = \frac{1}{1} [-1; \\
    1]; \]

\[ b_w = \frac{1}{4 - 1} [6 \cdot (xi^2 - 1); \\
    6 \cdot (3xi^2 - 2 - xi - 1); \\
    6 \cdot (1 - xi^2); \\
    6 \cdot (3xi^2 + 2 + xi - 1)]; \]

\[ c = \frac{1}{1 - 2} [6; \\
    1 + (3xi - 1); \\
    -6xi; \\
    1 + (3xi + 1)]; \]

\[ d = 2 \cdot (3 - 3); \\
    1 + 3; \\
    -6; \\
    1 + 3; \]

% Internal forces at Gauss points
\[ N_{nl} = \text{prop}(1) \cdot \left( \frac{1}{2} \right) \cdot (b_w \cdot w)^2; \]
\[ N_{lin} = \text{prop}(1) \cdot (b_u \cdot u); \]
\[ N = N_{lin} + N_{nl}; \]
\[ M = \text{prop}(2) \cdot c \cdot w; \]

% Internal forces at Nodes
\[ U_{nl} = N_{nl} \cdot b_u \cdot v(j, 2) + U_{nl}; \]
\[ U = N \cdot b_u \cdot v(j, 2) + U; \]
\[ W_{nl} = (N \cdot b_w \cdot w \cdot b_w \cdot v(j, 2) + W_{nl}) \cdot \text{Use full "N" here!} \]
\[ W = \left( N \cdot (b_w \cdot w \cdot b_w \cdot v(j, 2) + W); \right. \]

% NL-content of Stiffness matrix
\[ K_{uw_{nl}} = \text{prop}(1) \cdot (b_w \cdot w) \cdot b_u \cdot b_w \cdot v(j, 2) + K_{uw_{nl}}; \]
\[ K_{ww_{nl}} = (\text{prop}(1) \cdot (b_w \cdot w) \cdot b_w \cdot b_w \cdot b_w \cdot v(j, 2) + K_{ww_{nl}}) \cdot v(j, 2) + K_{ww_{nl}}; \]

% Lin. Content of Stiffness matrix
\[ \text{if mode} == 0 \]
\[ K_{uu} = \text{prop}(1) \cdot b_u \cdot b_u \cdot v(j, 2) + K_{uu}; \]
\[ K_{ww} = \text{prop}(2) \cdot c \cdot c \cdot v(j, 2) + K_{ww}; \]
\[ \text{end} \]

% Determine the outputs
\[ \text{if mode} == 0 \]
\[ K_{ww} = K_{ww} + K_{ww_{nl}}; \]
\[ K_{uu} = K_{uu} + K_{uw_{nl}}; \]
\[ K([1 4], [1 4]) = \frac{1}{2} \cdot K_{uu}; \]
\[ K([2 3 5 6], [2 3 5 6]) = \frac{1}{2} \cdot K_{ww}; \]
\[ K([1 4], [2 3 5 6]) = \frac{1}{2} \cdot K_{uw}; \]

Master of Science Thesis

R.J. Dedden
102 \[ \mathbf{K}([2 \ 3 \ 5 \ 6],[1 \ 4]) = \frac{1}{2} \mathbf{K}_{\text{u}}. \]
103 \[ \mathbf{f} = [\mathbf{U}(1);\mathbf{W}(1);\mathbf{U}(2);\mathbf{W}(3);\mathbf{W}(4)] \star \frac{1}{2}; \]
104 \end
105
106 \[ \mathbf{f}_{\text{nl}} = [\mathbf{U}_{\text{nl}}(1);\mathbf{W}_{\text{nl}}(1);\mathbf{W}_{\text{nl}}(2);\mathbf{U}_{\text{nl}}(2);\mathbf{W}_{\text{nl}}(3);\mathbf{W}_{\text{nl}}(4)] \star \frac{1}{2}; \]
107
108 \textbf{if} \ mode =\\= 1
109 \ \textbf{end}
110 \ \textbf{end}
111 \textbf{end}
112 \textbf{else}
113 \textbf{end}
114
115 \textbf{if} \ mode =\\= 0
116 \ \textbf{varargout}(1) = \{f\};
117 \ \textbf{varargout}(2) = \{\mathbf{K}\};
118 \ \textbf{varargout}(3) = \{f_{\text{nl}}\};
119 \ \textbf{else}
120 \ \textbf{varargout}(1) = \{f_{\text{nl}}\};
121 \ \textbf{varargout}(2) = \{\mathbf{K}_{\text{nl}}\};
122 \ \textbf{end}

\textbf{Listing B.6: beam1\_mass\_matrix.m}

1 \textbf{function} \ [\mathbf{M}] = beam1\_mass\_matrix(\text{node1},\text{node2},\text{prop})
2
3 \% Nonlinear Beam Element like "beam1\_int\_forces.m", but with
4 \% mass matrix added for dynamics.
5 \% input: \text{node1}=[x1 \ y1]
6 \% \text{node2}=[x2 \ y2]
7 \% \text{prop}(3) = \rho
8 \%
9 \% output \text{M}= mass matrix
10 \%
11 \% references: Crisfield pag 204-5
12 \%
13 \text{l} = \text{sqrt}((\text{node1}(1,1)-\text{node2}(1,1))^2+(\text{node1}(1,2)-\text{node2}(1,2))^2);
14 \% Mass matrix \text{M} (linear)
15 \text{M} = (\text{prop}(3) \star 1)/420 \star ... 
16 [140, \ 0, \ 0, \ 70, \ 0, \ 0; 
17 \ 0, \ 156, \ 22*1, \ 0, \ 54, \ -13*1; 
18 \ 0, \ 22*1, \ 4*1.\^2, \ 0, \ -13*1, \ -3*1.\^2; 
19 \ 70, \ 0, \ 0, \ 140, \ 0, \ 0; 
20 \ 0, \ 54, \ 13*1, \ 0, \ 156, \ -22*1; 
21 \ 0, \ -13*1, \ -3*1.\^2, \ 0, \ -22*1, \ 4*1.\^2];
B-4  Miscellaneous Functions

B-4-1  Element Assembly Function

Listing B.7: reassemble.m

1  function varargout = reassemble(model, varargin)
2  
3  % Reassemble elementary forces and stiffness matrices into
4  % global vectors / matrices.
5  %=========================================================================
6  % Modes of operation:
7  %=========================================================================
8  % 1) VECTOR MODE
9  % -- On (a single / arrays of) column vector(s) --
10  %
11  % [b,...] = reassemble(model,a,...)
12  % In vector mode, the assembly operation is carried out
13  % only on rows. Hence when the input size is nxm,
14  % the resulting output has dimension
15  % pxm,
16  % with p<n
17  %--------------------------------------------------------
18  % 2) MATRIX MODE
19  % -- On a symmetric matrix --
20  %
21  % [B,...] = reassemble(model,A,'matrix,...)
22  % In matrix mode, the assembly operation is carried out
23  % both on rows and columns. When the input size is nxn,
24  % the resulting output has dimension
25  % pxp,
26  % with p<n
27  %--------------------------------------------------------
28  % WARNING!
29  % The function is not able to check symmetry itself. In
30  % order to make it aware of the matrix mode, the input
31  % string 'matrix' should be passed after each given
32  % matrix input.
33  %=========================================================================
34  % Rob Dedden
35  % Februari 2012
36  %=========================================================================
37  % Unpack structs
38  mdl_data = model.mdl_data;
39  elements = model.elements;
40  
41  if nargin == 1
42     disp('not enough input arguments');
43     return
44  
45  % Determine the number and shape of the input(s)
elseif nargin > 1
    ndof = mdl_data.ndof;
    %varargout = {};
    % loop over the inputs
    i = 1;
    k = 0;
    while i <= nargin-1
        % Increase 'i' whenever varargin{i} is followed by a
        % char type input.
        if ischar(varargin{i})
            i = i + 1;
        else
            % determine input shape
            dim = size(varargin{i});
            I = varargin{i};
            % MATRIX MODE:
            if i < nargin-1 & strcmp(varargin{i+1},'matrix')
                if eq(dim(1),dim(2)) == true
                    O = zeros(ndof);
                    INDEX = int32(horzcat(elements.index));
                    INDEX_UNA = int32(horzcat(elements.index_una));
                    % Assemble the inputs:
                    for p = 1:length(INDEX_UNA);
                        for q = 1:length(INDEX_UNA);
                            O(INDEX(p),INDEX(q)) = O(INDEX(p),INDEX(q))
                                +...
                            I(INDEX_UNA(p),INDEX_UNA(q));
                        end
                    end
                    %
                    i = i + 1;
                    k = k + 1;
                else
                    disp('non-square matrix!');
                    disp('check input no.');
                    disp(num2str(i+1));
                    continue;
                end
                % VECTOR MODE
            else
                O = zeros(ndof,dim(2));
                % Assemble the inputs:
                for j = 1:size(elements,2);
                    index = elements(j).index;  % Assembled indices
                    index_una = elements(j).index_una;  % Unassembled indices
                    O(index,:) = 0(index,:) + I(index_una,:);
                end
                k = k + 1;
            end
            varargout(k) = {O};
            i = i + 1;
        end
    end
B-4 Miscellaneous Functions

B-4-2 Assembled DEIM Element Find Function

Listing B.8: find_elements.m

```matlab
function [deim_elements] = find_elements(elements, n_elem, dofs, phi)
% Function that finds the finite elements that correspond to the
% indices selected by the DEIM algorithm. This function is used
% to deal with the mismatch between dofs used in the DEIM-algorithm
% and the nodal dofs in the finite element model.

k = 0;
% Find indices of free unassembled dofs in model.mdl_data.freedofs_una
[TF] = find(ismember(horzcat(elements.index), dofs));
% Select, out of the above set, the deim dofs subset
deim_dofs = TF(phi);

for i = 1:n_elem
    %B = logical(find(ismember(phi, elements(i)), 1, 'first'));
    B = ~isempty(find(ismember(elements(i).index_una, deim_dofs), 1, 'first'));
    % Note: find stops at the first positive (nonzero) result. This
    % should
    % increase speed.
    if B == true
        k = k + 1;
        % append element to set of deim_elements
        deim_elements(k) = i;
    end
end
```

B-4-3 Unassembled DEIM Element Find Function

Listing B.9: find_elements_as.m

```matlab
function [deim_elements] = find_elements_as(elements, n_elem, dofs, phi)
% Function that finds the finite elements that correspond to the
% indices selected by the DEIM algorithm. This function is used
% to deal with the mismatch between dofs used in the DEIM-algorithm
% and the nodal dofs in the finite element model.

k = 0;
% Find indices of free unassembled dofs in model.mdl_data.freedofs_una
% [TF] = find(ismember(horzcat(elements.index), dofs));
% Select, out of the above set, the deim dofs subset
deim_dofs = dofs(phi);

for i = 1:n_elem
    %B = logical(find(ismember(phi, elements(i)), 1, 'first'));
end
```
16    B = isempty(find(ismember(elements(i).index, deim_dofs),1,’first’));
17    % Note: find stops at the first positive (nonzero) result. This should
18    % increase speed.
19    if B == true
20        k = k + 1;
21        % append element number to set of deim_elements
22        deim_elements(k) = i;
23    end
24 end
25
26 % ||
27 % ||
28 % \||//
29 % \||//
30 % \|
Bibliography


List of Acronyms

CAD      Computer Aided Design
CDC      Control Data Corporation
DEIM     Discrete Empirical Interpolation Method
dof      Degree Of Freedom
FEA      Finite Element Analysis
FEM      Finite Element Method
GSVD     Generalised Singular Value Decomposition
IFS      Interpolation of Function Snapshots
MOR      Model Order Reduction
MPE      Missing Point Estimation
PDE      partial differential equation
POD      Proper Orthogonal Decomposition
SVD      Singular Value Decomposition