Improved convergence of forward and inverse finite element models

ENG5041P

Preslav Aleksandrov - 2248797A

A project presented for the degree of
MEng Civil Engineering

School of Engineering
University of Glasgow
Scotland
2020/2021

Abstract

Forward and inverse models are used throughout different engineering fields to predict and understand the behaviour of systems and to find parameters from a set of observations. These models use root-finding and minimisation techniques respectively to achieve their goals. This paper introduces improvements to these mathematical methods to then improve the convergence behaviour of the overarching models when used in highly non-linear systems. The performance of the new techniques is examined in detail and compared to that of the standard methods. The improved techniques are also tested with FEM models to show their practical application. Depending on the specific configuration of the problem, the improved models yielded larger convergence basins and/or took fewer steps to converge.

Contents

List of Figures

List of Tables

Acknowledgements

Thanks to my advisor Dr. Ankush Aggarwal, whose help, guidance and input enabled me to produce this paper.

Chapter 1

Introduction

1.1 Background

Mathematical modelling is an important pillar of engineering and science. It allows one to define the behaviour of various systems in terms of governing equations, such as the balance of energy, balance of momentum, compatibility and so on. Except in a few special cases, like a statically determined truss, these governing equations cannot be solved analytically, and computational techniques are necessary to find a solution. Mathematical models can be broadly divided into two types: forward models, which one can use to predict the behaviour of a system, and inverse models, which can be used to find certain system properties.

Constructing a forward model is done by utilising governing equations together with initial and boundary conditions to predict the behaviour of a system. A simple example of the use of forward models is finding the deflection of a beam at a certain location along its length. For this, one must have the beam's material and section properties, its loading and support configuration, a constitutive model (stress-strain relation) and governing equations, such as those provided by the Euler–Bernoulli bending theory.

Inverse models, as the name suggests, do the opposite. They use governing equations together with observations of the system's behaviour to determine some property of a system. Using the beam example, one can create an inverse model to find the modulus of the beam material, given multiple observations of the beam's bending behaviour and the same system of equations used in the forward model example.

Even though the difference in the two cases may seem subtle, two different mathematical methods underlay their solutions. These methods are, in the case of forward modelling, root-finding and, in the case of inverse modelling, minimisation. These two methods will be explored in depth through this paper and various solution procedures will be investigated.

1.1.1 Available solutions

There are many ways to solve root-finding and minimisation problems. According to Nocedal and Wright [1], there are multiple families of methods which can be used. They include Newtonian methods like the BFGS method, Conjugate Gradient methods like the Fletcher–Reeves method, and Trust-Region methods such as the Levenberg–Marquardt method. All of them have particular pros and cons such as differences in computational load, memory usage and so on. The particular focus of this paper will be on two algorithms: the Newton-Raphson (NR) method for root-finding and the Gauss-Newton (GN) method for minimisation.

These methods are popular due to their simplicity and convergence behaviour [2] [3]. However, there are configurations in which these methods perform sub-optimally. Specifically, both methods undergo performance losses when dealing with highly non-linear systems or when a starting guess, which is required to initialise the methods, is taken too far away from the solution of the system [4].

A particular example of when a forward model, using NR as the root-finding algorithm, undergoes performance losses is when the object being modelled is made from a material which exhibits hyper-elasticity [4]. Such a material can be biological soft tissues which usually undergo large strains with small applied stresses [5] or it could be a polyurethane rail pad which exhibits similar behaviour [6]. Both of these materials share the same material model, which is described in more detail in subsection 4.4.1.

When solving an inverse model, which utilises the same hyper-elastic material, with the GN algorithm, performance loss will also be experienced. This is due to the fact that the direction used by GN to determine the correct parameter value is a low-order approximation of the most optimal one. Improving the performance of GN and NR in those highly non-linear systems will be the topic of this paper.

1.1.2 Literature survey

This paper builds on the work done by Aggarwal & Pant [7], which shows how the non-linearity of a function can be reduced to improve the convergence behaviour of root-finding algorithms. Here, an alternate multivariate derivation of their method will be proposed. The paper uses general information about the NR and GN methods from Nocedal & Wright [1], and Süli & Mayers [8]. The derivation of non-linear FEM was based on the one given by Belytschko et. al [9]. Two data sets were used for testing from Martins et. al [10] and Szurgott & Jarzebski [6].

1.2 Aims and Objectives

The main objective of this paper is to improve the convergence of forward and inverse finite element models through the improvement of their underlying computational solution algorithms. In order to do this, this paper will firstly familiarise the reader with basic concepts of iterative root-finding and minimisation methods by providing definitions and showing some basic examples. Next, the paper will focus on two modifications, one for each method, which can improve the performance of NR and GN when dealing with highly non-linear systems and finally, this paper will show the practical application of these algorithms in forward and inverse finite element models.

1.3 Outlines

This project is broken down into three main parts, prefaced by this introduction and followed by a discussion and conclusion chapter. The three main parts are as follows:

- **Root-Finding**
  In this section, the paper will present some of the basic concepts of root-finding and provide a simple example in the univariate space. Afterwards, the paper will cover the derivation of the proposed modification and the testing methodology will be shown. It will also explain how the results should be read and show which test functions were used and why. Finally, a comparison will be made between the proposed improvement and the existing method by presenting the results attained from testing.

- **Minimisation**
  Here, some basic concepts of minimisation will be covered. Then a simple example of linear-least squares minimisation will be shown. Afterwards, the derivation of the improvement will be presented with explanations of the different steps taken. Then the methods of testing will be covered and the set of sample functions will be shown together with their true values used to generate the test data. Finally, a comparison will be made between the proposed improvement and the existing method by presenting the results obtained from testing.

- **Finite Element Methods**
  A brief description of finite element methods will be given. Then, the derivation of a non-linear finite element system will be shown. Finally, the paper will present a comparison between results obtained by using both conventional and improved methods.

1.4 Python and NumPy

The majority of calculations and experiments presented in this paper were done with Python 3 (3.8.3) and an additional package called NumPy (1.19.1) [11]. All random numbers were generated using Latin hyper-cube sampling and the LHS-MDU (1.1) package [12]. The following functions were used to calculate particular elements in the paper:

- Moore-Penrose inverse: numpy.linalg.pinv()
- Matrix rank: numpy.linalg.matrix_rank()
- Vector norm: numpy.linalg.norm()
- Tensor multiplication: numpy.einsum()
- Random variables: lhsmdu.sample()

Chapter 2

Root-finding

2.1 Introduction

2.1.1 Definition

Root-finding is the process of finding values of x which satisfy the following canonical equation:

$$f(x) = 0 \tag{2.1}$$

where $f(x)$ is any generic function and x is the set of inputs it uses. The values of x which satisfy the above are called zeros or roots. Shown in (2.2) is a generic quadratic equation with its output set to zero, where a, b and c are parameters of the equation and $a \neq 0$.

$$ax^2 + bx + c = 0 \tag{2.2}$$

Solving the above equation is by definition root-finding. There are many ways to approach a problem such as this one. However, in this case, the easiest and most common method is to write out the equation's closed-form algebraic expression, a common representation of which is:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2.3}$$

This formula always yields two roots and provides a rigorous method for solving any quadratic equation. However, if the degree of the polynomial is higher than four no closed-form solution is available, as proven by Niels Henrik Abel [13]. To identify the roots of a function that is not suited for numerical evaluation or that does not have a closed-form solution, iterative root-finding algorithms can be utilised. These algorithms always produce approximations of the roots and are not guaranteed to find all available roots. They also have various properties discussed below.

2.1.2 Convergence behaviour

The convergence behaviour of an iterative algorithm shows its ability to produce a convergent sequence. A convergent sequence is any infinite sequence (x_n) whose elements become arbitrarily close to each other as the sequence progresses. That is, all but a finite number of elements from the sequence are within a small distance ϵ from a certain value x^*, which is known as the limit of the sequence. The finite number of elements that lay beyond the predetermined distance ϵ show that there exists another number $\mathcal{N}$ after which the series is considered to have converged. If both x^* and $\mathcal{N}$ exist then the series is said to converge to x^* in $\mathcal{N}$ iterations. This can be mathematically expressed by:

$$(x_n) \to x^* \Leftrightarrow |x_n - x^*| < \epsilon, n > \mathcal{N} \tag{2.4}$$

2.1.3 Tolerance

The value of ϵ from the above can be rewritten in terms of tolerances, allowable amounts of variation, which are used by the algorithm as termination criteria. Those will be, firstly, relative tolerance t_r, i.e. the tolerance between the last two elements of a sequence and secondly, the absolute tolerance t_a i.e. the tolerance between the latest value of a series and a target value x^*.

$$|x_n - x_{n-1}| \leq t_r \tag{2.5}$$

$$|x_n - x^*| \leq t_a \tag{2.6}$$

In most cases, setting stricter tolerances (t_r and t_a set to smaller values than another arbitrary set of t_r and t_a) will increase the number of iterations necessary for the method to converge.

2.1.4 Iterations and Iterative methods

Iterative methods work because of a phenomenon called a fixed point. This is a point at which an arbitrary function's value is equal to its input. This can be expressed as:

$$x = g(x) \tag{2.7}$$

The property of functions to have fixed points is described by Brouwer's Fixed Point Theorem [14]. As any equation of the form $f(x) = 0$ can be rewritten as $x = g(x)$, by setting $g(x) = x + f(x)$, fixed points can occur in any root-finding problem. Although other theorems such as the Contraction Mapping Theorem [8] give further requirements which guarantee convergence, these will not be explored in this paper.

An iteration is a repetition of a process. Each iteration represents one execution of the entire algorithm and produces an output which is used by the subsequent iteration as a starting point. Iterative methods can continue indefinitely until some end criterion is reached. Particularly, in this paper, the termination criteria were defined in terms of tolerances above. Another factor that impacts the number of iterations is the speed at which the algorithm converges, which can be measured with the rate and order of convergence.

2.1.5 Rate and Order of Convergence

The rate and order of convergence of an algorithm are quantities that represent how quickly it approaches its limit. According to Süli & Mayers, there are two ways to calculate the speed at which an iterative algorithm converges [8]. Both ways are defined with respect to the error e_n of a series which will be defined here as:

$$e_n = |x_n - x^*| \tag{2.8}$$

Estimating the rate μ from the above is possible by taking the quotient of two consecutive error terms e_n and e_{n-1} when n is sufficiently large ($n \to \infty$):

$$\lim_{n\to\infty} \frac{e_n}{e_{n-1}} = \mu \tag{2.9}$$

Based on the value of μ, the algorithm's rate can be classified as linear if $\mu \in (0, 1)$, with a smaller rate being faster than a larger rate. In this case, one can also calculate:

$$p = -\log_{10} \mu \tag{2.10}$$

which is called the asymptotic rate of convergence. If $\mu = 0$, then the rate of the convergence is considered super-linear and if $\mu = 1$, then the rate of the convergence is considered sub-linear.

Another way to calculate the rate of convergence of an algorithm is by first finding the order of convergence, which attempts to describe the polynomial behaviour of the algorithm around the root x^*. The base equation for this calculation is similar to (2.9) with the addition of the exponent q which in this case represents the order of an algorithm:

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^q} = \mu \tag{2.11}$$

If the order of convergence is higher, it will take fewer iterations for a series produced by an iterative method to reach its limit. Methods can be classified by their order, for example, a method is said to have linear convergence if $q = 1$. Other classifications are presented below.

$$q = 2 \to quadratic$$
$$q = 3 \to cubic$$

The values of q and μ can be determined both mathematically and numerically from data.

Estimating rate and order

Estimating rate and order of convergence starts with the error definition as before. When n becomes sufficiently large from (2.11), the following is true:

$$e_{n+1} = \mu e_n^q \tag{2.12}$$

The same relation exists for the previous index pair:

$$e_n = \mu e_{n-1}^q \tag{2.13}$$

Taking the quotient of (2.12) and (2.13) gives:

$$\frac{e_{n+1}}{e_n} = \frac{\mu e_n^q}{\mu e_{n-1}^q} \tag{2.14}$$

The factor μ can be cancelled to arrive at:

$$\frac{e_{n+1}}{e_n} = \left(\frac{e_n}{e_{n-1}}\right)^q \tag{2.15}$$

Solving for q gives:

$$q = \frac{\log e_{n+1}/e_n}{\log e_n/e_{n-1}} \tag{2.16}$$

From the above, it can be seen that at least three iterations are needed before the order of convergence can be calculated. With q the rate of convergence μ can be found. This is done by rearranging (2.12) to:

$$\mu = \frac{|e_{n+1}|}{|e_n|^q} \tag{2.17}$$

In this paper, (2.17) and (2.16) will be used for analysis. Using the error definition, another property can be defined.

2.1.6 Basin of Attraction

The basin of attraction is a region of space from which any initial guess will ultimately lead to the same answer, formally known as an attractor x^*. In order for the basin of an attractor to have a positive radius $r > 0$, the following has to be true [15]:

$$\frac{e_{n+1}}{e_n} < 1 \tag{2.18}$$

Using the above and (2.11), one can derive an inequality which will define the basin of attraction of any iterative method, as:

$$\mu e_n^{q-1} < 1 \tag{2.19}$$

In this paper, the basin of attraction will be found numerically, by testing a method across a range of values and recording whether it converged. The paper's focus will be on systems of non-linear functions, which produce very discontinuous basins of attraction.

2.1.7 Non-linearity

A linear function, or otherwise known as a linear map, is one which satisfies the following properties:

$$f(a+b) = f(a) + f(b) \tag{2.20}$$

$$f(aT) = af(T) \tag{2.21}$$

Equation 2.20 demonstrates the principle of superposition, while equation 2.21 shows the principle of homogeneity. If a function does not follow the conditions above, it is considered non-linear. Another definition of a non-linear function is one which possesses a second or higher derivative. One could argue that if a function has a higher-order derivative than another, it is more non-linear. However, as there is no formal definition for the term "highly non-linear", this paper will consider any function $f(x)$ as highly non-linear if following holds true:

$$\frac{d^2 f(x)}{dx^2} \geq \lambda \frac{df(x)}{dx} \tag{2.22}$$

where λ is sufficiently large.

2.2 Existing univariate methods

Two common univariate root-finding methods are shown in this section. The univariate case is used to introduce the main subject of this section which is the Newton-Raphson method.

2.2.1 Newton-Raphson

The Newton-Raphson (NR) method, more commonly known as the Newton method, is an iterative root-finding algorithm. Its derivation begins by writing out the Taylor's series expansion of a function $f(x)$ at x^* where $f(x^*) = 0$. The Taylor's series expansion is as follows:

$$f(x^*) = f(x_n) + f'(x_n)(x_n - x^*) + \frac{1}{2}f''(x_n)(x_n - x^*)^2 + ... \tag{2.23}$$

where $'$ represent derivation with respect to the independent variable, x_n shows the current value of x, and $(.)_n$ is the index of the iteration number. Ignoring all terms after the second one, the following equation is left:

$$f(x^*) = f(x_n) + f'(x_n)(x_n - x^*) = 0 \tag{2.24}$$

One can denote the difference between x_n and x^* as:

$$(x_n - x^*) = \Delta x \tag{2.25}$$

From here, (2.24) can be rearranged to the NR fixed point iteration:

$$f'(x_n)\Delta x = -f(x_n) \tag{2.26}$$

The value of Δx can be found by dividing $-f(x_n)$ by $f'(x_n)$, and then Δx can be used to update the current value of x as:

$$x_{n+1} = x_n + \Delta x \tag{2.27}$$

2.2.2 Secant Method

The secant method (SM) is an iterative root-finding algorithm that uses successive secant lines to a given function to create an approximation of the gradient at the latest point. It can be seen as a finite difference approximation of NR and due to its simplicity, it has been used throughout history to solve equations [16]. The derivation of the secant method begins by choosing two potential roots of a function and evaluating the function at the choices. Afterwards, a secant line is drawn between the points and the x-intercept of the line is taken as the subsequent value in the series of $\{x_n\}$. This is expressed by:

$$x_{n+2} = x_{n+1} - \frac{x_{n+1} - x_n}{f(x_{n+1}) - f(x_x)}f(x_{n+1}) \tag{2.28}$$

which can be rearranged as:

$$\frac{f(x_{n+1}) - f(x_x)}{x_{n+1} - x_n}\Delta x = -f(x_{n+1}) \tag{2.29}$$

In the above, the difference between consecutive elements of x is expressed as $\Delta x = x_{n+2} - x_{n+1}$. The secant method is a unique algorithm when it comes to setting up. In one dimension, $N = 1$, SM needs two guess, $N + 1$, to initialise, where N represents the number of dimensions. This translates to higher dimensions. Hence, using this method in multivariate space becomes unviable due to the large number of initial guesses which need to be taken.

2.2.3 Example of root-finding

To demonstrate how root-finding works practically, an example is given. A typical case for root-finding is solving a third order polynomial such as the one shown below:

$$f(x) = x^3 + x^2 - 2x \tag{2.30}$$

This polynomial yields the curve shown in Fig. 2.1. The function has three roots at $\{-2, 0, 1\}$. In order to find one of the equation's roots, the Newton-Raphson iterative method will be used in this example. At each iteration the method produces an estimation of the root denoted by x_n, where $(.)_n$ shows the number

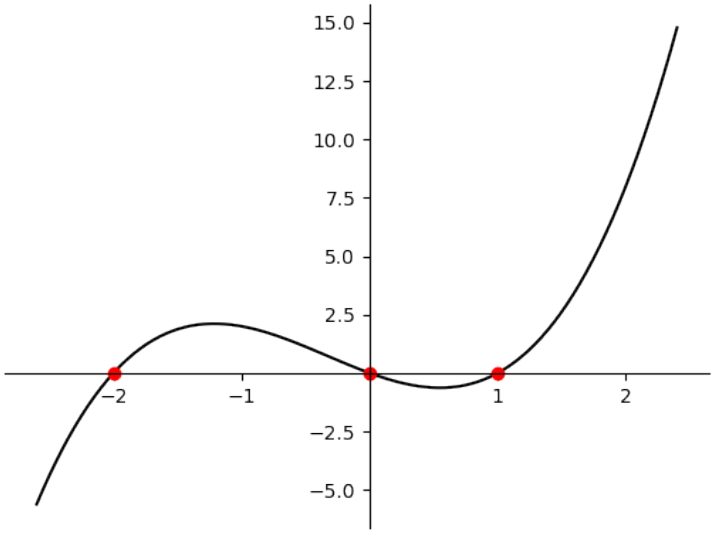

Figure 2.1: A graph of the cubic equation $y = (x-2)x(x+1)$ plotted in the range of [-3,3]. Red markers show the positions of the roots of the equation.

of iterations. The method also evaluates the function and its derivative at each step. The formulation of the Newton-Raphson method (NM) is:

$$f'(x_n)\Delta x = -f(x_n) \tag{2.31}$$

where Δx shows the difference between the current step x_n and the next step x_{n+1}, and f‘ denotes the derivative of f with respect to x. Solving for Δx allows one to find the next iteration value through:

$$x_{n+1} = x_n + \Delta x \tag{2.32}$$

In order to initialise the method, an initial value of x_0 needs to be chosen close to the actual value of the solution x^*. For the case at hand, an initial choice of $x_0 = 2$ was made. This will be used to evaluate the function and its derivative:

$$\begin{aligned} f(x_0) &= 8 \\ f'(x_0) &= 14 \end{aligned} \tag{2.33}$$

Using these values the next iteration of x can be calculated:

$$x_1 = x_0 - \frac{8}{14} = 2 - 0.571 = 1.429 \tag{2.34}$$

Then the process is repeated until the value of Δx becomes less than some previously predetermined tolerance value. Here this was taken as 10^{-3} because all values are reported to three decimal places. Table 2.1 shows the values at each step of the iteration process. The NR method converged in four iterations to the root closest to the starting choice. The steps taken by the method are graphically show in Fig.2.2.

iterations	x_n	$f(x_n)$	$f'(x_n)$	Δx
0	2.000	8.000	18.000	0.571
1	1.429	2.102	6.984	0.301
2	1.128	0.452	4.073	0.111
3	1.017	0.052	3.137	0.017
4	1.000	0.000	3.000	0.000

Table 2.1: Newton-Raphson iterations

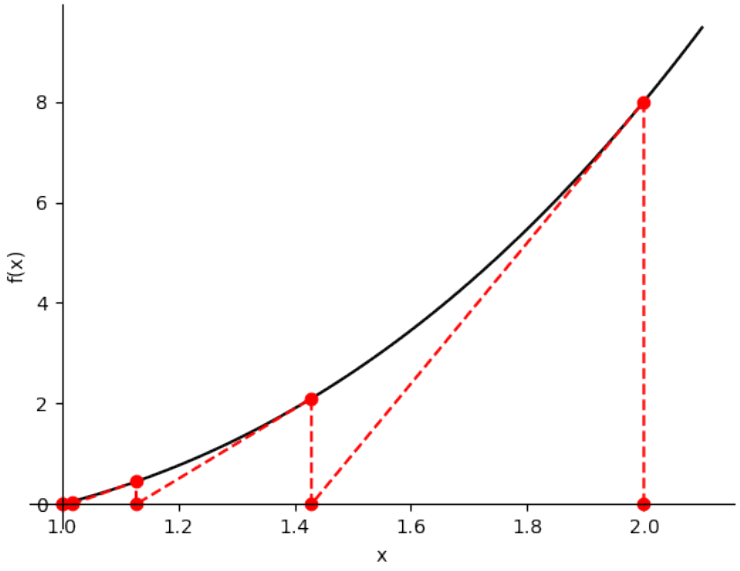

Figure 2.2: Diagram visualising the NR method iteration. When starting from a initial guess of 2, the method takes 4 steps to converge to the root.

2.3 Existing multivariate methods

There are multiple multivariate methods available which perform root-finding. However, the main focus of this paper will be the multivariate version of NR, which is discussed below.

2.3.1 Index notation

When describing tensors, this paper will use index notation. An index subscript, such as $(.)_i$, shows the number of the element in a first order tensor (vector). Multiple indices show higher order tensors, for example $(.)_{ij}$ shows a matrix with rows indexed by $(.)_i$ and columns indexed by $(.)_j$. Multiplying tensors with the same index indicates summation over that index and a subscript of a $(.)_{,j}$ shows derivation with respect to a variable with an index of $(.)_j$.

2.3.2 Newton-Raphson

NR in multivariate space ($x \in \mathbb{R}^N$) takes a similar form as before. The variable x becomes a vector of size N, which will be denoted by $x_i^{(n)}$, where i is the index of x, $(.)^n$ shows the iteration number and N is the number of dimensions. The function f also becomes a vector of size N and is denoted by $F_i(x^{(n)})$. The multivariate form of NR is shown below:

$$F_{i,j}(x^{(n)})\Delta x_i = -F_i(x^{(n)}) \tag{2.35}$$

where Δx_i is the step used to update the value of x_i.

2.4 Extended Method

This paper will propose a modification to NR by applying a non-linear modification aimed to reduce the non-linearity of a function to the base multivariate function F_i.

2.4.1 Non-linear modification P

In order to improve the performance of root-finding methods, the non-linearity of the problem can be reduced [7]. The way this is achieved is by using the definition of non-linearity in (2.22) shown here for convenience:

$$\frac{d^2 f(x)}{dx^2} \geq \lambda \frac{df(x)}{dx}$$

Rearranging the above gives:

$$\frac{\frac{d^2 f(x)}{dx^2}}{\frac{df(x)}{dx}} \geq \lambda \tag{2.36}$$

where λ is a factor of non-linearity. To reduce the non-linearity of the problem, λ can be set to zero. This can only be true when the numerator of (2.36) is zero and that in itself is true only if $f(x)$ is linear. Thus, a modification to the function is needed that will convert $f(x)$ to a linear function. This new converted function, $r(x, c)$, can be expressed through a conversion function [7], $P(x, c)$, as:

$$r(x, c) = mx - c = P(x, c)f(x) \tag{2.37}$$

From here $P(x, c)$ can be found as:

$$P(x, c) = \frac{mx - c}{f(x)} \tag{2.38}$$

The function $P(x, c)$ introduces new roots and eliminates the desired root. Hence, extra modifications are needed. As demonstrated by Aggarwal & Pant [7], the final modified form of the function is:

$$P(x, c) = \frac{x - c}{f(x) - f(c)} \tag{2.39}$$

The conversion eliminates the linear factor m by setting it to 1 to maintain generality and introduces $-f(c)$ in the denominator to eliminate the newly introduced root of $x = c$. Subsequently, the final expression of $r(x, c)$ becomes:

$$r(x, c) = P(x, c)f(x) = \frac{x - c}{f(x) - f(c)} f(x) \tag{2.40}$$

Multivariate expansion

The above-described method was defined in the univariate space. This paper will show the development of the method in the multivariate space.

The base equation (2.40) needs to be modified. This is done by changing the variables and functions into vectors:

$$\begin{aligned} x &\longrightarrow x_i \\ c &\longrightarrow c_i \\ f(x) &\longrightarrow F_i(x) \\ r(x, c) &\longrightarrow q_{ij}(x, c) \end{aligned} \tag{2.41}$$

and then rearranging (2.40) to:

$$q_{ij}(x, c) = (x_i - c_i)\left(\frac{F_j(x)}{F_j(x) - F_j(c)}\right) \tag{2.42}$$

where $i, j \in \{1, 2, 3, ..., N\}$ and N shows the dimension of the problem. Here a special rule is set where no summation over the index $(.)_j$ will be performed to maintain the proper shape of the answer.

2.4.2 Extended Newton-Raphson (ENR) method

With the multivariate form of the linear modification, ENR can be derived from (2.35) as:

$$q_{ij,k}(x^{(n)}, c)\Delta x_k^{(n)} = -q_{ij}(x^{(n)}, c) \tag{2.43}$$

To solve (2.43), the inverse of $q_{ij,k}$ needs to be found. However, obtaining the inverse cannot be naturally realised because $q_{ij,k}$ is a third-order tensor. A third-order tensor cannot act as a linear map from a vector space to itself and, thus, no such map can have an inverse. In order to to get the inverse, $q_{ij,k}$ needs to be modified to a second order tensor. One can set a matrix w_{Lk} to be equal to:

$$w_{Lk} = q_{ij,k} \tag{2.44}$$

where $L = N \times (i - 1) + j$, which leads to $L \in \{1, 2, 3, ..., N^2\}$; hence, the shape of w_{Lk} is $[N^2, N]$. This is a second-order tensor, so an inverse is possible. However, the shape of the tensor is guaranteed to be rectangular, so a regular inverse cannot be taken. For that reason, a generalised inverse, better

known as a Moore-Penrose inverse [17], has to be taken. This inverse is constructed using singular value decomposition (SVD) and is denoted by the superscript $(.)^+$. According to the generalised inverse theory of matrices [18], the matrix w_{Lk} will have a unique Moore-Penrose inverse. Once the inverse of w_{Lk} has been found, the function matrix p_{ij} has to be transformed to the modified function vector p_L, using the same $L = N \times (i-1) + j$ method. This will yield the final equation as:

$$\Delta x_k = w_{Lk}^{+} p_L \tag{2.45}$$

The construction of w_{Lk} is critical to the functioning of the algorithm. Hence, the derivation will be presented. For the equation to be solvable, the rank of the matrix has to be equal to the number of elements of x. In order to find what rank the final matrix will be, Sylvester's rank inequality will be used [19]. It states that if a_{ij} is a matrix of shape $[m \times n]$ and b_{jk} is a matrix of shape $[n \times p]$ then:

$$rank(a_{ij}) + rank(b_{jk}) - n \leq rank(a_{ij} b_{jk}) \tag{2.46}$$

Getting the derivative of q_{ik} begins by writing out its expression as:

$$w_{Lk} = \frac{\partial}{\partial x_k}\left[(x_i - c_i)\left(\frac{F_j(x)}{F_j(x) - F_j(c)}\right)\right] \tag{2.47}$$

Using the product rule, the equation is transformed into:

$$w_{Lk} = \frac{\partial (x_i - c_i)}{\partial x_k}\left(\frac{F_j(x)}{F_j(x) - F_j(c)}\right) + (x_i - c_i)\frac{\partial}{\partial x_k}\left(\frac{F_j(x)}{F_j(x) - F_j(c)}\right) \tag{2.48}$$

The fraction in the brackets will be set to equal a new tensor:

$$G_j = \left(\frac{F_j(x)}{F_j(x) - F_j(c)}\right) \tag{2.49}$$

Consequently, (2.48) can be rewritten as:

$$w_{Lk} = \frac{\partial (x_i - c_i)}{\partial x_k} G_j + (x_i - c_i)\frac{\partial G_j}{\partial x_k} \tag{2.50}$$

The expression of the partial derivative of G_j can then be written out as:

$$\frac{\partial G_j}{\partial x_k} = \left[\frac{F_{j,k}\left(F_j(x) - F_j(c)\right) - \frac{\partial}{\partial x_k}\left(F_j(x) - F_j(c)\right)F_j(x)}{\left(F_j(x) - F_j(c)\right)^2}\right] \tag{2.51}$$

The derivative of $F_j(x) - F_j(c)$ becomes equal to $F_{j,k}$ as the derivative of $F_j(c)$ with respect to x_k is equal to zero. This yields the final form of (2.51) as:

$$\frac{\partial G_j}{\partial x_k} = F_{j,k}\left(\frac{-F_j(c)}{(F_j(x) - F_j(c))^2}\right) \tag{2.52}$$

Here, the fraction in the brackets will be set to a new tensor:

$$H_j = \frac{-F_j(c)}{(F_j(x) - F_j(c))^2} \tag{2.53}$$

This gives the final form of (2.48) as:

$$w_{Lk} = \frac{\partial (x_i - c_i)}{\partial x_k} G_j + (x_i - c_i) F_{j,k} H_j \tag{2.54}$$

Solving the above yields:

$$w_{Lk} = \delta_{ik} G_j + (x_i - c_i) F_{j,k} H_j \tag{2.55}$$

where δ_{ik} is the identity matrix. Not summing over the index $(.)_j$ allows for this equation to preserve its rank. By applying Sylvester's inequality, the maximum rank of each tensor along the $(.)_k$ dimension is found to be 1. This means that once the $L = N \times (i-1) + j$ conversion is applied, the total rank of the new tensor w_{Lk} will be equal to the sum of the matrix ranks along $(.)_k$. This will coincide with the dimension of the problem, making the system determined. Using this derivation of the problem, the ENR method is fully formalised. As ENR applies a modification to the base function, the convergence properties of the method are identical to that of NR. Hence, this method retains super-linear convergence near the root. The results of its testing are presented below.

2.5 Results

In order to produce reliable results, a testing methodology was established.

2.5.1 Methods

- To create robust results, a uniform random sample of data was used for the initial choice. This is achieved through Latin hyper-cube sampling (LHS) [12] [20].
- Tolerances t_r for Δx and t_a for $F(x)$ were set to be equal to 10^{-6} and 0.001414 respectively. If each element of the result is ± 0.001 away from the actual result it is considered successful. Taking the norm of such a vector results in $\sqrt{0.001^2 + 0.001^2} = 0.001414$.
- All function were tested in the range of $[-50, 50]$.
- All plotted graphs have a resolution of $[512 \times 512]$ pixels.
- The iteration limit was set at 100 iterations.
- White areas represent starting points from which the algorithm did not converge within the predetermined steps (un-converged).
- Black areas represent starting points from which the algorithm diverged or encountered other problems such as numerical overflow in exponents or zero division.
- For graphs with multiple roots, each colour represents a convergence to a separate root and is denoted by r_i, where $(.)_i$ shows the index of the root. The opacity of the colour relates to the number of steps taken to converge.
- The horizontal axis of each graph shows the range of x_0. The vertical axis of each graph shows the range of x_1.
- The performance of the methods will be gauged based on the percentage of the area in which the algorithm converged. This percentage is later referred to as coverage.
- A set of equations was used to produce the results. The equations and their roots are as follows:

$$\begin{aligned} f_1(x) &= x_0^3 - 3x_0x_1^2 - 1 \\ f_2(x) &= 3x_0^2x_1 - x_1^3 \\ x^* &= (-0.500, 0.866), \\ &\quad (1.000, 0.000), \\ &\quad (-0.500, -0.866) \end{aligned} \tag{2.56}$$

$$\begin{aligned} f_1(x) &= e^{x_0} - x_1 \\ f_2(x) &= x_0 \times x_1 - e^{x_0} \\ x^* &= (1.000, 2.718) \end{aligned} \tag{2.57}$$

$$\begin{aligned} f_1(x) &= x_0^2 - \frac{1}{x_0} + x_1 \\ f_2(x) &= \frac{1}{x_1} + x_0 \\ x^* &= (1.260, -0.794) \end{aligned} \tag{2.58}$$

2.5.2 Choice of modification parameter c

Choosing a suitable value for the modification parameter c is vital to the improved convergence characteristics of ENR. A particular choice of c, whether it is a constant value or a function of x, will be referred to as a configuration of c. If c is picked close to x^*, then the convergence regions, which are shown below, are greatly improved. However, this requires prior knowledge of the roots of the system. If such knowledge is available then that configuration will outperform most others. However, to maintain generality and to help the method tackle systems whose roots are completely unknown, c will primarily be taken as a function of x. In this situation, knowledge of the system is necessary to pick a good relation between c and x, but that is more readily available as the general type of equation is known when setting up the system, i.e. exponential, high-order polynomial, etc. The sections below are going to compare different configurations of c used by ENR. This will show their effect on the convergence area.

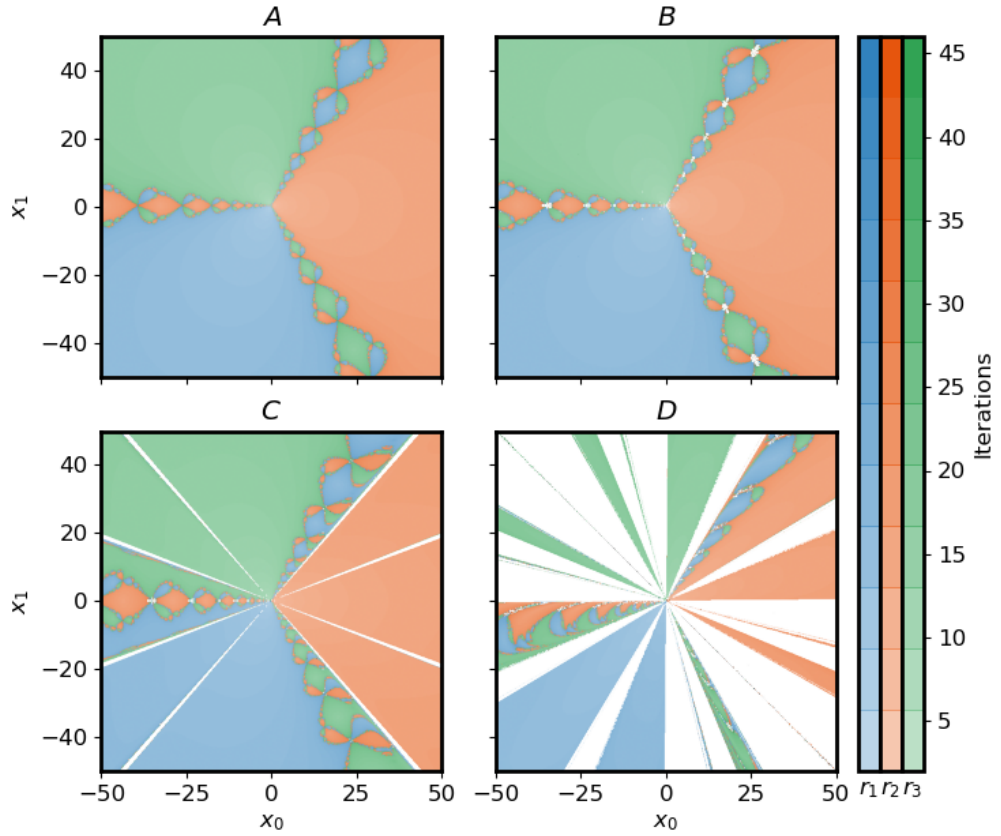

Figure 2.3: Figure showing results of (2.56). Panel A shows the system solved with NR. Panels B-D show the system solved by ENR with different configurations. Panel B uses $c_i = 2x_i$. Panel C uses $[c_0, c_1] = [2x_0, 3x_1]$. Panel D uses $c_i = x_i + 10^{-5}$. Panel B generates a graph which most closely resembles the original NR graph. The other two modification introduce grater distortions.

2.5.3 Validation

In order to validate ENR it had to be tested in a configuration which should not be affected by the modification. One such case is (2.56). For the predetermined range given above, NR provides 100% convergence coverage. It also creates a unique fractal shape which can be often seen in other papers dealing with NR. Hence, in order for the rest of the result to be valid, ENR has to perform almost identically to NR in (2.56). The performance of ENR is dependant on the choice of c. Here, multiple different arrangements of c will be presented together with the results from NR. This will allow for comparison not only between NR and ENR but also between the different choices of c. In Fig. 2.3-A the results for (2.56) evaluated by NR are presented. The NR method provides a 100% coverage of the $[-50, 50]$ range. A unique fractal pattern can also be seen. The second panel, Fig. 2.3-B, sets c as a function of x, where:

$$c_i = 2x_i \tag{2.59}$$

This yields a 99.75% coverage of the range. The white colour in the graph shows areas where the method exceeded the maximum iteration count. These zones are in the highly fractal border area which may explain the reason the method did not converge. The third panel, 2.3-C, shows an alternate c function:

$$[c_0, c_1] = [2x_0, 3x_1] \tag{2.60}$$

ENR with the above function only yielded a 97.20% coverage of the range. It also produced eight lines which radiate outwards from the centre point. These lines section off four slices which lay close to the diagonals of the graph. Their exact positions depend on the ratio between c_0 and c_1. If their ratio is one, then the sections lay exactly on the diagonals and produce a symmetric shape. If $c_0 < c_1$, then the sections rotate towards the horizontal axis. This can be seen in 2.3-C. Finally, if $c_0 > c_1$, the section rotates towards the vertical axis. The final panel, 2.3-D, shows the results when c was set to a small

perturbation of x as:

$$c_i = x_i + 10^{-5} \tag{2.61}$$

This yielded a coverage of 49.80%. The II and IV quadrants are sparsely covered. Using the small perturbation may have caused the new modified function r to have a very small value, as the modified function is proportional to the difference between x and c. Having the value of the modified function be a small number reduces the size of the step taken, which will in turn increase the number of iterations. This can be tested by setting the value of:

$$c_i = x_i + 1 \tag{2.62}$$

In this case, the plot of the results looks similar to 2.3-D. However, the coverage is up to 57.68%. This is due to the expansion of r_2 and r_3 convergence zones in the II and IV quadrants. This was further tested by adding 2 to the value of x and the converge grew to 59.42%. This shows that selecting a value far away from the initial guess is beneficial to the convergence properties of ENR. However, this does have diminishing returns as the distance increases.

Examining cases 2.3-A and 2.3-B, it can be seen that ENR produces the same fractal structure as NR and finds the same number of roots over the same domain. This validates that the method is functioning correctly.

2.5.4 Comparison

Exponential

The system (2.57) is one where NR fails to converge for a large area of the testing range. The variable x_0 determines the magnitude of the exponential terms, so (2.57) is exponential in x_0. However, the x_1 term is linear. It is expected for NR to have a large, smooth convergence range in x_1 and a narrow one in x_0. This is shown in Fig. 2.4-A. In the vertical direction, x_1, the graph covers the whole range with some discontinuities. In x_0, the covered range is a very slim strip close to the root. The total coverage of the range for this configuration by NR is 04.64%. The first configuration of ENR, shown in Fig. 2.4-B, c was set to:

$$c_i = \frac{1}{2} x_i \tag{2.63}$$

This yielded an improved range in the x_0 (non-linear) direction. The total coverage was brought up to 31.22% which is an increase of 26.58%. The modified shape is not fully continuous. However, there is a large smooth area in the I quadrant, which exhibits predictable behaviour i.e. when the initial guess is further away from the root, the number of iterations increases which can be seen by the darker shade around the edge of the range. There are also some white artefacts around the negative portion of the horizontal axis ($x_0 < 0$, $x_1 = 0$). This shows that the iteration limit was exceeded in that area. Unlike in the previous example, the case $c_i = 2x_i$ yielded a similar shape to $c_i = \frac{1}{2}x_i$ with a reduced coverage. That is why it is not shown in this graph. One of the possible explanations for why $\frac{1}{2}x_i$ performed better than $2x_i$ is that as this problem is exponential, small changes in x can cause a large difference in the result. This suggests that choosing smaller scalar modifiers can be beneficial. The third panel of the graph, Fig. 2.4-C, shows the same configuration as 2.3-C, where:

$$[c_0, c_1] = [2x_0, 3x_1] \tag{2.64}$$

Previously, this case caused distortions based on the ratio between the elements, but in the exponential system, the distortions are not present. The total coverage of this configuration amounted to 19.45%. It created a continuous region in the IV quadrant and repeating striations extending towards the upper limit of the x_0 range. Reducing the value of the second variable, as shown by the previous case, also improves the convergence properties of the method. For example, the total coverage of $c = [3x_0, 2x_1]$ is 46.18%. It extended the region of convergence in the II quadrant and reduced the spacing between the striations in the I and IV quadrants. The final configuration shown, in 2.3-D, is:

$$c_i = x_i + 1 \tag{2.65}$$

This configuration yielded a total coverage of 31.36%. Comparing panels B and D, it can be observed that both create a smooth region in the I quadrant. Both have an area in the II quadrant which is discontinuous and striations in the IV quadrant. The difference in the shapes seems to mimic the order of the c expression, i.e. case B appears to have an area generated by some quadratic relations and case D appears to have an area generated by a linear relation. In panel D the relation between x and c is a

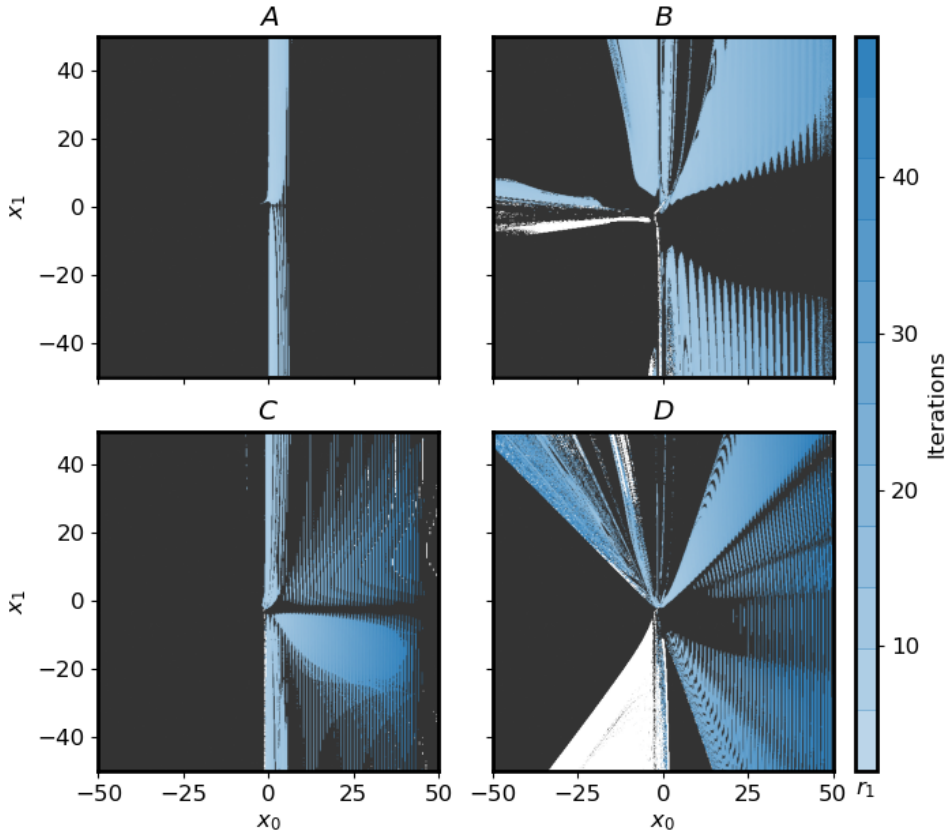

Figure 2.4: The figure shows results of (2.57) solved by NR ,panel A, and three configurations of ENR, panels B-D. Panel B represents a configuration of $c_i = \frac{1}{2}x_i$. Panel C shows the configuration of $c = [2x_0, 3x_1]$ and panel D shows the configuration of $c_i = x_i + 1$. All configurations of ENR produced a larger coverage of the range. Sections B and D also created un-converged regions in the III quadrant.

constant number which appears to have scaled to a linearly varying shape, whereas in panel B the linear relation between x and c appears to have scaled to a quadratically varying shape.

The results from this section show that in the exponential case the ENR method consistently produces a larger coverage of the test range. Different choices of c affect the shape but some general conclusions can be made, i.e. using smaller values when multiplying the exponential term yields better results. Generally, the convergence is improved if the choice of c is close to the root x^*. This can be seen in Fig. 2.5. By setting $c = (2, 1)$, a convergence coverage of 57.24% can be attained with two large, smooth converged zones in the II and IV quadrants. The results from the rest of the tested configurations are presented in Table 2.2, with some of their convergence diagrams attached in the appendix. The results from the rest of the tested configurations are presented in Table 2.2, with their convergence diagrams attached in the appendix.

Negative exponent

Negative exponents in a system, such as (2.58), lead to behaviour similar to some non-linear material models. For this system NR has discontinuous coverage of the test range. This can be seen in Fig. 2.6-A, where NR covers the whole test range with a disjointed pattern which exhibits some symmetries around the vertical and horizontal axes. The total coverage of NR in (2.58) was 36.63%. The second configuration, shown in 2.6-B, is that of:

$$c_i = 2x_i \tag{2.66}$$

This was chosen over the previous $c_i = \frac{1}{2}x_i$ because at smaller values a negative exponent function becomes more non-linear. When c was set to half x, the total coverage was 44.29%, whereas when it was set to double the value of x, the total coverage was 49.34%. The total coverage of this configuration is

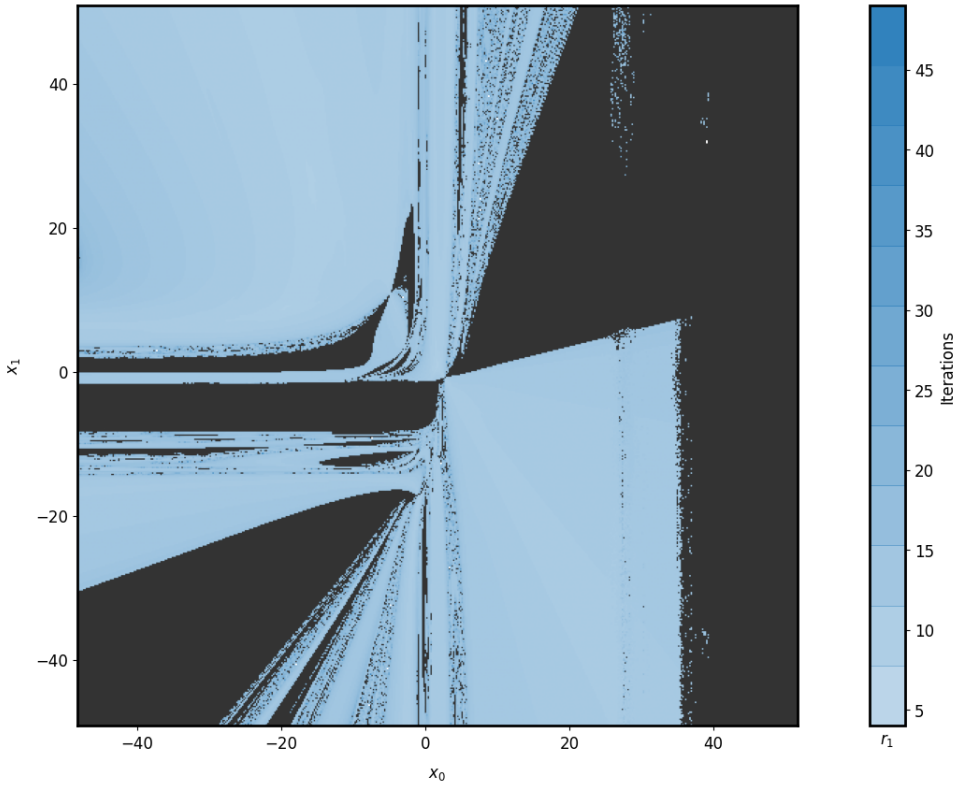

Figure 2.5: Figure showing results of (2.57) solved using ENR and a c configuration of $c = (2, 1)$. This yielded two large, smooth converged zones in the II and IV quadrants. The total coverage is 57.24%.

c	Coverage	Remarks
$None$	04.64%	Narrow convergence on x_0 axis
$x_i - 10^{-5}$	26.18%	Striations and un-converged region
$x_i + 10^{-5}$	26.14%	Striations and un-converged region
$[2x_0, 3x_1]$	19.45%	Heavy striations, no coverage of $-x_0$ direction
$[3x_0, 2x_1]$	46.18%	Large continuous regions along x_0 axis
$0.5x_i$	31.22%	Continuous regions and small un-converged region
$2x_i$	11.44%	Small expansion in $+x_0$ direction
$[1, 2]$	73.74%	Large smooth area due to vicinity of c to x^*
$[10, 20]$	48.31%	Large smooth area in the I and IV quadrants, repeating stripes in the II quadrant
x^2	05.45%	Wide strip of converged area close to $x_0 = 0$. Large area of un-converged points past $\|x_0\| > 20$

Table 2.2: Tabulated results for (2.56) with different c configurations and with brief remarks on the shape of the convergence region. The largest coverage is given by $c = [1, 2]$.

larger than that of NR. However, the shape of the convergence region is less well distributed as compared to NR. In 2.6-B, there are smooth regions close to $x_1 = 0$. There is also a large smooth area present in the III quadrant. The second configuration:

$$[c_0, c_1] = [2x_0, 3x_1] \tag{2.67}$$

is shown in 2.6-C. The total coverage was 41.62%. This configuration produced a large smooth area in the II quadrant, but reduced the density of converged points in the other three quadrants. However, it was still able to produce a larger coverage of the range than NR. The shape of the convergence basin for the given range is affected by the ratio of c_0 to c_1. If $[c_0, c_1] = [3x_0, 2x_1]$, then the coverage increases to 61.07% and smooth regions appear in all 4 quadrants. The third configuration presented in 2.6-D is:

$$c_i = x_i + 10^{-5} \tag{2.68}$$

This perturbation of x generated a coverage of 46.66%. This method changed the pattern of NR drastically but still managed to produce a larger coverage. There are large smooth areas in the I and IV quadrants.

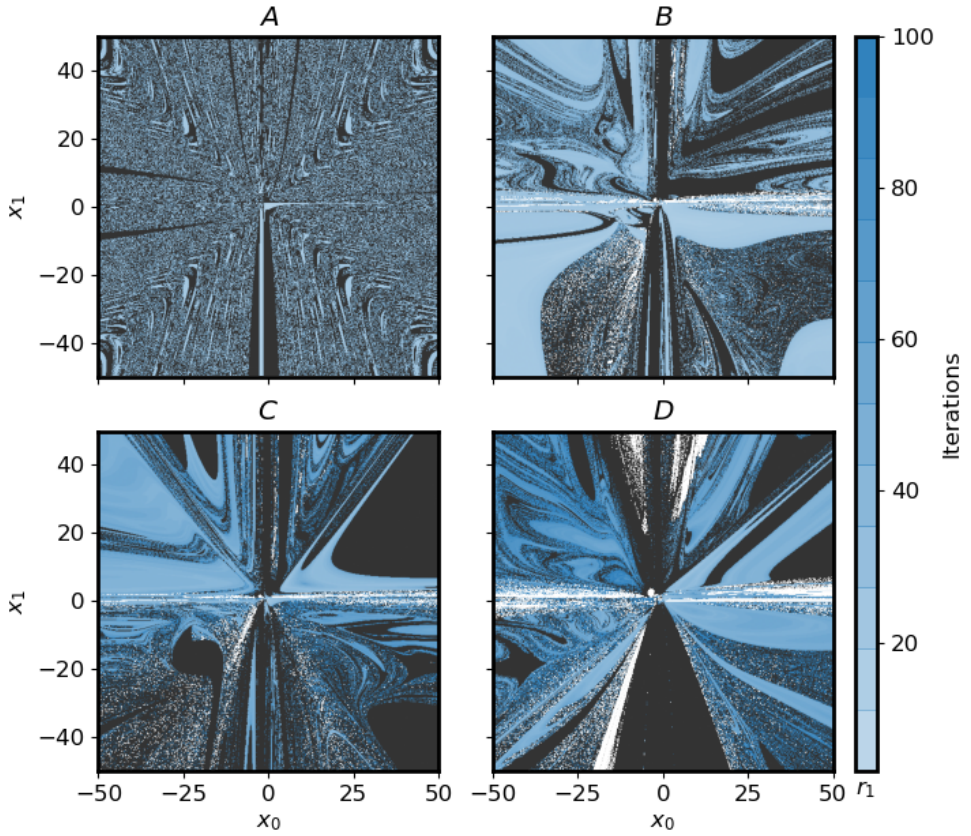

Figure 2.6: Figure showing results of (2.58) solved by both NR and ENR. NR yielded a discontinuous coverage of the range at 36.63% shown in panel A. Panels B-D show different configuration of ENR. These are: B - $c_i = 2x_i$, C - $[c_0, c_1] = [2x_0, 3x_1]$, D - $c_i = x_i + 10^{-5}$. All configurations of ENR were able to produce larger coverage than NR. However, their coverage shape was distorted and non-symmetric.

The rest of the test range in scattered by converged and un-converged points. These results show the positive effects of the linear modification used by ENR. Even though the convergence area produced by ENR is on average less well distributed than NR in this case, ENR was able to produce a larger coverage overall. The final configuration to be examined is when c was set to equal a constant $(-2, 1)$. This choice of c is relatively close to the true value of $x^* = (1.260, -0.794)$ and it created a very desirable shape with large smooth sections, shown in Fig. 2.7. It also reduced the number of iterations by almost a half. The total coverage was 44.88%. This shows that if a value of c is close to x^*, the convergence properties of the method improve drastically. This is also true if the initial guess x_0 is close to x^*, so using ENR with static values allows for two guesses at initial values. The results from the rest of the tested configurations are presented in Table 2.3, with some of their convergence diagrams attached in the appendix.

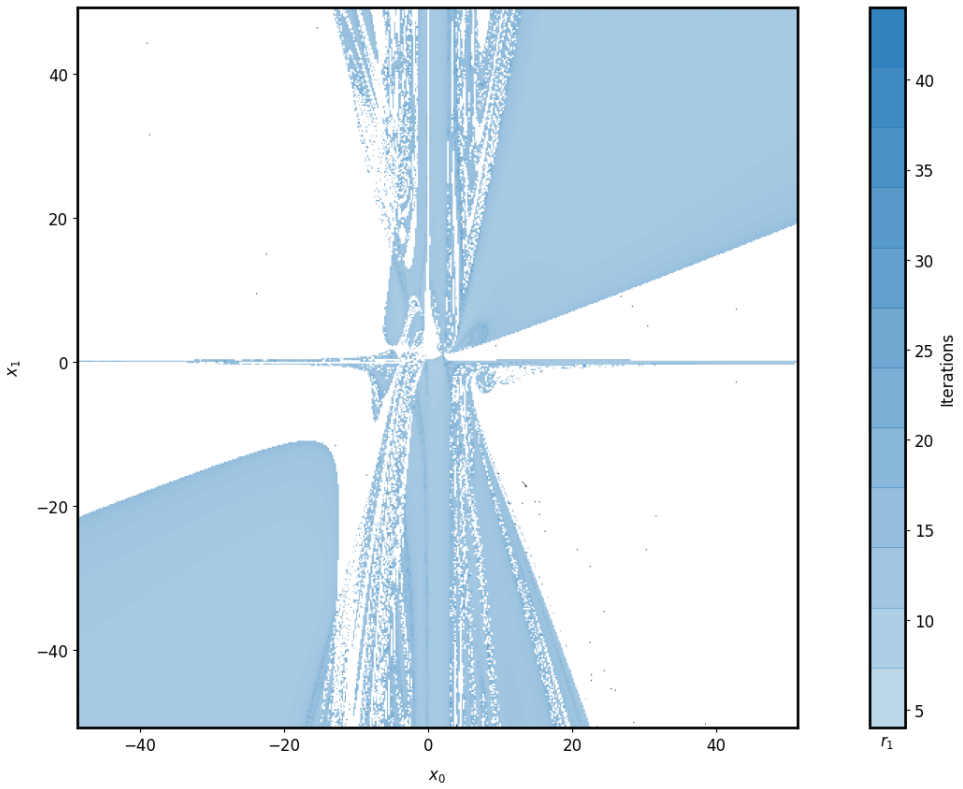

Figure 2.7: Figure showing results of (2.58) when solved using ENR with $c = [-2, 1]$. This configuration yields two large smooth regions in the I and III quadrants. It also changes the behaviour of the method across the range. Rather than ENR diverging, like in the previous cases (black area), it exceeds the iteration limit (white area). The total coverage of this configuration is 44.88%.

c	Coverage	Remarks
$None$	36.63%	Disjoint converged areas across range
$x_i - 10^5$	46.62%	Large distorted converged area with un-converged artefacts
$x_i + 10^5$	46.66%	Large distorted converged area with un-converged artefacts
$[2x_0, 3x_1]$	41.62%	Large smooth converged are in the II quadrant
$[3x_0, 2x_1]$	61.07%	Large converged areas in the I, III and IV quadrants
$0.5x_i$	44.29%	Large converged areas in the III and IV quadrants
$2x_i$	49.34%	Large converged areas in the III and IV quadrants
$[1, 2]$	13.58%	Smooth area in the I quadrant
$[10, 20]$	53.70%	Large heavily striated area

Table 2.3: Tabulated results for the different c configurations of (2.58) with brief remarks on the shape of the convergence region. The largest coverage is given by $c = [10, 20]$.

2.6 Conclusion

The ENR method provides improved convergence properties when compared to the NR method, as shown in subsection 2.5.4. In highly non-linear systems, ENR extended the coverage of the proposed test range and in some cases, depending on the choice of the modification parameter c, reduced the number of iterations needed to converge. The modification, however, can also reduce coverage when systems are not highly non-linear, as demonstrated in subsection 2.5.3. ENR, as shown by the derivation, is more computationally demanding than NR. However, due to the method's improved properties, it can serve as a useful technique when dealing with highly non-linear systems.

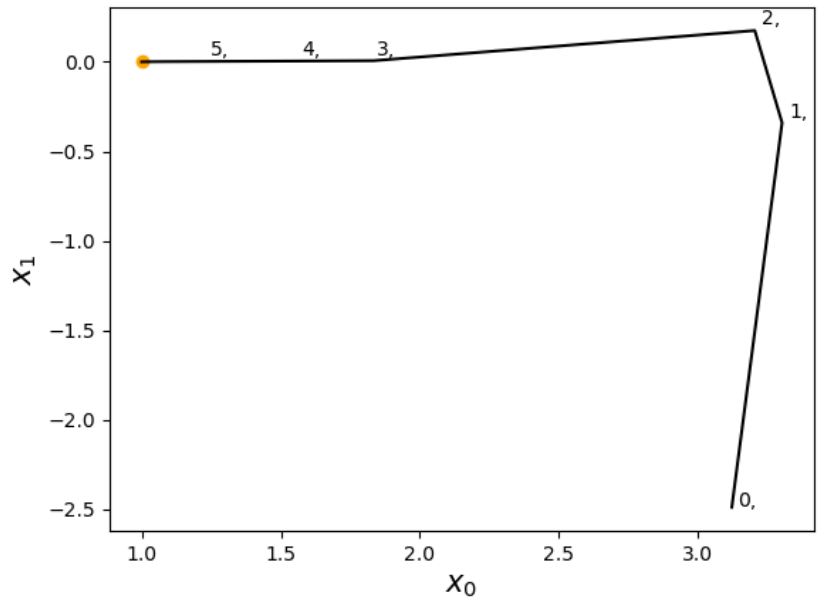

Figure 2.8: Figure showing steps taken by DS method for (2.56). Numbers indicate the number change in the current approximation of x^* and the orange dot shows the root of the system tested.

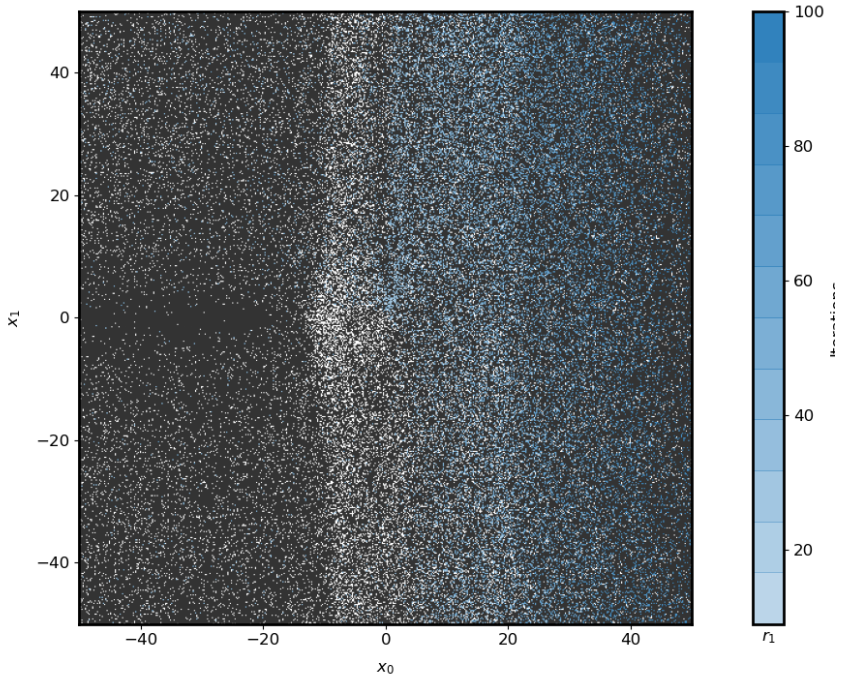

Figure 2.9: Figure showing convergence basin of the DS method.

2.7 Exploration of Diagonal Secant Method (DS)

As mentioned in subsection 2.2.2, the secant method with two initial values only exists in one dimension. If the secant method is to be expanded to the multivariate space, one has to increase the number of initial guesses. This makes this method very cumbersome for high dimensional problems. It was attempted to create a method based on the secant approximation which works in multiple dimensions by using only two initial guesses. This was attempted due to the secant method's low computational demand. The derivation of the method began by first taking two guesses denoted by $x_i^{(0)}$ and $x_i^{(1)}$. The system F_i was

evaluated at both points and the difference between both evaluations was recorded in a variable:

$$\Delta H_i = F_i(x^{(1)}) - F_i(x^{(0)}) \tag{2.69}$$

The difference in the two guesses was taken as:

$$\Delta X_i = x_i^{(n+1)} - x_i^{(n)} \tag{2.70}$$

From here, the secant step Δs_i is calculated as:

$$\Delta s_i = -\frac{-F_i(x^{(n+1)})}{\Delta H_i}\Delta X_i \tag{2.71}$$

where, as in the ENR derivation, no summation over $(.)_i$ was done. This yielded a method which was able to find roots in (2.56) and (2.57), but not in (2.58). The path which the DS method took to find a root of (2.56) with $x^{(0)} = (3, 2)$ and $x^{(1)} = (3.5, 1)$ is shown in Fig. 2.8. The method finds one element of the variable first and then searches for the other. This behaviour led to a step-like path to be taken to the root. The DS method was also tested across the same test range as ENR which produced Fig. 2.9. The second choice $x^{(1)}$ was made by choosing two random values in the range of $[-50, 50]$. Hence, the graph shows all initial choices $x^{(0)}$ paired with a random $x^{(1)}$. Fig. 2.9 shows that the coverage of the range is very poor and very discontinuous. The number of steps taken to converge varied widely and if $x_i^0 = x_i^1$ for any i, then the method will fail immediately. All of this combined meant that the method will not be expanded upon further in this paper.

Chapter 3

Minimisation

3.1 Introduction

3.1.1 Definition

Minimisation is a mathematical procedure used to locate local or global minima in a function. This involves calculating or estimating multiple derivatives of the function and using them to find a direction leading to a minimum. Minimisation is related to root-finding as it performs root-finding on the derivative of square error of a function rather than the function itself. The error of a function is expressed through the residual, which is the difference between a function's output $f(x, \theta)$, in its current configuration θ, and an observed result. This observed result can be interpreted as the output of the same function $f(x, \theta^*)$ with the observed configuration θ^* and with some added noise, arising from measurement errors. The goal of minimisation is finding the desired configuration. The process is called minimisation because, as the observed configuration is approached by the current one, the difference between $f(x, \theta)$ and $f(x, \theta^*)$ becomes smaller (minimised).

Graphically this can be seen as finding the location of zero slope of the base function. Using the function from the root-finding example, one can see the local extremums in Fig. 3.1. Finding the location of zero slope can lead to either of the two extremums in Fig. 3.1, which is why a minimisation algorithm is constructed in such a way to only find minima.

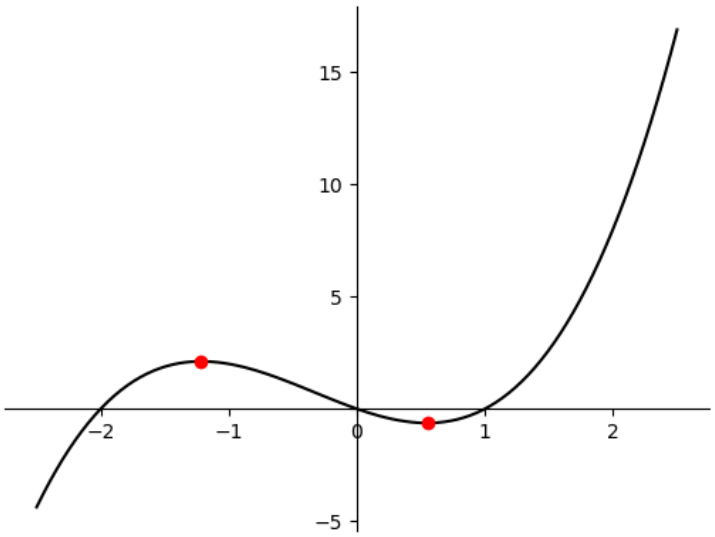

Figure 3.1: A graph of the cubic equation $y = (x-2)x(x+1)$ plotted in the range of [-3,3]. Red markers show the positions of local extrema in the range.

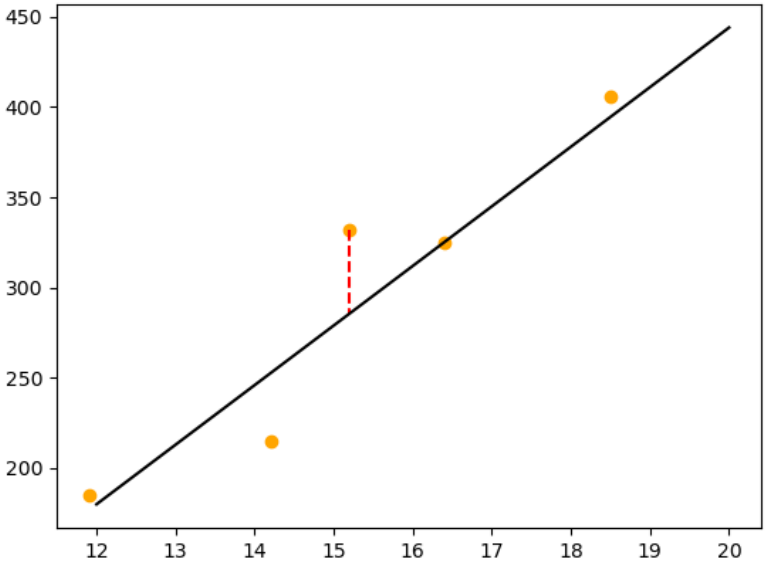

Figure 3.2: The figure shows a scatter of random data (yellow points) with a linear approximation of that data drawn (black line). One residual is shown on the graph by a dashed red line connecting one of the data points and the approximation line. The length of that line is the magnitude of the residual.

3.1.2 Error

In minimisation problems, error is the target parameter. The most common way of defining error is as the difference between an observed result and a calculated result. This difference between results is known as a residual and can be defined using $f(x, \theta^*)$, which is the function f evaluated with the desired configuration θ^*. This definition will be used, even though residuals can be defined in terms of empirical observations. All observed results for this paper were artificially generated using $f(x, \theta^*)$ and added noise. The residual can also be shown graphically, which is done in Fig. 3.2. There, one can see that the residual is the distance between the observed points and the proposed model (straight line). As minimisation tries to reduce the total error in a function, the residual needs to be taken all across the domain of x. Hence, the sum of all residuals is taken. To account for different signs which the residuals might have the residuals is are squared. This is known as the square error (SE) and can be calculated as:

$$SE = r_i r_i \tag{3.1}$$

This allows for an accurate estimation of the system's total error. By using this definition of error and previously explained concepts in section 2.1, existing minimisation methods can be explored.

3.2 Existing methods

3.2.1 Gauss-Newton (GN)

The Gauss-Newton method is an iterative minimisation algorithm based on the NR method. It performs root-finding on the derivative of the SE function, which locates a minimum. The GN method ignores second order derivatives of the residual to minimise computational costs, but that means that the GN step is an approximation to the most efficient direction. The derivation of GN shows this in more detail. Firstly, the residual is calculated, as:

$$r_i(\theta) = y_i - f(x_i, \theta) \tag{3.2}$$

Then the function to be minimised $F(\theta)$ is derived by summing the squared residuals and multiplying by a half. The factor is put here in order to eliminate a 2 which is produced by derivation later on.

$$F(\theta) = \frac{1}{2} r_i(\theta) r_i(\theta) \tag{3.3}$$

The desired value of θ is θ^*. In order to find θ^*, it is set equal to $\arg\min F(\theta)$ as:

$$\theta^* = \arg\min F(\theta) \tag{3.4}$$

The function $\arg\min$ represents the value of θ_j for which the following holds true:

$$\frac{\partial}{\partial \theta_j}\left[\frac{1}{2} r_i(\theta) r_i(\theta)\right] = 0 \tag{3.5}$$

The above equation can be simplified to produce the following:

$$\frac{\partial}{\partial \theta_j}\left[\frac{1}{2} r_i r_i\right] = r_i(\theta)\frac{\partial r_i}{\partial \theta_j}\bigg|_{\theta_j} = 0 \tag{3.6}$$

$$r_i(\theta)\frac{\partial r_i}{\partial \theta_j}\bigg|_{\theta_j} = r_i(\theta) r_{i,j}(\theta) = 0 \tag{3.7}$$

where $(.)_{,j}$ shows derivation with respect to index θ_j. Next, the Gauss-Newton iteration has to be derived. This can be done using the Taylor's series expansion of $r_i r_{i,j}$ around an initial guess of θ called θ^0. The expansion is written out as:

$$r_i(\theta^*) r_{i,j}(\theta^*) = r_i(\theta^0) r_{i,j}(\theta^0) + \frac{\partial}{\partial \theta_k}\left[r_i(\theta^0) r_{i,j}(\theta^0)\right]_{\theta_k^0} (\theta_k^* - \theta_k^0) + ... \tag{3.8}$$

Evaluating this expansion up to the second term yields:

$$r_i(\theta^*) r_{i,j}(\theta^*) = r_i(\theta^0) r_{i,j}(\theta^0) + (r_{i,k}(\theta^0) r_{i,j}(\theta^0) + r_i(\theta^0) r_{i,jk}(\theta^0))\Delta\theta_k \tag{3.9}$$

where $\Delta\theta_k = \theta_k^* - \theta_k^0$. This is the full expansion up to the second term and from here, one can derive GN by ignoring second order derivatives i.e. $(.)_{i,jk}$ and re-arranging to get:

$$r_i(\theta^*) r_{i,j}(\theta^*) = r_i(\theta^0) r_{i,j}(\theta^0) + r_{i,k}(\theta^0) r_{i,j}(\theta^0)\Delta\theta_k = 0 \tag{3.10}$$

From the above, the problem can be re-arranged again to get the final form of the GN method:

$$r_{i,k}(\theta^0) r_{i,j}(\theta^0)\Delta\theta_k = -r_i(\theta^0) r_{i,j}(\theta^0) \tag{3.11}$$

In the above equation, $\Delta\theta_k$ is the so called GN step. To find its value, LU (lower-upper) decomposition can be used. Practically, this was done using numpy.solve() which internally uses an algorithm called dgesv() that utilises LU decomposition. Once $\Delta\theta_k$ has been calculated it is used to update the initial guess θ^0 and every other consecutive guess until the termination criteria are reached. The update of θ is done by:

$$\theta_k^{(n+1)} = \theta_k^{(n)} + \Delta\theta_k \tag{3.12}$$

where the superscript shows the iteration number. The GN step is always in a decent direction. This can be shown by examining the gradient along the step. To find the gradient one can take the inner product of (3.11) with $\Delta\theta_j$ on both sides:

$$\Delta\theta_j r_{i,k}(\theta^0) r_{i,j}(\theta^0)\Delta\theta_k = -\Delta\theta_j r_i(\theta^0) r_{i,j}(\theta^0) \tag{3.13}$$

If $r_{i,k}(\theta^0)$ is full rank, then the matrix $r_{i,k}(\theta^0) r_{i,j}(\theta^0)$ is positive definite and therefore the left-hand side of the equations is always positive. Thus, on the right-hand side, $\Delta\theta_j r_i(\theta^0) r_{i,j}(\theta^0)$ is always negative. Hence, the GN method always has a step in the decent direction. To further explain how minimisation works, a simple example will be given.

3.2.2 Example of minimisation

A typical example of minimisation is finding the best fit of a curve. For this, a set of observations which represents the deflection of a cantilever I-beam will be used. The configuration of the problem is shown in Fig. 3.3. The data which will be used is shown in Table 3.2 and represents the deflection of the beam along its length. This data was created with an I (second moment of area) value of $2340cm^4$ with additional noise added to the result. The known properties of the beam are summarised in Table 3.1. The equation used to calculate the deflection and generate the data is:

$$y(x) = \frac{Px^2(3L - x)}{6EI} \tag{3.14}$$

Modulus E	$200GPa$
Length L	$2m$
Applied Force P	$10kN$

Table 3.1: Properties of beam used for Gauss-Newton example.

$x[m]$	0.000	0.500	1.000	1.500	2.000
$y[mm]$	0.000	0.490	1.781	3.606	5.698

Table 3.2: Observed data points used to evaluate the parameter of a deflection equation.

where y is the deflection, P is the force, L is the length, E is the modulus and I is the second moment of area. The parameter which will be fitted is the second moment of area I. It will be shown in $[cm^4]$ and for simplicity, all necessary unit conversions will be done in the background. This is a linear problem. Hence, GN should converge in one step, but as there is noise in the system more steps will be needed. Firstly, an initial guess of parameter needs to made. For continuity, the second moment of area will be denoted by θ. The above equation can be rewritten in terms of x and θ as:

$$f(x,\theta) = \frac{Px^2(3L-x)}{6E\theta} \tag{3.15}$$

For this example, the initial guess θ^0 will be taken to be:

$$\theta^0 = 2000cm^4 \tag{3.16}$$

From here, the residual can written in terms of y and f as:

$$r(\theta) = y - f(x,\theta) \tag{3.17}$$

Taking x and θ to be a vectors the above can be converted to:

$$r_i(\theta) = y_i - f(x_i,\theta) \tag{3.18}$$

Then the derivative of the residual with respect to θ can be written out as:

$$r_{i,j} = \frac{d(y_i(x) - f_i(x_i,\theta_j))}{d\theta_j} = -\frac{df_i(x_i,\theta_j)}{d\theta_j} \tag{3.19}$$

In this case, $r_{i,j}$ can be found analytically, but when this is unavailable a numerical derivative using finite differences can also be used. The closed form of $r_{i,j}$ is:

$$r_{i,j} = \frac{Px_i^2(3L-x_i)}{6E\theta_j^2} \tag{3.20}$$

Using the residual and its first derivative, one can calculate the GN step using (3.11). This step is used to update the initial guess θ^0 to produce the final answer. All GN steps and values of θ are presented in Table 3.3. The termination criterion was set when the step size becomes less than 10^{-3} because all values are reported to three decimal places. The result of the process yielded a value for $\theta \equiv I$ of $2339.921cm^4$ which satisfies the system. Using this value, one can see that the test beam used for this example was a UB203x133x25. The final estimated value of I was used to plot the deflection of the beam and the original data points were scattered on top. This can be seen in Fig. 3.4.

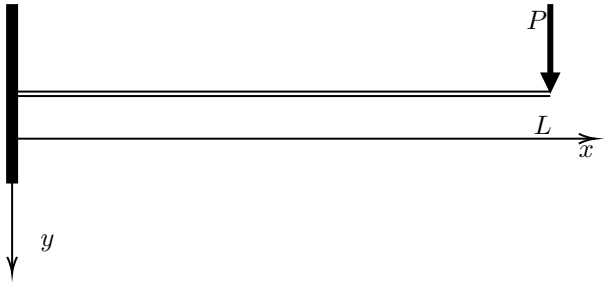

Figure 3.3: The figure shows the problem definition for the example solved via the Gauss-Newton method. It shows a cantilever beam of length L with an applied concentrated load P at one side.

Iter.	θ	$\Delta\theta$
0	2000	290.541
1	2290.541	48.339
2	2338.880	1.0416
3	2339.921	0.000

Table 3.3: This table shows the iterations of the GN method. At the fourth iteration the method converges.

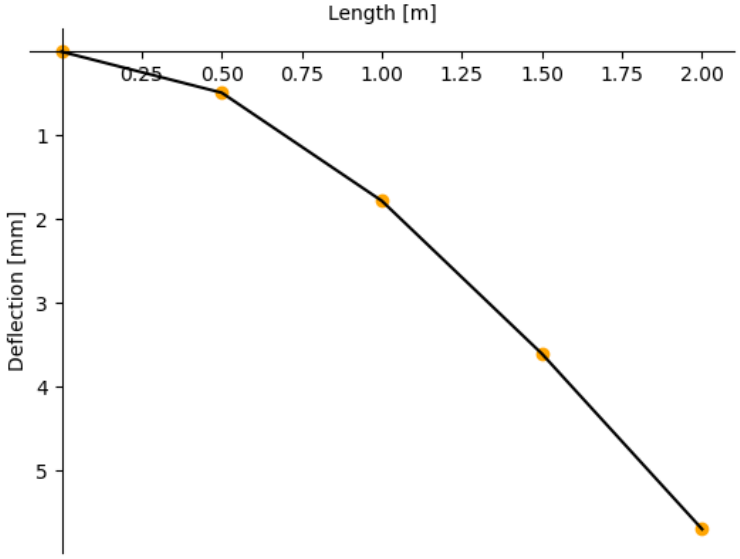

Figure 3.4: This graphic shows the data points (yellow) used by GN to find θ^*. The black curve shows the estimated value of the deflection function using the value of θ produced by GN after four iterations.

3.3 Corrected Gauss-Newton (CGN)

This project proposes a correction to the Gauss-Newton method which utilises the second derivative of the residual, which was omitted during the derivation, between (3.9) and (3.10). This will increase the computational complexity, but it will also improve the convergence behaviour in highly non-linear functions. The proposed improvement is a two step process which uses the original GN step to then calculate a corrected step using the second derivative. The derivation begins by getting the regular GN step, shown in (3.11). This is restated here for convenience.

$$r_{i,k}(\theta^0)r_{i,j}(\theta^0)\Delta\hat{\theta}_k = -r_i(\theta^0)r_{i,j}(\theta^0) \tag{3.21}$$

The only modification to this equation is the appearance of $\Delta\hat{\theta}_k$ which will be used to denote the original GN step. Once the GN step is found, the Taylor's series expansion of the residual is written out to the second term as:

$$r_i(\theta) = r_i(\theta^0) + r_{i,j}(\theta^0)\Delta\theta_j + \frac{1}{2}r_{i,jk}(\theta^0)\Delta\theta_j\Delta\theta_k \tag{3.22}$$

Then the corrected GN step $\Delta\theta_j$ can be factored out to produce:

$$r_i(\theta) = r_i(\theta^0) + \left(r_{i,j}(\theta^0) + \frac{1}{2}r_{i,jk}(\theta^0)\Delta\hat{\theta}_k\right)\Delta\theta_j \tag{3.23}$$

For convenience, the term in the brackets is set to equal s_{ij} and the equation is re-written again as:

$$r_i(\theta) = r_i(\theta^0) + s_{ij}(\theta^0)\Delta\theta_j \tag{3.24}$$

Substituting the new definition of $r_i(\theta)$ into (3.21) and noting that $r_{i,j}(\theta) \approx s_{ij}(\theta^0)$ and that:

$$s_{ij}(\theta^0) = \left(r_{i,j}(\theta^0) + \frac{1}{2}r_{i,jk}(\theta^0)\Delta\hat{\theta}_k\right) \tag{3.25}$$

the following can be derived:

$$s_{ik}(\theta^0)s_{ij}(\theta^0)\Delta\bar{\theta}_k = -r_i(\theta^0)s_{ij}(\theta^0) \tag{3.26}$$

where $\Delta\bar{\theta}_k$ denotes the CGN step. Its value can be found using the same methods as before, namely LU decomposition. If $s_{ik}(\theta^0)$ is full rank, then the the gradient along the step direction is always negative.

3.4 Results

3.4.1 Methods

- In order to create robust results, a uniform random sample of data is needed to create the initial guess. In this project, this is achieved through Latin hyper-cube sampling (LHS) which provides such uniform data in a given domain [20] [12].
- To make the results more general, the average of three samples was taken per specific domain.
- The distance parameter, which shows how far away the initial guess is from the actual parameters of the equation, is taken on a log scale.
- A noise parameter was introduced to the data, represented by a Signal to Noise ratio in decibels. The range was adjusted for each experiment.
- The resulting graphs use distance on the x axis and noise on the y axis. The steps are plotted on a contour plot with different colours representing a different amount of steps. Fractional steps are can be observed due to the averaging of results and due to linear interpolation used between data points.
- White space in the graphs represents areas where the method failed by exceeding 100 iterations steps. This region will be referred to as un-converged.
- A set of equations was used to generate data for testing. The equations, their true values θ^* and the sampling ranges s are shown:

$$\begin{aligned} f(x,\theta) &= \theta_0 x^3 + \theta_1 x^2 + \theta_2 x + \theta_3 + \theta_4 sin(x), \\ &\quad \theta^* = \{-0.001, 0.1, 0.1, 2, 15\} \\ &\quad s = [1, 10] \end{aligned} \tag{3.27}$$

$$\begin{aligned} f(x,\theta) &= \theta_0^3 x^3 + \theta_1^2 x^2 + \theta_2^2 x + \theta_3^3 + \theta_4 sin(x), \\ &\quad \theta^* = \{-0.001, 0.1, 0.1, 2, 15\} \\ &\quad s = [1, 10] \end{aligned} \tag{3.28}$$

$$\begin{aligned} f(x,\theta) &= \theta_0 x_0^{\theta_1} + \theta_2 x_1^{\theta_3}, \\ &\quad \theta^* = \{0.1, 4, 0.1, 2\} \\ &\quad s = [1, 10] \end{aligned} \tag{3.29}$$

Note: if a negative variable is raised to a non-integer power only the real part of the result is taken.

$$\begin{aligned} f(x,\theta) &= \theta_0 exp(-x_0/\theta_1) + \theta_2 exp(-x_1/\theta_3) \\ &\quad \theta^* = \{4, 2, 1, 10\} \\ &\quad s = [0.1, 10] \end{aligned} \tag{3.30}$$

3.4.2 Validation (Linear results)

The CGN method should only diverge from the GN method in functions which are non-linear in their parameters. If the parameters are linear, the second derivative of the residual $r_{r,jk} = 0$ and the newly created matrix $s_{ij} = r_{i,j}$. This leads to (3.11) being equivalent to (3.26). Hence, comparing result of both methods in such a linear function can be used as a baseline to see if CGN behaves as expected. An equation which is linear in its parameters is (3.27). There it is expected that both GN and CGN will produce the same results. The outcome of this test can be seen in Fig. 3.5 which shows two identical graphs. This means that CGN worked as expected and all consequent results can be taken as valid. Fig. 3.5 shows graphs where, when there is no noise added, both methods find the answer in one step. This is a known behaviour of the GN method, inherited by CGN. The region of dark blue near the top of both graphs is caused by the increased noise level.

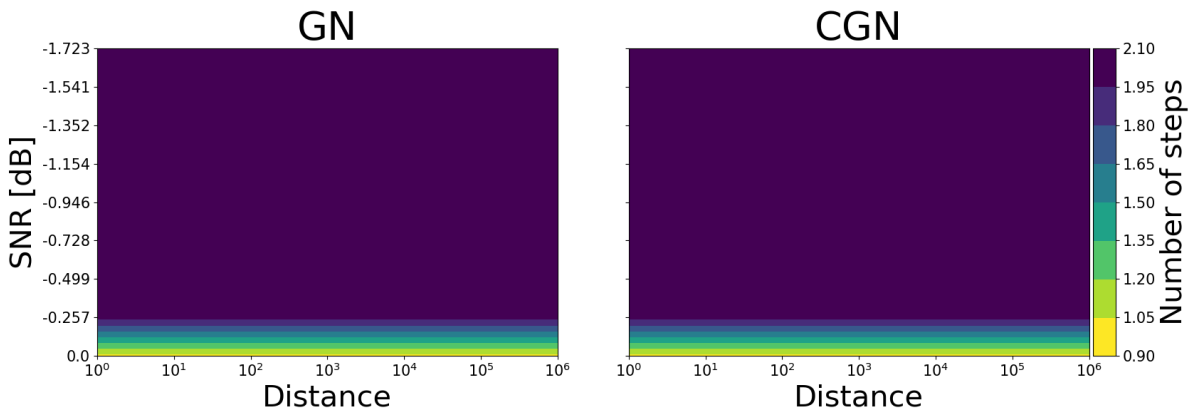

Figure 3.5: The figure shows the results of (3.27). The comparison between GN and CGN was plotted across a range of distances and noise. Both methods produce identical graphs, validating CGN through the linear cases. The information provided by the graphs shows that with the increase of noise both methods need more iterations to pass the tolerance criterion.

3.4.3 Comparison

In the comparison section, the two methods will be contrasted with each other. The first function which will be examined is (3.28). This function is a modification of (3.27), where powers were added to the parameters of x. The function was chosen to represent a classical polynomial case.

Polynomial

The results of the polynomial case are shown in Fig. 3.6. There, CGN consistently requires fewer steps to yield a result. Both methods share the same un-converged areas at large noise levels. They also share the discontinuous convergence area. Near the zero noise level, it can be seen that CGN produces a results with about 8 steps fewer than GN which is equal to a 25% increase in efficiency.

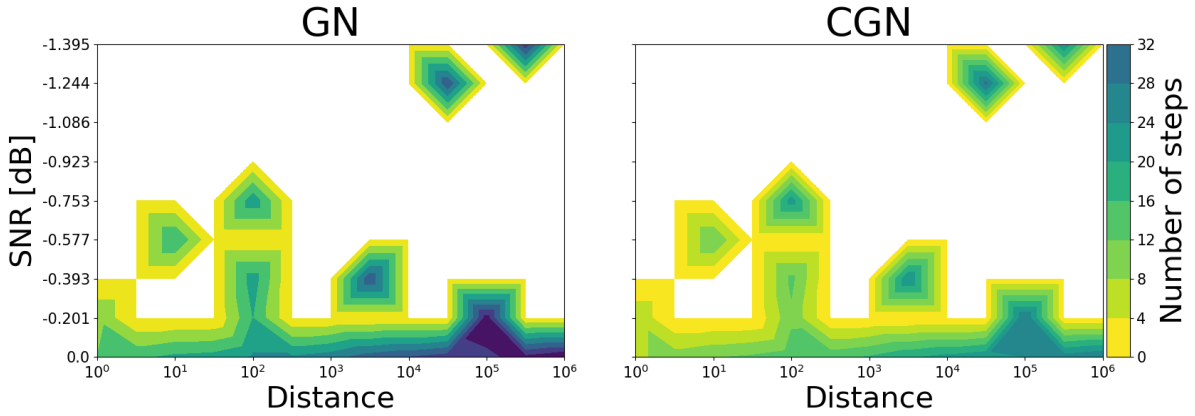

Figure 3.6: The figure shows results of (3.28). The comparison between GN and CGN was plotted across a range of distances and noise. As can be seen, CGN consistently took fewer steps to converge. This is particularly evident at large noise levels, where the CGN converged with almost half of the iteration steps. Both methods share the same un-converged area.

Exponential parameters

The next system to be examined is (3.29). This function has parameters as exponents and two different independent variables. This system was chosen as the initial guess heavily impacts the systems value and as such an initial guess which is far away from the desired value will take much longer to compute. The results of this test are shown in Fig. 3.7. There, it can be seen that CGN provides an expanded

convergence range as compared to GN. CGN provides full coverage on the noise range and covers initial distances of up to 10^2. This is contrasted by GN's coverage of a few discontinuous areas. These areas vary in distance and noise, with a maximum noise level covered of $-0.042dB$.

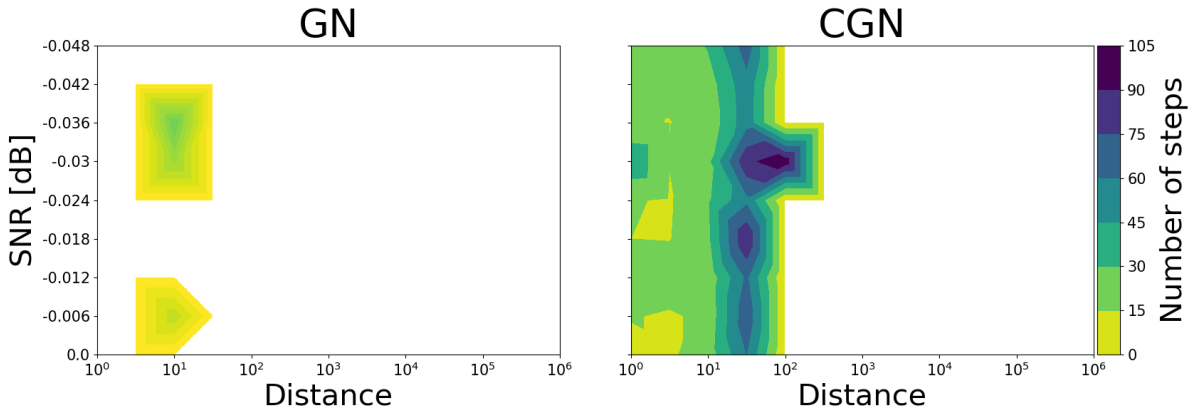

Figure 3.7: This figure shows the system described by (3.29). The system is a representation of a case which has exponential parameters. As can be seen, CGN provides a much larger and more continuous convergence range. As distance increases the number of steps needed increases as well. GN produced only two converged "islands" whose border areas took fewer steps to converge than the middle sections.

Negative exponent in exponential function

The final case to examined here is shown by (3.30). This system has parameters with negative exponents which are used as the argument of an exponential function. In testing this function, it was assumed that all elements of the initial guess are of the same magnitude, which does not show the worst case possible. For comparison purposes this will not affect the results, which are shown in Fig. 3.8. There it can be seen that again CGN takes fewer steps to converge. It also shows that CGN does not have any discontinuities across the test domain. This suggests that CGN can be a valid improvement not only by expanding the convergence basin but also reducing step count.

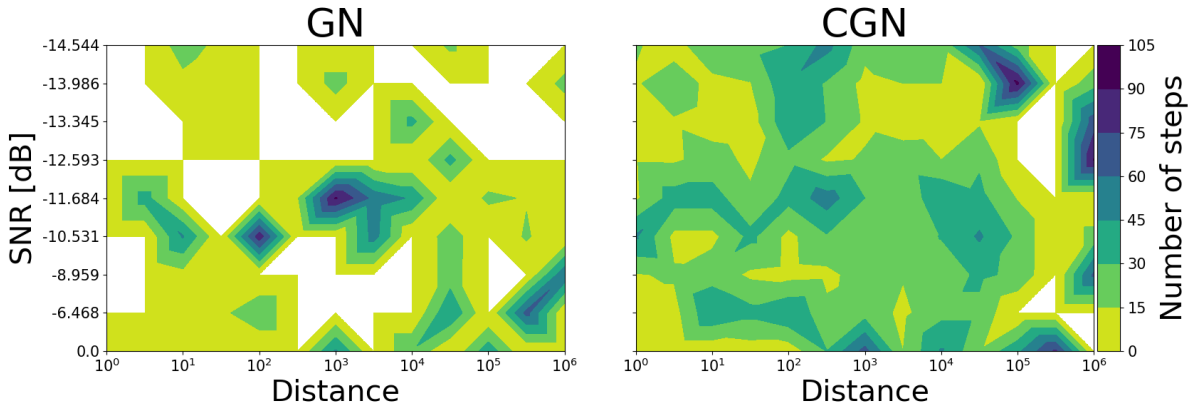

Figure 3.8: This figure shows the results from testing the system described by (3.30). The system represents a case where some of the parameters have negative exponents and are part of an exponential function. This case is similar to ones found in some constitutive models. The graphs show that CGN consistently took fewer steps to converge and provided better coverage. Also, due to large difference in iteration steps, CGN took less computation time on average to produce a result.

3.4.4 Rate and order of convergence

In this section, the rate and order of convergence of the CGN and GN methods will be compared. Both methods were tested with (3.28), the noise level was set at $0.0dB$ and an initial guess of the parameters

was taken as $[10, 10, 10, 10, 10]$. The results of the test are shown in Fig. 3.9 and Fig. 3.10. The first figure, Fig. 3.9, shows the calculated rate of convergence of both algorithms. As can be seen both of the methods exhibit a similar behaviour where the rate of convergence becomes smaller as the algorithm converges. The CGN method consistently provides a lower rate of convergence compared with GN. The second figure, Fig. 3.10, shows the calculated order of convergence of both methods. The order of both varies between 0.9 and 1.0. The order of convergence decreases as the algorithm converges. Looking at the behaviour of both algorithms, it can be deduced that both exhibit super-linear convergence. It is known that GN can reach quadratic convergence [21], but only when using an appropriate line-search algorithm on the GN step, which was not used in this experiment. It can be concluded that the CGN method, due to its consistently lower rate of convergence, provides an improvement to the convergence behaviour of the GN method.

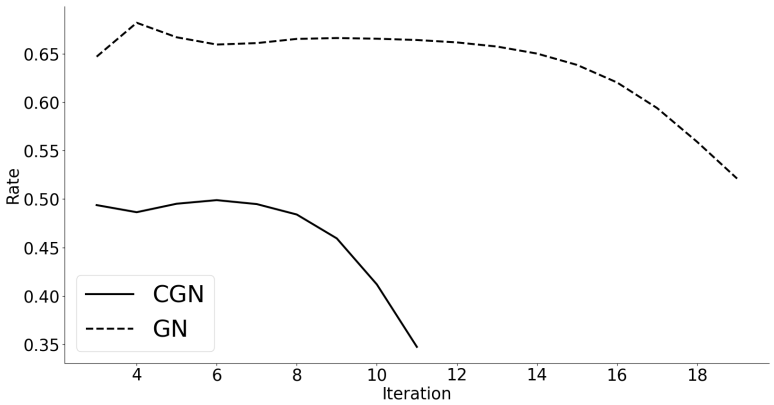

Figure 3.9: The figure shows a comparison between the calculated rate of CGN (solid line) and GN (dashed line). The rate of CGN is consistently lower than that of GN which leads to a quicker convergence. The graph shows the rate past the third iteration because, as explained before, three iterations are required to calculate the first instance of rate.

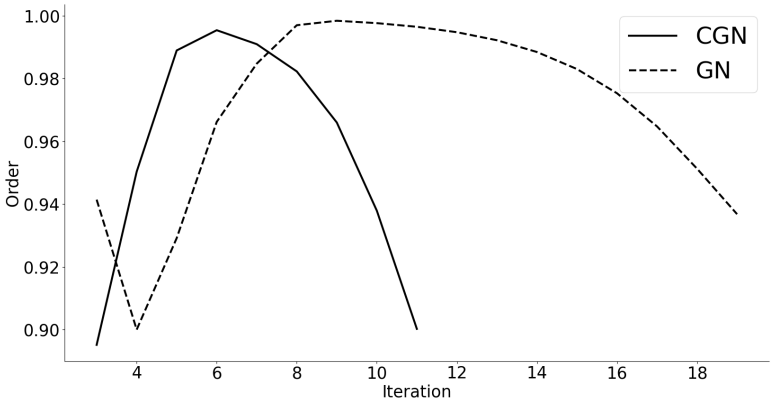

Figure 3.10: The figure shows a comparison between the calculated order of CGN (solid line) and GN (dashed line). The orders of CGN and GN behave similarly, both laying in the range of $[0.9, 1.0]$ with a similar decrease as the methods converge.The graph shows the rate past the third iteration because, as explained before, three iterations are required to calculate the first instance of order.

3.5 Conclusion

In this chapter, the paper has demonstrated that CGN preforms better than GN in a variety of test cases. In simple linear problems both methods perform identically. However, the added computational load of CGN makes it inefficient to use. In more complex systems, such as the three presented in this paper, CGN provides saving in terms of step count and convergence area. The CGN method also shows a lower

rate of convergence and a similar order of convergence to GN, which reinforces the fact that it converges in fewer iteration steps. The computational load of CGN is greater, but this can be offset by the reduced iteration count. In any case, this method can be used to increase the basin of convergence of the GN method and, based on the specific system, yield time savings as well.

Chapter 4

Finite Element Method (FEM)

4.1 Background

This paper has so far proposed improvements to NR which resulted in ENR, used for root-finding, and to GN which resulted in CGN, used for minimisation. Now, it will be shown how these new methods can be applied to forward and inverse models through FEM. This paper is concerned with stress analysis of objects with hyper-elastic behaviour. Hence, stretches λ and stresses P will be examined. The problem which will be solved in the deformation of a bar shown in Fig. 4.1. Assuming that the material is isotropic in the y and z direction and that it undergoes isochoric deformation (constant volume), which is explained later, one can model the bar as a one-dimensional object. Hence, FEM here will be defined for a one-dimension non-linear system. For the forward modelling case the following procedure will integrate

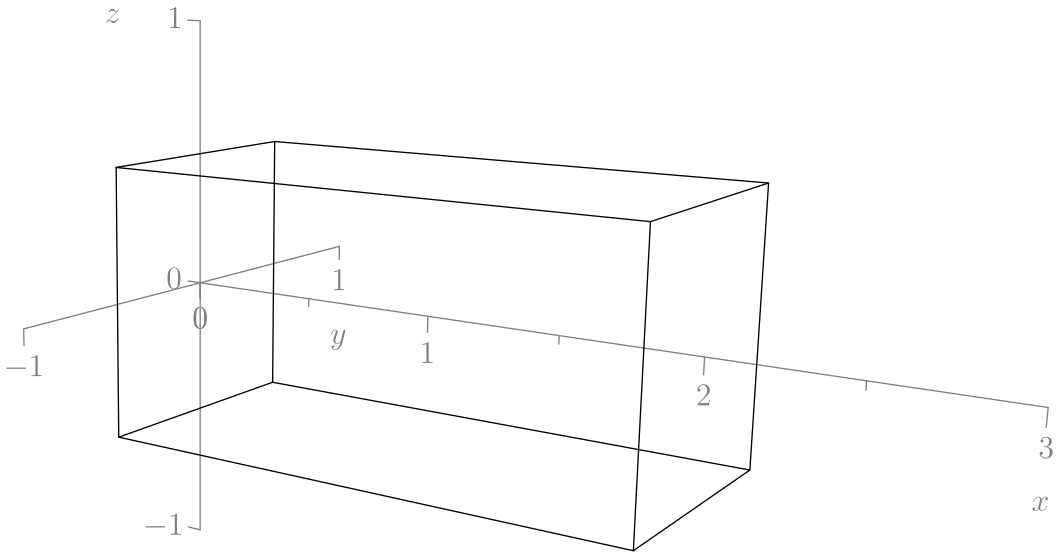

Figure 4.1: A representation of a bar in three dimensions.

ENR with FEM:

- Initially a FEM model will be constructed using the desired object.
- An initial guess of the location of each node will be taken. It will, in all cases coincide with the initial configuration, i.e. the starting guess of the deformed shape will be the un-deformed shape.
- The FEM model will return information which is equivalent to the previously used function vector $f(x)$ and its derivative.
- This information will be used by ENR to produce a better estimate of the locations of all the nodes.
- These new locations will be fed back to the FEM model and the cycle will continue until the resulting function vector $f(x)$ is within an absolute tolerance away from zero. In this case, the model has reached a state of static equilibrium.

For inverse models a similar procedure will be followed, but rather than varying the locations of the elements, the properties of the object will varied. Testing will be done directly on the material model which is equivalent to a FEM model with one element. The procedure is as follows:

- Initially, a FEM model will be constructed using the desired object and its properties.
- An initial guess of the material properties will be used by the FEM model to produce a result.
- The result will be used together with an observed value to calculate the residual.
- A finite difference method will be used to calculate the derivatives of the residual.
- The residual and its derivatives will be used to calculate the CGN step which will then be applied to the initial object properties.
- The new properties will be passed onto the FEM model and the cycle will continue until the residual becomes less than a predetermined tolerance.

4.2 Introduction

Many phenomena in science and engineering can be explained using partial differential equations (PDEs). These equations are constructed using various partial derivatives of a multivariate function together with many independent variables. They are produced naturally and are extensively used in engineering. Examples of where they appear can be seen in the study of groundwater flows or in stress analysis which will be examined in this paper [22]. Solving such PDEs by classical analytical methods for arbitrary shapes is almost impossible [23].

That is why in the 1960s the Finite Element Method (FEM) was developed [22]. FEM is a numerical method with which PDEs can be solved approximately. This is done by changing the domain of the problem from a continuous to a discretised one. During this process, the body which is being examined is broken down into a finite number of elements, from which the method gets its name, connected by nodes. This newly generated system of local sub-problems can be easily solved and then stitched back together to form the global solution.

If the system which will be analysed is already in its discrete form, i.e. a truss structure, a direct FEM can be applied. However, when that is not the case, a general approach is needed, where an arbitrary domain can be discretised.

4.2.1 Derivation

In order to solve a problem of non-linear continua, such as the one at hand, a non-linear formulation of FEM is needed. There are two possible formulations, the Lagrangian and the Eulerian. The difference between the two is in the way the elements, generated by the discretisation of the continuum, behave. Lagrangian elements deform with the material; that is, the location of the nodes is coincident with the material points; whereas Eulerian nodes remain fixed in space, so their location does not change with the material's deformation. The example given in Fig. 4.2 shows this difference graphically. There, it can be seen that in the Lagrangian formulation, because the boundary nodes are coincident with the material points, one has to apply boundary conditions to the nodes, whereas in Eulerian formulation the boundary nodes are not necessarily coincident with the material boundary. Hence, one has to apply boundary conditions at points which are not nodes. Due to this difference, this paper will use the Lagrangian formulation and more specifically the total Lagrangian formulation.

As described before, one cannot directly discretise a domain. There are specific steps that need to be taken. Like with linear FEM models, firstly, the strong form of the system needs to derived, which includes the governing equations and the boundary conditions. Then, the weak form has to be derived through integration of the strong form with reduced requirements for continuity. Afterwards, the model can be discretised and solved.

4.2.2 Nomenclature

As the element is expected to experience large strains, a way to describe both the current and initial material locations of the material points is needed. Hence, the initial locations will be denoted by X and

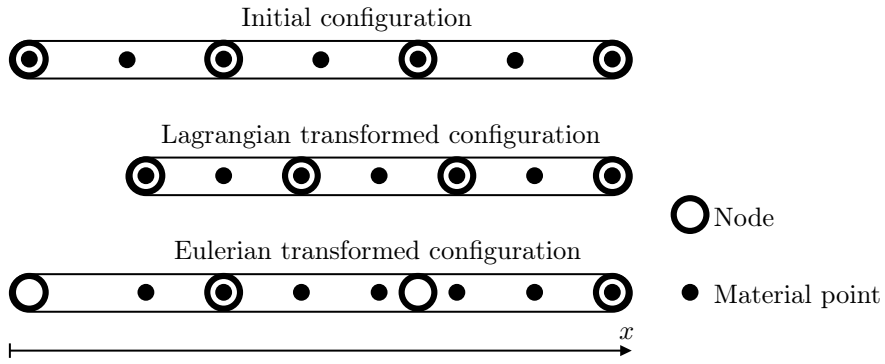

Figure 4.2: Figure showing the difference between Lagrangian transformations and Eulerian Transformations.

the current locations will be denoted by x. In both cases, these parameters show a location along the element. The way to convert from one to the other can be done by a function called $\phi(X,t)$ which will act a map between x and X. As can be seen, ϕ is a function of time as well as a function of the location. This is necessary because the system reacts to the applied loads and deformations, and changes the internal forces. In the case of static equilibrium, which is explored here, t does not need to represent actual time, but can instead just be a monotonically increasing variable. Because an iterative method will be used to solve the non-linear system, t will coincide with the number of iterations. The final relation of x, ϕ and X can then be written out as:

$$x = \phi(X,t) \tag{4.1}$$

The displacement of the object will be denoted by $u(X,t)$ which is known as the trial function and is equal to the difference between x and X. From this, one can derive the deformation gradient F with regards to the material domain as:

$$F = \frac{\partial \phi}{\partial X} \tag{4.2}$$

Taking the determinant of the deformation gradient gives the Jacobian determinant J, which is:

$$J = det(F) \tag{4.3}$$

and shows the volumetric change of the object. The deformation gradient is a measure of deformation and it can be converted to a parameter known as a stretch, which will be used by the material model to relate stretches to stresses. This is done by using the Polar Decomposition Theorem. The conversion is based on the fact that if the deformation gradient F is non-singular with a positive determinant, it can be expressed as the product of a rigid body rotation R and a deformation U [24] as:

$$F = RU \tag{4.4}$$

The stretch matrix, used to establish the deformation gradient, is defined as:

$$U = (F^T F)^{0.5} \tag{4.5}$$

The fractional power of a matrix is used in terms of spectral representation. It is computed by first transforming the matrix into its principal coordinates, where the matrix is diagonal with its eigenvalues on the diagonal. The fractional power is applied to all the diagonal terms, and the matrix is transformed back. The rotation R can then be given by:

$$R = FU^{-1} \tag{4.6}$$

If F shows the deformation gradient in three dimensions then, because U is positive definite and symmetric, it will have three real positive eigenvalues λ_1, λ_2 and λ_3, called the principal stretches, and a corresponding triplet of orthonormal eigenvectors r_1, r_2 and r_3. In the case at hand, no rotation will happen, so:

$$R = FU^{-1} = I \tag{4.7}$$

From this it can be seen that if no rotation exists, $U = F$. Hence, the components of F in the principal basis are the principal stretches. In the one-dimensional case examined here, the principal stretch of each component can be calculated directly by setting:

$$\lambda_1^e = \frac{x_2^e - x_1^e}{X_2^e - X_1^e} \tag{4.8}$$

The ratio of the stretches shows what type of deformation the object has undergone. Previously, it was mentioned that the bar object will undergo isochoric (constant-volume) deformation, which now can be defined in terms of stretches as:

$$J = \lambda_1 \lambda_2 \lambda_3 = 1 \tag{4.9}$$

4.2.3 Stress

Stress in the total Lagrangian formulation is taken as the first Piola-Kirchhoff stress P which is equal to:

$$P = \frac{\partial W}{\partial F} - pF^{-1} \tag{4.10}$$

where p denotes the Lagrangian multiplier, which represents hydro-static pressure, and W is the strain energy density given by the material model. When a material model is defined, one can solve for the parameter p and use it to find the stress tensor P.

4.2.4 Governing equations

There are four governing equations which affect the system at hand [9]. These are:

- Conservation of mass
- Conservation of momentum
- A measure of deformation
- A constitutive equation which is a material model that relates deformation to stresses

There is an additional compatibility requirement, which in this case is that the deformation has to be continuous (C^1 in the strong form and C^0 in the weak form). For stress analysis, conservation energy can be ignored as deformation as a process can dissipate energy as heat but it will be assumed that the effects of that energy loss are negligible.

Conservation of mass

The conservation of mass equation can be written as:

$$\rho J = \rho_0 \tag{4.11}$$

This equation requires that the mass of the original configuration is the same as the mass of the current configuration. The variable ρ shows the current density of the object, whereas ρ_0 shows the initial density.

Conservation of momentum

Because the area of the element is taken as a constant along the length of the beam, the conservation of momentum formula becomes:

$$P(X,t)_{,X} + \rho_0 b_i = \rho_0 u_{,tt} \tag{4.12}$$

where:

$$P(X,t)_{,X} = \frac{\partial P(X,t)}{\partial X} \tag{4.13}$$

In the case of static equilibrium, $\rho_0 u_{,tt}$ vanishes.

4.3 One-dimensional case

In the one-dimensional case, some of the above definitions are modified. The material is assumed to undergo isochoric (constant volume) deformation. Hence, the deformation gradient will become:

$$F = diag\left[\lambda, \frac{1}{\sqrt{\lambda}}, \frac{1}{\sqrt{\lambda}}\right] \tag{4.14}$$

Also, in the one-dimensional case there are no stresses in the y and z direction, which means that the stress vector P becomes:

$$P = diag[P^x, 0, 0] \tag{4.15}$$

This allows the Lagrangian multiplier p to be determined. Moreover, the one-dimensional case modifies the governing equations.

Final governing equation

From the above definitions, it can be seen that the equations which govern the behaviour of the object in 1D static equilibrium applications are the conservation of momentum and the material model. The later can be substituted into the former to produce the final governing equation:

$$P^x(X,t)_{,X} + \rho_0 b = 0 \tag{4.16}$$

The above is a boundary value problem and is the target of the FEM model. To finalise the formulation of the problem, the boundary conditions and initial conditions are needed.

Boundary and Initial conditions

In the one-dimensional case, the boundary consists of two points which are situated at the ends of the domain. In the model, these points are represented by X_a and X_b, shown in Fig. 4.4. A boundary will be denoted by Γ. There are two types of boundary, the first is called a displacement boundary and is denoted by Γ_u. The other boundary is called a traction boundary where the traction force is applied. It is denoted by Γ_t. Traction and displacement cannot be prescribed at the same boundary. Furthermore, to be able to properly describe the traction boundary, the unit normal n^0 has to be defined. It has a value of -1 at X_a and a value of 1 at X_b. Values prescribed at the boundaries are denoted by a superposed bar. With these definitions the boundary conditions can be written as:

$$u = \bar{u} \; on \; \Gamma_u \tag{4.17}$$

$$n^0 P = \bar{t^0} \; on \; \Gamma_t \tag{4.18}$$

The traction force here has a superscript $(.)^0$ to distinguish it from time and to show that it is defined over the original area of the bar. In the one-dimensional bar case, the prescribed boundaries are:

$$u(X_a,t) = 0 \; on \; \Gamma_u \tag{4.19}$$

$$n^0 P(X_b,t) = P(X_b,t) = \bar{t^0} \; on \; \Gamma_t \tag{4.20}$$

The final set of definitions needed are the initial conditions at $t = 0$. As the body is initially at rest, displacement u and the velocity $\dot{u}$ are zero.

Continuity

The continuity of the trial solution u and the later shown test function δu is vital to the functioning of FEM. That is why the topic of continuity will be examined here.

A function is considered to be C^n continuous if its n-th derivative exists and is continuous over the entire domain. The derivative of a C^n function will be C^{n-1} continuous. If a function C^0 is piece-wise continuously differentiable, its derivative C^{-1} is piece-wise continuous, i.e. C^{-1} is continuous except at specific points called "jumps". The location of these jumps in the C^{-1} function correspond to the location of "kinks" in the C^0 function. A kink is a sharp change is the gradient of the function. Kinks are not exclusive to C^0 and they can also exist in C^{-1} functions. A C^1 function has no kinks or jumps and is continuously differentiable. The differences between these functions can be seen in Fig. 4.3 and are summarised by Table 4.1.

Function	Kinks	Jumps	Type
C^{-1}	Yes	Yes	piece-wise continuous
C^0	Yes	No	piece-wise continuously differentiable
C^1	No	No	continuously differentiable

Table 4.1: Continuity of functions based on C^n

The strong form of FEM, which will be formalised below, requires that the trial solution and test function are C^1 continuous, whereas the weak formulation only requires that they are C^0 continuous. That is why the weak formulation is called "weak" because it reduces the continuity requirements for u and δu.

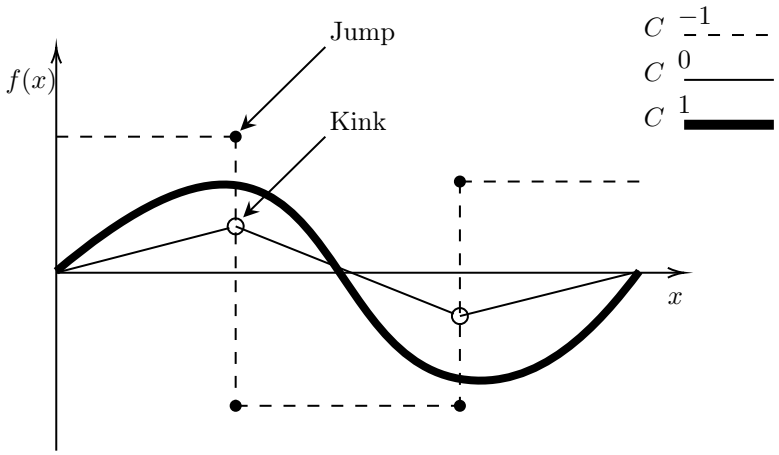

Figure 4.3: A graph showing the differences between different continuity levels.

4.3.1 The Strong form

Using the definition from above, the strong form of the domain can be defined for the one-dimensional bar under static loading as follows:

$$P^x(X,t)_{,X} + \rho_0 b = 0 \tag{4.21}$$

$$u(X_a,t) = 0 \; on \; \Gamma_u \tag{4.22}$$

$$n^0 P(X_b,t) = P(X_b,t) = \bar{t^0} \; on \; \Gamma_t \tag{4.23}$$

To solve the system the strong from has to be converted to the weak form. This will be shown in the next section.

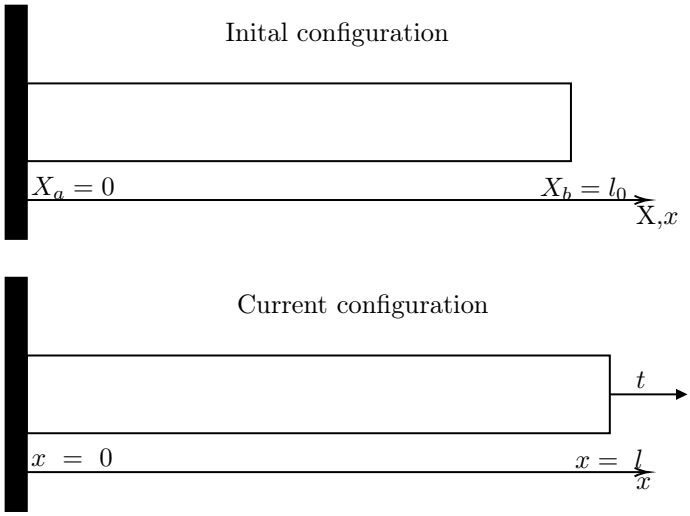

Figure 4.4: Figure showing boundary conditions of a 1D bar.

4.3.2 The Weak Form

The momentum equation cannot be discretised directly by the finite element method. In order to discretise this equation, a weak form, often called the principle of virtual work, is needed [23]. It is equivalent to the momentum equation and the traction boundary conditions. To derive the weak form, the strong form has to be multiplied by some arbitrary function $\delta u(X)$, known as the test function. The arbitrary nature of this function makes sure that the weak and strong form are equivalent. The weak form also has to satisfy the previously described continuity requirements and to vanish on Γ_u. After the multiplication, the integral of both sides can be taken over the domain of the object. This will yield the following:

$$\int_{X_a}^{X_b} \delta u \left[P_{,X} + \rho_0 b\right] dX = 0 \tag{4.24}$$

In the above, the short hand P was used to denote the result of the material model evaluated at X and at time t. The first term of the equation can be separated and the whole equations can be rearranged to:

$$\int_{X_a}^{X_b} \delta u P_{,X} dX + \int_{X_a}^{X_b} \delta u \, \rho_0 b \, dX = 0 \tag{4.25}$$

The first term can be expanded using the fundamental theorem of calculus. This will produce:

$$\delta u n^0 P\Big|_{\Gamma} - \int_{X_a}^{X_b} \delta u_{,X} P dX + \int_{X_a}^{X_b} \delta u \, \rho_0 b \, dX = 0 \tag{4.26}$$

Evaluating the first term across Γ and combining the two separate integrals yields the weak form of the momentum equation and the the traction boundary. The test function disappears at Γ_u and the nominal stress P at Γ_t is equal to the prescribed traction force $\bar{t^0}$. Therefore, the final version of the weak form is:

$$\delta u \bar{t^0}\Big|_{\Gamma_t=0} + \int_{X_a}^{X_b} [\delta u \, \rho_0 b - \delta u_{,X} P] \, dX = 0 \tag{4.27}$$

The above equation can be separated into two parts. The first:

$$\delta W^{ext} = \int_{X_a}^{X_b} \delta u \, \rho_0 b \, dX + \delta u \bar{t^0}\Big|_{\Gamma_t=0} \tag{4.28}$$

shows the external virtual work done on the bar and the second:

$$\delta W^{int} = \int_{X_a}^{X_b} \delta u_{,X} P \, dX = \int_{X_a}^{X_b} \delta F P \, dX \tag{4.29}$$

which shows the internal virtual work. Using δW^{ext} and δW^{int}, the weak form can be rewritten as:

$$\delta W = \delta W^{ext} - \delta W^{int} = 0 \tag{4.30}$$

where δW denotes virtual work.

4.3.3 Finite Element Discretisation

With the weak form of the problem, the domain can be discretised. This is done by using the finite element interpolation of the test and trial functions. The domain of the problem, which is contained within $[X_a, \, X_b]$, is broken down into e elements, where $e \in [1, n_e]$ with $n_e + 1$ nodes. The nodes are denoted by X_I, where $(,)_I$ shows the index of the nodes. The nodes of the generic element will be denoted by X_i^e, where $(.)^e$ shows the element and $i \in [1, \, m]$ shows the index of the nodes, and m shows the number of nodes per element. In the one-dimensional bar case, $m = 2$. The total domain is denoted by Ω, whereas the domain of each element, comprising of $[X_1^e, \, X_2^e]$, is denoted by Ω_e.

Approximation of the trial solution and test function

For completeness, the approximation of the trial solution for an element $u^e(X, t)$ has to be at least a linear function to meet the completeness condition. This approximation at a node will be called a nodal displacement which is:

$$u^e(X, t) = c_1^e(t) + c_2^e(t) X \equiv \hat{u}^e(t) \tag{4.31}$$

where the $\hat{u}$ denotes the approximation of trial function evaluated at a particular X. The time variable t will be omitted from here on. As a one-dimensional element has two nodes and two nodal parameters, there are an equal number of nodes X_1, X_2 and nodal parameters c_1, c_2. That means that the nodal parameters can be uniquely expressed in terms of nodes. The matrix of nodes can then be described as:

$$M_{ij}^e = \begin{cases} 1 & j = 1 \\ X_j^e & j = 2 \end{cases} \tag{4.32}$$

The element parameter vector c^e has to be such that (4.31) is true. That means:

$$c^e = [c_1^e \; c_2^e]^T = \left[\frac{u_2^e X_1^e - u_1^e X_2^e}{X_1^e - X_2^e} \; \frac{u_1^e - u_2^e}{X_1^e - X_2^e} \right]^T \tag{4.33}$$

With M_{ij}^e and c_j^e defined, the next step will be to define the vector of nodal displacements d^e. This vector will be equal to the product of M_{ij}^e and c_j^e. That is expressed by:

$$d_i^e = M_{ij}^e c_j^e \tag{4.34}$$

Then, a way will be needed to relate the displacement $u^e(X,t)$ to the nodal displacements d_i^e. The general equation of displacement (4.31) can be rearranged in matrix form to:

$$u^e(X,t) = p_i(X)c_i^e(t) \tag{4.35}$$

where $p_i(x)$ is equal to 1 when $i = 1$ and equal to X when $i = 2$. Now, both equations (4.34) and (4.35) can be rearranged in terms of c. This means that the following system can be created:

$$c_i^e = d_i^e(M_{ij}^e)^{-1} \tag{4.36}$$

$$c_i^e = u^e(X,t)p_i^{-1}(X) \tag{4.37}$$

Setting both right-hand sides equal to each other, the relation between $u^e(X,t)$ and d_i^e can be expressed as:

$$u^e(X,t) = p_i(X)(M_{ij}^e)^{-1}d_j^e \tag{4.38}$$

From the above equation, the first two terms can be isolated and set equal to a new vector N^e. That vector is known as the shape function. It is equal to:

$$N_i^e = p_j(X)(M_{ij}^e)^{-1} = \begin{bmatrix}1 & X\end{bmatrix} \frac{1}{X_2^e - X_1^e} \begin{bmatrix}X_2^e & -X_1^e \\ -1 & 1\end{bmatrix} = \frac{1}{l_0^e}\begin{bmatrix}X_2^e - X & X - X_1^e\end{bmatrix} \tag{4.39}$$

From here, the final form of the general element displacement can be found as:

$$u^e(X,t) = N_i^e d_i^e = \frac{1}{l_0^e}\begin{bmatrix}X_2^e - X & X - X_1^e\end{bmatrix}\begin{bmatrix}\hat{u}_1^e \\ \hat{u}_2^e\end{bmatrix} \tag{4.40}$$

The rate of displacement $u_{,X}^e(X,t)$ can be found by taking the derivative of the above. As $\hat{u}$ is a function of time, only the first terms will be affected by the derivation. Hence, it will take the form of:

$$u_{,X}^e(X,t) = N_{i,X}^e d_i^e = \frac{1}{l_0^e}\begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}\hat{u}_1^e \\ \hat{u}_2^e\end{bmatrix} \tag{4.41}$$

The same method for deriving the general element trial solution can be applied to the test function. It will yield the following:

$$\delta u^e(X,t) = N_i^e \delta d_i^e = \frac{1}{l_0^e}\begin{bmatrix}X_2^e - X & X - X_1^e\end{bmatrix}\begin{bmatrix}\hat{\delta u_1}^e \\ \hat{\delta u_2}^e\end{bmatrix} \tag{4.42}$$

$$\delta u_{,X}^e(X,t) = N_{i,X}^e \delta d_i^e = \frac{1}{l_0^e}\begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}\hat{\delta u_1}^e \\ \hat{\delta u_2}^e\end{bmatrix} \tag{4.43}$$

Global approximation

Once all local approximations have been derived, they have to be converted back into the global domain. This process is known as scattering. It distributes all local elements Ω^e for $e = 1, 2, ..., n_e$ to the global domain Ω. This is done using the connectivity matrix L_e. The same matrix is also used in the process called gathering, which relates the global nodal displacements to the local ones. The element nodal displacements are related to global nodal displacements by:

$$u^e = L^e u \tag{4.44}$$

$$u = \sum_{e=1}^{n_e} (L^e)^T u^e \tag{4.45}$$

The connectivity matrix is a Boolean matrix, which means it only has 0 and 1 elements and has a shape of $m \times n_e + 1$ in the one-dimensional case. This matrix shows where the position of the local element's nodes X_i^e are in the global node vector X_I. This matrix has to be constructed on a per element basis for

each specific discretisation of the domain Ω. The global function can also be approximated by the global nodal values by using global shape functions. Hence, the trial solution $u(X,t)$ can be approximated by:

$$u(X,t) = \sum_{I=1}^{n_e+1} N(x)_I u(t)_I \tag{4.46}$$

where the global shape function $N(x)_I$ is equal to:

$$N(x) = \sum_{e=1}^{n_e} (L^e)^T N(x)_i^e \tag{4.47}$$

The same procedure can be applied to the test function δu which will yield:

$$\delta u(X,t) = \sum_{I=1}^{n_e+1} N(x)_I \delta u(t)_I \tag{4.48}$$

Nodal forces

The virtual work terms of the weak form can be expressed as the product of a virtual displacement and a virtual force. This will allow for the derivation of the victual force elements:

$$\delta W = \sum_{I=1}^{n_e+1} \delta u_I f_I \tag{4.49}$$

where f_I represents the virtual force at the node I and δu_I represents the virtual displacement at the same node. The above can be applied to any of the specific work elements to derive their force components. For internal work the equation is:

$$\delta W^{int} = \sum_{I=1}^{n_e+1} \delta u_I f_I^{int} = \int_{X_a}^{X_b} \delta u_{,X} P \, dX \tag{4.50}$$

The test function $\delta u(X,t)_{,X}$ can be replaced by its equivalent nodal components $N(x)_I$ and $\delta u(t)_I$. As the $\delta u(t)_I$ term is a function of time, it can be taken away from the integral as a constant. This will yield:

$$\int_{X_a}^{X_b} \delta u_{,X} P \, dX = \sum_{I=1}^{n_e+1} \delta u(t)_I \int_{X_a}^{X_b} N_{I,X} P \, dX \tag{4.51}$$

Substituting this back into (4.50), the following can be derived:

$$\delta W^{int} = \sum_{I=0}^{n_e+1} \delta u_I f_I^{int} = \sum_{I=0}^{n_e+1} \delta u(t)_I \int_{X_a}^{X_b} N_{I,X} P \, dX \tag{4.52}$$

Here, because of the arbitrary nature of the test function, it can be deduced that:

$$f_I^{int} = \int_{X_a}^{X_b} N_{I,X} P \, dX \tag{4.53}$$

The same procedure is applied to the external virtual work vector δW^{ext} and the virtual external force vector f^{ext} can be defined as:

$$f_I^{ext} = \int_{X_a}^{X_b} N_I \rho_0 b \, dX + N_I \bar{t^0}\Big|_{\Gamma_t=0} \tag{4.54}$$

These global force vectors can be found from the elemental force vectors similarly to what was shown in (4.45), as:

$$f_I = \sum_{e=1}^{n_e} (L^e)^T f_i^e \tag{4.55}$$

This will allow for the derivation of an equation that will give the local element force. The above will be expressed for a specific force, in this case the virtual internal force:

$$f_I^{int} = \sum_{e=1}^{n_e} (L^e)^T (f^{int})_i^e = \sum_{e=1}^{n_e} (L^e)^T \int_{X_1^e}^{X_2^e} N_{i,X}^e P \, dX \tag{4.56}$$

From the above it can be seen that the local element internal virtual force:

$$(f^{int})_i^e = \int_{X_1^e}^{X_2^e} N_{i,X}^e P \, dX \tag{4.57}$$

The same procedure is applied to the external force and this will yield:

$$(f^{ext})_i^e = \int_{X_1^e}^{X_2^e} N_i^e \rho_0 b \, dX + N_i^e \bar{t}^0 \Big|_{\Gamma_t=0} \tag{4.58}$$

Solving the element nodal forces requires the evaluation of an integral. This integral may not be solvable analytically. That is why numerical integration will be used. A popular choice for solving integrals in FEM is Gaussian-Quadrature. It is used because of the accuracy it provides when evaluating the integral of polynomials which are frequent in FEM.

4.3.4 Solution procedure

The weak form was expressed with the help of virtual work terms in (4.30). Later, it was shown that the virtual work terms are equal to the product of a virtual displacement and a virtual force. This will allow for (4.30) to be rewritten as:

$$\sum_{I=1}^{n_e+1} \delta W_I = \sum_{I=1}^{n_e+1} \delta u_I f_I^{ext} - \delta u_I f_I^{int} = 0 \tag{4.59}$$

The common factor δu can be taken out to yield:

$$\sum_{I=1}^{n_e+1} \delta u_I (f_I^{ext} - f_I^{int}) = 0 \tag{4.60}$$

Now, the displacement boundary conditions can be enforced. They were defined as:

$$\begin{aligned} u &= \bar{u} \; on \; \Gamma_u \\ \delta u &= 0 \; on \; \Gamma_u \end{aligned} \tag{4.61}$$

Because the test function δu vanishes on Γ_u and is zero everywhere else, the following can be deduced:

$$f_I^{ext} - f_I^{int} = 0, \; I = 2, 3, ..., n_e + 1 \tag{4.62}$$

From here the final solution procedure can be constructed. It is as follows:

1. Gather element nodal displacements u^e
2. Compute the measure of deformation F
3. Compute stress via the material model
4. Compute internal nodal forces
5. Compute external nodal forces
6. Compute the total force per element: $f^e = f_e^{ext} - f_e^{int}$
7. Scatter element nodal forces to global matrices.
8. Check if solution is reached by: $f_I^{ext} - f_I^{int} = 0, \; I = 2, 3, ..., n_e + 1$
   If yes, then u is the actual displacement vector.
   Otherwise, go to step 1.

4.4 Hyper-Elastic Constitutive models

Many different constitutive models can be developed for non-linear elasticity. Also, because many different stress and deformation measures for finite strain are available, the same constitutive equations can be written out in multiple ways. This section will cover the derivation of two constitutive models which will be used by the FEM model to test the proposed improvements. The models are the compressive Mooney-Rivlin (MR) model [25] and the Veronda-Westmann (VW) model. MR is a hyper-elastic material model, originally developed to describe the behaviour of rubber, currently used to explain various material behaviours such as the behaviour of polyurethane [6] which is extensively used as a bearing material in bridges and under rail tracks [26]. VW is a model commonly used for the response of biological tissues [4]. All models will be expressed in terms of the first Piola-Kirchoff stress, denoted by P, and principal stretches.

4.4.1 Mooney-Rivlin

The compressive Mooney-Rivlin (MR) model defines the strain energy density $W(F)$ as:

$$W(F) = \frac{\mu}{2}\left[\nu(J^{-2/3}I_1 - 3) + (1-\nu)(J^{-4/3}I_2 - 3)\right] + \frac{K}{2}(\ln J)^2 \tag{4.63}$$

In the above, μ is the effective shear modulus, K is the bulk modulus, and ν is a dimensionless parameter which belongs to $[0, 1]$. I_n represents the invariants of the unimodular component of the left Cauchy-Green deformation tensor. F is the deformation gradient and $J = det(F) = \lambda_1 \times \lambda_2 \times \lambda_3 = 1$. This entire expression can be simplified for uniaxial stretch and isochoric deformation for P as:

$$P(\lambda) = \mu\nu\left(\lambda - \frac{1}{\lambda^2}\right) + \mu(1-\nu)\left(1 - \frac{1}{\lambda^3}\right) \tag{4.64}$$

4.4.2 Veronda-Westmann

The Veronda-Westmann (VW) model was chosen for its exponential behaviour. It defines strain energy density as:

$$W(F) = \frac{A}{B}\left[exp(B(J^{-2/3}I_1 - 3)) - 1\right] - \frac{A}{2}\left(J^{-4/3}I_2 - 3\right) + \frac{K}{2}(\ln J)^2 \tag{4.65}$$

In the above, K represents the bulk modulus and A and B are stiffness parameters. The stress equation is:

$$P(\lambda) = 2A\left(\lambda - \frac{1}{\lambda^2}\right) exp\left(B\left(\lambda^2 + \frac{2}{\lambda} - 3\right)\right) - A\left(1 - \frac{1}{\lambda^3}\right) \tag{4.66}$$

4.5 Results

4.5.1 Forward Models

Methods

The FEM model was constructed using 5 linear elements to discretise a one-dimensional bar of length 2m. The system is modelled as one-dimensional as the deformation is assumed to be isochoric and the example material is taken to be isotropic in the y and z axes of the coordinate system. The discretised system is shown in Fig. 4.5. There, each vertical rectangle represents a node of the discretised object, each internal arrow represents a body force, and the final external arrow shows the traction force. They will not necessarily be applied to each configuration and specificities will be given for each test case. All models will have a prescribed displacement of $u(0) = 0$ on Γ_u

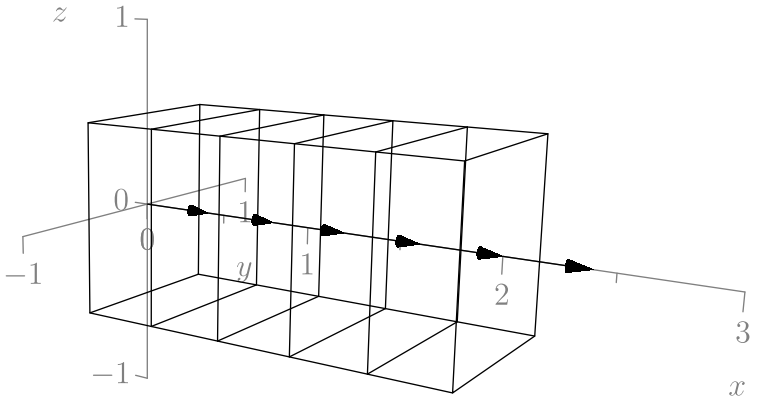

Figure 4.5: Base FEM discretisation

Validation

The validation is done by testing NR and ENR on a constitutive model such that both methods successfully converge. One such model can be taken to be the linear elastic model. This model relates stresses and stretches, which will internally be converted to strains, using a constant parameter known as the modulus of elasticity E. In this configuration, it is expected that both solving methods will find an answer

within the pre-determined iteration limit for sufficiently small applied loadings. The full configuration of the test case is shown in Table 4.2. In order for ENR to work with FEM, the distance between each pair of elements of c needs to be different, which is why a parameter ζ is added to each element. ζ is a small random variable of the order of 10^{-3}. Using this configuration, both methods were able to produce

Property	Value
Constitutive model	Linear Elastic
Elastic modulus E	$100Pa$
Length	$2m$
Body loading	0
Traction loading	20Pa
Initial guess x_0	$x_0 = X$
c value	$c = 2X + \zeta$

Table 4.2: Table showing the test configuration for a linear elastic constitutive model.

a result in 1 iteration. This means that ENR works as expected in the linear case and will produce valid results. The total length of the deformed member was found to be $2.4m$, which can also be verified by hand. Knowing that E relates stresses and strains, the following relation can be written out:

$$\epsilon = \frac{\sigma}{E} = \lambda - 1 \tag{4.67}$$

The total length of the new member can then be found using the definition of λ as the ratio of deformed length to original length. Hence:

$$l = l_0\lambda = l_0(\epsilon + 1) = 2.0 \times \left(\frac{20}{100} + 1\right) = 2.4m \tag{4.68}$$

where l_0 represents the original length of the member and l shows the deformed length. The deformed shape of the member has been drawn in Fig. 4.6.

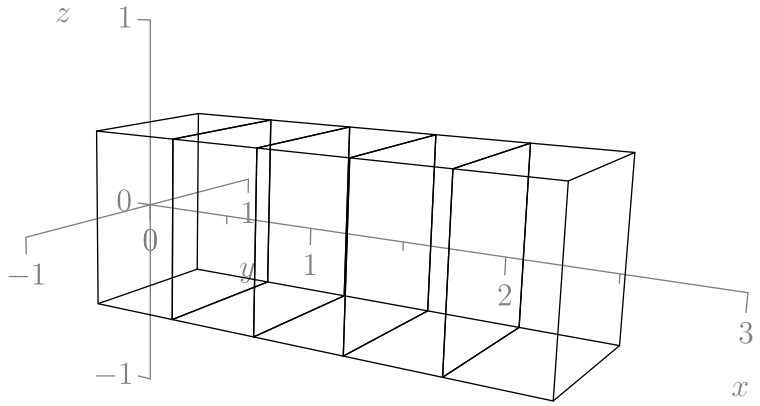

Figure 4.6: Validation FEM discretisation

Comparison

In this section, the performance of ENR and NR will be compared in FEM models using hyper-elastic constitutive models. The first comparison will be using the Veronda-Westmann model, which exhibits non-linear behaviour in tension. That is why only tensile loading (positive) will be applied.

Veronda-Westmann The VW model has two parameters, A and B, which define the behaviour of a material. These parameters are stiffness parameters and will be set to:

$$\begin{aligned} A &= 2.48446MPa \\ B &= 0.16860 \end{aligned}$$

which represent typical values for modelling a silicone rubber material [10]. The specific configuration for this test will be shown in Table 4.3. With this configuration, ENR converged to a solution in 3

iteration steps, whereas NR took 6 steps to converge. The choice of c affected this result heavily. It was found that if c was taken as $c = 2X + \zeta$, ENR would not converge at all. This lead to further testing of c configurations which revealed that, for this particular case, in order for ENR to converge, c must be taken as at least $c = 3.2X + \zeta$. Past that value, tested up to a value of $c = 10X + \zeta$, ENR converges within 3 iteration steps. The reason for this is probably the exponential nature of VW. When an exponent is taken at a small enough value, it produces an answer which often lays within the predetermined tolerance check. This leads an iterative algorithm to converge to a false root. This behaviour can be seen in Fig. 2.4, where both methods with different configurations were unable to converge when an initial guess was taken in the negative x_0 range. The deformed shape produced by this configuration is shown in Fig. 4.7.

Property	Value
Constitutive model	VW
A	2.48446 MPa
B	0.16860
Length	$2m$
Body load	5MPa/m
Traction load	5MPa
Initial guess x_0	$x_0 = X$
c value	$c = 4X + \zeta$

Table 4.3: Table showing the test configuration used for the Veronda-Westmann example.

There, the total length of the deformed bar is $4.451m$.

In the next step of the testing, the loadings were increased in order to make convergence more difficult to achieve. Hence, the body and traction load was taken as $20MPa$. This caused NR to converge in 34 steps, whereas ENR took 8 steps to converge. In this test, the elongation was 319.16%. Further increasing the load to $30MPa$ as a body and traction loading caused NR to converge in 69 steps, whereas ENR took 9 steps to converge. The elongation was at 346.7%. These results shows that with a adequate choice of c, ENR can outperform NR by up to 7 times the iteration count. Now, the behaviour of ENR

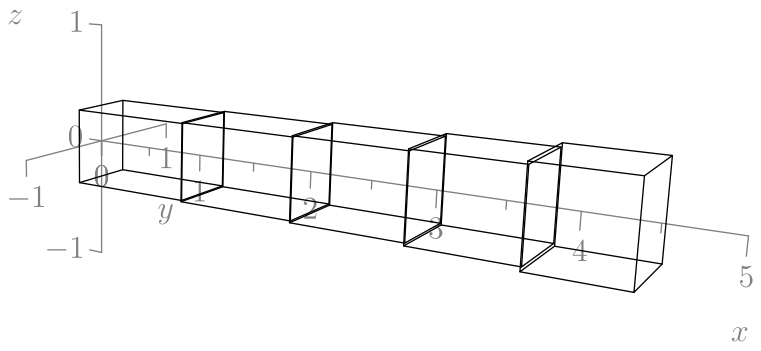

Figure 4.7: The figure shows the deformed shape of the test model using the VM material model and the loadings shown in Table 4.3.

will be examined in a model which behaves in a hyper-elastic manner when under compressive (negative) loads.

Mooney-Rivlin The Mooney-Rivlin constitutive model has a wide variety of usage cases, one of which is the modelling of polyurethane. The MR model has two parameters, μ and ν. Their values for polyurethane are $5.289MPa$ and 0.6417 respectively [6]. Those were used with a compressive body and traction load of $5MPa$ to test the performance of NR and ENR. The c value was set equal to $0.5X+\zeta$. The full configuration of the experiment is presented in Table. 4.4. The test result shows that, for this configuration, ENR produced a result in 4 iteration, whereas NR failed to converge. The deformed shape from the FEM model is presented in Fig. 4.8. These results show that the modification, present in ENR, provides an improvement to the convergence. The next step in the testing of MR was to increase the loading and see at which points the ENR method would fail. The next loading configuration was taken as $20MPa$ in compression both as a body and traction load. This caused ENR to fail. The configuration of c was then modified to $c = 0.4X + \epsilon$ which enabled ENR to converge in 6 steps. This result, together with the results from paragraph 4.5.1, can be interpreted as bracketing behaviour of ENR. In paragraph 4.5.1

Property	Value
Constitutive model	MR
mu	5.289MPa
nu	0.6417
Length	$2m$
Body loading	-5MPa/m
Traction loading	-5MPa
Initial guess x_0	$x_0 = X$
c value	$c = 0.5X + \zeta$

Table 4.4: Table showing the test configuration used for the Mooney-Rivlin example.

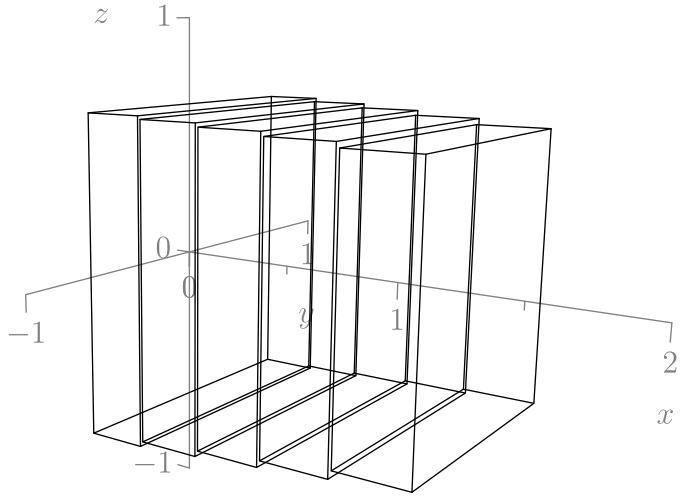

Figure 4.8: The figure shows the deformed shape of the test model using the MR material model and the loadings shown in Table 4.4.

as extension cases were tested, ENR converged in fewer steps than ER when the guess of the modification parameter c, if interpreted as geometrical location, laid beyond the deformed object. In this section, the same case arises when testing compressive loads and the MR model.

Modification parameter analysis The above results created a need for a method of estimating an adequate value of c, which will enable one to use ENR without multiple configuration testing. The two material models were tested independently in an attempt to extrapolate a formula that will provide a suitable choice of c. To test MR, the configuration presented in Table 4.4 was used together with a range of c parameters starting from $c = 0.1X + \zeta$ and finishing at $c = X + \zeta$ with a step change to the variable in front of X, called ϕ, equal to 0.1. This yielded the graph shown in Fig. 6.2 in the Appendix, where the horizontal axis represents ϕ and the vertical shows the number of steps taken. There it can be seen that after $\phi = 0.5$ the method converges with an increasing number of iterations, meaning that in order for ENR to converge, $phi < 0.5$. Interpreted geometrically, this will lead to $c < 1m$. This can be compared with the compressed length of the member which is $0.970m$. The result shows that, in order for ENR to converge, a choice of c which places it beyond the actual deflected configuration is needed.

The same analysis was done for the VW model. In this case, the loading was taken as $20MPa$ in tension both as a body load and traction load and the parameters A and B were taken to be the same as in Table 4.3. The initial value for ϕ was set to one and a step increase of 0.5 was used until $\phi = 10$ was reached. The result from this experiment are shown in Fig. 6.1. There, it can be seen that value of ϕ beyond which ENR converges is 4, meaning that $c > 4X + \zeta = 8m$. This can be compared with the deflected length of the member which in this case came out to be $6.383m$. This shows that c must be taken such that it extends beyond the correct answer by 25%.

From the data gathered from these experiments, it can be seen that setting a c value that lays beyond the deformed length yields consistently good results. A specific equation for this value was not derived as it is application specific. One strategy to always get a result is to over-estimate the deflection and use that as the value of c, i.e. in the MR model one can set $c = 0.01X + \zeta$ and guarantee convergence,

except when the compressed object's length becomes less 1% of the original. As was shown by Fig. 6.2, it may not be the most efficient way to choose c, but yields consistent results nonetheless.

4.5.2 Inverse Models

Methods

Inverse model testing will be based on the previously described CGN and GN methods. Results here will be presented in the same way they were presented in section 3.4. However, rather than using a set of equations, the above described material models will be adopted as test cases.

Validation

The validation of the CGN method will be done by comparing its results to the GN method in the linear case. Here, this will be the linear elasticity equation $\sigma = E\epsilon$, where ϵ represents the strain which is internally derived from the stretch λ. The results from this test are presented in Fig. 4.9. As can be seen, the two methods provided the same results which validate the CGN method in the linear elasticity case. Furthermore, the figure shown here is similar to the one shown in subsection 3.4.2.

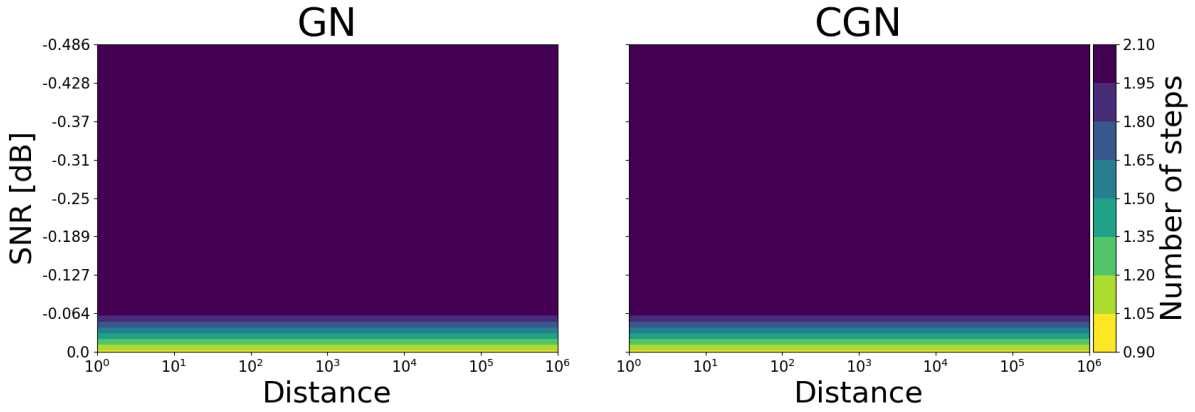

Figure 4.9: Figure showing comparison between CGN and GN when tested on a linear elastic model.

Comparison

In this section, the performance of CGN and GN will be compared. The first comparison case will be with the VW material model using the parameters shown in paragraph 4.5.1. Ten samples were taken from the test range of $[2, 10]$ to generate the test data. All tests were done on the constitutive equations directly rather than on a FEM model which uses them. However, the result shown are still valid when applied to a FEM model.

Veronda-Westmann As can be seen in Fig. 4.10, the CGN reduces the number of steps needed in regions of low noise levels. It also increases the converged area in regions of noise level higher than $-0.22dB$. This shows that CGN improves the convergence properties when applied to the VW material model.

Mooney-Rivlin The second test that will be made is on the MR material model. Here, ten samples were taken from the test range $[0.3, 0.9]$ to generate the test data. The test range was changed to represent the compressive range of the MR model. The results of the test are shown in Fig. 4.11. There it can be seen that CGN provides an increased convergence area. However, the iteration count at low distances has increased. This shows that CGN improves the convergence area when applied to the MR material model.

4.6 Conclusion

This chapter showed how a non-linear FEM model can be derived and how the two corrected methods can be used to create forward and inverse models. The test result for forwards models revealed that the ENR

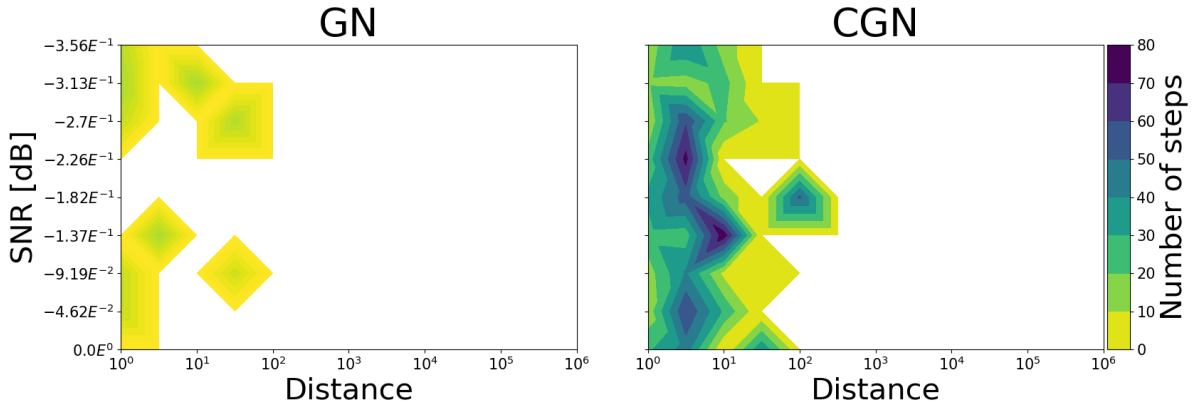

Figure 4.10: Figure showing comparison between CGN and GN when tested on the Veronda-Westmann material model. The CGN method produced a larger coverage of the test range. However, it also increased the iteration count at lower distances.

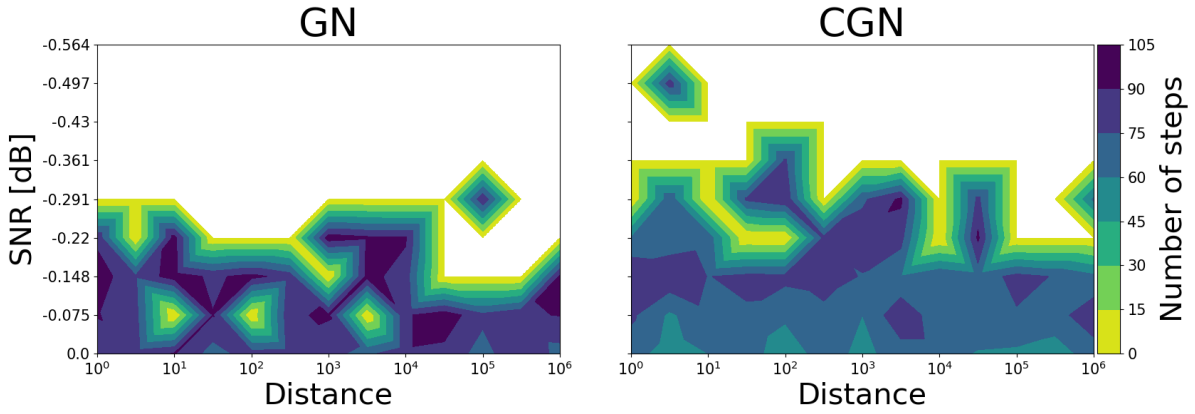

Figure 4.11: Figure showing comparison between CGN and GN when tested on the Mooney-Rivlin material model. The CGN method produced a larger coverage of the test range and decreased the number of iterations needed to reach a result across the test range.

method improves the convergence behaviour of the FEM model when the materials exhibits hyper-elastic behaviour. This chapter, furthermore, showed the extrapolation of a strategy for selecting an adequate value of the modification parameter c, which allows ENR to converge for configurations which cause NR to fail. The CGN was also shown to provide an improvement over the standard GN method in highly non-linear systems in either total coverage of the test range or in integration count or both depending on the system.

Chapter 5

Discussion & Conclusion

5.1 Discussion

The aim of this paper was to propose new methods which can improve the convergence of forward and inverse finite element models. The paper has shown that the two proposed methods possess improved convergence properties when compared to their original counterparts. These improved convergence properties can then translate to improved convergence of the overarching forward and inverse models that this paper began with. When constructing a forward model of an object that is made of a material that is hyper-elastic, the usage of the extended Newton-Raphson method can greatly improve the convergence properties of the model. The degree to which this improvement will be realised is based on the choice of the parameter c, for which a strategy has been proposed. The improvement added by ENR can be twofold. It can, firstly, reduce the iteration count needed to find an answer and, secondly, it can expand the convergence basin of the problem allowing for more extreme configurations to be tested.

When creating an inverse model, the same improvements can be seen when using the corrected Gauss-Newton method. It was shown in multiple examples that the method yielded a wider convergence range and a reduced iteration count. The CGN method also proved to be more resilient to noise in the data. This can make the CGN a vital tool when creating inverse models using data that has a high degree of noise.

Future development of the methods proposed by this paper can be concentrated on finding alternative formulation of the non-linear modification function. A few different functions were tested in the development of this paper. However, their performance was inadequate due to their insufficient continuity. The non-linear modification can also be applied to other root-finding methods such as BFGS. During the course of this project, an attempt was made to create an multivariate version of the secant method which does not require a large number of starting choices. This was achieved via taking the main diagonal of the approximation of the Jacobian matrix. A similar strategy has been proven to work in quasi-Newtonian methods [27], so pursuing a simplified multivariate version of the secant method may yield valuable results. A line search algorithm can be applied to the CGN step to further improve the convergence behaviour. It is known that the standard GN method can approach quadratic convergence with an appropriate line search. Hence, if an appropriate line search is used with CGN better than quadratic convergence could be achieved due to CGN's improved rate.

5.2 Conclusion

In conclusion, this paper has covered the improvement of the Newton-Raphson root-finding algorithm through a non-linear modification aimed to reduce the non-linearity of a function. It also covered how the Gauss-Newton step can be corrected to produce improved convergence behaviour. Finally, the paper showed how to derive a non-linear version of FEM and apply the two improved approaches to forward and inverse models.

This paper showed that the ENR method can perform better than NR in a variety of highly non-linear functions. When ENR was applied to a FEM model, the paper showed that for the same initial starting point as NR, ENR was able to converge to a solution in fewer steps. For some configurations in which NR failed to converge, ENR was able to produce an answer from the same starting point. This paper, additionally, showed a bracketing behaviour exhibited by ENR when it comes to the choice of c. This led

to the development of a strategy for selecting c which produced consistent results.

The paper also demonstrated that the CGN method extends the convergence basin of GN in a variety of highly non-linear functions, yet performs identically in the linear case. CGN was also shown to reduce the iteration count needed to reach an answer in some cases. When applied to the two material models examined by this paper, CGN was, again, able to produce a larger convergence basin and a reduced step count. This suggests that it can be used as an alternative to GN when dealing with highly non-linear systems, such as those presented in this paper.

Both improved methods require extra computational resources. However, their convergence properties may make them a suitable choice for analysis of plastics, soft tissues and other materials which exhibit similar hyper-elastic behaviour.

Bibliography

[1] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, 2006.

[2] A. Galántai. The theory of newton's method. *Journal of Computational and Applied Mathematics*, 124(1):25 – 44, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[3] BARBARA BLASCHKE, ANDREAS NEUBAUER, and OTMAR SCHERZER. On convergence rates for the Iteratively regularized Gauss-Newton method. *IMA Journal of Numerical Analysis*, 17(3):421–436, 06 1997.

[4] Yue Mei, Daniel E. Hurtado, Sanjay Pant, and Ankush Aggarwal. On improving the numerical convergence of highly nonlinear elasticity problems. *Computer Methods in Applied Mechanics and Engineering*, 337:110 – 127, 2018.

[5] P. A. L. S. Martins, R. M. Natal Jorge, and A. J. M. Ferreira. A comparative study of several material models for prediction of hyperelastic properties: Application to silicone-rubber and soft tissues. *Strain*, 42(3):135–147, 2006.

[6] Piotr Szurgott and Lukasz Jarzebski. Selection of a hyper-elastic material model - a case study for a polyurethane component. *Latin American Journal of Solids and Structures*, 16, 01 2019.

[7] Ankush Aggarwal and Sanjay Pant. Beyond newton: A new root-finding fixed-point iteration for nonlinear equations. *Algorithms*, 13(78), 2020.

[8] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[9] T Belytschko, WK Liu, B Moran, and K Elkhodary. *Nonlinear Finite Elements for Continua and Structures*, chapter 1,2. John Wiley & Sons, Ltd, 2014.

[10] Pedro Martins, Renato Natal Jorge, and Antonio Ferreira. A comparative study of several material models for prediction of hyper-elastic properties: Application to silicone-rubber and soft tissues, 12 2012.

[11] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[12] Sahil Moza. sahilm89/lhsmdu: Latin hypercube sampling with multi-dimensional uniformity (lhsmdu), July 2020.

[13] Niels Henrik Abel. Démonstration de l'impossibilité de la résolution algébrique des équations générales qui passent le quatrième degré. *Œuvres Complètes de Niels Henrik Abel*, 2, 1826.

[14] R. B. Kellogg, T. Y. Li, and J. Yorke. A constructive proof of the brouwer fixed-point theorem and computational results. *SIAM Journal on Numerical Analysis*, 13(4):473–483, 1976.

[15] Adam B. Levy. *Attraction in Numerical Minimization*, pages 33–38. Springer, Cham, 2018.

[16] Joanna M. Papakonstantinou and Richard A. Tapia. Origin and evolution of the secant method in one dimension. *The American Mathematical Monthly*, 120(6):500–518, 2013.

[17] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, 1955.

[18] Guorong Wang, Yimin Wei, and Sanzheng Qiao. *Structured Matrices and Their Generalized Inverses*, pages 225–231. Springer Singapore, Singapore, 2018.

[19] George Matsaglia and George P. H. Styan. Equalities and inequalities for ranks of matrices. *Linear and Multilinear Algebra*, 2(3):269–292, 1974.

[20] Jared L. Deutsch and Clayton V. Deutsch. Latin hypercube sampling with multidimensional uniformity. *Journal of Statistical Planning and Inference*, 142(3):763 – 772, 2012.

[21] Ake Bjorck. *9. Nonlinear Least Squares Problems*, pages 339–358. SIAM, 1996.

[22] Glyn James. *Advanced modern engineering mathematics*, chapter 9, pages 723–724, 802–814. Pearson Prentice Hall, 3 edition, 2011.

[23] Jacob Fish and Ted Belytschko. *A First Course in Finite Elements*, chapter 1-5. John Wiley & Sons, Ltd, 2007.

[24] Rohan Abeyaratne. Continuum mechanics, volume ii of lecture notes on the mechanics of solids, 2012.

[25] M. Mooney. A theory of large elastic deformation. *Journal of Applied Physics*, 11(9):582–592, 1940.

[26] K Naidu, J Sai, Maruthi Vinay, M Reddy, M Chandra, and Smita Singh. Comparative study of materials for reinforcement in bridge pier bearings. *International Journal of Civil Engineering and Technology*, 10:912–919, 04 2019.

[27] Neculai Andrei. Diagonal approximation of the hessian by finite differences for unconstrained optimization. *Journal of Optimization Theory and Applications*, 185, 06 2020.

Chapter 6

Appendix

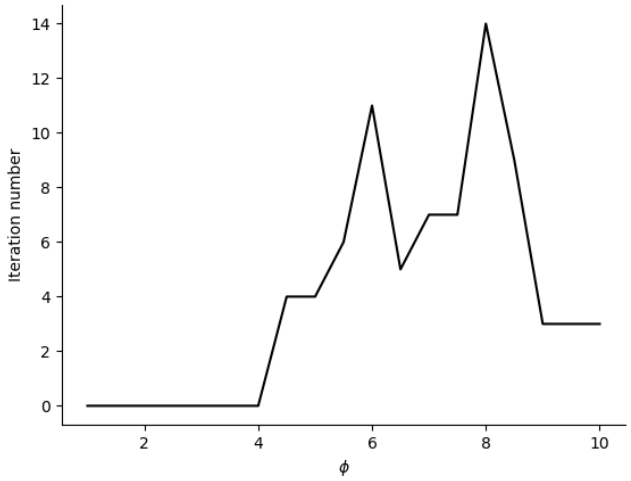

Figure 6.1: The figure shows the tested range for the ϕ parameter for the VW material model.

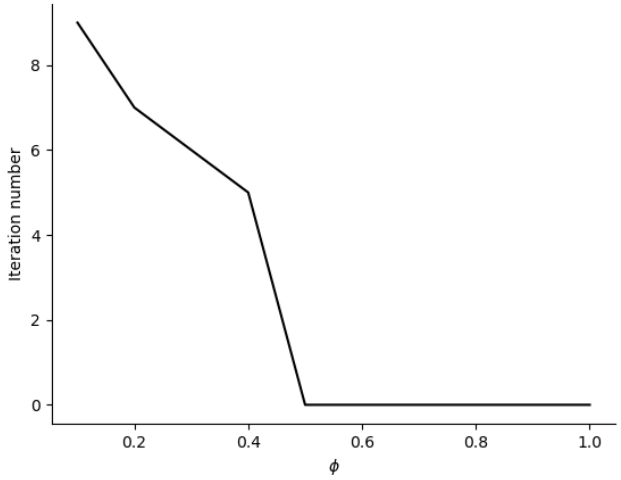

Figure 6.2: The figure shows the tested range for the ϕ parameter for the MR material model.

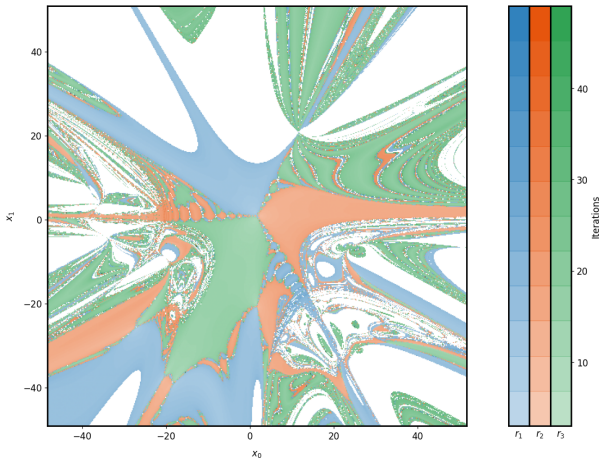

Figure 6.3: Figure showing results of (2.56) for $c = (10, 20)$

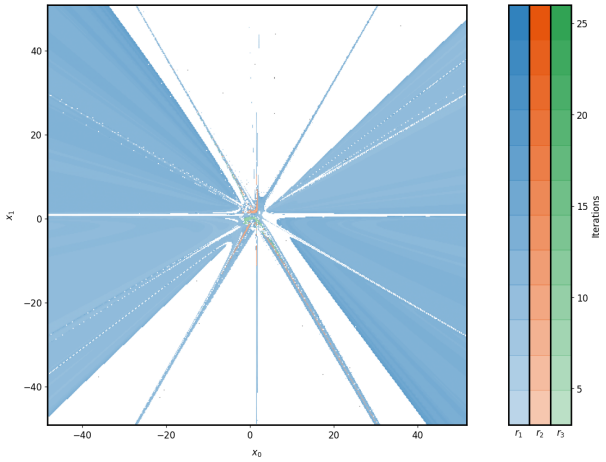

Figure 6.4: Figure showing results of (2.56) for $c = (2, 1)$

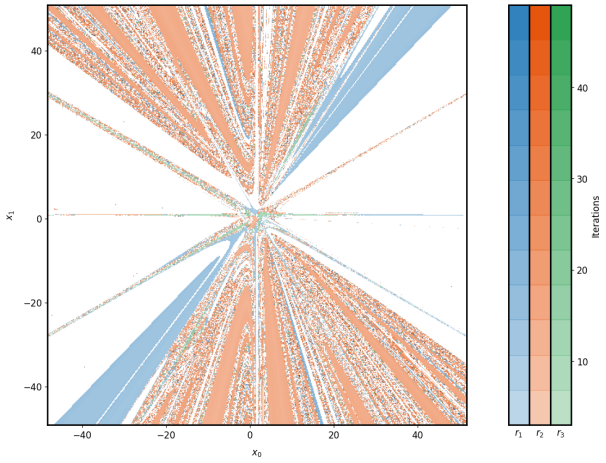

Figure 6.5: Figure showing results of (2.56) for $c = (1, 2)$

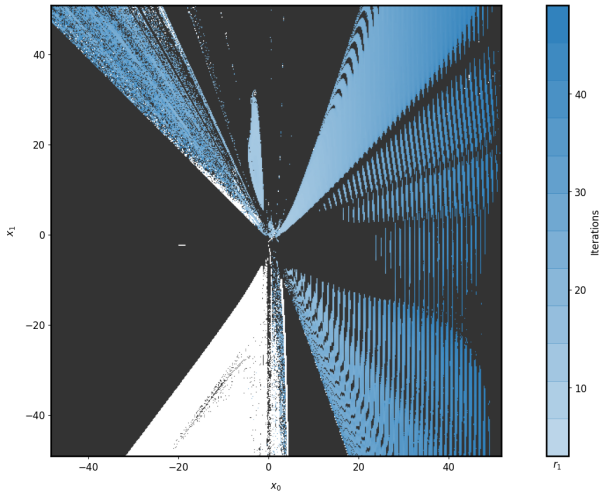

Figure 6.6: Figure showing results of (2.57) for $c = x - 10^{-5}$

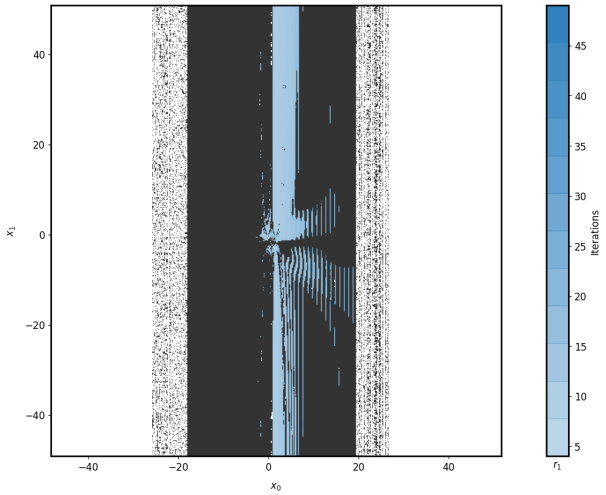

Figure 6.7: Figure showing results of (2.57) for $c = x^2$

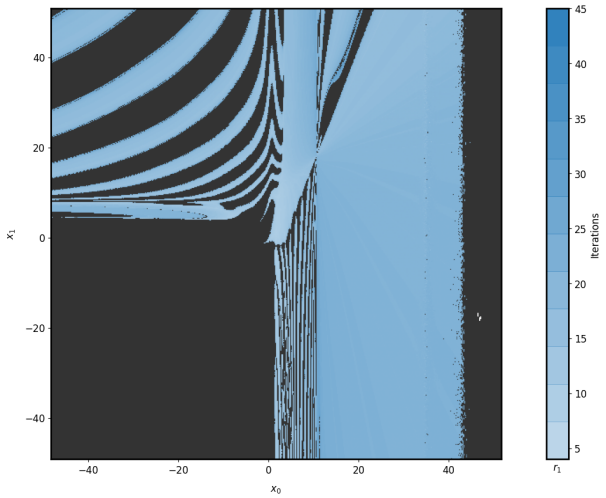

Figure 6.8: Figure showing results of (2.57) for $c = (10, 20)$

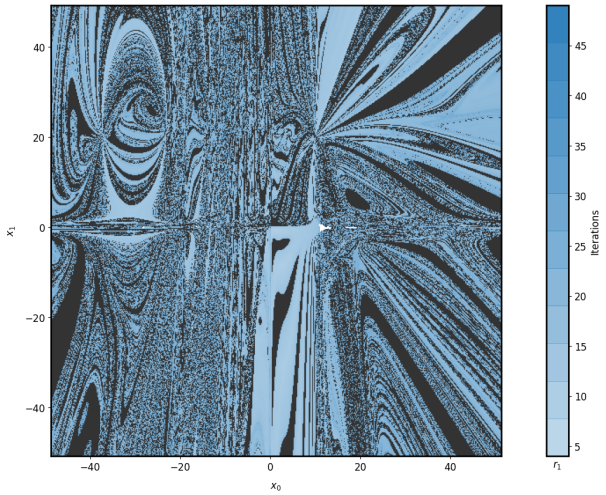

Figure 6.9: Figure showing results of (2.58) for $c = (10, 20)$

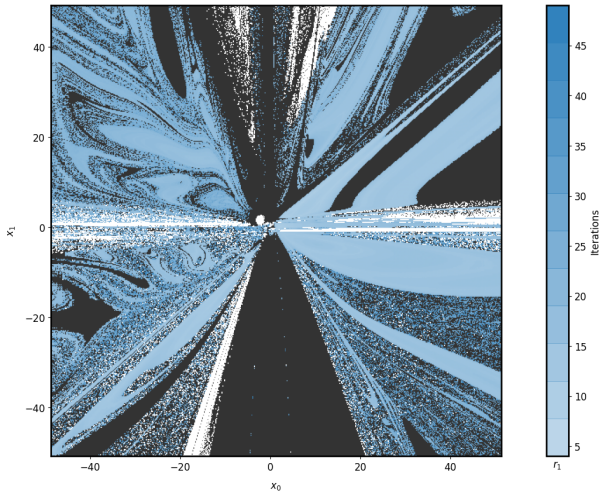

Figure 6.10: Figure showing results of (2.58) for $c = x - 10^{-5}$

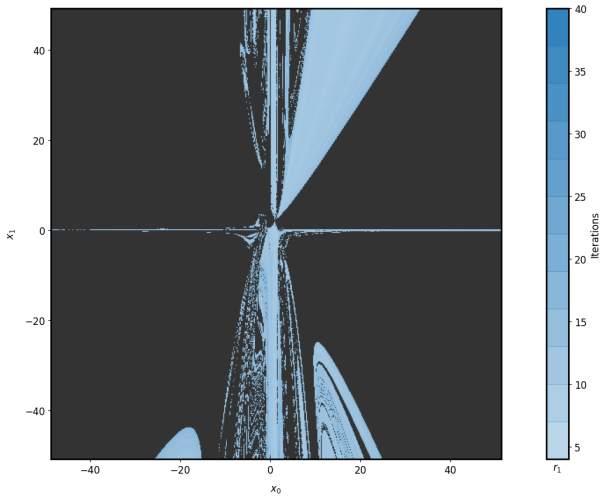

Figure 6.11: Figure showing results of (2.58) for $c = (1, 2)$