NONLINEAR PROGRAMMING SOLVERS FOR HYBRID FINITE ELEMENT

A Thesis in

Civil Engineering

by

Payal Thukral

© 2017 Payal Thukral

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2017
The thesis of Payal Thukral was reviewed and approved* by the following:

Konstantinos Papakonstantinou  
Assistant Professor of Civil Engineering  
Thesis Advisor  

Gordon P. Warn  
Associate Professor of Civil Engineering  

Ali Memari  
Professor of Civil Engineering  

Patrick J. Fox  
John A and Harriette K. Shaw Professor  
Department Head of Civil & Environmental Engineering  

*Signatures are on file in the Graduate School
ABSTRACT

The focus of this work is on analyzing and developing nonlinear solvers for performing nonlinear structural analysis for large displacements in both elastic and inelastic cases. The response of a structure to a load application is shown by its equilibrium path which may include snap back and snap through behavior. Material and geometric nonlinearities are taken into consideration while developing the response path of a multi-element structure with sections discretized using fiber elements. Traditionally, Newton’s method is employed for solving the system of nonlinear equations but it comes with certain challenges. The response determination becomes difficult when stiffness matrix becomes singular at turning point. It also requires the calculation of the inverse of a Hessian matrix, which is costly. Newton’s method gives quadratic convergence but as the scale of the structure increases, resorting to Newton’s method becomes difficult.

These limitations motivate us to explore new solvers. Hence, in this study we analyze and develop various nonlinearly constrained optimization solvers for a recently suggested hybrid finite element. In particular, we compare the performance of conjugate gradient method with or without preconditioning, Sequential Quadratic Programming method and augmented Lagrangian method. For the case of structural response with snap back and snap through behavior, a new method called the implicit path continuation method is developed to ensure path continuation and solution convergence. The various solvers are then validated by obtaining responses of three benchmark structural problems with large displacements and rotations, and comparing the results with the conventional Newton’s method and a variant of Newton’s method with submatrices.
TABLE OF CONTENTS

List of Figures ......................................................................................................................... vi
List of Tables ........................................................................................................................... viii
Acknowledgements .................................................................................................................. ix

Chapter 1 INTRODUCTION ................................................................................................... 1

1.1 General ....................................................................................................................... 1
1.2 Research ..................................................................................................................... 3
1.3 Tasks .......................................................................................................................... 4

Chapter 2 LITERATURE REVIEW ...................................................................................... 5

2.1 Finite Element Analysis ............................................................................................. 5
   2.1.1 Displacement based finite element method ..................................................... 6
   2.1.2 Force based finite element method .................................................................. 7
2.2 Nonlinear Programming ............................................................................................. 8
   2.2.1 One Dimensional Unconstrained Optimization ............................................... 11
   2.2.2 Unconstrained optimization ............................................................................ 13
   2.2.3 Constrained optimization ................................................................................ 25

Chapter 3 PROBLEM STATEMENT ..................................................................................... 31

3.1 Finite Element problem .............................................................................................. 31
3.2 Method 1- Newton’s method ..................................................................................... 37
3.3 Method 2-Newton’s method with submatrices .......................................................... 39
3.4 Method 3-Conjugate gradient method with and without Preconditioning ................. 43
3.5 Method 4- Sequential quadratic programming method .............................................. 44
   3.5.1 Null space calculation ..................................................................................... 46
   3.5.2 Preconditioning of matrix ................................................................................. 46
   3.5.3 Algorithm to solve Problem 2 ......................................................................... 47
   3.5.4 Hybrid Finite element problem ....................................................................... 48
3.7 Method 5-Augmented Lagrangian method ................................................................ 49
3.8 Iterative scheme for traversing snap back and snap through ...................................... 50
3.8.1 Method 6 - Arc Length method with Newton's iteration .................................. 50
3.8.1.1 Rik's method [17] .......................................................................................... 54
3.8.1.2 Crisfield's method [15] .................................................................................. 54
3.8.2 Method 7 - Implicit path continuation: Modified Homotopy method ............. 56
   3.8.2.1 Homotopy - Motivation for this method ......................................................... 56
   3.8.2.2 Implicit path continuation method for post critical behavior determination for hybrid finite element analysis ................................................................. 58

Chapter 4 RESULTS ....................................................................................................... 63
   4.1 Cantilever with vertical load at free end ............................................................... 63
   4.2 Cantilever with moment at free end ..................................................................... 66
   4.3 Toggle frame ......................................................................................................... 67
   4.4 Lee's Frame ......................................................................................................... 70

Chapter 5 CONCLUSIONS ............................................................................................. 72
LIST OF FIGURES

Figure 1 Material Nonlinearity ................................................................................................ 2
Figure 2 Geometric Nonlinearity ............................................................................................. 2
Figure 3 Coordinate system, displacement and deformations ................................................. 6
Figure 4 Local and global minima ........................................................................................... 10
Figure 5 Golden section method .............................................................................................. 12
Figure 6 Descent direction ....................................................................................................... 13
Figure 7 Steepest descent iterates ............................................................................................ 14
Figure 8 Minimum of a quadratic function.............................................................................. 15
Figure 9 Comparison of steepest descent method and conjugate gradient method ................. 17
Figure 10 Contours of a quadratic function with high condition number ............................. 18
Figure 11 Conjugate gradient method...................................................................................... 20
Figure 12 Newton's method iterates ......................................................................................... 22
Figure 13 Quasi Newton method iterates using DFP ............................................................... 23
Figure 14 Reissner beam theory .............................................................................................. 32
Figure 15 Finite element description and discretized fiber elements in a section.................... 32
Figure 16 Load displacement curves showing snap through and snap back behavior (LP: Limit Point, TP: Turning Point) ................................................................................................. 51
Figure 17 Load control increment step .................................................................................... 51
Figure 18 Crisfield arc length method ..................................................................................... 52
Figure 19 Zero path curve ....................................................................................................... 57
Figure 20 Predictor-corrector iterate for implicit path continuation method ......................... 60
Figure 21 Cantilever with vertical load .................................................................................... 64
Figure 22 Load-displacement curve for cantilever with vertical load for elastic case............. 64
Figure 23 Load-displacement curve for cantilever with vertical load for inelastic case........... 65
Figure 24 Cantilever with moment at free end ........................................................................ 66
Figure 25 Deformed shape for cantilever with moment applied at end.........................66
Figure 26 Load-displacement curve for Cantilever with moment at free end for elastic case.67
Figure 27 Toggle frame with point load ......................................................................68
Figure 28 Load-displacement curve for toggle frame for elastic case .........................68
Figure 29 Load-displacement curve for toggle frame for inelastic case ......................69
Figure 30 Lee Frame....................................................................................................71
Figure 31 Load-displacement curve for Lee's frame for elastic case.........................71
LIST OF TABLES

Table 1: Conjugate Gradient method ....................................................................................... 16
Table 2: Preconditioned Conjugate Gradient Method ............................................................. 18
Table 3: Nonlinear Conjugate Gradient Method ...................................................................... 19
Table 4: Sequential Quadratic Programming Method ............................................................. 28
Table 5: Augmented Lagrangian Method ................................................................................ 30
Table 6: Newton’s Method ...................................................................................................... 38
Table 7: Newton’s Method with Submatrices .......................................................................... 42
Table 8: Reduced preconditioned conjugate gradient method to solve Problem 2 .............. 47
Table 9: SQP method for finite element problem ................................................................. 48
Table 10: Crisfield’s method .................................................................................................... 55
Table 11: Implicit path continuation method for finite element problem .............................. 60
ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to Dr. Konstantinos Papakonstantinou, my graduate advisor, for providing me with this opportunity and guidance in my research work. He has been a tremendous mentor and helped me in all walks of graduate life. Without his encouragement and constant guidance this thesis would not have been possible. I am grateful to Dr. Gordon P. Warn for his support as an academic mentor and thesis committee member. I would like to thank Dr. Ali Memari for his valuable comments and guidance. I would like to thank Dr. Jeffrey Laman for sharing his knowledge with me in all aspects of my academic life and constantly motivating me to give my best.

I extend my gratitude to my graduate school colleagues and friends for their moral support and motivation which helped me a lot in the process of completing this thesis. A special mention goes to Charalampos Andriotis and Priyanka Patki for their willingness to share their knowledge and constantly being a pillar of support. Finally, a big thank you to my husband and my parents for their patience, understanding and unconditional support which was my source of inspiration while pursuing my master’s degree.
Chapter 1
INTRODUCTION

1.1 General

In structural analysis, there is an increasing demand for performing analysis of large scale structural systems with nonlinear behavior. The load response of a structure can be seen physically in the form of deflections, rotations or vibrations. For a preliminary analysis, the material behavior is considered linearly elastic, i.e. stress (axial or bending) varies linearly with the corresponding strain and upon unloading the body recovers back to its original configuration. This is an idealized model which is very straightforward for obtaining structural element behavior. The stiffness matrix is constant in this case. The structural deformations can be also superimposed for this linear case.

However, for real life problems with large loads many complexities arise due to nonlinearities. Nonlinear phenomena arise due to physical variables related by nonlinear quantities. In nonlinear structural analysis, the relationship between the stress and strain becomes nonlinear and the stiffness matrix does not remain constant. When a structure is tested for large loads up to collapse, we observe bifurcations and instabilities in structural response. Hence, we define two types of nonlinearities: material and geometric. Material nonlinearly is commonly seen in all elements. As the name suggests, this type of nonlinearity is attributed to material behavior, as the force increases the stress-strain behavior becomes nonlinear. In many structures, geometric nonlinearity is also observed due to large displacements and large rotations. Geometric nonlinearity arises when the change in geometry becomes significant and is taken into account in equilibrium and compatibility equations. For small deformations, geometric nonlinearity does not come into picture as it introduces very small error. But as the displacements and rotations increase in magnitude,
the frame of reference is significantly different from the original one and the stress-strain curve of body is nonlinear.

To completely define the behavior of continuum problems, it is required to have deformation information for all the sections of the body. This is a very cumbersome task since initially we know only
about the applied loading and the general body configuration. To obtain structural deformations and internal forces for a body, we need a solution strategy so that we can define, analyze and obtain structural responses. Hence finite element methods are used which are very popular in several engineering fields and provide approximation to boundary value problems.

In displacement based finite element methods, compatibility is strictly enforced due to their formulation, but equilibrium of forces is not exact which results in certain errors. In force-based finite element methods, the nodal forces are unknowns and exact values of forces are obtained. Since compatibility is implemented by integrating, the deformations have now some errors in this case. In order to eliminate the limitations of these methods, mixed and hybrid finite element methods have been proposed in several works. From the available studies in the literature there is still need for development of methods that can overcome several limitations of current techniques.

1.2 Research

This work is based on developing computational solutions which enhance the efficiency of hybrid finite element methods. In this regard, we explore various optimization techniques which are being successfully implemented in other fields of engineering. The aim is to explore optimization algorithms for solving the finite element problems by improving upon the classical methods. We want to improve the accuracy of solution and computational cost. In this regard, a hybrid finite element model was developed by [1] for large displacements and rotations and Newton’s method has been used to iteratively obtain the structural response. Hence the overall goal of this work is to search for alternative methods than second order Newton’s methods in solving problems with multiple degrees of freedom based on this recently suggested hybrid element.
1.3 Tasks

1. The new hybrid finite element formulation will be modeled and analyzed using Newton’s constrained optimization technique.

2. The Newton’s equation will be solved using conjugate gradient method and preconditioned conjugate gradient method.

3. The hybrid finite element problem will also be analyzed using sequential quadratic programming method.

4. The hybrid finite element problem will also be analyzed using augmented Lagrangian method.

5. For snap back and snap through case, Crisfield’s arc length method will be implemented and a new implicit path continuation method will be developed to capture structure behavior in these cases.

6. The methods mentioned above will be used to check cantilever beam with vertical load, cantilever beam with moment at free end, toggle frame and Lee’s frame for elastic case and for inelastic case.

7. A comparison of the performance of the above mentioned methods will be provided and future directions will be discussed.
Chapter 2
LITERATURE REVIEW

2.1 Finite Element Analysis

Finite element methods in structural engineering are based on the principle of virtual work or principle of minimum potential energy. The principle of virtual work states that: For a body to be in equilibrium, the external virtual work due to real forces moving through virtual displacements is equal to the internal virtual work due to the internal stresses integrated over the volume of the body. The principles of work and energy are utilized to derive the classical finite element equations. The total potential energy for a body is:

\[ \Pi = U + W \]  

(2.1)

where Strain energy = \( U = \frac{1}{2} \int_{V} \sigma^T \varepsilon dV \) and

Work potential = \( W = - \int_{V} u^T f dV - \int_{A} u^T T dV - \sum u^T P \)

Where the quantities \( \sigma, \varepsilon, u, f, T, P, dV \) correspond to stress, strain, displacement, body force, surface force, point load and differential volume. The fundamental requirements in mechanics to form finite element equations are:

1. Equilibrium of body should be satisfied
2. Kinematic Compatibility should be satisfied
3. A body should follow stress strain constitutive relationship defined by the material

Now, the governing variables for finite element analysis will be described before discussing displacement-based and force-based element.
For a beam element, the displacement field \( \tilde{u} \) is given by:

\[
\tilde{u}(x) = \begin{pmatrix} u(x) \\ v(x) \\ w(x) \end{pmatrix}
\] (2.2)

Where \( u(x) \) is the displacement in x-direction, \( v(x) \) is the displacement in y-direction, \( w(x) \) is the displacement in z-direction.

The corresponding deformations \( \tilde{d} \) are:

\[
\tilde{d}(x) = \begin{pmatrix} \varepsilon(x) \\ \kappa_y(x) \\ \kappa_z(x) \end{pmatrix} = \begin{pmatrix} \frac{du(x)}{dx} \\ \frac{d^2w(x)}{dx^2} \\ \frac{d^2v(x)}{dx^2} \end{pmatrix}
\] (2.3)

Where \( \varepsilon(x) \) is the axial strain along the reference axis and \( \kappa_y(x) \), \( \kappa_z(x) \) are the curvature about y and z axis. The internal force field \( \tilde{D}(x) \) is given by:

\[
\tilde{D}(x) = \begin{pmatrix} N(x) \\ M_y(x) \\ M_z(x) \end{pmatrix}
\] (2.4)

\( N(x) \) is the axial force and \( M_y(x) \), \( M_z(x) \) are the bending moments. Many finite element models have been developed so far like displacement based, force based and hybrid method.

**2.1.1 Displacement based finite element method**

This is the classical method of finite element analysis. The basic steps towards formulation of displacement-based finite element are described: The continuous element is discretized into finite elements and displacement field is interpolated in terms of nodal displacements.

\[
\tilde{u}(x) = N_q(x)q
\] (2.5)
\( q \) are the displacement degrees of freedom, \( N_q(x) \) is the interpolation matrix which is usually cubic Hermitian polynomials for beam element and linear shape functions for truss element. The deformation field is interpolated in terms of the global nodal displacement vector.

\[
\ddot{d}(x) = B(x)q
\]  

(2.6)

\( B(x) \) is the strain-displacement matrix to express deformations in terms of displacements. Stress-strain relationship can be written as:

\[
\tilde{D}(x) = k_Ed(x)
\]  

(2.7)

\( k_E \) is the material rigidity matrix. The principle of virtual displacement leads to equilibrium condition. The stiffness matrix of element \( k \) can be obtained as:

\[
k = \frac{\partial F}{\partial q} = \int B^T(x)k_E(x)B(x)dx
\]  

(2.8)

\[ F = \int B(x)^T D(x)dx \]

(2.9)

\( F \) is the force vector. We have obtained the element stiffness matrix. We can assemble the stiffness matrices of several elements to obtain global stiffness matrix

### 2.1.2 Force based finite element method

The internal force field is described in terms of nodal forces, \( F \)

\[
\tilde{D}(x) = b(x)F
\]  

(2.10)

\( b(x) \) is the matrix of force interpolation functions

\[
\ddot{d} = f_E(x)F
\]  

(2.11)

Where \( f_E = k_E^{-1} \), \( f_E \) is the flexibility matrix. Applying the principle of virtual forces leads to compatibility which gives:

\[
q = \int b^T(x)\ddot{d}(x)dx
\]  

(2.12)
The flexibility matrix of element is given by

\[ f = \frac{\partial q}{\partial F} = \int b^T(x) f_x(x) b(x) dx \]  
\hspace{1cm} (2.13)

In displacement based method, since compatibility is strictly enforced due to the formulation, but equilibrium of forces is not exact which results in error. In force- based method, the nodal forces are unknowns and exact values of forces are obtained. Since compatibility is implemented by integrating, hence the deformations have some errors. In order to get rid of these deficiencies, mixed and hybrid finite element methods have been proposed in other works such as [2], [3], [4], [5], [6] and [7].

\[ \text{2.2 Nonlinear Programming} \]

As discussed in Section 1.2, we aim to enhance computational efficiency of the iterative solution method so now we will explore several nonlinear programming solvers. Firstly, we present some fundamentals of nonlinear programming and then describe constrained optimization algorithms. Nonlinear programming/optimization involves the minimization or maximization of an objective function in the presence of equality and inequality constraints. A general nonlinearly constrained optimization problem is defined as:

\[ \text{Minimize } f(x) \]  
\hspace{1cm} (2.14)

Subject to \( g_i(x) \leq 0 \) for \( i = 1, \ldots, m \)

\[ h_j(x) = 0 \] for \( j = 1, \ldots, l \)

\[ x \in \mathbb{R}^n \]

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}, g_i : \mathbb{R}^n \rightarrow \mathbb{R}, h_j : \mathbb{R}^n \rightarrow \mathbb{R} \] is a smooth function.

\( f \) is the objective function, \( g_i (i = 1, \ldots, m) \) is the inequality constraint, \( h_j (j = 1, \ldots, l) \) is the equality constraint. In general, in a nonlinear problem, the functions \( f(x), g_i(x), h_j(x) \) are nonlinear functions. We
begin with an initial point, $x_0$, and generate a sequence of iterates $x_k$ using an optimization algorithm till we obtain a solution point $x$ which satisfies the optimality conditions and the stopping criteria.

For unconstrained problem where the equality and inequality constraints are not present, the necessary and sufficient optimality conditions at a local minimum are, $\nabla f(x) = 0$, where $\nabla$ is the gradient, and the $n \times n$ symmetric matrix, Hessian of function $H(x)$ is positive semidefinite. For a constrained optimization problem, we define the Lagrangian function $L(x, \nu, \lambda)$ as the sum of the original objective function and the constraint functions multiplied by factors called Lagrange multipliers,

$$L(x, \nu, \lambda) = f(x) + \sum_{i=1}^{m} \nu_i g_i(x) + \sum_{j=1}^{l} \lambda_j h_j(x) \quad (2.15)$$

$\lambda_i$ and $\nu_j$ are the Lagrange multipliers corresponding to equality and inequality constraints. For this constrained optimization problem, the optimality conditions are the Karush Kuhn Tucker conditions or KKT conditions given below

1. The gradient of Lagrangian function is 0, $\nabla L(x, \nu, \lambda) = \nabla f(x) + \nu^T \nabla g(x) + \lambda^T \nabla h(x)$
2. $\nu_i g_i(x) \geq 0$
3. $\nu_i \geq 0$
4. $h_j = 0$ for $j = 1, 2, \ldots, l$.

Any point that satisfies these conditions is called KKT point. KKT point is a stationary point for the problem which can be a maximum, minimum or saddle point.

The solution methods for this problem are discussed in detail now. Our aim to study constrained optimization algorithms but these methods involve solution of sequence of unconstrained optimization methods. The general outline of optimization methods is given below:

1. Starting from an initial point $x_0$.
2. Solve for a descent direction, $d_k$

3. Solve for step length, $\alpha_k$ such that the value of the function $f(x_k + \alpha_k d_k)$ is a minimum. This process is called line search

4. Update $x_{k+1} = x_k + \alpha_k d_k$ and repeat the algorithm till convergence is attained.

A vector, $d_k$, is called direction of descent, if there exists a $\delta > 0$ such $f(x_k + \alpha d_k) < f(x_k)$ for all $\alpha_k \in (0, \delta)$. $\alpha_k$ is obtained by unconstrained minimization of $f(x_k + \alpha d_k)$.

Descent direction computation methods can involve derivatives or can be used without derivatives. But the general intuition says that line search is along the direction of negative gradient, $-\nabla f(x_k)$ i.e. maximum decrease in function value is seen when we take a step in direction of negative of gradient. We will see many methods in this chapter which are developed with this thought process.

While using iteration methods, it is necessary to establish a metric to define the efficiency and performance of algorithms. The algorithm will converge to the solution point $\bar{x}$ for $k \to \infty$, where $x_k$ is the solution for $k^{th}$ step. We should evaluate the rate of convergence. We define the error at $k^{th}$ step as $e^k = (x^k - \bar{x})$ and $\mu$ is the rate of convergence. Different rates of convergence are defined below:

1. Linear convergence $\lim_{k \to \infty} \frac{|e_{k+1}|}{|e_k|} = \mu (0 < \mu < 1)$

![Figure 4 Local and global minima](image-url)
2. Quadratic convergence \( \lim_{k \to \infty} \frac{|e_{k+1}|}{|e_k|^2} = \mu (\mu > 0) \)

3. Cubic convergence \( \lim_{k \to \infty} \frac{|e_{k+1}|}{|e_k|^3} = \mu (\mu > 0) \)

There is also the concept of local and global convergence. Some algorithms stop when they attain the local minimum point, which is called local convergence. Some algorithms can solve for global minimum point to achieve global convergence.

We also define two types of unconstrained methods. In the first method, we first find a search direction which is the descent direction and then perform line search along this direction to obtain minimum. The second class is trust region methods, where we find an approximate model for the objective function within a trust region. The trust region is evaluated for fitness and if the model is found to be a good approximation of the original objective then the trust region is expanded and the model is used to find the optimum, otherwise the trust region is contracted till we obtain a good model.

### 2.2.1 One Dimensional Unconstrained Optimization

These methods are used to perform line search which is equivalent to perform unconstrained optimization for one variable problem. See [8] for more details.

\[
\min_{\alpha} f(x_k + \alpha d_k)
\]  \hspace{1cm} (2.16)

#### 2.2.1.1 Golden section method

Golden section method is a line search method which progressively narrows the interval for finding the optimum value of a unimodal function. A unimodal function increases monotonically for \( x < x_m \) and decreases monotonically for \( x > x_m \) or vice versa. In this method, at each step we have two points \( x_1, x_2 \) and the function values \( f(x_1), f(x_2) \). Obtain a third point \( x_3 = \tau x_1 + (1-\tau)x_2 \) where \( \tau \) is the golden ratio 0.6182. If \( f(x_2) < f(x_1) < f(x_3) \), then the next point will lie between \( x_3, x_2 \). Obtain
Figure 5 Golden section method

\[ x_4 = (1 - \tau)x_1 + \tau x_2. \] If \( f(x_2) < f(x_3) \), then the new interval of uncertainty is \( x_1, x_4 \). At each step the bracket narrows and finally converges to the optimum point. This method is advantageous because it does not require first or second order information about the function, however it is applicable only for unimodal functions which is not so commonly seen in practical applications.

### 2.2.1.2 Polynomial based methods

The basis for this method is that a quadratic or cubic polynomial is used to approximate the interval \( (x_1, x_2) \) in which the minimum is to be found. Based on the knowns function values at the known discrete points, a quadratic fit is obtained. This polynomial is then minimized to obtain the optimum point. If function and gradient information is available, then cubic fit can be obtained and then cubic polynomial is minimized to obtain the optimum.

### 2.2.1.3 Armijo Line Search

As mentioned before, the goal of line search is to find \( \alpha_k \) to minimize \( f(x_k + \alpha d_k) < f(x_k) \). In practice, we do not need to accurately solve \( \min_{\alpha} f(x_k + \alpha d_k) \). In Armijo’s line search, we solve for \( \alpha \) such that sufficient decrease in objective function is obtained. This is an inexact line search method and
prevents the step length from being too large or too small. We start with a large value of $\alpha$, then iteratively reduce the value till there is sufficient decrease in objective function. The algorithm is shown below:

1. Given $\beta \in (0, 1)$, initial step length $\alpha_0 = 1$, current point $\mathbf{x}_k, \nabla f(\mathbf{x}_k), \mathbf{d}_k$

2. If $f(\mathbf{x}_k + \alpha \mathbf{d}_k) \leq f(\mathbf{x}_k) + \beta \alpha f'(\mathbf{x}_k)\mathbf{d}_k$, then $\alpha_k = \alpha_0$, otherwise $\alpha_k = \frac{\alpha_0}{2}$

3. Repeat step 2 till convergence.

### 2.2.2 Unconstrained optimization

Unconstrained optimization refers to the problems in following form:

\[
\text{Minimize } f(\mathbf{x})
\]

\(\mathbf{x} \in \mathbb{R}^n \rightarrow \mathbb{R} \) is a smooth function

Now we will describe several multivariable unconstrained optimization algorithms.

#### 2.2.2.1 Steepest descent

This is the oldest method for finding the minimum of a function and was proposed by Cauchy in 1847. It is also called gradient descent method. For a function $f(\mathbf{x})$, the descent direction is given by

\[
\mathbf{d}_k = -\nabla f(\mathbf{x}_k) = -\left[\frac{df(\mathbf{x}_k)}{dx_1}, \frac{df(\mathbf{x}_k)}{dx_2}, \ldots, \frac{df(\mathbf{x}_k)}{dx_n}\right].
\]

![Figure 6 Descent direction](image)
The direction of steepest descent is given by \( \mathbf{d}_k = -\frac{\nabla f(x_k)}{\| \nabla f(x_k) \|} \), which is defined for a point \( x_k \) such that \( \nabla f(x_k) \neq 0 \). Starting with a point \( x_k \), the next iteration point is given by

\[
x_{k+1} = x_k + \alpha \mathbf{d}_k
\]

(2.18)

\( \mathbf{d}_k \) is the descent direction and \( \alpha \) is the step length (\( \alpha \geq 0 \)). The method is terminated when \( \| \nabla f(x_k) \| < \epsilon \) (\( \epsilon \) is of the order \( 1 \times 10^{-5} \)). Steepest descent methods have linear rate of convergence. Iterations of Steepest descent method are obtained for the problem:

\[
\min_{x} (x_1 - 2)^4 + (x_1 - 2x_2)^2
\]

(2.19)

This is shown in Figure 7 where zigzagging phenomenon can be seen. This method performs well when it is away from solution point, but when it is close to the solution, it takes small steps as the value of \( \| \nabla f(x_k) \| \) becomes very small near solution. It has poor convergence and has

Figure 7 Steepest descent iterates
to be terminated. Other unconstrained optimization methods like conjugate gradient and Newton’s method give faster convergence. For more details see [8].

2.2.2.2 Conjugate gradient method

Conjugate gradient method was introduced by Hestenes and Stiefel in 1952. Initially this method was proposed to solve system of linear equations,

\[
Ax = b
\]

where \( b \) is a real, symmetric, positive-definite matrix. The use of this method for unconstrained optimization was motivated by the minimization of a positive definite quadratic function, which is equivalent to solving the linear equation system since it is solved by making the gradient zero. A quadratic minimization problem is shown below:

\[
\min x \ b^T x - \frac{1}{2} x^T Ax
\]

Figure 8 shows the minimum of a quadratic function. This method can be interpreted as an improvement over the steepest descent method which has issues with convergence. At each iteration, the

![Figure 8 Minimum of a quadratic function](image)
The steepest descent method follows the direction of negative gradient, \(-\nabla f(x_k)\). The conjugate gradient methods use the conjugate direction for descent which are now explained.

The concept of conjugacy is important for this method. For a matrix \(A\), which is \(nxn\) symmetric, we can define vectors \(p_1, p_2, \ldots, p_n\) to be conjugate if they are linearly independent and if \(p_i^T A p_j = 0\) for \(i \neq j\) i.e. they are orthogonal w.r.t. to the inner product. The conjugate gradient algorithm is described. We show below the algorithm to solve the problem of minimizing \(b^T x - \frac{1}{2} x^T A x\).

**Table 1: Conjugate Gradient method**

1. Choose a tolerance \(\varepsilon\) and start with an initial point \(x_0\)
2. Define the residual at step 0, \(r_0 = A x_0 - b\). At the first step, the negative of gradient is the search direction, so \(p_0 = r_0\)
3. Define step length parameter \(\alpha_0 = \frac{r_0^T r_0}{p_0^T A p_0}\)
4. The next iteration point is \(x_1 = x_0 + \alpha_0 p_0\)
5. Define the residual at the next step as: \(r_1 = r_0 - \alpha_0 A p_0\). If \(r_1 < \varepsilon\), quit the loop.
6. Otherwise, define the conjugate direction orthogonalization parameter \(\beta_0 = \frac{r_1^T r_1}{r_0^T r_0}\)
7. The conjugate direction at next step is \(p_1 = r_1 + \beta_0 p_0\)
8. Repeat steps 3 through 7 until solution point is obtained.

The set of conjugate directions \(p_0, p_1, \ldots, p_n\) is obtained from the algorithm. For a quadratic minimization problem in \(\mathbb{R}^n\), this method takes exactly \(n\) steps. For example, for the problem,
min \( x_1^2 + 2(x_2 - 3)^2 \), conjugate gradient method takes 2 steps and steepest descent method takes 4 steps. This problem is implemented and results shown in Figure 9.

When the matrix \( A \) is not positive definite, we can modify the equation (2.20) as follows:

\[
A^T A x = A^T b
\]

(2.22)

\( A^T A \) is a symmetric, positive definite matrix, so solving this equation is equivalent to solving \( A x = b \).

For more information on this method, see [9], [10] and [11].

### 2.2.2.2.1 Preconditioning

The method described in the previous section is for linear systems and unconstrained minimization of quadratic problems. The performance of this method depends on the condition number of the matrix, which is the ratio of the largest and smallest eigenvalues of matrix \( A \). When the condition number is large, it requires a large number of steps to reach the optimum point. To improve the performance of this method and to obtain solution in less number of steps, we perform preconditioning operation. Figure 10 shows the contour lines of a quadratic function with high condition number. In this, \( Ax = b \) is replaced by \( M^{-1} Ax = M^{-1} b \). \( M \) is a symmetric, positive definite matrix. The algorithm for preconditioned conjugate gradient is shown:

![Figure 9 Comparison of steepest descent method and conjugate gradient method](image-url)
Table 2: Preconditioned Conjugate Gradient Method

1. Choose a tolerance $\varepsilon$ and start with an initial point $x_0$

2. Define the residual at step 0, $r_0 = Ax_0 - b$, compute the preconditioned residual $z_0 = M^{-1}r_0$. At the first step, the negative of gradient is the search direction, so $p_0 = z_0$

3. Define step length parameter $\alpha_0 = \frac{r_0^Tz_0}{p_0^TAp_0}$

4. The next iteration point is $x_1 = x_0 + \alpha_0 p_0$

5. Define the residual at the next step as: $r_1 = r_0 - \alpha_0 Ap_0$. For $r_k < \varepsilon$, quit the loop.

6. Otherwise, compute preconditioned residual $z_1 = M^{-1}r_1$ and define the conjugate direction orthogonalization parameter $\beta_0 = \frac{z_1^Tr_1}{z_0^Tz_0}$

7. The conjugate direction at next step is $p_1 = z_1 + \beta_0 p_0$

8. Repeat steps 3 through 7 until convergence is obtained.

Figure 10 Contours of a quadratic function with high condition number
2.2.2.2 Nonlinear conjugate gradient method

The method developed is very useful for linear systems and minimizing quadratic functions, however many problems are nonlinear unconstrained optimization. The algorithm to solve the nonlinear optimization problem $\min_{x} f(x)$ is given below:

**Table 3: Nonlinear Conjugate Gradient Method**

1. Choose a tolerance $\varepsilon$ and start with an initial point $x_0$.
2. Define the gradient descent direction at step 0, $p_0 = -\nabla f(x_0)$.
3. Define the orthogonalization parameter $\beta$ for conjugate direction using (2.23), (2.24) or (2.25).
4. The next iteration point is $x_i = x_0 + \alpha_0 p_0$ where $\alpha_0$ is obtained by line search $\min_{\alpha} f(x_0 + \alpha p_0)$
5. For the next step, the descent direction is: $p_i = p_0 + \beta_0 p_0$ which is conjugate to previous direction.
6. Calculate $g_i = -\nabla f(x_i)$. For $g_i < \varepsilon$, quit the loop.
7. Repeat steps 4 through 6 until solution point is obtained.

Nonlinear conjugate gradient method is slower than conjugate gradient and takes more than $n$ steps to reach solution. Several formula are available for updating the conjugate gradient direction and more details can be found in [8]:

1. Hestenes and Steifel gave this formula

   $$\beta_k^{HS} = -\frac{\Delta x_n^T (\Delta x_n - \Delta x_{n-1})}{p_{n-1}^T (\Delta x_n - \Delta x_{n-1})}$$  \hspace{1cm} (2.23)

2. Fletcher and Reeves proposed the formula
\[ \beta_k^{PR} = \frac{\Delta x_n^T \Delta x_n}{\Delta x_{n-1}^T \Delta x_{n-1}} \]  

(2.24)

Figure 11 Conjugate gradient method

3. Polak-Ribiere’s formula

\[ \beta_n^{PR} = \frac{\Delta x_n^T (\Delta x_n - \Delta x_{n-1})}{\Delta x_{n-1}^T \Delta x_{n-1}} \]  

(2.25)

This algorithm is implemented for the same problem as in steepest descent method. Conjugate gradient method takes 11 steps to reach the solution whereas steepest descent method takes 25 steps.

2.2.2.3 Newton’s method

Newton’s method is also used to solve optimization problems. Newton’s method uses a quadratic approximation of function \( f \) at \( x_k \). The quadratic approximation of function can be written using Taylor’s theorem as:

\[ q(x) = f(x_k) + \nabla f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k) \]  

(2.26)
At the optimum point of unconstrained objective function, $\nabla q(x) = 0$. So, $x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k)$. It uses an initial guess point $x_0$, the derivative of function at the point $\nabla f(x_0)$, Hessian of function $\nabla^2 f(x_0)$ to obtain a sequence of $x_n$ to converge to a good estimate of solution point $\bar{x}$.

The rate of convergence of Newton’s method is quadratic, hence it is much faster than steepest descent method and conjugate gradient method. However, the computational cost of each step is considerable. Newton’s method results in local convergence, but not global convergence. For multi-dimensional unconstrained optimization problem, the iterates are found using:

$$x_{k+1} = x_k - Hf(x_k)^{-1} \nabla f(x_k)$$  \hspace{1cm} (2.27)

$Hf(x_k)$ is the Hessian of the function at $x_k$. Newton’s method is used to solve the problem previously solved using steepest descent method and conjugate gradient method and it reaches the solution in 2 steps.

Newton’s method requires the calculation of inverse of Hessian which makes it a cumbersome algorithm. There are methods which can approximate the Hessian for a function, since this calculation can be computationally expensive. These methods are called Quasi-Newton methods. For more information on this method, see [9].
2.2.2.4 Quasi Newton methods

Quasi-Newton methods is a class of methods used to solve optimization problems when the Hessian calculation is costly. So, an approximation of the Hessian matrix is used. Several formulae are available to approximate the Hessian $H_k$ or inverse of Hessian $B_k$. The Quasi-Newton step is written as:

$$x_{k+1} = x_k - B_k \nabla f(x_k) \quad (2.28)$$

$B_k$ is a positive definite symmetric matrix which approximates the inverse of Hessian $H_k$.

1. DFP (Davidon Fletcher Powell) method:

   In this method, the inverse of Hessian matrix is approximated as shown:

   $$B_{k+1} = B_k + \frac{\Delta x_k \Delta x_k^T}{\Delta x_k y_k} - \frac{B_k y_k y_k^T B_k}{y_k^T B_k y_k} \quad (2.29)$$

   Where $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$

2. BFGS method (Broyden, Fletcher, Goldfarb, and Shanno):
In this method, the inverse of Hessian matrix is approximated as shown:

\[
B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T \Delta x_k} - \frac{B_k \Delta x_k (B_k \Delta x_k)^T}{x_k^T B_k \Delta x_k}
\]  

(2.30)

Where \( y_k = \nabla f(x_{k+1}) - \nabla f(x_k) \)

Quasi-Newton method is used to solve the problem previously solved using steepest descent method, conjugate gradient method and Newton’s method it reaches the solution in 5 steps.

For more information on this method, see [9].

2.2.2.5 Trust region methods

A trust region is a region around a point where the quadratic approximation of the objective function can be trusted within tolerance limits. This trust region is a hypersphere of radius \( \Delta_k \). In trust region method,

![Figure 13 Quasi Newton method iterates using DFP](image)

an approximation of the objective function (usually a quadratic approximation) is found within a trust region of given initial radius. We then minimize the quadratic model, if the step size is smaller than trust radius, then that step will be chosen. If the step is larger than trust radius, then the step size is equal to trust radius.
After taking the next step, function is evaluated at the new point, by comparing the movement of actual objective function and the model. If the approximation is good, then the trust region is expanded and the model is accepted. If the approximation is not good, then the trust region is contracted till a good model is obtained. The quadratic subproblem for a minimization of function \( f(x) \) at \( x = x_k \) is:

\[
\min_d m = f_k + \nabla f(x_k)^T d + \frac{1}{2} d^T Hf(x_k)d
\]

s.t. \( \| d \| \leq \Delta_k \)

At a given step \( d_k \), the performance of trust region is evaluated using a merit function and defining the ratio

\[
\rho_k = \frac{f(x_k) - f(x_k + d_k)}{m_k(0) - m_k(\Delta_k)}
\]

We define Cauchy point, \( d_k^C \) as the one which minimizes the quadratic model, \( m(d) \).

\[
d_k^C = -\tau \frac{\Delta_k}{\| \nabla f_k \|} \nabla f_k
\]

where \( \tau_k = 1 \) if \( \nabla f_k^T Hf(x_k) \nabla f_k < 0 \), otherwise \( \tau_k = \min \left\{ \frac{\| \nabla f_k \|^3}{\Delta_k \nabla f_k^T Hf(x_k) \nabla f_k}, 1 \right\} \)

Using Cauchy point method is similar to performing a line search and then selecting the direction of steepest descent. In essence, this method performs poorly like method of steepest descent. Hence, modifications are required. Choice of minimum of \( m(d) \) depends on the trust region radius. If \( H(f(x_k)) \) is positive definite, and \( \| d_k \| < \Delta_k \), then then minimum point is the Newton step,

\[
d^* = d_B = -Hf(x_k)^{-1}\nabla f(x_k) . \text{If} \| \Delta_k \| < d_B, \text{then} \ d^* = -\Delta \frac{\nabla f(x_k)}{\| \nabla f(x_k) \|}
\]

We now formulate the trajectory as:

\[
d(\tau) = \tau d^U, 0 \leq \tau \leq 1
\]

\[
d^U + (\tau - 1)(d_B - d^U), 1 \leq \tau \leq 2
\]
A critical component of any trust region algorithm is the modification of trust region radius at each iteration. The algorithm is:

1. Begin with trust region radius $\Delta_k$, parameter $\eta \in [0, \frac{1}{4}]$, ratio of actual and predicted function reduction $\rho_k$

2. If $\rho_k < 0.25$, update the trust radius $\Delta_{k+1} = \frac{1}{4} \Delta_k$

3. Else if $\rho_k > 0.75$ and $\|p_k\| = \|\Delta_k\|$, update the trust radius $\Delta_{k+1} = \min(2\Delta_k, \Delta_{tr})$.

4. Otherwise, the trust radius is $\Delta_{k+1} = \Delta_k$

5. If $\rho_k > \eta$ then update the point as $x_{k+1} = x_k + p_k$ Otherwise, use $x_{k+1} = x_k$

2.2.3 Constrained optimization

Using (2.15), for an equality constrained optimization problem, we define the Lagrangian function $L(x, \lambda)$ as

$$L(x, \lambda) = f(x) + \sum_{j=1}^{J} \lambda_j h_j(x)$$

(2.35)

$\lambda_j$ are the Lagrange multipliers corresponding to equality constraints. For this constrained optimization problem, the optimality conditions are reiterated below:

1. The gradient of Lagrangian function is 0, $\nabla L(x, \lambda) = \nabla f(x) + \lambda^T \nabla h(x)$

2. $h_j = 0$ for $j = 1, 2, J$.

Any point that satisfies these conditions is called KKT point. KKT matrix for constrained optimization problem is:
\[
\begin{pmatrix}
\nabla_x L(x, \lambda) \\
\mathbf{h}(x)
\end{pmatrix} = 0
\tag{2.36}
\]

**2.2.3.1 Newton’s method**

Newton’s method is used in constrained optimization where we want to solve the KKT conditions described in (2.36). We solve the KKT optimality conditions using Newton’s method. The iterates are:

\[
\begin{pmatrix}
x_{k+1} \\
\lambda_{k+1}
\end{pmatrix} =
\begin{pmatrix}
x_k \\
\lambda_k
\end{pmatrix} -
\begin{pmatrix}
\nabla^2_x L(x, \lambda) & \mathbf{h}(x) \\
\mathbf{h}(x) & \mathbf{0}
\end{pmatrix}^{-1}
\begin{pmatrix}
\nabla L(x, \lambda) \\
\mathbf{h}(x)
\end{pmatrix}
\tag{2.37}
\]

The rate of convergence of Newton’s method is quadratic. The method provides only local convergence which means that if the solution point is far away then the algorithm doesn’t work. Computation of Hessian matrix is expensive. For more details see [8].

**2.2.3.2 Sequential quadratic programming method (SQP)**

Sequential quadratic programming methods were first introduced in 1970s. These are a class of programming methods which are Newton-like and are used to solve nonlinearly constrained optimization problems. In this method, the optimization problem is approximated at an initial point, say \(x_k\), by a quadratic model of the problem and the optimum point of the subproblem is the solution. If the original problem is

Minimize \(f(x)\)

Subject to \(h_j(x) = 0 \text{ for } j = 1, \ldots, l\)

This problem is transformed into a quadratic programming (QP) subproblem. At \(x = x_k\), QP subproblem is given as,

\[
\min_p \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2_{xx} L(x_k, \lambda_k) p
\]

Subject to \(h(x_k) + \nabla h(x_k)^T p = 0\)

\tag{2.39}
The new iterate is given by \( x_{k+1} = x_k + p \). The KKT conditions of this problem are:

\[
\begin{pmatrix}
\frac{d^2 f_k}{dx^2} + \lambda^T_k \frac{d^2 h_k}{dx^2} & \frac{dh_k}{dx} \\
\frac{dh_k}{dx} & 0
\end{pmatrix}
\begin{pmatrix}
dx \\
d\lambda
\end{pmatrix}
= -\begin{pmatrix}
\frac{df_k}{dx} \\
h_k
\end{pmatrix}
\tag{2.40}
\]

For equality constrained problems, the quadratic subproblem is equivalent to solving a linear system of equations, which can be solved using Newton’s method or conjugate gradient method. The analytical solution of the QP is computationally intense, so special algorithms should be chosen for solving it numerically. We now present a proof to show that the iterates for constrained optimization problem solved using Newton’s method and sequential quadratic programming are the same. KKT optimality conditions for the optimization problem are \( \nabla L(x, \lambda) = 0 \) where Lagrangian is given in Equation (2.35)

Newton’s step to solve optimality conditions for \( (x, \lambda) \) is,

\[
\begin{pmatrix}
\frac{d^2 L}{dx^2} & \frac{d^2 L}{dx d\lambda} \\
\frac{d^2 L}{d\lambda dx} & \frac{d^2 L}{d\lambda^2}
\end{pmatrix}
\begin{pmatrix}
dx \\
d\lambda
\end{pmatrix}
= -\begin{pmatrix}
\frac{dL}{dx} \\
\frac{dL}{d\lambda}
\end{pmatrix}
\tag{2.41}
\]

Expand the Lagrangian,

\[
\begin{pmatrix}
\frac{d^2 f + \lambda^T \frac{d^2 h}{dx^2}}{dx^2} + \frac{d^2 f + \lambda^T \frac{d^2 h}{dx^2}}{dxd\lambda} + \frac{d^2 f + \lambda^T \frac{d^2 h}{dx^2}}{d\lambda dx} + \frac{d^2 f + \lambda^T \frac{d^2 h}{dx^2}}{d\lambda^2} \\
\frac{df}{dx} + \frac{\lambda^T dh}{dx}
\end{pmatrix}
\begin{pmatrix}
dx \\
d\lambda
\end{pmatrix}
= -\begin{pmatrix}
\frac{df}{dx} + \lambda^T \frac{dh}{dx} \\
h
\end{pmatrix}
\tag{2.42}
\]

At the point \( x = x_k, \lambda = \lambda_k \), the matrix is written as,

\[
\begin{pmatrix}
\frac{d^2 f_k + \lambda_k^T \frac{d^2 h_k}{dx^2}}{dx^2} & \frac{dh_k}{dx} \\
\frac{dh_k}{dx} & 0
\end{pmatrix}
\begin{pmatrix}
dx \\
d\lambda
\end{pmatrix}
= -\begin{pmatrix}
\frac{df_k}{dx} + \lambda_k^T \frac{dh_k}{dx} \\
h_k
\end{pmatrix}
\tag{2.43}
\]

This matrix is equivalent to these set of equations:
\[
\left( \frac{d^2 f_k}{dx^2} + \lambda_k^T \frac{d^2 h_k}{dx^2} \right) dx + \frac{dh_k}{dx} d\lambda = -\frac{df_k}{dx} - \lambda_k^T \frac{dh_k}{dx}
\]
\[
\frac{dh_k}{dx} dx = -h_k
\]  

Taking \( \frac{dh}{dx} d\lambda \) to right hand side

\[
\left( \frac{d^2 f_k}{dx^2} + \lambda_k^T \frac{d^2 h_k}{dx^2} \right) dx = -\frac{df_k}{dx} - \lambda_k^T \frac{dh_k}{dx}
\]
\[
\frac{dh_k}{dx} dx = -h_k
\]  

In matrix form, this is written as,

\[
\begin{pmatrix}
\frac{d^2 f_k}{dx^2} + \lambda_k^T \frac{d^2 h_k}{dx^2} & \frac{dh_k}{dx} \\
\frac{dh_k}{dx} & 0
\end{pmatrix}
\begin{pmatrix}
dx \\
\lambda_{k+1}
\end{pmatrix}
= -
\begin{pmatrix}
\frac{df_k}{dx} \\
h_k
\end{pmatrix}
\]

This corresponds to KKT conditions for the QP problem:

\[
\min \nabla f_k^T p + \frac{1}{2} p^T \nabla^2 x L_k p
\]
\[
s.t. \frac{dh_k}{dx}^T \quad p + h_k = 0
\]

This is the main QP subproblem. Hence the iterates of SQP and Newton’s method are the same.

Hence SQP also has the quadratic convergence of Newton’s method. SQP does not require a feasible initial point which makes it a good choice for nonlinear programming. SQP methods described above provide only local solution like Newton’s method.

The basic algorithm of SQP for equality constrained problem is shown below:

**Table 4: Sequential Quadratic Programming Method**

1. Select initial values for variables \( x_0 \) and Lagrange multipliers \( \lambda_0 \)
2. Formulate the QP subproblem using the values for $\nabla f(x_k), \nabla^2_{xx} L(x_k, \lambda_k), h(x_k), \nabla h(x_k)$ and solve for $p_k$ (descent direction).

3. Choose a merit function $\phi$ and perform line search to solve for $\alpha$ such that

$$\phi(x_k + \alpha p_k) < \phi(x_k) \quad (2.48)$$

4. Update $x_{k+1} = x_k + \alpha p_k, \lambda_{k+1} = \lambda_k + \alpha p_k$

5. Check for convergence and repeat

We can also solve the SQP problem using other methods like Reduced Preconditioned Conjugate Gradient method which is explained in context of the finite element problem, in which we solve the QP for step 2 in Table 4. Step 3 is optional and is needed when we want global convergence of the problem.

Any optimization problem with inequality constraints can be modified into equality constraints with active set strategy or using slack variables. Initial Lagrange multiplier estimate can be obtained:

$$\lambda_o = -[\nabla h(x_o)^T \nabla h(x_o)]^{-1} \nabla h(x_o)^T \nabla f(x_o) \quad (2.49)$$

It was mentioned that SQP is a locally convergent algorithm. But we are looking for global solutions. To do so we use merit function which ensures that we reach global optimum by making sure that merit function $\phi$ decreases. The decrease in value of merit function implies that progress is made. An example of merit function is shown with the L1 merit function: $\phi_1 = f(x) + \rho \| h(x) \|$ This is presented in more detail in [12].

### 2.2.3.3 Augmented Lagrangian method

Augmented Lagrangian method belongs to the category of penalty methods which are used to solve constrained optimization. In penalty methods, the original problem is transformed to an unconstrained optimization by adding an additional term for each constraint to the original objective to constrained optimization. When the constraint is violated, this term is positive and when constraint is satisfied, this term...
is zero. Hence, the minimum of the unconstrained problem and original problem are same. Using Augmented Lagrangian method, the transformed unconstrained optimization is:

$$\min_x f(x) + \sum_{j=1}^{n} \lambda_j h_j(x) + \frac{1}{2} \mu \sum_{j=1}^{n} [h_j(x)]^2$$  \hspace{1cm} (2.50)

Where \( \lambda_j \) are Lagrange multipliers and \( \mu \) is the penalty parameter.

This becomes a nonlinear problem and is solved like unconstrained optimization problem. This problem can be solved using a number of unconstrained programming methods like Newton’s method, method of steepest descent, conjugate gradient and trust region method. For computational purposes, it is best to use a first order method like nonlinear conjugate gradient.

An algorithm is shown for solving the problem using Augmented Lagrangian method.

**Table 5: Augmented Lagrangian Method**

1. At step \( k \), choose initial guess \( x = x_k \), penalty parameter \( \mu = 10 \), and Lagrange multiplier \( \lambda = \lambda_k \), \( \tau \) is a parameter to update the penalty parameter.

2. Using nonlinear conjugate gradient method using Table 3, solve the unconstrained minimization problem below for \( x_{k+1} \)

$$\min_x f(x) + \sum_{j=1}^{n} \lambda_j h_j(x) + \frac{1}{2} \mu \sum_{j=1}^{n} [h_j(x)]^2$$

3. Update the Lagrange multiplier \( \lambda_{k+1} = \lambda_k + \mu h(x_k) \)

4. Update the penalty parameter \( c = c\tau \)

5. Repeat steps 1 to 4 till convergence is achieved

See [13] for detailed explanation on this method.
3.1 Finite Element problem

Various finite element methods were described in Chapter 2. These approaches have several limitations and necessitate the development of a new formulation. The classical displacement-based approach interpolates the displacement field and satisfies compatibility but not equilibrium for the case of inelastic problems with large displacement. The solution converges slowly and so this method is computationally costly. In force-based approach, force shape functions are used, satisfying equilibrium, nevertheless for geometrically nonlinear problems, there are issues. A nonlinear hybrid finite element formulation was recently developed by [1] for large displacement analysis of structural frames. This formulation satisfies equilibrium and compatibility simultaneously to analyze a structure till collapse with the following desirable characteristics:

1. Structure is discretized using coarse mesh, hence fewer finite elements
2. Due to coarse mesh, there are fewer unknowns, hence computationally fast
3. A computationally efficient iterative algorithm to solve the finite element problem

This nonlinear hybrid finite element is described now. First, we develop the kinematic relations to be used. Kinematic quantities describe the displacements and deformations of the element. Kinematic relations are the equations of motion which describe the interrelationships between the kinematic quantities. A 2-dimensional beam element has 6 degrees of freedom, displacement vector $\mathbf{u}$ is given as,

$$
\mathbf{u} = [u_x, u_y, \theta_1, u_x, u_y, \theta_2]
$$

(3.1)
where $u_{x1}$ and $u_{x2}$ is the translation in $x$-direction at left and right node, $u_{y1}$ and $u_{y2}$ is the translation in $y$-direction at left and right node, and $\theta_1$ and $\theta_2$ is the rotation about $z$-axis at left and right node. The geometrical element is modeled in Figure 14 where the initial element length is $dl$, deformed element length is $ds = (1 + \epsilon)dl$. $u_x, u_y, \theta$ are the displacements in $x$-direction, $y$-direction and rotation fields along the element length in coordinate system shown Figure 14. By using Reissner beam theory [14], the beam deformations are related by following equations:

**Figure 14 Reissner beam theory**

**Figure 15 Finite element description and discretized fiber elements in a section**
In $x$-direction,  
\[ \cos \theta = \frac{du_x + dl}{(\varepsilon + 1)dl} \]  
(3.2)

In $y$-direction,  
\[ \sin \theta = \frac{du_y}{(\varepsilon + 1)dl} \]  
(3.3)

$\varepsilon$ is the axial deformation in curved configuration. Integrating the above equations along the length of element, we get,

\[ u_{x2} - u_{x1} = \int_0^l [(\varepsilon + 1) \cos \theta - 1] dl \]  
(3.4)

\[ u_{y2} - u_{y1} = \int_0^l (\varepsilon + 1) \sin \theta dl \]  
(3.5)

The curvature $\kappa$ is determined using Euler- Bernoulli equation,

\[ \kappa = \frac{d^2 \theta}{dl^2}, \quad \theta_2 - \theta_1 = \int \kappa dl \]  
(3.6)

The set of equations (3.4),(3.5),(3.6) define the element compatibility relations. Note that, for small displacements, $\theta \rightarrow 0$, $\cos \theta \rightarrow 1$, so Reissner beam equations get modified as,

\[ \frac{du_x}{dl} = \varepsilon, \quad du_y = 0, \quad \frac{d\theta}{dl} = \kappa \]

Hence, Reissner beam theory reduces to Euler-Bernoulli beam theory for small displacements.

To obtain the equilibrium of a system, minimize the total potential energy, $f_{TPE}$

**Objective function ($f$):**  
\[ f_{TPE} = \int_0^l W(\varepsilon, \kappa) dl - P^T u \]  
(3.7)

(Total Potential energy = Strain energy + External work)

where $f$ is the objective function, $W(\varepsilon, \kappa)$ is the strain energy per unit element length, $P$ is the vector of point loads on the nodes in the global coordinates, $u$ is the vector of nodal displacements in global coordinates given by (3.1). The total potential energy is treated as the objective function which is to be
minimized, the equilibrium conditions define the stationary point for this function. In addition to this we subject the problem to compatibility constraints in (3.4),(3.5),(3.6). In undeformed shape, the local axes of the element make an angle $\gamma$ with the global axes of the structure. The integrals are evaluated numerically using Gaussian quadrature with $np$ Gauss points, $x_i$ is the location of Gauss point and $w_i$ are the Gaussian weights. In matrix form, compatibility is written as:

**Equality constraints (h):**

$$
Cu - B^T \int_0^L L_i dl = 0
$$

$$
C = \begin{pmatrix}
-1 & 0 & 0 & 1 & 0 & 0 \\
0 & -1 & 0 & 0 & 1 & 0 \\
0 & 0 & -1 & 0 & 0 & 1 \\
\end{pmatrix}, \quad
B = \begin{pmatrix}
\cos \gamma & \sin \gamma & 0 \\
-\sin \gamma & \cos \gamma & 0 \\
0 & 0 & 1 \\
\end{pmatrix}, \quad
L_1 = \begin{pmatrix}
\sum_{i=1}^{np} w_i (\varepsilon_i + 1) \cos \theta_i - l \\
\sum_{i=1}^{np} w_i (\varepsilon_i + 1) \sin \theta_i \\
\sum_{i=1}^{np} w_i \kappa_i \\
\end{pmatrix}
$$

(3.9)

$h$ is the vector of equality constraints in (3.8). The rotation $\theta_i$ at Gauss point is given by an interpolation scheme using curvature $\kappa$.

$$
\theta_i = \theta_i + \sum_{j=1}^{np} \rho_{ij} \kappa_j
$$

$$
\rho = l \begin{pmatrix}
g_1 & g_2 & \cdots & g_{np} \\
g_1^2 & \frac{2}{np} & \cdots & \frac{g_{np}^2}{np} \\
g_1^3 & \frac{3}{np} & \cdots & \frac{g_{np}^3}{np} \\
\vdots & \vdots & \ddots & \vdots \\
g_1^{np} & \frac{np}{np} & \cdots & \frac{g_{np}^{np}}{np} \\
\end{pmatrix} W^{-1}
$$

Where

$$
W = \begin{pmatrix}
1 & g_1 & \cdots & g_1^{np-1} \\
\vdots & \vdots & \ddots & \vdots \\
1 & g_{np} & \cdots & g_{np}^{np-1} \\
\end{pmatrix}
$$

(3.10)

where $W$ is the Vandermonde matrix, $g_i = \frac{x_i}{l}$, $l$ is the element length.

The original problem is now an equality constrained nonlinear optimization written below:
Minimize \( f_{TPE} = \int_{0}^{l} W(\varepsilon, \kappa) dl - P^T u \) subject to \( Cu = B^T \int_{0}^{l} L_i dl \) \( (3.11) \)

The focus of this work is to find computationally efficient algorithms to process the solution of (3.11). As discussed in Section 2.2.3, we get the optimum point of equality constrained optimization problems using KKT optimality conditions. The Lagrangian \( L \) (2.35) of the problem in (3.11) is given as:

\[
L = \int_{0}^{l} W(\varepsilon, \kappa) dl - P^T u + \lambda^T h \\
\text{where } h = Cu - B^T \int_{0}^{l} L_i dl
\]  \( (3.12) \)

\( \lambda = (\lambda_1, \lambda_2, \lambda_3) \) \( (3.13) \)

\( \lambda \) is vector of Lagrange multiplier and \( \lambda_1, \lambda_2, \lambda_3 \) are the components of \( \lambda \). For this thesis, Lagrangian refers to \( L \) in (3.12). KKT optimality conditions are (2.36):

\[
\nabla L = 0 \quad \text{or} \quad \begin{bmatrix} \nabla_{\varepsilon} L \\ \nabla_{\kappa} L \\ \nabla_{u} L \\ \nabla_{\lambda} L \\ \end{bmatrix} = 0
\]  \( (3.14) \)

The strain energy derivatives are required for calculation of \( \nabla L \),

\[
\frac{dW(\varepsilon, \kappa)}{d\varepsilon_i} = N_i, \quad \frac{dW(\varepsilon, \kappa)}{d\kappa_i} = M_i
\]  \( (3.15) \)

\( N_i \) and \( M_i \) are the axial force and bending moment at Gauss point \( i \). To evaluate the axial force and bending moment at each section for an inelastic element, fiber discretization is performed to capture material inelastic behavior. Using Euler-Bernoulli beam theory assumptions that plane sections remain plane, the strain, \( \varepsilon_{j} \) at each fiber level is a function of axial deformation \( \varepsilon_i \), curvature \( \kappa_i \) at the Gauss points and distance of fiber element from centroid \( y_{j} \).

\[
\varepsilon_{j} = \varepsilon_i - y_{j} \kappa_i
\]  \( (3.16) \)
The stress resultants, $N_i$ and $M_i$, are computed using mid-point rule as given below:

$$
(N_i, M_i) = \left( \sum_{j=1}^{n_f} A_j \sigma_{f_j} \sum_{j=1}^{n_f} A_j y_{f_j} \sigma_{f_j} \right)
$$

(3.17)

where $A_j$ is the section area of fiber element, $\sigma_{f_j}$ is the stress in the fiber element $j$, $n_f$ is the number of fiber elements. User can specify a unique stress-strain relationship for the element.

Overview flowchart:

- Problem Statement:
  - Objective function ($f$): $f_{\text{JPE}} = \int_0^h W(\epsilon, \kappa) dl - P^T u$ subject to $h = 0$

- Newton's method
- Conjugate gradient method
- SQP
- Newton's method with submatrices
- Preconditioned Conjugate gradient method
- Crisfield's Arc length method
- Implicit Path continuation
The flowchart shown above outlines the solution algorithms explored in this thesis. Some comments regarding the algorithms corresponding to the numbers in circles are below:

**The problem statement corresponds to (3.11), which is equivalent to solving (3.14).**

1. Equation (3.14) is solved using Newton’s method
2. Hybrid finite element problem is modified and sparsity of Hessian matrix is exploited to formulate submatrices to solve the problem using Newton’s method
3. Equation (3.24) is solved using conjugate gradient method
4. Equation (3.24) is solved using preconditioned conjugate gradient method which modify eigen structure of Hessian matrix to easily solve the problem
5. Equation (3.11) is transformed into a quadratic problem and solution obtained using SQP. Newton’s method and SQP are similar in convergence
6. Preconditioning of matrices is a common operation in Preconditioned conjugate gradients and SQP
7. Newton’s method is implemented with Crisfield’s [15] formulation to obtain solution paths with snap through and snap back behavior
8. Implicit path continuation methods uses a predictor-corrector approach to cross critical points

### 3.2 Method 1- Newton’s method

We want to obtain the equilibrium path of a structure by using (3.11) for incremental loading \( P \). We will use load control method, where at each increment, \( P \) is known and the hybrid finite element problem (3.11) will be solved. We will use Newton’s method, which was described in Section 2.2.3.1. This method requires gradient of Lagrangian (3.12), \( \nabla L \) and Hessian of the Lagrangian, \( \nabla^2 L \). The vector of unknowns \( y \) is given as,

\[
y = [\varepsilon, \kappa, u, \lambda]
\]  

(3.18)
The gradient of Lagrangian $L$ is,

$$
\nabla L = \begin{pmatrix}
\nabla_x L \\
\nabla_u L \\
\nabla_{xu} L
\end{pmatrix} = \begin{pmatrix}
\nabla_x f + \nabla_x h \\
\nabla_u f + \nabla_u h \\
\n\nabla_u f + \nabla_u h
\end{pmatrix} = \begin{pmatrix}
N + \nabla_x h \\
M + \nabla_u h \\
-P + \nabla_u h
\end{pmatrix}
$$

(3.19)

(3.19) is expanded below at Gauss point $y_i$:

$$
N_i + \nabla_{ai} h = N_i + \lambda_1 \cos \theta_i + \lambda_2 \sin \theta_i
$$

(3.20)

$$
M_i + \nabla_{ui} h = \lambda_1 + \lambda_2 \sum_{j=1}^{np} \frac{W_j}{W_i} (\varepsilon_j + 1) \cos(\theta_j + \gamma) \frac{d\theta_j}{d\kappa_j} - \lambda_3 \sum_{j=1}^{np} \frac{W_j}{W_i} (\varepsilon_j + 1) \sin(\theta_j + \gamma) \frac{d\theta_j}{d\kappa_j}
$$

(3.21)

$$
-P + \nabla_u h = -P + \lambda_T (C - B^T \sum_{j=1}^{np} \frac{W_j}{W_i} \frac{dL}{d\mathbf{u}})
$$

(3.22)

$$
\nabla^2 L = \begin{pmatrix}
\nabla_{xx}^2 L & \nabla_{ux}^2 L & \nabla_{uu}^2 L & \nabla_{xu}^2 L \\
\nabla_{ux}^2 L & \nabla_{xx}^2 L & \nabla_{uu}^2 L & \nabla_{uux}^2 L \\
\nabla_{ux}^2 L & \nabla_{ux}^2 L & \nabla_{uu}^2 L & \nabla_{uuu}^2 L \\
\n\nabla_{xx}^2 L & \nabla_{xx}^2 L & \nabla_{uu}^2 L & \nabla_{uuu}^2 L
\end{pmatrix}
$$

(3.23)

**Table 6: Newton’s Method**

To solve finite element problem in (3.11) using Newton’s method, follow the algorithm below:

1. At the initial stage, the structure is not deformed hence, $\varepsilon^0 = 0$, $\kappa^0 = 0$, $u^0 = 0$, $\lambda^0 = 0$,

   $$
y^0 = \begin{pmatrix}
\varepsilon^0 \\
\kappa^0 \\
u^0 \\
\lambda^0
\end{pmatrix}
$$

2. Apply load $P = P^0$

3. Compute the gradient of Lagrangian $\nabla_y L$ using (3.19)

4. Compute the Hessian given by: $\nabla^2_{yy} L$ using (3.23)
5. Solve for increment $dy^i$ using,

$$dy^i = -[\nabla_{yy}^2 L]^{-1} \nabla_y L$$

where $dy^i = [d\varepsilon^i \, d\kappa^i \, du^i \, d\lambda^i]^T$

6. Update $y^{i+1} = y^i + dy^i$ and update $\varepsilon, \kappa, u, \lambda$

7. Repeat steps 1 to 5 till $\|\nabla_y L\|_2 < \text{tolerance}$

8. Increment the load $P$

Now, we will look into other methods which can be used to solve the finite element optimization problem (3.11).

### 3.3 Method 2-Newton’s method with submatrices

A incremental iterative procedure was developed by [1] based on the beam element described in Section 3.1 and a nonlinear program was also developed to implement structural problems. This is called Newton’s method with submatrices. In addition to the given variables, it also introduces the rotation, $\theta_i$ at Gauss points as unknowns. Additional equality constraints which correspond to equation (3.10) are included. The new set of unknowns is $(\varepsilon \, \kappa \, \theta \, u \, \lambda)$ and deformation $z_i = (\varepsilon_i \, \kappa_i \, \theta_i)$ at each Gauss point, $u = [u_1, u_2, u_3, u_4, u_5, u_6]$. For this method, the Newton’s step is given below, where

$$(d_\varepsilon \, d_u \, d_\lambda)^T$$

is the descent direction,

$$
\begin{pmatrix}
\nabla_{zz}^2 L & 0 & \nabla_z h \\
0 & 0 & \nabla_u h \\
\nabla_z h & \nabla_u h & 0
\end{pmatrix}
\begin{pmatrix}
d_z \\
d_u \\
d_\lambda
\end{pmatrix}
= -
\begin{pmatrix}
\nabla_z f \\
\nabla_u f \\
h
\end{pmatrix}
$$

(3.25)

Where, as before, $L$ is the Lagrangian of the new problem, $f$ is the new objective function and $h$ is the set of new equality constraints. The new approach exploits the sparsity of the Hessian of Lagrangian to
obtain the explicit form of stiffness matrix. As will be shown below, the stiffness matrix eases the calculation of displacements and sequentially, the dual variables and element deformations are computed. Hence, the original problem is elegantly converted to a classic structural analysis problem solved using Newton’s method.

For the new set of unknowns, the equality constraints are

$$ \mathbf{h} = \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \end{pmatrix} = 0 $$  \hspace{1cm} (3.26)

$$ \mathbf{h}^1 = \begin{pmatrix} u_4 - u_1 - \sum_{i=1}^{np} w_i (\varepsilon_i + 1) \cos \theta_i + l \\ u_2 - u_1 - \sum_{i=1}^{np} w_i (\varepsilon_i + 1) \sin \theta_i \\ u_6 - u_2 - \sum_{i=1}^{np} w_i \kappa_i \end{pmatrix} = \begin{pmatrix} \theta_1 - u_3 - \sum_{j=1}^{n} \rho_{1j} \kappa_j \\ \theta_2 - u_3 - \sum_{j=1}^{n} \rho_{2j} \kappa_j \\ \vdots \\ \theta_n - u_3 - \sum_{j=1}^{n} \rho_{nj} \kappa_j \end{pmatrix} $$  \hspace{1cm} (3.27)

$\mathbf{h}^1$ is similar to set of constraints in method 1. $\mathbf{h}^2$ is set of the constraints introduced to account for equation (3.10). The gradient of constraint matrix w.r.t. nodal displacement $\mathbf{u}$ is shown below:

$$ \nabla_{\mathbf{u}} \mathbf{h}^1 = \begin{pmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix} \quad \nabla_{\mathbf{u}} \mathbf{h}^2 = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & -1 & 0 & 0 & 0 \end{pmatrix} \quad \nabla_{\mathbf{u}} \mathbf{h} = \begin{pmatrix} \nabla_{\mathbf{u}} \mathbf{h}^1 \\ \nabla_{\mathbf{u}} \mathbf{h}^2 \end{pmatrix} $$  \hspace{1cm} (3.28)

The Hessian of Lagrangian $L$ w.r.t. $\mathbf{z}$, $\nabla_{\mathbf{zz}}^2 L$ is shown below:

$$ \nabla_{\mathbf{zz}}^2 L = \text{diag}(\nabla_{\mathbf{z}\mathbf{z}}^2 L) $$  \hspace{1cm} (3.29)

Where

$$ \nabla_{\mathbf{z}\mathbf{z}}^2 L = w_i \begin{pmatrix} \kappa_i^i & \lambda_i \sin \theta - \lambda_2 \cos \theta \\ \lambda_i \sin \theta - \lambda_2 \cos \theta & (\varepsilon_i + 1)(\lambda_1 \cos \theta + \lambda_2 \sin \theta) \end{pmatrix} $$  \hspace{1cm} (3.30)

$k_i$ is the tangent stiffness of the cross-section.
\[ k_i^j = \begin{pmatrix} \sum_{i=1}^{nf} E_i^i A_i & -\sum_{i=1}^{nf} E_i^i A_i y_{fi} \\ -\sum_{i=1}^{nf} E_i^i A_i y_{fi} & \sum_{i=1}^{nf} E_i^i A_i y_{fi}^2 \end{pmatrix} \] (3.31)

The gradient of constraint matrix \( h \) w.r.t. deformation matrix \( z \) is shown below:

\[
\nabla_z h^1 = \begin{pmatrix} -\cos \theta & -\sin \theta & 0 \\ 0 & 0 & -1 \\ (\varepsilon + 1) \sin \theta & -(\varepsilon + 1) \cos \theta & 0 \end{pmatrix} \] (3.32)

\[
\nabla_z h^2 = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \rho_{i1} & \rho_{i2} & \cdots & \rho_{im} \\ 0 & 0 & \cdots & 0 \end{pmatrix} \] (3.33)

The gradient of objective function \( f \) w.r.t. \( z_i \) is,

\[
\nabla_{z_i} f = w_i \begin{pmatrix} N_i \\ M_i \\ 0 \end{pmatrix} = D^i \] (3.34)

The gradient of objective function \( f \) w.r.t. \( u \) is,

\[
\nabla_u f = P \] (3.35)

The tangential stiffness matrix of each element is \( K \),

\[
K = \nabla_u h F^{-1} \nabla_u h^T \] (3.36)

where \( F \) is the flexibility matrix for Lagrange multipliers,

\[
F = \sum_{i=1}^{np} \frac{1}{w_i} \nabla_z h^T (k_i^j)^{-1} \nabla_z h \] (3.37)

The Lagrange multiplier are updated as, \( \lambda^{k+1} = \lambda^{k}_{00} + (F)^{-1} \nabla_u h^T d_u \)

\( d_u \) is the vector of nodal displacement. \( \lambda^{k}_{00} \) are Lagrange multipliers related to internal nodal forces as shown below,
\[ \lambda_{01}^k = (F)^{-1}(h_{res} - \sum_{i=1}^n \nabla_z h^T (k_i^z)^{-1} D^i) \]  
\[ \nabla_u h_{01} = P_{in} \]

\( P_{in} \) are the internal nodal force in each element. Finally, the solution algorithm for this method is shown below:

**Table 7: Newton’s Method with Submatrices**

1. Start with \( \varepsilon, \kappa, \theta, u, \lambda \).
2. Begin load step \( P \).
3. For all Gauss points, compute \( \nabla_z h \) using (3.32), the tangential stiffness of each element \( k_i^z \) using (3.31), stress resultants using \( \nabla z_i f \) using (3.34).
4. Compute constraint residual \( h_{res} \).
5. Compute \( F, K \) using (3.37) and (3.36).
6. Compute internal forces \( P_{m} \) using (3.39).
7. Assemble the tangential structural stiffness matrix \( K_{str} \).
8. Assemble the structural internal forces, \( P_{str} \).
9. Compute \( d_p = P_{str} - P_m \).
10. Compute \( d_u = (K_{str})^{-1} d_p \).
11. Update Lagrange multipliers \( \lambda^{k+1} \).
12. For all elements, compute \( z \).
13. Check convergence and go to next iteration.
As compared to Method-1, Newton’s method with submatrices has several advantages. The size of the matrix to be inverted is smaller. Due to the sparsity of matrices in this method, this method is better than Method 1 theoretically.

3.4 Method 3-Conjugate gradient method with and without Preconditioning

From the previous section we saw that the original finite element problem was transformed into an optimization problem (3.11) and the optimality conditions (3.14) can be solved to get the solution point using Newton’s method and Newton’s method with submatrices. The disadvantage of Newton’s method is that it requires the calculation of inverse of Hessian matrix $\nabla^2 L$ in Method 1 and tangential stiffness matrix in Method 2. We are searching for computationally advanced and efficient algorithms which are better than Newton’s method. We first test conjugate gradient method described in Section 2.2.2.2. This method does not require computation of matrix inverse. This method converges slow, but for large scale problem they are better than Newton’s method. For conjugate gradient method, number of multiplications per iteration is $O(n)$ vs $O(n^3)$ for Newton’s method.

The conjugate gradient will be utilized to solve for step 5 in Table 6. The conjugate gradient method requires a positive definite matrix, but the matrix of equation (3.23) is indefinite. To make the algorithm useful, we use Conjugate gradient with normal equations.

This concept is now explained. Conjugate gradient is applicable only for the system $Ax = b$, where $A$ is a positive definite matrix. If this is not available, then pre-multiply this equation with $A^T$ to get the equation:

$$A^T Ax = A^T b$$  \hspace{1cm} (3.40)

$A^T A$ is a symmetric positive definite matrix, hence conjugate gradient can be applied here. Since the condition number of $A^T A$, $\kappa(A^T A)$ is same as $\kappa^2(A)$, hence there can be problem in convergence. For the finite element problem, we precondition with matrix $\nabla^2 L^T \nabla^2 L$. 
In this work, we will compare some optimization algorithms and test their efficiency against standard Newton’s method. Direct solvers are based on the factorization of the coefficient matrix, they are robust and give good results in reasonable time. But as the scale of the matrix increases, the storage demands and number of operations performed are huge. Hence, we resort to using iterative measures. Iterative measures have less number of operations as compared to direct methods. Hence, preconditioning is required.

The constrained optimization algorithms which are used are:

1. Sequential quadratic programming
2. Augmented Lagrangian method

3.5 Method 4- Sequential quadratic programming method

In this section, we will solve the hybrid finite element optimization problem of equation (3.11) using sequential quadratic programming method discussed in Section 2.2.3.2. The hybrid finite element problem is an equality constrained optimization of the form shown below:

\[
\begin{align*}
\text{Min } f(\epsilon, \kappa, u) \\
\text{Subject to } h(\epsilon, \kappa, u) = 0
\end{align*}
\]  

(3.41)

Using Section 2.2.3.2, above general problem is transformed into the quadratic problem 1 at the initial point \(v_k = (\epsilon_k, \kappa_k, u_k)\) whose solution will yield \(v_{k+1} = v_k + s\) which is given below:

Problem 1

\[
\begin{align*}
\min_s \begin{pmatrix}
\nabla_{\epsilon} f \\
\nabla_{\kappa} f \\
\nabla_{u} f
\end{pmatrix}^T \begin{pmatrix}
\nabla_{\epsilon}^2 L & \nabla_{\epsilon \kappa}^2 L & \nabla_{\epsilon u}^2 L \\
\nabla_{\kappa \epsilon}^2 L & \nabla_{\kappa}^2 L & \nabla_{\kappa u}^2 L \\
\nabla_{u \epsilon}^2 L & \nabla_{u \kappa}^2 L & \nabla_{uu}^2 L
\end{pmatrix} \begin{pmatrix}
s \\
s \\
s
\end{pmatrix} + \frac{1}{2} s^T \begin{pmatrix}
\nabla_{\epsilon}^2 L & \nabla_{\epsilon \kappa}^2 L & \nabla_{\epsilon u}^2 L \\
\nabla_{\kappa \epsilon}^2 L & \nabla_{\kappa}^2 L & \nabla_{\kappa u}^2 L \\
\nabla_{u \epsilon}^2 L & \nabla_{u \kappa}^2 L & \nabla_{uu}^2 L
\end{pmatrix} \begin{pmatrix}
s \\
s \\
s
\end{pmatrix}
\end{align*}
\]
Subject to \( \nabla \mathbf{h}^T \mathbf{h} + \nabla \mathbf{u}^T \mathbf{h} \mathbf{s} + \mathbf{h}^T = 0 \)

\[
\begin{align*}
\mathbf{h}(\mathbf{e}, \mathbf{k}, \mathbf{u}) &= \left\{ 
\begin{array}{l}
u_{x2} - u_{x1} - \sum_{i=1}^{np} w_i (\varepsilon_i + 1) \cos \theta_i + l \\
u_{y2} - u_{y1} - \sum_{i=1}^{np} w_i (\varepsilon_i + 1) \sin \theta_i \\
\theta_2 - \theta_1 - \sum_{i=1}^{np} w_i \kappa_i
\end{array}
\right. \\
&= \left\{ 
\begin{array}{l}
u_{x2} - u_{x1} - \sum_{i=1}^{np} w_i (\varepsilon_i + 1) \cos \theta_i + l \\
u_{y2} - u_{y1} - \sum_{i=1}^{np} w_i (\varepsilon_i + 1) \sin \theta_i \\
\theta_2 - \theta_1 - \sum_{i=1}^{np} w_i \kappa_i
\end{array}
\right.
\end{align*}
\tag{3.42}
\]

This can also be written in terms of vector \( \mathbf{v}_k \),

\[
\min_{\mathbf{s}} \nabla \mathbf{v}_k \mathbf{f}(\mathbf{v}_k)^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T \nabla \mathbf{v}_k^2 \mathbf{L}(\mathbf{v}_k) \mathbf{s}
\]

Subject to \( \nabla \mathbf{v}_k \mathbf{h}(\mathbf{v}_k)^T \mathbf{s} + \mathbf{h}(\mathbf{v}_k) = 0 \) \tag{3.43}

In [16], an approach for solving quadratic minimization problems with linear equality constraints is suggested. This solution technique will first be implemented for the quadratic problem 2 shown below and based on that, an algorithm to solve Problem 1 will be developed.

Problem 2

\[
\begin{align*}
\text{Min} & \quad \mathbf{c}_k^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{H}_k \mathbf{x} \\
\text{Subject to} & \quad \mathbf{A}_k \mathbf{x} = \mathbf{b}_k
\end{align*}
\tag{3.44}
\]

where \( \mathbf{c}_k \in \mathbb{R}^n \), \( \mathbf{H}_k \) is a \( nxn \) symmetric matrix, \( \mathbf{A}_k \) is an \( m \times n \) matrix of rank \( m \), and \( \mathbf{b}_k \) is an \( n \times 1 \) vector.

We solve for \( \mathbf{x}_k \) as described next. Initialize the problem with feasible point \( \mathbf{x}_k \), and values of \( \mathbf{c}_k, \mathbf{H}_k, \mathbf{A}_k, \mathbf{b}_k \). It was suggested in [16] that Problem 2 can be transformed into equation (3.45) shown below:

\[
\mathbf{\bar{H}}_k \bar{p} = -\bar{c}_k
\tag{3.45}
\]

where \( \mathbf{\bar{H}}_k = \mathbf{Z}_k^T \mathbf{H}_k \mathbf{Z}_k, \bar{c}_k = \mathbf{Z}_k^T \mathbf{c}_k \). Here \( \mathbf{Z}_k \) is the null space of matrix \( \mathbf{A}_k \) such that:

\[
\mathbf{A}_k \mathbf{Z}_k = 0
\tag{3.46}
The linear system in (3.45) is solved for $\mathbf{p}_{k+1}$ using preconditioned conjugate gradient method discussed in Section 2.2.2.2.1. The solution point, $\mathbf{x}_{k+1}$, is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + Z\mathbf{p}_{k+1}$$  \hspace{1cm} (3.47)

This technique solves the reduced system of (3.45) by a preconditioned conjugate gradient method and does not require matrix inversions. We discuss the null space calculation and preconditioning operation below:

**3.5.1 Null space calculation**

In the above algorithm, we require the calculation of the Null Space of a Matrix $\mathbf{A}_k$ such that it satisfies (3.46). The null space contains all the vectors $\mathbf{a}_z$ such that $\mathbf{A}_k \mathbf{a}_z = 0$. Various methods are available to compute the null space matrix. A general concept of null space of a matrix is explained now. Take a general $m \times n$ matrix $\mathbf{G}$. The QR decomposition of matrix $\mathbf{G}$ is $\mathbf{G} = \mathbf{QR}$, where $\mathbf{Q}$ is a $m \times m$ orthogonal matrix, $\mathbf{R}$ is a $m \times n$. The matrix $\mathbf{G}$ is given by:

$$\mathbf{G} = \mathbf{QR} = (\mathbf{Q}_1 \mathbf{Q}_2) \begin{pmatrix} \mathbf{R}_1 \\ 0 \end{pmatrix}$$  \hspace{1cm} (3.48)

$\mathbf{Q}_1$ and $\mathbf{Q}_2$ contain orthonormal vectors. $\mathbf{Q}_2$ contains orthonormal vectors in the null space of $\mathbf{G}$.

**3.5.2 Preconditioning of matrix**

We perform preconditioning by sparse factorization of the augmented system as,

$$\begin{pmatrix} \mathbf{C} & \mathbf{A}_k^T \\ \mathbf{A}_k & 0 \end{pmatrix} \begin{pmatrix} \mathbf{z}_r \\ \mathbf{w} \end{pmatrix} = \begin{pmatrix} \mathbf{r} \\ 0 \end{pmatrix}$$  \hspace{1cm} (3.49)

where $\mathbf{r}$ is the residual vector, $\mathbf{z}_r$ is the preconditioned residual vector, $\mathbf{C}$ is a sparse symmetric positive definite approximation of Hessian matrix $\mathbf{H}_k$ and more details can be found in [16]
3.5.3 Algorithm to solve Problem 2

In [16], the algorithm in Table 8 has been proposed to solve (3.45). In essence, this algorithm is a preconditioned conjugate gradient method similar to Table 2.

Table 8: Reduced preconditioned conjugate gradient method to solve Problem 2

1. Choose a stopping tolerance and a feasible starting point \(x_k\) and obtain \(Z_k\)

2. Set initial value of the residual vector to:
\[
r = -Z_k^T (c_k + H_k x_k)
\]

3. Set initial value of correction vector \(p = 0\) and direction vector \(d = 0\)

4. Apply preconditioner to the residual to obtain preconditioned residual \(z_r\) as, \(z_r = P(r)\)

Start loop:

5. Determine the value of \(\gamma = d^T Z_k^T H_k Z_k d\)

6. Calculate the step length factor \(\alpha\) along search direction as \(\alpha = \frac{r^T z_r}{\gamma}\)

7. Now obtain the correction point \(p\) as, \(p = p + \alpha d\)

8. Update \(r = r - \alpha Z_k^T H_k Z_k, \text{ if } r < \text{tolerance}\)

9. Update the preconditioned residual \(z_r = P(r)\)

10. Update the conjugate direction orthogonalization parameter \(\beta = \frac{r^T z_r}{r^T r_{r-}}\)

11. Update the conjugate gradient direction as, \(d = z_r + \beta d\)

12. Obtain the solution point \(x_{k+1} = x_k + Z_k p\)

13. Repeat steps 5 through 12
3.5.4 Hybrid Finite element problem

Due to similarity in formation of Problem 1 and Problem 2, we use the solution strategy described above to solve Problem 1 in (3.42) which represents the hybrid finite element problem. The corresponding equation to (3.45) is:

$$
Z^T \begin{bmatrix}
\nabla_{\varepsilon_0}^2 L & \nabla_{\varepsilon_0}^2 L & \nabla_{\varepsilon_0}^2 L \\
\nabla_{\kappa_0}^2 L & \nabla_{\kappa_0}^2 L & \nabla_{\kappa_0}^2 L \\
\nabla_{u_0}^2 L & \nabla_{u_0}^2 L & \nabla_{u_0}^2 L
\end{bmatrix} p = -Z^T \begin{bmatrix}
\nabla_{\varepsilon_0} f \\
\nabla_{\kappa_0} f \\
\nabla_{u_0} f
\end{bmatrix}
$$

(3.50)

Where $Z$ is given by $(\nabla_{\varepsilon} h \ \nabla_{\kappa} h \ \nabla_{u} h)Z = 0$

(3.51)

The solution is $s = s + Zp$ and update vector $v$ in (3.43) as $v_{k+1} = v_k + s$

Equation (3.50) shown above intuitively represents a structural equilibrium equation which is modified in the null space of the compatibility constraint matrix which is solved using preconditioned conjugate gradient algorithm.

Table 9: SQP method for finite element problem

1. Choose a stopping tolerance and a feasible starting point $v_0 = (\varepsilon_0 \ \kappa_0 \ u_0), \lambda_0$ and obtain $Z_0$

2. Set initial value of the residual vector to:

$$
r = -Z_0^T \begin{bmatrix}
\nabla_{\varepsilon_0} f_0 \\
\nabla_{\kappa_0} f_0 \\
\nabla_{u_0} f_0
\end{bmatrix} + \begin{bmatrix}
\nabla_{\varepsilon_0}^2 L_0 & \nabla_{\varepsilon_0}^2 L_0 & \nabla_{\varepsilon_0}^2 L_0 \\
\nabla_{\kappa_0}^2 L_0 & \nabla_{\kappa_0}^2 L_0 & \nabla_{\kappa_0}^2 L_0 \\
\nabla_{u_0}^2 L_0 & \nabla_{u_0}^2 L_0 & \nabla_{u_0}^2 L_0
\end{bmatrix} \begin{bmatrix}
\varepsilon_0 \\
\kappa_0 \\
u_0
\end{bmatrix}
$$

3. Set initial value of correction vector $p = 0$, direction vector $d = 0$

4. Apply preconditioner to the residual to obtain preconditioned residual $z_r$ as, $z_r = P(r)$

Start loop:

5. Determine the value of $\gamma = d^T Z_0^T \begin{bmatrix}
\nabla_{\varepsilon_0}^2 L_0 & \nabla_{\varepsilon_0}^2 L_0 & \nabla_{\varepsilon_0}^2 L_0 \\
\nabla_{\kappa_0}^2 L_0 & \nabla_{\kappa_0}^2 L_0 & \nabla_{\kappa_0}^2 L_0 \\
\nabla_{u_0}^2 L_0 & \nabla_{u_0}^2 L_0 & \nabla_{u_0}^2 L_0
\end{bmatrix} Z_0 d$
6. Calculate the step length factor along search direction as $\alpha = \frac{r^T z_r}{\gamma}$

7. Now obtain the point $p$ as, $p = p + \alpha d$

8. Update $r_\perp = r$, $z_\perp = z_r$, then update the residual vector as

$\mathbf{r} = \mathbf{r}_\perp - \alpha \mathbf{Z}_0^T \begin{pmatrix} \nabla^2_{xx} L_0 & \nabla^2_{xk} L_0 & \nabla^2_{xu} L_0 \\ \nabla^2_{xk} L_0 & \nabla^2_{kk} L_0 & \nabla^2_{ku} L_0 \\ \nabla^2_{xu} L_0 & \nabla^2_{ku} L_0 & \nabla^2_{uu} L_0 \end{pmatrix} \mathbf{Z}_0$, if $\mathbf{r} < \text{tolerance}$, quit the loop

9. Update the preconditioned residual $z_r = P(r)$

10. Update the conjugate direction orthogonalization parameter $\beta = \frac{r^T z_r}{r^T z_r}$

11. Update the conjugate gradient direction as, $d = z_r + \beta d$

12. Obtain $s = s_0 + Z_0^T p$ and update $v = v_0 + s$ and obtain $v = (\varepsilon, \kappa, u)$

13. Repeat steps 5 to 12

14. For the feasible point $v$, update Lagrange multipliers as, $\lambda = -[\nabla h(v)^T \nabla h(v)]^{-1} \nabla h(v)^T \nabla f(v)$ using (2.49)

**3.7 Method 5-Augmented Lagrangian method**

See Augmented Lagrangian algorithm in Section 2.2.3.3. The problem is converted to an unconstrained optimization shown in (3.52) and solved using a first order nonlinear conjugate gradient iteration using Table 3.

The parameters used here are initial penalty parameter $c$ is 10 and updating factor for penalty parameter $\tau$ is 10.

$$\min_{\varepsilon, \kappa, u, \lambda} \sum w_i W(\varepsilon, \kappa) - P^T u + \lambda^T h(\varepsilon, \kappa, u) + \frac{1}{2} \| h(\varepsilon, \kappa, u) \|^2$$
\[ \mathbf{h}(\varepsilon, \kappa, u) = \begin{cases} u_{x_2} - u_{x_1} - \sum_{i=1}^{np} w_i (\varepsilon_i + 1) \cos \theta_i + l \\ u_{y_2} - u_{y_1} - \sum_{i=1}^{np} w_i (\varepsilon_i + 1) \sin \theta_i \\ \theta_2 - \theta_1 - \sum_{i=1}^{np} w_i \kappa_i \end{cases} \]  

(3.52)

3.8 Iterative scheme for traversing snap back and snap through

3.8.1 Method 6-Arc Length method with Newton’s iteration

Jacobian matrix is singular near limit points for geometrically nonlinear frames or in elastic-plastic solids. The problem we are referring to involves the solution of a nonlinear system in order to trace the equilibrium path of the structure. The ordinary treatment of problem can incur problems when there are limit points or bifurcation points. Some examples of load displacement curve are shown in Figure 16 which cannot be solved by Newton’s method completely. These are snap back and snap through behavior.
Figure 16 Load displacement curves showing snap through and snap back behavior (LP: Limit Point, TP: Turning Point)

Figure 17 Load control increment step
Newton’n method does not perform well near limit points. Hence, displacement control methods have been used. In the load control method, the load is kept constant during each step whereas in the displacement control method, displacement is kept constant during each increment. But this also did not give satisfactory results for snap-back and snap-through cases. Hence, the step forward was to develop a method which has both load and displacement increments. In this direction, Riks [17] suggested the arc length method in its most crude form. In the arc-length method, the load-factor is modified at each iteration so the equilibrium path is followed till convergence is obtained. Various forms of arc length method were then developed by other researchers.

The original problem has $N$ variables. In the arc length method, there are $N+1$ variables as the load factor is also an unknown since we will keep updating the displacement and load factor to follow the correct solution path. For the $N+1$ variables, we need $N+1$ equations, hence an arc length constraint is introduced. Crisfield's arc length formulation which is widely used in software has quadratic constraint.

For incremental loading, we write the load vector $\mathbf{P}$ in equation (3.11) as,

$$\mathbf{P} = \xi \mathbf{q}$$

(3.53)
\( \xi \) is the load factor and \( q \) is the fixed load vector, such that that ratio of load components is fixed as defined in \( q \) and the magnitude of incremental load is scaled by \( \xi \). When load control and displacement control cannot capture the response curve for

\[
\nabla L(y, \xi) = 0 \quad \text{or} \quad \begin{pmatrix}
\nabla_{\epsilon} L \\
\nabla_{\kappa} L \\
\nabla_{u} L \\
\nabla_{h} L
\end{pmatrix} = 
\begin{pmatrix}
\nabla_{\epsilon} L \\
\nabla_{\kappa} L \\
\nabla_{u} (\lambda^{T} h) + \nabla_{u} f \\
\nabla_{h} L
\end{pmatrix} = \begin{pmatrix}
\nabla_{\epsilon} L \\
\nabla_{\kappa} L \\
\nabla_{u} (\lambda^{T} h) - \xi q \\
\nabla_{h} L
\end{pmatrix} = 0
\]

in (3.14), where \( y = (\epsilon \ \kappa \ u \ \lambda) \), we can resort to arc length methods. We can decompose the gradient of Lagrangian in above equation as,

\[
\nabla_{\text{int}} L - \xi \nabla_{\text{ext}} L = 0
\]

where \( \nabla_{\text{int}} L = \begin{pmatrix}
\nabla_{\epsilon} L \\
\nabla_{\kappa} L \\
\nabla_{u} (\lambda^{T} h) \\
\nabla_{h} L
\end{pmatrix}, \quad \nabla_{\text{ext}} L = \begin{pmatrix}
0 \\
0 \\
q \\
0
\end{pmatrix}
\]

We define Residual \( R(y, \xi) \) and express the system in (3.54) as,

\[
R(y, \xi) = \nabla_{\text{int}} L - \xi \nabla_{\text{ext}} L = 0
\]

We introduce the term \( R(y, \xi) \) for corrections because we are interested in the general case where the system of equations is not in equilibrium and therefore the difference expresses the residual vector, which we then use to find corrections to our solution.

We start with initial value of \( y, \xi \) and calculate increments \( \Delta y, \Delta \xi \) such that \( R(y + \Delta y, \xi + \Delta \xi) = 0 \),

\[
R(y + \Delta y, \xi + \Delta \xi) = \nabla_{\text{int}} L (y + \Delta y) - (\xi + \Delta \xi) q
\]

Since increments \( \Delta y, \Delta \xi \) do not immediately satisfy (3.54), recalculate Residual with subincrement

\( \delta y, \delta \xi \).
\[
R(y + \Delta y, \xi + \Delta \xi, \delta y) = \nabla L_{\text{int}}(y + \Delta y + \delta y) - (\xi + \Delta \xi + \delta \xi)q
\]  \quad (3.57)

We want the residual in (3.57) to be zero, so expand (3.57) by using Taylor’s theorem as,

\[
\nabla L_{\text{int}}(y + \Delta y) + \left[ \frac{\partial \nabla L_{\text{int}}(y)}{\partial y} \right] \delta y - (\xi + \Delta \xi + \delta \xi)q = 0
\]  \quad (3.58)

Substitute \( \left[ \frac{\partial \nabla L_{\text{int}}(y)}{\partial y} \right] = K_T \), we obtain the new form of equilibrium equation as,

\[
K_T \delta y - \delta \xi q = -[\nabla L_{\text{int}}(y + \Delta y) - (\xi + \Delta \xi)q] = -R(y + \Delta y, \xi + \Delta \xi)
\]  \quad (3.59)

### 3.8.1.1 Rik’s method [17]

The equilibrium equation (3.59) must be satisfied and in addition to this we have the arc length constraint, \( \psi \) is the scaling factor,

\[
(\Delta y + \delta y)^T (\Delta y + \delta y) + \psi^2 (\Delta \xi + \delta \xi)^2 = l^2
\]  \quad (3.60)

\( l \) is the arc length. Linearize (3.60) and use (3.59) to obtain the system of equations given below:

\[
\begin{bmatrix}
K_T & -q \\
2\Delta y^T & 2\psi^2 \Delta \xi (q^T \cdot q)
\end{bmatrix}
\begin{bmatrix}
\delta y \\
\delta \xi
\end{bmatrix}
= - \begin{bmatrix}
R(y + \Delta y, \xi + \Delta \xi) \\
A
\end{bmatrix}
\]  \quad (3.61)

where \( R(y + \Delta y, \xi + \Delta \xi) = [\nabla L_{\text{int}}(y + \Delta y) - (\xi + \Delta \xi)q] \) and

\[
A = \Delta y^T \Delta y + \psi^2 \Delta \xi^2 (q^T \cdot q) - l^2
\]

This matrix is not symmetric and at limit points, it does not become singular. This method can be easily implemented using Newton’s method.

### 3.8.1.2 Crisfield’s method [15]

Crisfield proposed an alternative formulation for the Arc length method (3.60). He modified the equilibrium equation in (3.59) as given below,

\[
\delta y = -[K_T]^{-1} \psi \nabla L_{\text{int}}(y + \Delta y) - (\xi + \Delta \xi)q + [\delta \xi]^T [K_T]^{-1} \psi \cdot q
\]  \quad (3.62)
This equation can be written as,
\[
\delta y = \delta y_{\text{in}} + \delta \xi \delta y_{\text{ex}}
\]  
(3.63)

Where \( \delta y_{\text{in}} = -[K_T]^{-1}_{y+\Delta y} [\nabla L_{\text{int}} (y + \Delta y) - (\xi + \Delta \xi)q] \) and \( \delta y_{\text{ex}} = [K_T]^{-1}_{y+\Delta y} q \)

\( \delta y_{\text{in}} \) and \( \delta y_{\text{ex}} \) can be calculated immediately for known values of \([K_T]_{y+\Delta y}, \nabla L_{\text{int}} (y + \Delta y), \xi, \Delta \xi, q\).

Then substitute (3.63) into (3.60), to obtain an equation of the form,
\[
a_1 \delta \xi^2 + a_2 \delta \xi + a_3 = 0
\]  
(3.64)

Where
\[
a_1 = \delta y^T_{\text{ex}} \delta y_{\text{ex}} + \psi^2 (q^T q)
\]
\[
a_2 = 2(\Delta y + \delta y_{\text{in}})^T \delta y_{\text{ex}} + 2\psi^2 \Delta \xi (q^T q)
\]
\[
a_3 = (\Delta y + \delta y_{\text{in}})^T (\Delta y + \delta y_{\text{in}}) + \psi^2 \Delta \xi^2 (q^T q) - \Delta l^2
\]

(3.64) is solvable for \( \delta \xi \), and then calculate \( \delta y \) using (3.63). Now, use the algorithm in this table

**Table 10: Crisfield’s method**

1. Initiate the iteration with initial point \( y = y_0 \), initial load value \( \xi = \xi_0 \), arc length \( l \), \( \Delta y = 0 \), \( \Delta \xi = 0 \)
2. Initial value of \( \delta y_{\text{in}} = 0 \), \( \delta y_{\text{ex}} = [K_T]^{-1}_{y+\Delta y} q \)
3. With the above values, solve arc length equation (3.64) for \( \delta \xi \)
4. Choose the value of \( \delta \xi \) with same sign as \( K_T \)
5. Then update the load increment as \( \Delta \xi = \delta \xi \), update \( \delta y \) using (3.63)
6. Update \( \Delta y = \delta y \), update the point \( y = y + \Delta y \) and load increment \( \xi = \xi + \Delta \xi \)
7. Check residual for tolerance limits and if there is no convergence go to the next step otherwise quit loop
8. With the values of \( y \) and \( \zeta \), calculate \( \delta y_{in} = -[K_T]^{-1}_{y+\Delta y} [\nabla L_{int} (y+\Delta y) - (\zeta + \Delta \zeta)q] \) and \\
\( \delta y_{ex} = [K_T]^{-1}_{y+\Delta y} q \)

9. Solve arc length equation (3.64) for \( \Delta \zeta \)

10. Choose the correct solution, based on projection on previous step.

11. Then update the load increment as \( \Delta \zeta = \Delta \zeta + \delta \zeta \), update \( \delta y \) using (3.63), update \( \Delta y = \Delta y + \delta y \) , update the point \( y = y + \Delta y \) and load increment \( \zeta = \zeta + \Delta \zeta \)

12. Check for convergence, otherwise repeat steps 8 to 11

For further explanation, see [18], [19] and [20]

3.8.2 Method 7- Implicit path continuation: Modified Homotopy method

The implicit path continuation method will be used to analyze the structure when instabilities like snap back and snap through are seen. The concept behind this method is based on a homotopy class of methods, in which a system of algebraic equations is solved by varying an artificial parameter. This parameter is similar to load factor \( \xi \) for problems related to work in this thesis. Varying the parameter \( \xi \) linearly is similar to load control method and varying this parameter on the basis of the distance traversed on the equilibrium curve will help analyze structural behavior after critical point. First the homotopy basics are described and then we develop the formulation to solve the hybrid finite element problem in (3.14).

3.8.2.1 Homotopy-Motivation for this method

Homotopy methods, in general, are used to obtain solution point \( \bar{x} \) for nonlinear algebraic equations \( F(x) = 0 \) Homotopy function \( H(x, \eta) \) is defined as:

\[
H(x, \eta) = \eta F(x) + (1-\eta)G(x), \ \eta \in (0,1)
\]
Where $\eta$ is an artificial parameter (scalar), $G(x)$ is a function which can be easily solved. For $\eta=0$, the solution of $H(x,\eta) = 0$ is $G(x_0) = 0$. For $\eta = 1$, the solution is $F(\bar{x}) = 0$. We start from $(x,\eta) = (x_0,0)$ and iteratively proceed to $(x,\eta) = (x,1)$. For each value of $\eta$, we get the solution of $H(x,\eta) = 0$. At $\eta = 1$, we solve for $F(\bar{x}) = 0$. We can achieve this result by linearly varying $\eta$. However, when the path from $(x_0,0)$ to $(\bar{x},1)$ has turning points, then we introduce a parameter $t$, such that $x = x(t), \eta = \eta(t)$.

In Figure 19, we need to introduce curve parameter $t$, to trace complete curve. The set of all points $(x,\eta)$ is called zero path.

Homotopy methods are globally convergent and seek the solution starting from an arbitrary point $x_0$. These methods also come in handy when the Jacobian matrix becomes singular and Newton’s method does not work. Also, Newton’s method only gives local convergence. These methods are explained in more detail in [21], [22], [23] and [24].

![Figure 19 Zero path curve](image)
**3.8.2.2 Implicit path continuation method for post critical behavior determination for hybrid finite element analysis**

The implicit path continuation method is now implemented for the finite element problem. To solve (3.12) we want to evaluate the algebraic system (3.14), also shown in (3.65). As before, \( y \) is the vector of unknowns \((\varepsilon \ k \ u \ \lambda)\) discussed in (3.18) and load vector \( P \) in equation (3.12) as \( P = \xi q \) also discussed in (3.53), \( \xi \) is the load factor and \( q \) is the fixed load vector. Compared to homotopy method, we have a natural parameter \( \xi \).

\[
\nabla L(y, \xi) = 0 , \quad \text{or} \quad \begin{bmatrix} \nabla_\xi L(y, \xi) \\ \nabla_k L(y, \xi) \\ \nabla_u L(y, \xi) \\ \nabla_\lambda L(y, \xi) \end{bmatrix} = 0 \quad (3.65)
\]

For each value of \( \xi \), we can solve the nonlinear problem (3.65). In homotopy, we are interested in the final solution at \( \eta = 1 \), but in finite element problem, we want all the solutions for \( \nabla L(y, \xi) = 0 \) which will form the equilibrium path. For capturing instabilities, parameterize \( y \) and \( \xi \) as \( y = y(t), \xi = \xi(t) \). \( t \) is a measure of length along the curve. Differentiate (3.65) w.r.t to \( t \), to get

\[
\begin{bmatrix} \nabla_{\xi\xi} L \\ \nabla_{\xi k} L \\ \nabla_{\xi u} L \\ \nabla_{\xi \lambda} L \\ \nabla_{kk} L \\ \nabla_{ku} L \\ \nabla_{ua} L \\ \nabla_{\lambda\lambda} L \end{bmatrix} \begin{bmatrix} \frac{d\xi}{dt} \\ \frac{d\varepsilon}{dt} \\ \frac{d\kappa}{dt} \\ \frac{d\lambda}{dt} \\ \frac{du}{dt} \\ \frac{d\kappa}{dt} \\ \frac{d\lambda}{dt} \end{bmatrix} = 0 \quad (3.66)
\]
For an increment in curve length $\Delta t$, it is easy to see that,

$$ (\Delta t)^2 = (\Delta \varepsilon \Delta \kappa \Delta \mathbf{u} \Delta \lambda)(\Delta \varepsilon \Delta \kappa \Delta \mathbf{u} \Delta \lambda)^T + (\Delta \xi)^2 $$

or

$$ 1 = \left( \frac{\Delta \varepsilon}{\Delta t} \frac{\Delta \kappa}{\Delta t} \frac{\Delta \mathbf{u}}{\Delta t} \frac{\Delta \lambda}{\Delta t} \right) \left( \frac{\Delta \varepsilon}{\Delta t} \frac{\Delta \kappa}{\Delta t} \frac{\Delta \mathbf{u}}{\Delta t} \frac{\Delta \lambda}{\Delta t} \right)^T + \left( \frac{\Delta \xi}{\Delta t} \right)^2 \tag{3.67} $$

Taking limit both sides, \( 1 = \left( \frac{d\varepsilon}{dt} \frac{d\kappa}{dt} \frac{d\mathbf{u}}{dt} \frac{d\lambda}{dt} \right) \left( \frac{d\varepsilon}{dt} \frac{d\kappa}{dt} \frac{d\mathbf{u}}{dt} \frac{d\lambda}{dt} \right)^T + \left( \frac{d\xi}{dt} \right)^2 \tag{3.68} \)

We have the following equivalent set of equations:

$$ \nabla L(y(t), \xi(t)) = 0 $$

$$ \begin{pmatrix} \nabla_{\varepsilon}^2 L & \nabla_{\kappa}^2 L & \nabla_{\mathbf{u}}^2 L & \nabla_{\lambda}^2 L & \nabla_{\xi}^2 L \\ \nabla_{\varepsilon \kappa}^2 L & \nabla_{\kappa \kappa}^2 L & \nabla_{\kappa \mathbf{u}}^2 L & \nabla_{\kappa \lambda}^2 L & \nabla_{\kappa \xi}^2 L \\ \nabla_{\varepsilon \mathbf{u}}^2 L & \nabla_{\mathbf{u} \mathbf{u}}^2 L & \nabla_{\mathbf{u} \mathbf{u}}^2 L & \nabla_{\mathbf{u} \lambda}^2 L & \nabla_{\mathbf{u} \xi}^2 L \\ \nabla_{\varepsilon \lambda}^2 L & \nabla_{\lambda \lambda}^2 L & \nabla_{\lambda \mathbf{u}}^2 L & \nabla_{\lambda \lambda}^2 L & \nabla_{\lambda \xi}^2 L \end{pmatrix} \begin{pmatrix} \frac{d\varepsilon}{dt} \\ \frac{d\kappa}{dt} \\ \frac{d\mathbf{u}}{dt} \\ \frac{d\lambda}{dt} \end{pmatrix} = 0 $$

$$ \begin{pmatrix} \frac{d\varepsilon}{dt} \\ \frac{d\kappa}{dt} \\ \frac{d\mathbf{u}}{dt} \\ \frac{d\lambda}{dt} \end{pmatrix} = \begin{pmatrix} \frac{d\xi}{dt} \end{pmatrix}^2 = 1 \tag{3.69} $$

The continuation algorithm starts from a known solution and uses a predictor-corrector scheme to find subsequent solutions at different $\xi$ values. \( T = \begin{pmatrix} \frac{d\varepsilon}{dt} & \frac{d\kappa}{dt} & \frac{d\mathbf{u}}{dt} & \frac{d\lambda}{dt} \end{pmatrix} \) is tangent to the solution branch.

Firstly, we find the tangent vector, then the prediction for solution can be easily made by taking a step length from the current point in the direction of the tangent. After obtaining the prediction, a method for correcting the predicted solution is needed. The correction is always taken perpendicular to the tangent obtained during calculation of prediction point. The predictor-corrector iterates for one loop of algorithm are shown in Figure 20. The table below describes the procedure to obtain the solution of this set of equations:
Table 11: Implicit path continuation method for finite element problem

1. Initialize the value of \((\varepsilon \quad \kappa \quad u \quad \lambda) = (\varepsilon_k \quad \kappa_k \quad u_k \quad \lambda_k)\), \(\xi = \xi_k\)

2. Say \(y_z = (\varepsilon \quad \kappa \quad u \quad \lambda \quad \xi)^T\) and \(\frac{dy_z}{dt} = \begin{pmatrix} \frac{dc}{dt} \\ \frac{d\kappa}{dt} \\ \frac{d\mu}{dt} \\ \frac{d\lambda}{dt} \\ \frac{d\xi}{dt} \end{pmatrix}^T\), Substituting these values in (3.69), we get the following equations,

\[
\begin{pmatrix}
\nabla_{\varepsilon\varepsilon}^2 L & \nabla_{\varepsilon\kappa}^2 L & \nabla_{\varepsilon u}^2 L & \nabla_{\varepsilon \lambda}^2 L & \nabla_{\varepsilon \xi}^2 L \\
\nabla_{\kappa\varepsilon}^2 L & \nabla_{\kappa\kappa}^2 L & \nabla_{\kappa u}^2 L & \nabla_{\kappa \lambda}^2 L & \nabla_{\kappa \xi}^2 L \\
\nabla_{u\varepsilon}^2 L & \nabla_{u\kappa}^2 L & \nabla_{u u}^2 L & \nabla_{u \lambda}^2 L & \nabla_{u \xi}^2 L \\
\nabla_{\lambda\varepsilon}^2 L & \nabla_{\lambda\kappa}^2 L & \nabla_{\lambda u}^2 L & \nabla_{\lambda \lambda}^2 L & \nabla_{\lambda \xi}^2 L \\
\nabla_{\xi\varepsilon}^2 L & \nabla_{\xi\kappa}^2 L & \nabla_{\xi u}^2 L & \nabla_{\xi \lambda}^2 L & \nabla_{\xi \xi}^2 L \\
\end{pmatrix}
\frac{dy_z}{dt} = 0 \quad \text{and} \quad \frac{dy_{\xi}}{dt} = 1
\]

3. Take a vector \(y_z e\), which is not perpendicular to \(\frac{dy_z}{dt}\) (it is easy to choose this vector as any non-perpendicular vector). Normalize \(\frac{dy_z}{dt}\) and define \(v_z\) as: \(\frac{dy_z}{dt} = \pm \frac{v_z}{|v_z|}\) so that \(\frac{dy_z}{dt}\) is unit vector and, \(y_z^T v_z = 1\)
4. Now from step 2 and 3, we obtain the equation system given below and solve for \( y_z \) and find

\[
\frac{dy_z}{dt}
\]

which is the tangent to solution branch,

\[
\begin{pmatrix}
\nabla^2_{xx} L & \nabla^2_{ux} L & \nabla^2_{uy} L & \nabla^2_{ux} L & \nabla^2_{uy} L \\
\nabla^2_{ux} L & \nabla^2_{xx} L & \nabla^2_{uy} L & \nabla^2_{ux} L & \nabla^2_{uy} L \\
\nabla^2_{ux} L & \nabla^2_{uy} L & \nabla^2_{xx} L & \nabla^2_{ux} L & \nabla^2_{uy} L \\
\nabla^2_{ux} L & \nabla^2_{uy} L & \nabla^2_{ux} L & \nabla^2_{xx} L & \nabla^2_{uy} L \\
y_{z=1} & y_{z=2} & y_{z=3} & y_{z=4} & y_{z=5}
\end{pmatrix}
\begin{pmatrix}
z \\
v \\
0 \\
0 \\
1
\end{pmatrix}
= 
\begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
1
\end{pmatrix}
\text{ and obtain } \frac{dy_z}{dt} = \pm \frac{y_z}{|y_z|}

5. The next iterate should follow the direction of curve, so check the following condition,

\[
\frac{dy_z}{dt} \geq 0
\]

\[
\begin{pmatrix}
\varepsilon_k - \varepsilon_{k-1} \\
k_k - k_{k-1} \\
\lambda_k - \lambda_{k-1} \\
\zeta_k - \zeta_{k-1}
\end{pmatrix}
\]

6. Obtain the predictor \((\varepsilon^0_k, k^0_k, u^0_k, \lambda^0_k, \zeta^0_k)\) by multiplying tangent vector \(\frac{dy_z}{dt}\) to step length \(\Delta t\) and adding to initial point \((\varepsilon_k, k_k, u_k, \lambda_k, \zeta_k)\) as,

\[
\begin{pmatrix}
\varepsilon^0_{k+1} \\
k^0_{k+1} \\
u^0_{k+1} \\
\lambda^0_{k+1} \\
\zeta^0_{k+1}
\end{pmatrix} = \begin{pmatrix}
\varepsilon_k \\
k_k \\
u_k \\
\lambda_k \\
\zeta_k
\end{pmatrix} + \frac{dy_z}{dt} \Delta t
\]

(3.70)

7. Iteratively find the correction \((\Delta \varepsilon^m, \Delta k^m, \Delta u^m, \Delta \lambda^m, \Delta \zeta^m)\), the correction is always made perpendicular to the tangent \(\frac{dy_z}{dt}\)
\[
\begin{pmatrix}
\nabla^2_{xx} L & \nabla^2_{xk} L & \nabla^2_{xu} L & \nabla^2_{uk} L & \nabla^2_{uu} L \\
\nabla^2_{kk} L & \nabla^2_{kk} L & \nabla^2_{ku} L & \nabla^2_{uk} L & \nabla^2_{uu} L \\
\nabla^2_{uu} L & \nabla^2_{uu} L & \nabla^2_{uu} L & \nabla^2_{uu} L & \nabla^2_{uu} L \\
\nabla^2_{kk} L & \nabla^2_{kk} L & \nabla^2_{kk} L & \nabla^2_{kk} L & \nabla^2_{kk} L \\
\frac{dy_{z1}}{dt} & \frac{dy_{z2}}{dt} & \frac{dy_{z3}}{dt} & \frac{dy_{z4}}{dt} & \frac{dy_{z5}}{dt}
\end{pmatrix}
\begin{pmatrix}
\Delta \varepsilon^m \\
\Delta \kappa^m \\
\Delta \mu^m \\
\Delta \lambda^m \\
\Delta \xi^m
\end{pmatrix}
= \begin{pmatrix}
-\nabla_{xx} L \\
-\nabla_{kk} L \\
-\nabla_{uu} L \\
-\nabla_{xk} L \\
0
\end{pmatrix}
\]

8. The corrected values \( (\varepsilon, \kappa, \mu, \lambda, \xi) \) are obtained as follows till convergence is achieved:

\[
\begin{pmatrix}
\varepsilon_{k+1}^{m+1} \\
\kappa_{k+1}^{m+1} \\
\mu_{k+1}^{m+1} \\
\lambda_{k+1}^{m+1}
\end{pmatrix}
= \begin{pmatrix}
\varepsilon_{k+1}^0 \\
\kappa_{k+1}^0 \\
\mu_{k+1}^0 \\
\lambda_{k+1}^0
\end{pmatrix} + \begin{pmatrix}
\Delta \varepsilon^m \\
\Delta \kappa^m \\
\Delta \mu^m \\
\Delta \lambda^m
\end{pmatrix}
\]

\[
\xi_{k+1}^{m+1} = \xi_{k+1}^{m+1} + \Delta \xi^m
\]
Chapter 4
RESULTS

In this chapter, we present an assessment of the solution procedures discussed in Chapter 3. The solution methods are implemented in computer program to analyze structural problems. In [1], the same structural problems have been worked out using Newton’s method on hybrid finite element and with the arc length scheme for snap through and snap back behavior. Comparisons with OpenSees are also provided in [1]. In this thesis these results from [1] are obtained using plot digitization technique and are used to validate the results of the discussed algorithms in this thesis.

We check the algorithms in this thesis for the following problems:

1. Cantilever beam with vertical load at free end (elastic and inelastic)
2. Cantilever beam with moment at free end (elastic)
3. William’s toggle frame (elastic and inelastic)
4. Lee’s frame (elastic)

In above, all the codes used are in house codes except, MATLAB [25] program ‘quadprog’ which is used to solve the QP subproblem in SQP algorithm.

4.1 Cantilever with vertical load at free end

A cantilever is subjected to point load at the free end as shown in Figure 21 and the deformed shape of beam for large displacement is also shown. The load displacement curve of the beam is seen in Figure 22 and Figure 23. For analysis, 350 load increments of 1kN are used. The beam is modeled as a single element and integration is performed with 5 Gauss integration points. The beam length is 0.5-meter, width of section is 0.03-meter, height of section is 0.03-meter. The modulus of elasticity is 200 MPa. For inelastic
case, each section is discretized into 30 fibers and the yield stress is 2MPa. The computational tolerance used for iterations is $10^{-5}$. The beam is subjected to a gradually increasing load and elastic and inelastic analysis is performed. Initially, the load varies linearly with the displacement, but with increase in load, the axial stiffness of the beam becomes significant and the load response is nonlinear. With further increase in load, the cantilever gets stiffer.

![Figure 21 Cantilever with vertical load](image1)

![Figure 22 Load-displacement curve for cantilever with vertical load for elastic case](image2)
In Figure 23, we see that response path of Newton’s method, Newton’s method with submatrices, Conjugate gradient, preconditioned conjugate gradient, SQP overlap with hybrid finite element results using Newton’s method from [1]. For details on Opensees analysis with more elements see [1]. The runtime for 150 increments of 1kN for Newton’s method, Newton’s method with submatrices and SQP are same, about 3.5 seconds. In one Newton’s iteration, 6 conjugate gradient steps are required to reach convergence and 1 preconditioned conjugate gradient step is needed.
4.2 Cantilever with moment at free end

A cantilever is subjected to moment at the free end as shown in Figure 24. The load displacement curve of the beam is seen in Figure 26. For analysis, 350 load increments of 1kN are used. The beam is modelled as a single element and integration is performed with 5 Gauss integration points. The modulus of elasticity is 200 MPa. The computational tolerance used for iterations is $10^{-5}$. The beam is subjected to a gradually increasing load and elastic analysis is performed. This is a case of curling beam, where the beam completes a full circle when $M_{\text{curl}} = n \left( \frac{2EI}{L} \right)$. Figure 25 shows cantilever rolled up when moment $M_{\text{curl}}$ is reached.

![Figure 24 Cantilever with moment at free end](image)

![Figure 25 Deformed shape for cantilever with moment applied at end](image)
In Figure 26, we see that response path of Newton’s method, Newton’s method with submatrices, Conjugate gradient, preconditioned conjugate gradient, SQP overlap with hybrid finite element results using Newton’s method from [1]. For details on Opensees analysis with more elements see [1]. The runtime for 350 increments of 1kN for Newton’s method, Newton’s method with submatrices and SQP are same, about 2 seconds. In one Newton’s iteration, 5 conjugate gradient steps are required to reach convergence and 1 preconditioned conjugate gradient step is needed.

### 4.3 Toggle frame

Toggle frame consists of two inclined beams connected and loaded at a common load as shown in Figure 27. The load displacement curve of the beam is seen in Figure 28 and Figure 29. For analysis 100 load increments of $10^3$ kN are used. Each beam is modelled as a single element and integration is performed with 5 Gauss integration points. For the frame, $l_1$ is 6 meter and $l_2$ is 0.4 meter. The width of section is 0.3-meter, height of section is 0.3-meter. The modulus of elasticity is 200 MPa. For inelastic case, each
section is discretized into 30 fibers and the yield stress is 2MPa. The computational tolerance used for iterations is $10^{-5}$. The beam is subjected to a gradually increasing load and elastic and inelastic analysis is performed. We observe snap through behavior for inelastic analysis.

![Figure 27 Toggle frame with point load](image)

Figure 27 Toggle frame with point load

![Figure 28 Load-displacement curve for toggle frame for elastic case](image)

Figure 28 Load-displacement curve for toggle frame for elastic case
In Figure 28, we see that response path of Newton’s method, Newton’s method with submatrices, Conjugate gradient, preconditioned conjugate gradient, SQP, Crisfield’s arc length method, Implicit path continuation overlap with hybrid finite element results using Newton’s method from [1]. For details on OpenSees analysis with more elements see [1]. The runtime for 70 increments of $10^3$ kN for Newton’s method, Newton’s method with submatrices and SQP are same. In one Newton’s iteration, 40 conjugate gradient steps are required to reach convergence and 1 preconditioned conjugate gradient step is needed. This problem was analyzed using Crisfield’s arc length method (AL) and Implicit path continuation (IPC) and as shown in chart, Crisfield’s arc length method took 8 seconds and implicit path continuation took 2 seconds to reach same value of peak load.

Figure 29 Load-displacement curve for toggle frame for inelastic case
In Figure 29, we see that response path of Crisfield’s arc length method, Implicit path continuation overlap with hybrid finite element results using Arc length method from [1]. Results from Newton’s method, Newton’s method with submatrices, Conjugate gradient, preconditioned conjugate gradient, SQP are plotted till peak value $26e3$ kN. For details on Opensees analysis with more elements see [1]. The runtime for 26 increments of $10^3$ kN for Newton’s method, Newton’s method with submatrices and SQP are same. In this problem, snap through behavior is observed, hence complete solution path is obtained using Crisfield’s arc length method and implicit path continuation method. As shown in chart, Crisfield’s arc length method took 4 seconds and implicit path continuation took 2 seconds to reach same value of load.

### 4.4 Lee’s Frame

Lee’s frame system is show in Figure 30. The beam and column have length of 1.2-meter, section width of 0.03 meter and section height 0.02 meter. Modulus of elasticity is 70.6 MPa and elastic analysis is performed. The initial frame configuration and various deformed configurations are shown. The column is discretized with 5 equal length beam elements, while the beam is discretized with 5 equal length beam elements to the left of the point load and 5 equal length beam elements to the right of the point load. We observe snap through and snap back behavior for elastic analysis. We use load control scheme like arc length and implicit path continuation method to solve this problem. The displacement plotted against load is the vertical displacement of the node under the point load.
In Figure 31, we see that response path of Implicit path continuation overlaps with hybrid finite element results using arc length method from [1]. For details on OpenSees analysis with more elements see [1].
Chapter 5
CONCLUSIONS

In this thesis, several nonlinear programming algorithms have been explored to solve problems with the recently developed hybrid finite element which has previously used Newton’s method and its variant. The Newton’s method has some limitations for large structures and there are possibilities to utilize other nonlinear programming solvers which have been successfully used in several other fields. In this work, theoretical and numerical comparisons have been made between methods involved in analyzing three benchmark structural problems using conjugate gradient methods (with and without preconditioning), sequential quadratic programming method, augmented Lagrangian method and a suggested implicit path continuation method.

Conjugate gradient method in their natural form requires more iterations when compared with preconditioned conjugate gradients method, however cost per iteration is less since it involves only matrix-vector multiplications. Preconditioned conjugate gradient method is faster and demands less iterations since preconditioning of Hessian matrix improves the matrix eigen-structure. Sequential Quadratic Programming method has also been implemented. As shown theoretically in Section 2.2.3.2, this algorithm has similar convergence properties with Newton’s method. The preconditioning treatment of the quadratic matrix in SQP improves the performance of the algorithm. From the numerical results, we observe that SQP, Newton’s method and Newton’s method with submatrices have a comparative runtime performance for all the test problems. An advantage of SQP method is that there are no matrices inversions involved. Rather the problem is suitably modified into a linear system and can be solved by preconditioned conjugate gradient technique. As mentioned, preconditioning of matrix leads to better convergence of sub-iterations in each QP iteration. It should be also noted that we are utilizing a hot start approach where an initial guess
for the increment is obtained from the solution of the previous increment. This is useful for locally convergent algorithms. If however, the initial guess is far from the solution, then global convergence is necessary. SQP methods can be modified by using merit functions and line searches to be globally convergent. In addition, the SQP method has a Hessian matrix in its formulation which is smaller than the full Hessian in Newton’s method.

Some disadvantages of SQP method are that it requires computation of null space of matrix which can be tedious. Comparisons can be made between SQP and Newton’s algorithm with submatrices. In the latter, we follow an algorithm to sequentially obtain the displacements, dual variables and deformations, whereas in SQP all the variables are updated simultaneously. This saves time and may potentially lead to better convergent properties. Moreover, in SQP, the preconditioning operation improves the behavior of Hessian matrix as compared to Newton’s method with submatrices.

Augmented Lagrangian method was found to be slower for the problem which was tested. Also, for every new iteration, penalty parameters need to be adjusted or tuned. Unlike SQP where second order derivatives retain the essential properties of the problem, augmented Lagrangian is an approximation of the objective and constraints. At each iteration updating the penalty parameter is needed, so the new problem may be completely different form the old one which may hinder convergence.

For snap back and snap through behavior, we used Crisfield’s arc length method and the suggested implicit path continuation method. The motivation behind implicit path continuation method is the introduction of a curve parameter which is generally used in homotopy methods to solve problems which encounter turning points in their solution path. These two methods are able to overcome singularity points and are able to ensure path continuity of the solution. Crisfield’s method uses an arc length constraint in the problem, whereas implicit path continuation method uses a predictor-corrector approach using Newton’s iterations. A tangent to the curve is used to obtain a predictor, and final solution is found by making corrections based on the predicted step. The new method has better runtime performance as compared to Crisfield’s arc length method. We also experimented with performing iterations in implicit
path continuation method using trust-region dogleg algorithms. The computational time for trust-region dogleg method is very high.

Based on the conclusions of this thesis, we can deduce that nonlinear programming has an important role and further scope in structural analysis problems and applications, and the shortcomings of Newton’s method can be tackled by suitable algorithms which have same quadratic convergence as Newton’s method and are also computationally efficient. In this regard, as a future work, the implicit path continuation method and SQP methods can be unified to formulate a numerically advantageous method that can also address issues with turning points and bifurcations. Implicit path continuation can also be combined with Levenberg-Marquadt algorithm to efficiently find the iterates. We are showing this schematically using the same flowchart as before:
BIBLIOGRAPHY


