Addis Ababa University

College of Natural and Computational Sciences

Computational Science Program

BROYDEN AND QUASI-NEWTON METHODS FOR NONLINEAR SYSTEMS OF ALGEBRAIC EQUATIONS AND EXTENSION TO HOMOTOPY AND OPTIMIZATION APPLICATIONS

BY: BETELEHEM MENGISTU

3/12/2020

ABSTRACT

In the field of science and engineering, researches often come across with problems model by system of nonlinear algebraic equations that cannot solve easily. Numerical convergence for many problems, typically solved by the Newton-Raphson algorithm, is sensitive to the initial guess and need computations of Jacobi and its inverse at each iteration. Emphasis in the present work is placed on the alternative approach, such as quasi-Newton, Homotopy Method and optimization method.

The choice of the optimal numerical method, which ensures the best convergence rate with minimum error for the corresponding system of nonlinear equations, is discussed. Effectiveness of the method is demonstrated by comparing the results norm errors of each method for our sample problem implemented using a MATLAB Program.

In order to avoid computation of Jacobian matrix at each iteration that is one problem of Newton method, Broyden method that use an approximation matrix that is updated at each iteration in place of the Jacobian matrix. Moreover, to reduce the large step size that leads to wrong path, we use Safeguarded and Damped methods for Newton as well as Broyden's method. In addition to this, we use different Modified Broyden's or Quasi-Newton method to solve SNAE. The modified Broyden's method gives a better optimal solution than both pure Newton and Broyden's methods.

In order to avoid failure of convergence of poor starting guess of problem for Newton method as well as Broyden's method we apply Numerical algorithm like NHAM that combining Newton method and Homotopy analysis method with Euler and Runge-Kutta type to increase the range of initial value and efficiency of convergence. The results from NHAM with RK6 NHDE steps were more accurate than other NHAMs.

Finally, solve our sample SNLAE by globally convergent, Optimization methods in order to get good and sufficient initial guess for other methods. The result from Levenberg-Marquard method, were significantly reliable and more accurate than Steepest Decent, BFGS and DFP methods, however, computation cost is high.

ACKNOWLEDGMENT

I would like thank my creator GOD help me to prepare this paper. Secondly i would like to say thank my supervisor professors Okey Oseloka Onyejekwe. He was an invaluable source of knowledge, guidance, encouragement and support throughout both the course of this thesis and my graduate studies. Finally I would like thanks a lot for my family and my friends to your appreciation and help.

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURE

ACRONYMS

- SNAE – System of Nonlinear Algebraic Equation.
- HAM - Homotopy Analysis Method.
- NHAM - Numerical Homotopy Analysis Method
- SNLAE – System of Nonlinear Algebraic Equation.
- SSNAE - Solving System of Nonlinear Equation.
- NLAEs - Nonlinear Algebraic Equation.
- ODE – Ordinarily Differential Equation.
- NAEs - Nonlinear Algebraic Equations.
- QMN – Quasi Modified Newton.
- TSMM – Trapezoidal, Simpson and Midpoint Method.
- QN – Quasi Newton.
- MNM – Modified Newton Method.
- MHM – Modified Homotopy Method.
- GHN – Generalized Homotopy Method.
- MHPM – Multi-Stage Homotopy Perturbation Method.
- RK – Runge-Kutta.
- DEP – Davidon-Fletcher-Powell.
- BFGS – Broyden-Fletcher-Gloldfarb-Shanno.
- HBFGS – Hessian Broyden-Fletcher-Gloldfarb-Shanno.
- BC – Broyden Class.
- MBC – Modified Broyden Class.
- f – Function.
- x_0 - Initial Value.
- x^* - Approximate Solution.
- J - Jacobian Matrix.
- A^{-1} - Invers of Matrix.
- λ - Lambda Parameter.
- NS- Norm of step size
- NF- Norm of function

CHAPTER ONE

Introduction

Mathematics generally is playing a big role, which cuts across all aspects of human life: social, engineering, economic and physical lives. Algebra is one of the most critical aspects of mathematics.

Over the years, we have been taught on how to solve equations using various algebraic methods. These methods include the substitution method and the elimination method. Other algebraic methods that can be executed include the quadratic formula and factorization [1]. In Linear Algebra, we learned that solving systems of linear equations can be implemented by using row reduction as an algorithm. However, when these methods are not successful, we use the concept of numerical methods.

Numerical methods are used to approximate solutions of equations, when exact solutions cannot be determined via algebraic methods. They construct successive approximations that converge to the exact solution of an equation or system of equations.

In this paper, we will describe many numerical methods to solve nonlinear algebraic equations and compare the methods depending on a certain criteria.

Newton–Raphson method is the most popular technique for solving nonlinear equations. It has quadratic rate of convergence, but one of its major disadvantages of this methods is that a computation of Jacobian matrix at each iteration. In order to diminish this problem, quasi-Newton method of rank-1can be used, which is presented by Broyden [59].

Quasi-Newton methods, those that are generalizations of the one-dimensional secant method, have been found to be very successful methods for solving nonlinear algebraic systems. Over the last decade, great deals of progress have made in determining very effective quasi-Newton methods that declines its weakness. One weakness of Broyden method is that, it is not self-correcting; this problem also gets improvement by Abu-Hour [60] that declines this problem.

The Newton and quasi-Newton methods, which are widely used, have advantage on their high speed of convergence once the initial guess of the root, which is sufficiently close to

exact solution to ensure convergence. In order to overcome, the choice of a better guesses of exact solutions of both methods several effort and progress have made, HM by **Liao** [61], HM can solve nonlinear algebraic systems and converge to a desired solution without close choice of initial guess. Another method that avoid initial guess problem of the methods is Optimization methods, like Steepest Decent, BFGS, DFP and Levenberg-Marquardt propose by authors see [62, 63, 64, 65].

Both Newton and Quasi-Newton methods are locally convergent algorithms, Damped Newton methods by [66] and Damped Broyden's that improves local convergence method to global convergent.

In the following, we consider numerical methods like Newton's, its variation like quasi-Newton methods and their modification to solve NLAEs. The result obtained by modified Broyden methods, have efficient and optimal than the other methods, however, its computation cost is very high.

The remainder of this introduction provides a very brief background on NLAEs, certain procedures for solving them numerically, and basic concept associated with algebraic systems

1.1. Background

1.1.1. Root finding method

One of the most common problems encountered in engineering or any other filed analysis is that given a function $f(x)$, find the values of x for which f (x) = 0. [6] The solution (values of x) is known as the roots of the equation. Root finding involves searching for the location where the function equals to zero.

There are two types of methods available to find the roots of algebraic.

1. **Direct Methods:** give the exact value of the roots in a finite number of steps. Direct methods determine all the roots at the same time.
2. **Indirect or Iterative Methods:** are based on the concept of successive approximations. Iterative methods determine one or two roots at a time.

The indirect or iterative methods are further divided into two categories: bracketing and open methods. Among the open methods, the Newton method is the most common method for successive approximation and the most popular method for solving a non-linear equation. This method has a high rate of convergence to a solution [2]. In general, numerical methods suggest that iterative solutions of nonlinear equations are more powerful than direct method.

***Definition* 1.1.**

An ***iterative method*** is a procedure that is repeated over and over again, to find the root of an equation or find the solution for system of equations [1].

1.1.2. The different between linear and nonlinear equation

Linear equation:

- Linear equation is the form of $ax+b=0$.
- A graphed of linear equation is a straight line.
- Its degree is always 1.

Non-linear equations: The form non-linear equation is $ax^2+by+c=0$.

- The graph is a curve.
- The degree of one of its term is different from 1.

***Definition* 1.2.**

***Algebraic Equation*:** is an equation, to obtained zero a sum of a finite number of terms each one of which is a product of positive integral power of the variables.

Definition 1.3.

Single nonlinear equation: solving a single equation in a single variable. Single variable are easier to solve than multi-variable problems.

***Definition* 1.4.**

System of nonlinear equation: is a set or collection of equation that you deal with all together at once.

1.1.3. A non-linear system of algebraic equations

When solving systems of two equations, the real solutions will represent the coordinates of the points where the graphs of the two functions intersect [1].

A system of nonlinear equations is simple analytical technique of the form $f(x)=0$, the function f is in the case of n variable $x_1, x_2, x_3, ...x_n$ which makes all function are zero.

$$\begin{aligned} f_1(x_1, x_2, x_3, ...x_n) &= 0 \\ f_2(x_1, x_2, x_3, ...x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, x_3, ...x_n) &= 0 \end{aligned} \qquad (1)$$

This system of equations can be written in vector form as

$$F(x)=0 \qquad (2)$$

$$F = (f_1, f_2, ... f_n) \qquad R^n \to R^n$$

$$x = (x_1, x_2, x_3, ... x_n) \qquad (3)$$

Where $(x_1, x_2, ..., x_n) \in R^n$ and each f_i is a nonlinear real function, $i = 1, 2, ..., n.$. We will use the term ***root* or *solution*** frequently to describe the final result of solving the systems.

1.1.4. Numerical system of nonlinear algebraic equations

Generally, numerical method is a mathematical tool to solve numerical problems. The implementation of a numerical method is to check an appropriate convergence in a programming language is called numerical algorithm.

Numerical approximation methods are usually needed for solving systems of equations when the equations are nonlinear. Solving a system of nonlinear equations is a problem that is avoided when possible, customarily by approximating the nonlinear system by a system of linear equations. When this is unsatisfactory, the problem must be tackled directly.

Nonlinear equations cannot in general be solved analytically. In this case, therefore, the solutions of the equations must be approached using iterative methods. [2]. Therefore, we are forced to solve it by approximation approach of iterative methods. In this paper, we will see some of iteration methods, which use to find the root of the equation.

The ***Jacobian matrix*** is a key component of numerical methods. In the next section will see how we compute Jacobian matrix numerically and we will try to compare with approximate computed of Jacobian matrix.

***Definition* 1.4**

The *Jacobian matrix* is a matrix of f is defined to be an $m \times n$ matrix, denoted by J , whose $(i, j)^{th}$ entry is $J_{ij} = \dfrac{\partial f_i}{\partial x_j}$, or explicitly, is called first order partial derivatives [1]

$$J = \left[\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \quad (4)$$

Hessian Matrix

Hessian matrix is a matrix of second order partial derivatives [1] $H = \left[\frac{\partial^2 f}{\partial x_i \partial x_j} \right]_{ij}$ such that

$$H(x) = \begin{pmatrix} \frac{\partial^2 f_1}{\partial x_1^2} & \frac{\partial^2 f_1}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f_1}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f_2}{\partial x_2 \partial x_1} & \frac{\partial^2 f_2}{\partial x_2^2} & \cdots & \frac{\partial^2 f_2}{\partial x_2 \partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f_n}{\partial x_n \partial x_1} & \frac{\partial^2 f_n}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f_n}{\partial x_n^2} \end{pmatrix} \quad (5)$$

1.1.5. Norms of Vectors

Let $\mathbf{X} \in R^n$ where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad (6)$$

Norms: provide tools for measuring errors.

There are two types of norm of vector

1. The l_2 norm for the vector x is called the **Euclidean norm**. It represents the length of the vector denoted by

$$\|x\| = \|x\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \quad (7)$$

2. The l_∞ norm represents the absolute value of the largest component in the vector x denoted by

$$\|x\|_\infty = \max_{1\le i\le n} |x_i|. \qquad (8)$$

Name	**Definition**	**MATLAB**
Euclidean norm	$\rightarrow \|x\| = \|x\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$	$\rightarrow norm(x)$
1-norm	$\rightarrow \|x\|_1 := \|x_1\| + ... + \|x_n\|$	$\rightarrow norm(x,1)$
∞-norm, max norm	$\rightarrow \|x\|_\infty := \max\{\|x_1\|, ..., \|x_n\|\}$	$\rightarrow norm(x,\inf)$

1.1.6. Rate of Convergence

Definition 1.7.

Rate of convergence measures how fast the error sequence goes to zero. [6].

If a sequence $(x_1, x_2, ...x_n)$ converges to α value r and if there exist real numbers $\lambda > 0$ and $\alpha \ge 1$ such that

$$\lim_{x\to\infty} \frac{|x_{n+1} - r|}{|x_n - r|^\alpha} = \lambda \qquad (9)$$

Then we say that α is the **rate of convergence** of the sequence.

1.**1.8. Types of rate of convergence**

Iterative methods can be classified by the *rate of convergence*.

Let $\{x_n\} \subseteq R^n$ and $x^* \in R^n$. Then

i. $x_n \rightarrow x^*$ ***q-quadratically*** if $x_n \rightarrow x^*$ and $k > 0$ there is such that

$$\|x_{n+1} - x^*\| \le k \|x_2 - x^*\|^2 \qquad (10)$$

ii. $x_n \rightarrow x^*$ ***q-superlinearly*** if

$$\lim_{n\to\infty}\frac{\left\|x_{n+1}-x^*\right\|}{\left\|x_n-x^*\right\|}=0 \qquad (11)$$

iii. $x_n \to x^*$ ***q-linearly with q-order*** $\sigma \in (0,1)$ if

$$\left\|x_{n+1}-x^*\right\| \le \sigma \left\|x_n-x^*\right\| \qquad (12)$$

for n sufficiently large.

we will see that often a method that is more slowly convergent, in terms of the convergence types defined above, can have a cost/iterate so low that the slow iteration is more efficient.

Sometimes errors are introduced into the iteration that is independent of the errors in the approximate solution.

Unlike Newton's method, Broyden's method as well as all of the Quasi-Newton methods converges super linearly.

1.2. Problem Definition

Nonlinear systems of equations have occurred in many important fields such as engineering, mechanics, medicine, chemistry, and robotics. In this study, we solve systems of nonlinear algebraic equations (SNAE) by using different numerical methods, such as Newton's method, Broyden's method(Quasi-Newton method), Modified Broyden Method, Newton Homotopy analysis method and optimization method. Finally, compare the accuracy and efficiency of each method by apply different methods on a sample problem defined by equation (13).

Problems: [Ishiharek (2001)]. $F(x); R^3 \to R$ is defined by

$$F(x)=\begin{cases} 4x_1-2x_2+x_1^2-3 \\ -x_1+4x_2-x_3+x_2^2-3 \\ -2x_2+4x_3+x_3^2-3 \end{cases} \qquad (13)$$

1.3. Objective of study

This paper was conduct to achieve the following objectives

- ✓ To code the algorithms of Newton-Raphson's, Quasi-Newton (Broyden's), Safeguarded and Damped for Newton's and Broyden's, Modified Broyden's, Trapezoidal, Simpson and Midpoint (TSMM), NHAM with Euler, RK2,RK4,RK5 and RK6 NHDE steps, steepest Decent, BFGS,DFP and Levenberg-Marquardt of optimization methods by using MATLAB program.
- ✓ To apply, the above numerical methods in solving test problem equation (13).
- ✓ To compare, the performance of algorithms by a certain criteria.
- ✓ Analyze the results of simulation and determine the efficiency of each method.

1.4. Scope of study

In this study, we will solve the nonlinear systems of algebraic equations by using several methods. The first method is Newton and Boyden's method are the most common method for solving the nonlinear systems of equations, in addition to this add Damped and Safeguarded method to minimize their limitation. The second one is quasi-Newton method, which is modified from the Newton's method and used modified Newton type, Trapezoidal Broyden's method and TSMM method. The third method will use in this study is the Newton Homotopy analysis Method (NHAM) to get a good initial point. This method is mixing Euler and Rung-kutta type to find the slope and finally get the approximate solution. The last method is Optimization method. Optimization means finding the best solution among many feasible solutions that are available to us. So, to solve the problem we use steepest descent, Levenberg-Marquardt methods, BFGS and DEP method.

1.5. Significance of study

Recently, nonlinear systems of algebraic equations appear in numerical applications frequently. Normally, we will use Newton's method and Broyden's method to solve the nonlinear systems of algebraic equations. In this study, in addition to modified Broyden's method we employ additional method, Newton Homotopy analysis method and Optimization method to solve SNAE. So, this study will illustrate and understand more about the performance and efficiency of those methods.

1.6. Methodology and Approach of the study

The remainders of the thesis chapters are organize as follows.

Chapter 2 contains an overview of literature that supports our study. Its focus on review different numerical methods by different authors in different years, which are used as a base for our study, like Newton-Raphson and Quasi-Newton and their modifications, the global convergent method HAM, different optimization methods and other related previous works.

Chapter 3 gives an overview of the basic numerical method, Newton, and their safeguard and Damped Newton methods. It also presents Newton method drawbacks, and it explains how to diminish this problem by using Safeguarded and Damped methods. Moreover, it contains variant of Newton method called Quasi-Newton. Finally, apply the methods on a test problem defined in section (13) and compare the result obtained from each method.

Chapter 4 gives an overview of HAM and NHAM. The different variant of NHAM; which are solving with different Runge-Kutta methods in order to solve NHDE. Finally presents the application of the numerical methods, NHAM and its variant on the test problem and their numerical results present in both numeric and graphical forms, finally comparison of methods also done here.

Chapter 5 gives overview of Optimization methods, Steepest Decent, Davidon-Fletcher-Powell's (DFP), Broyden, Fletcher, Goldfarb, and Shannon (BFGS), Levenberg-Marquardt (LM) methods are discussed in order to solve NLSAEs. These methods apply on a test problem, their numerical results are displayed for these methods by using a MATLAB program, and their results are compared.

Chapter 6 summarizes the overall methods, comparison of the overall methods by using numerical results and outlines possible directions for further research.

CHPTER TWO

2. LITERATURE REVIEW

2.1. System of Nonlinear Algebraic equation

Okorie and Charity [7]: the purpose of this work was to determine the best method of solving nonlinear equations and iterative method. The work outlined use four methods. They proof the limitation and cost of the system and show their performance. Finally, they recommended that the Newton's method is the best method of solving the nonlinear because of its high rate of convergence.

Wongee [8], on this researcher recommends and explain that nonlinear systems of equations to solve in numerical application. However, such nonlinear systems of equations are difficult to be solved either exactly or numerically. There for they recommend that we could apply several methods to solve the nonlinear systems of equations numerically like Newton's method, quasi-Newton method, and Homogony continuation method.

In the same fashion Hernandez *et al* [9] explain the development of a tool (application) to solve SNAE. SNE are usually complex and can't be solved analytically or inexact way. So in this paper shows the implementation of four numerical algorithms: Conjugate gradient method, Newton method, Ostrowski's method and Homotopy continuation methods to solve the method.

Shan and Atluri [10] in this paper they solved an iterative algorithm of SNAEs using ODEs with an optimization of vector-form: $F(x)=0$ based on an invariant manifold defined in the space of (x,t) in terms of the residual norm of vector.

2.2. Newton for System of Non-linear Algebraic Equation

Tjalling [11] ,this expository paper traces the History and development of the Newton Raphson method for SSNAE through letters, and publications of Isaac Newton, Joseph Raphson, and Thomas Simpson are show that how Newton's formulation differed from the iterative process of Raphson, and that Simpson was the first to give a general formulation, in terms of fluxional calculus, applicable to no polynomial equations. Simpson's extension of the method to systems of equations is exhibited.

Nusrat et al [12] defined and show that Newton's method is the most well-known method for solving nonlinear equations. Later Bhar *et al* [13] and recently Okorie and Charity[7] explaining and implementing that Newton's method is the best numerical approach for solving Nonlinear System of Algebraic Equations (NLSAEs).

At the same time, Shififul [15] explain and show Newton-Raphson Method. Apply this most powerful and popular root finding method to show the faster convergence rate (quadratic convergence). Finally he suggested that if the function is well behaved near the solution and the initial approximation is chosen carefully. But one of the main drawbacks of Newton's method is that it needs to evaluate the derivative at each iteration, which is sometimes complicated and computationally undesirable.

Luzanin and Rapajic [16]: show in this method was to use a given number of inner iteration for calculating approximation of the inverse Jacobine matrix, because it has an important influence on the convergence rate of the method. Local q-linear, q-superlinear and q-quadratically convergence of the modification of the general Newton method is proved.

2.3. Broyden's method

Remain [17], also was focus on by using two numerical methods involved in solving systems of nonlinear equations. First, they were applying Newton's method for solving multivariable equations, which involves using the Jacobian matrix. Secondly, they examine and show Boyden's method. Finally they cover the limitation of Newton method by use Broyden update method.

Muhammad*et al* [18], they use a new approach for solving system of nonlinear equations especially large-scale. This approach was based on true Jacobian for initial iterations and Broyden's update for subsequent iteration.

Towaiq *et al* **[**19], in this paper they propose two efficient algorithms based on Broyden's methods by using the central finite difference and modification of Newton's method for solving systems of nonlinear equations. The most significant features of these algorithms are to introduce their simplicity and excellent accuracy of the new approach.

Mamat *et al*[3] he also recommended that to solve quasi-Newton or Broyden like method; we can use the trapezoidal rule to solve system of nonlinear equations.

Osinuga [58], In this paper they introduce a variant of the Broyden-like method is proposed using the weighted combination of the Trapezoidal, Simpson and Midpoint quadrature rules, known as TSMM. The numerical tests confirm that TSMM is promising when subjected to comparison with other Broyden-like methods based on these rules.

2.4. Quasi Newton method

Hour *et al* in [21], applied their idea to improved methods for Quasi Newton's method (QN) called Quasi Modified Newton's (QMN) of type 1 and type 2to obtain an approximate solution for systems of nonlinear equations. The most significant features of these methods are their simplicity and excellent accuracy.

Dennis and More [22] show this paper was an attempt to motivate and Justify quasi-Newton method as useful modification of newton's method for general and gradient nonlinear system of equations**.**

2.5. Modified newton method

Satya [23] explained and show modified Newton method to get highly efficient, insensitive to the initial condition, to find the solutions with a very small the residual error. In general, it is difficult to choose a good initial condition for most large systems of NAEs. However they applied modified Newton method (MNM) and the modified Homotopy method (MHM).

Akintayo [24] he modified Newton's method for solving non-linear programing problems was presented in his work. The scheme constructed from the Taylor's series expansion and Adomian decomposition method. The proposed method is found to be reliable and converges faster than the Newton's method, as well as some of the existing modifications of the Newton's method for solving nonlinear optimization problem.

2.6. Newton Homotopy Analysis Method of SSNAE

Alshorman *et al* [25]; in this work, they highlight the key concepts in using the Homotopy analysis method (HAM) as a methodology used to construct efficient iteration formulas for solving nonlinear equations. The proposed method was experimentally characterized

according to a set of determine parameter which affect the systems. The result show that the potential and limitations of the new method and imply directions for future work.

Hosseini [26] a new iterative method for solving the system nonlinear equations was suggested based on Newton-Raphson Method and Homotopy analysis method.

Hanim *et al*[27]; In this project, they introduce Newton-Homotopy method using start-system was implemented to solve several nonlinear problem by showing the current result in terms of number of iterations and rate of convergence rates.

Hecter[28]; In this study paper also apply a new tool know as Generalized Homotopy method (GHM) for the solution of nonlinear differential equations was presented. In order to assess the benefits of this proposal, two nonlinear problems are solved and compared against other semi-analytic or numerical methods, the results show that GHM is a powerful tool and capable to generate highly accurate solutions.

Abbasbandy [29], they present an efficient numerical algorithm for solving nonlinear algebraic equations based on Newton–Raphson method and Homotopy analysis method. Hashim and Sazzad [30] also introduce a new version of Homotopy algorithms has been proposed which was more efficient than previous algorithms, because the start point can be choose arbitrary.

Recently, Chowdhury [31] proposed different approach, linear and non-linear stiff systems of ordinary differential equations are solved by the multi-stage Homotopy perturbation method (MHPM). The MHPM is tested for Runge-Kutta type method demonstrates the promising capability.

Izadias *et al* [32] this paper shows other approach utilizing. Newton Method and Homotopy Analysis Method (HAM), were proposed for solving nonlinear system of equations. Accelerating the rate of convergence of HAM, and obtaining a global quadratic rate of convergence are the main purposes of this approach. In addition to this the numerical results demonstrate efficiency, performance and the great freedom of selecting the initial guess the main quality of the method.

2.7. Optimization method

Abdullahi [32] this thesis gives a review of the methods for solving SNEs and unconstrained minimization of real-valued functions. Methods for solving systems of NAEs include; Newton's method, Quasi-Newton's method and Diagonal Broyden-like and Homotopy and continuation method. For unconstrained minimization it covers: Steepest Descent method, the Fletcher-Reeves and Gradient methods, the modification Newton's method and Quasi-Newton BFGS and DEP method using Analytic line search method to calculate the step length.

Mohd [33], He applied that the conjugate gradient method plays an important role in solving large-scaled problems and the quasi-Newton method is known as the most efficient method in solving unconstrained optimization problems. Therefore, in this paper, the new hybrid method between the conjugate gradient method and the quasi-newton method for solving optimization problem is suggested.

Al-Baali and Grandinett [34]; this study aims consider a family of damped quasi-Newton methods for solving unconstrained optimization problems. This damped technique modifies the Hessian approximations and maintains the global and superliner convergence property of a restricted class of quasi-Newton methods for convex functions is tested on a set of standard unconstrained optimization problems. Finally, it is shown that the damped technique improves the performance of quasi-Newton methods substantially in some robust cases (as the BFGS method) and significantly in certain inefficient cases (as the DFP *method)*

Taheri and Mammadov [35] show a new algorithm was proposed for the solutions of systems of nonlinear equations. This algorithm uses the combination of the gradient and the Newton methods with parameters adjusted. Also they use the gradient method due to its global convergence property. Finally the Newton methods speed up the convergent rate and compare the two methods to know their performance.

Jureccko and Duda [36] solved SNE with the use of optimization method like newton method, trust regine method, levenberg-Marquardt method and Damped Newton method for the search direction.

2.8. Steepest Descent

Kamoh and Dauda [14]; show that two basic methods of approximating the solutions of nonlinear systems of algebraic equations. The Steepest Descent method was a way of obtaining good and sufficient initial guess which is in turn used for the Broyden's method. On the other hand Broyden's method replaces the Newton's method which requires the use of the inverse of the Jacobian matrix at every new step of iteration. Inverse matrix is directly determined at each step by up-dating the previous inverse.

De Sterck [37] show and prof that steepest descent preconditioning was considered for recently proposed nonlinear generalized minimal residual (N-GMRES) optimization algorithm for unconstrained nonlinear optimization. Two steepest descent preconditioning variants were proposed. Firstly employs a line search, whereas the second employs a predefined small step.

Olver and Trogdon [38], also used Steepest descent method to solve numerical method.

2.9. Levenberg-Marquardt and BFGS

Asral and Ibrahim [39]; in this paper they present a new line search method known as the HBFGS method, which uses the search direction of the conjugate gradient method with the quasi-Newton updates. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) update is used as approximation of the Hessian matrix for the methods. The new algorithm is compared with the BFGS method in terms of iteration count and CPU-time. They also proved that the algorithm was globally convergent.

Wang *et al* [40] show in this paper a limited memory BFGS algorithm was presented for solving large-scale symmetric nonlinear equations, when a line search technique without derivative information is used. The global convergence of the proposed algorithm established under some suitable conditions.

Waziri [41] this paper presents a globally convergent hyper plan-BFGS method for solving nonlinear system of equation. The attractive attributes of our method were due to singularity free requirements and global convergence properties.

Ahookhosh et al [42] they describe and analysis Levenberg-Marquardt methods for solving SNAEs. More specifically, they proposed an adaptive formula for Levenberg-Marquardt parameter and analysis the local convergent of the method.

Bergou *et al* [43] The Levenberg-Marquardt algorithm was one of the most popular algorithms for the solution of nonlinear least squares problems. Motivated by the problem structure in data assimilation, they consider in this paper the extension of the classical Levenberg-Marquardt algorithm to the scenarios where the linearized least squares sub problems are solved inexactly and the gradient model is noisy and accurate only within a certain probability.

CHAPTER THREE

3. METHOD FOR SOLVING SYTSEM OF NONLINEAR ALGEBRIC EQUATIONS

The solution of systems of nonlinear algebraic equations applying in any field of science, engineering, or applied mathematics problems are that are encountered frequently [44].

In addition to this, systems of nonlinear algebraic equations are abundant in many applications requiring numerical simulation, and more robust and efficient methods for solving SNAE are continuously being sought.

.

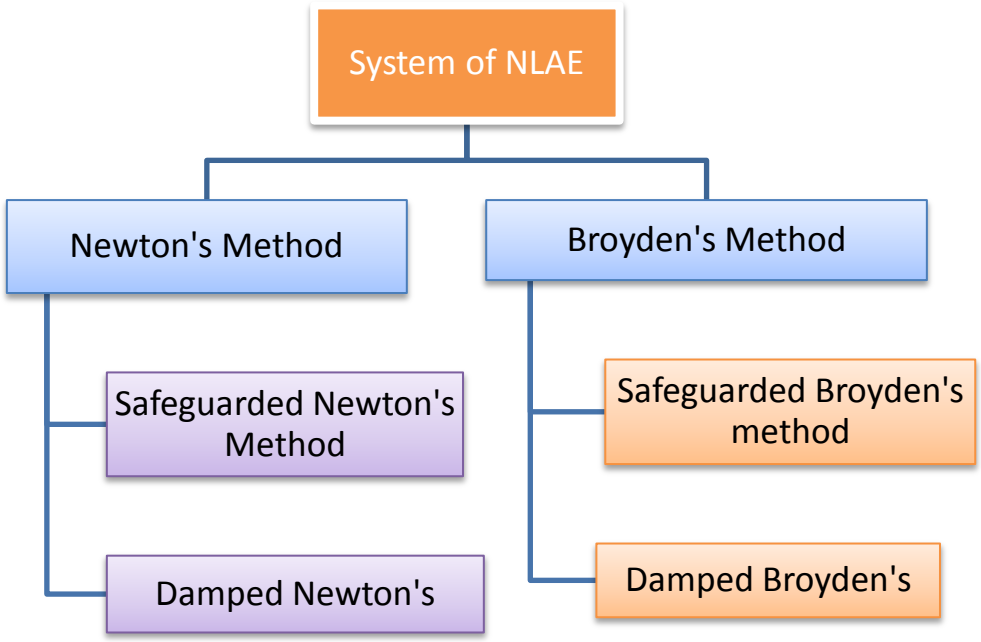


Figure 1: Schematic diagram of Newton's and Broyden's method.

3.1.NEWTON'S METHOD

The fundamental idea of Newton's method is to approximate the original function $f(x)$ by straight line. The key step in Newton method is to find appoint where the tangent line cross the x-axis, which means solving $f(x)=0$.

Newton's method uses a simple idea to provide a powerful tool for fixed point analysis. The idea is that we can use tangent lines to approximate the behavior of f near the root. The method goes as follows. We start with a point x_0, close to a root of f. The line tangent to f at x_0 will intersect the x-axis at some point (x_1, 0) [45]. The x-coordinate of this intersection point x_1, should be closer to the root of f than x_0. The process is shown in Figure 1.1.

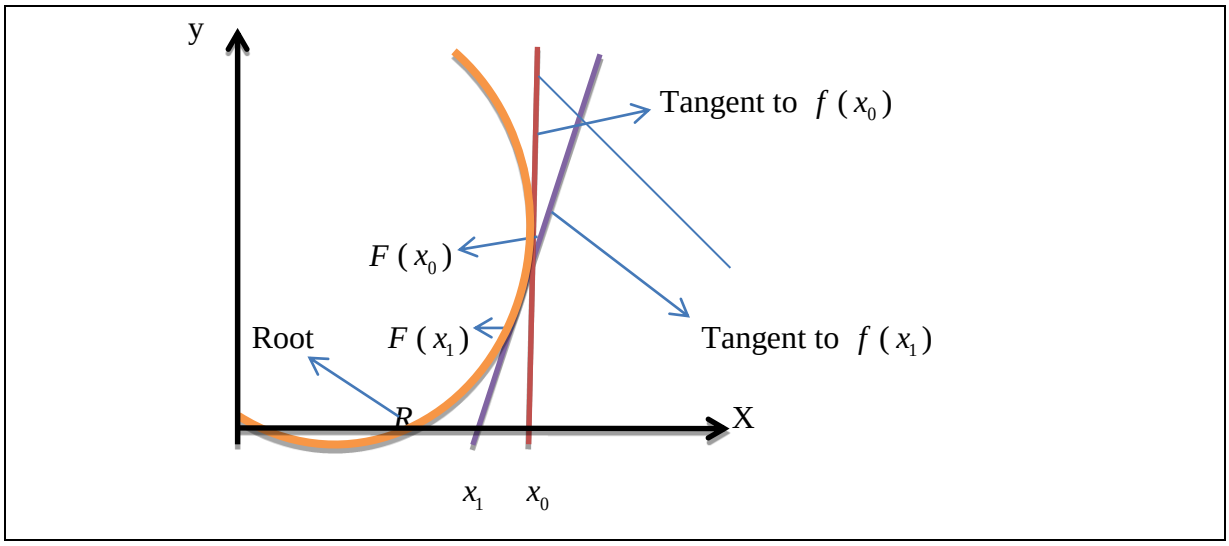


Figure 2: Showing Newton's approximation solution by using tangent line.

Based on this iteration process, we can find and get the approximate solution of the root. Moreover, Newton's method is the most popular method for solving a system of non-liner equation and this method has a high rate of convergence to a solution.

3.1.1. Derivation of Newton's method from Taylor series

The Taylor polynomial for $f(x)$ is

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)(x - x_0)^2}{2!} + ... \quad (14)$$

As the function approaches a root, higher-order terms of the Taylor polynomial will approach zero. Therefore, we can consider the higher-order terms trivial [46]. We truncate these terms to get a linear approximate to $f(x)$, give as

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0). \quad (15)$$

Since we are solving for a root, we set $f(x) = 0$ and solve for x by:

$$0 = f(x_0) + f'(x_0)(x - x_0). \quad (16)$$

From this we can derive the Newton Method, gives as

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \quad (17)$$

When n variables are involved, we need to approximate a *vector function* $F(x)$ by some linear function $F = Jx + c$, where J is a $n \times n$ matrix and c is some vector of length n.

The technique for approximating F by a linear function is to use the first two terms in Taylor series expansion. Give the value of F and its partial derivatives with respect to x at some point x_i, we can approximate the value at some point x_{i+1} by the two first term in a Taylor series expansion around,

$$F(x_{i+1}) \approx F(x_i) + \nabla F(x_i)(x_{i+1} - x_i) \quad (18)$$

The expression ∇F is the matrix of all the partial derivatives of F.

This creates an iterative process to more accurately approximation a root.

3.1.2. Newton for single equation

The Newton method is based on the principle that if the initial guess of the root of $f(x)=0$ is at x_i , then if one draws the tangent to the curve at $f(x_i)$, the point x_{i+1} where the tangent crosses the x -axis is an improved estimate of the root like Figure 2..

From the equation of eq(14) we can drive,

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad (19)$$

Equation (19) is called the Newton formula for solving single nonlinear equations of the form $f(x)=0$ .So starting with an initial guess x_i , one can find the next guess, x_{i+1}, by using equation (19). One can repeat this process until one finds the root within a desirable error tolerance.

3.1.3. For system of equation for Newton's method for SNAE

First, we need to introduce the concept of the Jacobian matrix (also known as the matrix of first partial derivatives)

In this paper we can take systems of equations and express those systems in the form of matrices and vectors. Therefore, we can express the nonlinear system as a matrix with a corresponding vector.

(1) Let F be a function which maps $\Re^n$ *to* $\Re^n$

$$F(x_1, x_2, ..., x_n) = \begin{bmatrix} f_1(x_1, x_2, ..., x_n) \\ f_2(x_1, x_2, ..., x_n) \\ \vdots \\ f_3(x_1, x_2, ..., x_n) \end{bmatrix} \qquad (20)$$

(2) Let $x_i \in \Re^n$. Then x represents the vector

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{21}$$

Where $x_i \in \Re^n$ and $k = 1, 2, ..., n$

(3) We know that $F(x)$ is the Jacobian matrix of F.

Then the Jacobian matrix $J_f(x) = \dfrac{\partial f(x)}{\partial x}$ is a $m \times m$ matrix whose (i, j) element is given by

$$J(x)^{-1} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{22}$$

The Newton iteration for systems of nonlinear equations takes the form

$$x_{k+1} = x_k - J^{-1}F(x_k) \tag{23}$$

Where $k = 1, 2....n$ represents the iteration, $x_i \in \Re^n$, F is a vector function, and $J(x)^{-1}$ is the inverse of the Jacobian matrix .

Then update

$$x_k = x_k + s_k \tag{24}$$

Where $S_{(k)} = -J(x_k) / F(x_{(k)})$

Newton's method is quadratically convergent.

Generally, the procedure of Newton's method for solving systems nonlinear algebraic equation follows the following main step.

- Set an initial guess $x_{(k)}$.
- Solve the linear system $-J_{(k)}^{-1}S_{(k)} = F(x_{(k)})$
- Compute or update solution $x_{(k+1)} = x_{(k)} + s_{(k)}$

A significant weakness of Newton's method is that, for each iterations a Jacobian matrix must be computed, so [19] this method is very expensive and has the following **disadvantages:**

1. Need a good initial solution $x_{(0)}$ close to the solution x^*.
2. Requires $n^2 + n$ function evaluation at each iteration (n^2 for Jacobian matrix and n for $F(x)$).
3. $J(x_{(k)})$, must be nonsingular for all k and $J(x^*)$ is invertible.
4. Need to compute n^2 partial derivative for $J(x_{(k)})$ and $J^{-1}(x_{(k)})$ at each step.

The **advantage** of this method is that $\{x^{(k)}\}_{k=0}^{\infty}$ **converges quadratically** to x^* and the scheme above is self-corrective when $J(x_{(k)})$ is non-singular.

3.1.4. Problem Definition

Apply the following sample problem to find numerical result of system nonlinear algebraic equation.

Problems: [Ishiharek (2001)]. $F(x); R^3 \to R$ is defined by

$$F(x) = \begin{cases} 4x_1 - 2x_2 + x_1^2 - 3 \\ -x_1 + 4x_2 - x_3 + x_2^2 - 3 \\ -2x_2 + 4x_3 + x_3^2 - 3 \end{cases}$$

With initial guess $x_0 = [1,1,1]^T$, Error Tolerance 0.0001 and maximum number of iteration are 25.

Algorithm Pure Newton's Method

Step 1: Choose an initial estimate x_0 $\max iteration = 25$ $toi = 0.0001$,

pick Error tolerance $\varepsilon > 0$, and set$k$:= 0:

Step2: Solve $-J^{-1}F(x_k) = s_k$ to find step (derivative of Jacobian matrix and function F)

Step3: find update $x_{k+1} = x_k + s_k$

Step4: Repeat the following sequence of steps until $err(k) < tol$

Step5: Numerical output

Numerical Output

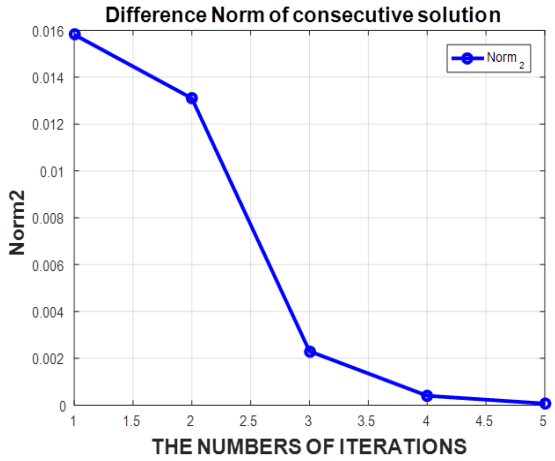
For comparison purpose, pure Newton's method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs, number of iteration is 25 and error tolerance of 0.0001.

Table 1: NS Vs NF and norm-2 of error for Pure Newton method

Pure Newton Method Norms			
K	NS	NF	Norm-2
1	0.72048401940794082000	0.01581672496379229300	0.015817
2	0.00399489581514759180	0.01309842229558167700	0.013098
3	0.00316883817161130050	0.00230332754821952880	0.002303
4	0.00055668869294983050	0.00040417804397699169	0.000404
Norm of F 7.095062153998490e-05			
CPU Time Newton Method = 0.0468003			

As shown in table 1, the MATLAB code for Newton method, it shows the norm error of the solution, step size and function are continuously decreases and converge to a solution for a sample problem with in 4 iteration. Moreover, we can observe that the number of iteration is smaller than other methods.

Figure 3.A **Figure 3.B**



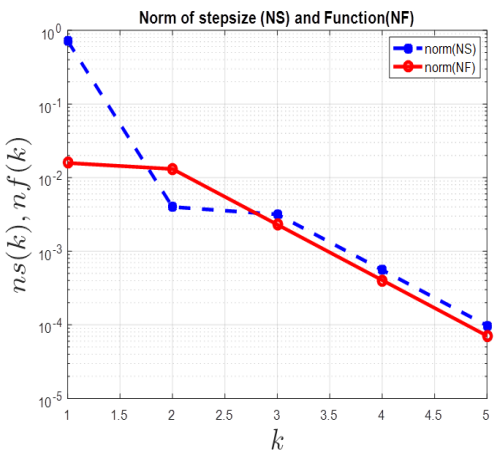


Figure 3: pure Newton's method of norm-2 and NS Vs NF.

As we seen in figure 3A and 3B, Newton method converges to problem (13) within 4 iterations. However, when we see norm error of step size and norm error of function, as well as norm of the solution graph are decreases continuously to the solution.

3.2.SAFEGUARDED NEWTON METHODS FOR SYSTEM OF NLAE

If the initial estimate of the root is poor, then Newton's method may diverge. In these instances, the stability of Newton' method can often be enhance, at a cost, by monitoring the iterate to ensure that they improve rather than deteriorate with each iteration.

Since the norm of the function value $\|f(x)\|$ is precisely zero at a root, one may view an literate's as yielding an improvement if it reduces this norm. [48]If an iterate increases the norm of the function value, then one can cut the step length prescribed by the newton method in half, and continue cutting it in half, until the revised iterate yields an improvement.

Cutting the step length in half, when necessary, prevents Newton's method from taking a large step in the wrong direction, something that can occur early in execution if the starting value is poor or the function is irregular. Newton methods that implement this feature are said to be **safeguarded Newton's method**.

So that the safeguarded Newton's method needs for Newton Method without any up and down line in the graph, it can move smoothly by correcting the direction. We will compare their different with safeguarded and without safeguarded Newton method in numerical output.

To approximate the solution of test problem defined by equation (eq 13), we wrote an algorithm for Safeguarded Newton method as shown below.

Algorithm of Newton with Safeguard method

Step 1: Choose an initial estimate x_0 $\max iteration = 25$ $toi = 0.0001$, and set *k*:= 0:

Step2: Solve $-J^{-1}F(x_k) = s_k$ to find step (derivative of Jacobian matrix and function F)

Step3: *if* k :1 *upto* $\max iteration$

update $x_{k+1} = x_k + s_k$

if $norm\ f_k > f_{k+1}$ $s_{k+1} = s_k / 2$

else $norm\ f_k < tol$ Stop

Step4: Numerical output

Numerical Output

For comparison purpose, Safeguarded Newton's method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

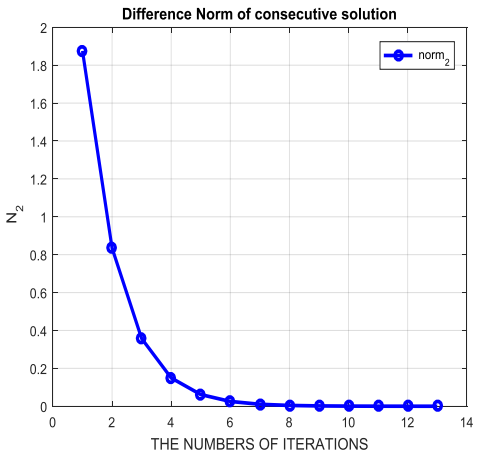
Table 2: NS Vs NF and norm-2 of error for Newton safeguard method

Newton Safeguard Method Norms			
K	NS	NF	Norm_2
1	0.18012100485198521000	1.87543814606506180000	1.875438
2	0.09691094100112303300	0.83507360234244854000	0.835074
3	0.04687426956599485800	0.35753751755793162000	0.357538
4	0.02094834583292123600	0.14950629920794184000	0.149506
5	0.00893799243830142860	0.06174631380699381600	0.061746
6	0.00372564297854929880	0.02533616596238180700	0.025336
7	0.00153565465195038650	0.01035487626221338700	0.010355

8	0.00062930188205290074	0.0042186420457737480	0.004219
9	0.00025692101704022233	0.0017132086479259736 0	0.001713
10	0.00010455739205835732	0.00069317810472415006	0.000693
11	0.00004240784113094783	0.00027920268179133350	0.000279
12	0.00001713220072970612	0.00011182354515060786	0.000112
Norm of F 4.446360116061855e-05			
CPU Time Newton safeguard Method = 1.3260085			

As shown in table 2, the MATLAB code for Newton safeguard method, it shows the norm error of the solution, step size and function are continuously decreases, it can converge for a sample problem with in 12 iteration. Moreover, we can observe that the number of iteration is longer than pure Newton method.

Figure 4.A **Figure 4.B**



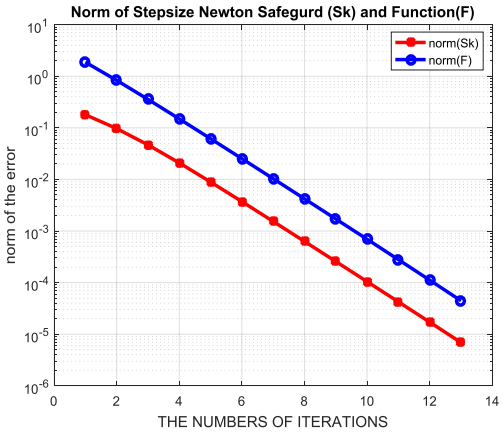


Figure 4: Newton Safeguard norm_2 and NS Vs NF .

As we seen in figure 4A and 4B, Newton safeguard method converges to the solution within 12 iterations. However, when we see norm error of step size and norm error of function, as well as norm of the solution are decreases continuously and move smoothly to the solution.

3.3.USING DAMPED NEWTON'S METHOD FOR SOLVING SNAE

Damping is the way to limit vibrations and is essential for protecting the system in which it operates. In addition, plays a crucial role in fixing the borderline between stability and instability in many dynamical systems.

The damped Newton method is modification to the Newton method that damped the step size, when the solution is not in the area of local convergence and has better results for global convergent.

As mentioned before, the quadratic convergence of the Newton method is local. It has no global convergence properties [49]. This can be a problem for large time steps or small radius of convergent due to complicated algebraic equations.

To achieve convergence for 'bad' initial guesses, one may globalize Newton by introducing a damping parameter λ that extended to a Damping Newton method, which can reduce step size[50].

This leads to the following algorithm

$$J(x_k)s_k = -F(x_k), \quad k = 0,1,2,..... \tag{25}$$

$$x_{k+1} = x_k + \lambda s_k, \quad \lambda \in (0,1) \tag{26}$$

In order to create an efficient and robust solution method for problems, this iteration must be combined with an adaptive step length control. Within such a procedure the damping factors λ_k should be chosen in such a way that the iteration x_k approach successively the solution x^*

.

Where $0 < \lambda_{(k)} \leq 1$ is chosen such that

$$\left\| F(x_{(k+1)}) \right\| < \left\| F(x_{(k)}) \right\| \tag{27}$$

In practical problems this Damped Newton's algorithm convergence, if that $\lambda_{(k)} = 1$, we obtain the original Newton's method and the process how to change λ is called line search procedure.

Algorithm: Damped Newton Method

Step 1: Choose an initial estimate x_0 $\max iteration = 25$ $toi = 0.0001$, set k:= 0:

$\lambda = 0.1$ initial Damping factor

$\min relaxation\ facter = 1e-10$ (minimal permitted damping factor)

$convergent\ criter = 1e-4$ (max permitted damping factor)

$F(x)$ (nonlinear system of function)

$J(x)$ (Jacobian of the function)

Step 2: if i=1 up to k Compute step size Sk by starting very small number and get x_1

$$sk = sqrt(eps) * x(i)$$
$$xi = x(i) + sk$$

Compute, Numerical Jacobian matrix

Step 3: compute step size $\rightarrow$ $s = -J / F1$, $x_{i+1} = x_i + lambda * s$ for s_k (Numerical Jacobian matrix & F)

While $err > convergent\ critera\ and\ k < \max iter$

Step 4: compute next iteration $\rightarrow$ $sk_2 = sqrt(eps) * x_{i+1}$, $x_{i+2} = x_{i+1} + sk_2$

Step 5: If $\max abs(lam) * s_{k+1} < \max abs(s)$

$lam = lamb / 2$;

Step 6: else compute the next iteration until we get the max iteration

Step 7: Numerical output

Numerical Output

Damped Newton's method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. Using inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 3: NS Vs NF and norm_2 of error for Damped Newton's method

Damped Newton Method Norms			
k	NS	NF	Norm_2
1	0.06212660435224746900	3.37885722485194150000	1.950784
2	0.11542264062631814000	2.85202623592814190000	1.646618
3	0.19835120591793734000	2.03458863623574080000	1.174670
4	0.28636069868026998000	1.10484929988005050000	0.637885
5	0.18276889514168998000	0.54921888104419803000	0.317092
6	0.12748723909361417000	0.39113738319838903000	0.225823
7	0.06894640708160244800	0.21722908314778130000	0.125417
8	0.04306340026633177800	0.13382839102105981000	0.077266
9	0.02771076747085155600	0.08784401924739994300	0.050717
10	0.01447948351624241100	0.04467642533015685200	0.025794
11	0.01025967531514643100	0.03219487236178666300	0.018588
12	0.00554526267645081940	0.01738027847138223100	0.010035
13	0.00346226861582214470	0.01073968112271453700	0.006201
14	0.00222259642189659090	0.00702917878750619180	0.004058
15	0.00116751443309429210	0.00360129303506429020	0.002079
16	0.00082689696372455288	0.00260102405082797890	0.001502
17	0.00044375169346891699	0.00138895748020593050	0.000802
18	0.00028007333477501450	0.00086925268228148125	0.000502
19	0.00017804850147661242	0.00056293673663157442	0.000325
20	0.00009410462849888113	0.00029019530947488093	0.000168
21	0.00006663370010954759	0.00020976651637001582	0.000121
22	0.00003550255756444007	0.00011099816458086094	0.000064
Solver converged.			
CPU Time for Damped newton is 0.0468003			
Norm of F 1.1100e-04			

As shown in table 3, the output of the MATLAB code Damped Newton method, it shows the norm error of solution, the step size and function are decreases continuously, and it can converge for a sample problem with in 22 iterations. Moreover, we can observe that the number of iteration is two times longer than pure Newton method.

Figure 5.A

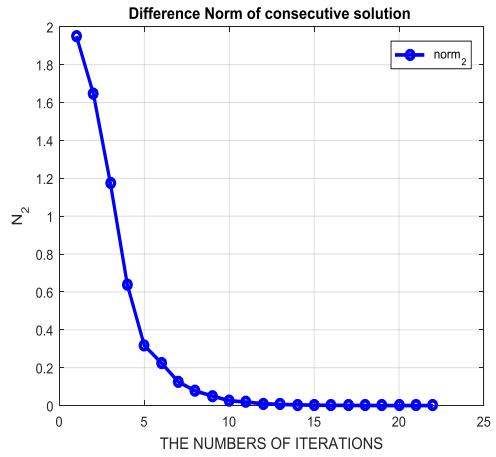


Figure 5.B

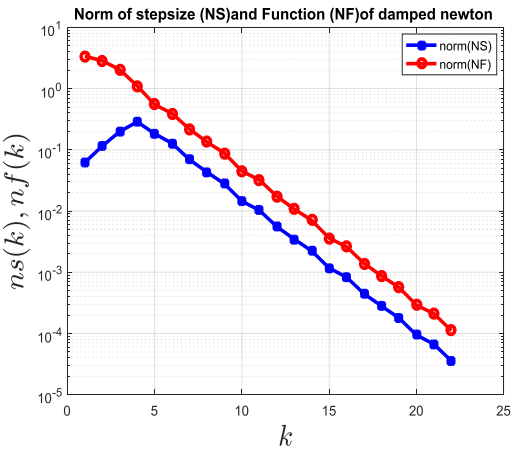


Figure 5: Damped Newton method of norm_2 and NS Vs NF.

As shown in figure 5A and 5B, Damped Newton method converges for our problem within 22 iterations. However, when we see norm error of step size and norm error of function, as well as norm of the solution graph are decreases continuously and move smoothly to the solution.

COMPARISONS OF NUMERICAL OUTPUT FOR SOLVING SNLAE OF PURE NEWTON, NEWTON SAFEGUARDED AND DAMPED NEWTON METHODS

Iteration	Norm_2 error for solving system of Nonlinear Algebraic Equation		
	Pure Newton method	Newton with safeguarded	Damped newton method
1	0.015817	1.875438	1.950784
2	0.013098	0.835074	1.646618
3	0.002303	0.357538	1.174670
4	0.000404	0.149506	0.637885
5		0.061746	0.317092
6		0.025336	0.225823
7		0.010355	0.125417
8		0.004219	0.077266
9		0.001713	0.050717

10		0.000693	0.025794
11		0.000279	0.018588
12		0.000112	0.010035
13			0.006201
14			0.004058
15			0.002079
16			0.001502
17			0.000802
18			0.000502
19			0.000325
20			0.000168
21			0.000121
22			0.000064
CPU time	0.0468003	1.3260085	0.0468003
Norm of F	7.095062153998490e-05	4.446360116061855e-05	1.1100e-04

Table 4: Comparisons of Newton, Safeguarded and Damped method Euclidian norm.

Generally, when we summarized these three methods, Pure Newton method has convergent property with small number of iteration and small CPU time. However, norm of F is large. When we see, Newton Safeguard and Damped Newton methods have use a large number of iterations. In order to correct the wrong direction and to help the graph move smoothly of Newton method uses the Safeguarded and Damped method.

3.4.BROYDEN'S METHOD FOR SYSTEM OF NAE

In the previous section, the one we examined the numerical method known as Newton's method. We established that one of the major disadvantages of this method was that Jacobian matrix, $J(x)$ and its inverse must be computed at each iteration. We therefore want to avoid this problem. [1]There are methods known as Broyden method or Quasi-Newton methods that use an approximation matrix that is updated at each iteration in place of the Jacobian matrix computation.

This indicates that the form of the iterative process for Broyden's method is almost the same that used in Newton's method. The only exception being that an approximation matrix B_k is implemented instead of J(x).

Definition 3.4.1

Broyden's method: [20] is a method for solving the system of nonlinear equation and is designed to improved Newton's method with respect to storage and approximation of the Jacobian. Broyden's method is generalization of the secant method to multiple dimensions. Once the initial value x_0 is obtained, then the next approximation x_1 is calculated as the same manner as newton method. In order to obtain x_2 the secant method uses update at the place of Jacobian.

To determine x_2, the matrix $J(x_1)$ is replaced by B_k as follows [1]:

$$x_2 = x_1 - B_k^{-1}F(x_1) \tag{28}$$

Once the x_i is determined, then the new approximation x_{i+1} is obtained by executing the following

$$B_{k+1} = B_k + \frac{(s_k - B_k y_k)}{s_k^t B_k y_k} s_k^t B_k \tag{29}$$

Where $y_k = F(x_k) - F(x_{k-1})$ and $s_k = x_k - x_{k-1}$.

Hence, the number of scalar function evaluation is reduced from $n^2 + n$ to n calculations are still required to solve the associated $n \times n$ linear system:

$$B_k s_{k+1} = -F(x_k) \tag{30}$$

The reduction form the quadratic convergence of Newton's method to superlinearly convergence can be improved by employing a matrix inversion formula of Sherman and Morrison.

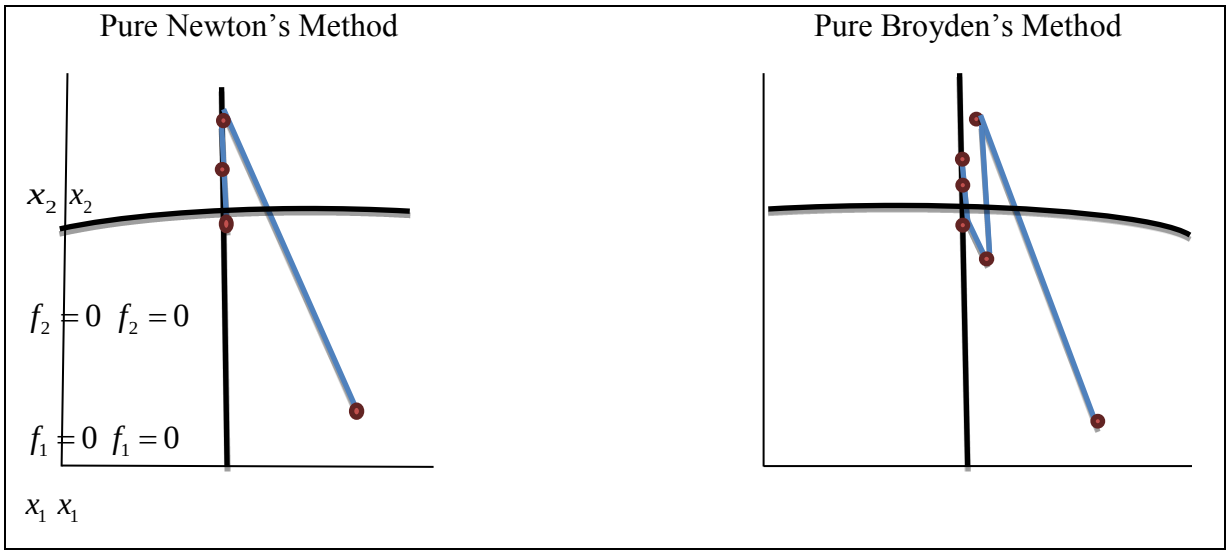


Figure 6: Convergence paths for Newton and Broyden's Methods

Advantage of Broyden's method

Reduction of computations is the main advantage of Broyden's method. More specifically, [1] the way the inverse of the approximation matrix, B_k^{-1} can be computed directly from the previous iteration, $B_{k=1}^{-1}$ reduces the number of computations needed for this method in comparison to Newton's Method.

Disadvantage of Broyden's method

One thing that is seen as a disadvantage of this Broyden method, it does not converge quadratically and where the initial iterate is far from a solution. This may mean that more iteration may be needed to reach the solution, when compared to the number of iterations Newton's method requires. Another disadvantage of Broyden's method has not self-correcting property. This means that in contrast to Newton's method have self-correct by it itself. This may cause only a slight inaccuracy in the iterations compared to Newton's, but the final iteration will be the same.

Algorithm: Pure Broyden's Method

Step 1: Choose an initial estimate $x_0 = 1$, identity matrix B_0, $err\,tol = 0.00001$ and Set $k \leftarrow 0$

Step 2: Compute function $F(x_k)$

Step 3: Compute the secant approximation method to get the next iterate.

$x_2 = x_1 - B_k^{-1} F(x_1)$

Step 4: Compute $s_k = x_k - x_{k-1}$

Step 5: Compute $y_k = F(x_k) - F(x_{k-1})$

Step 6: Compute Broyden's update $B_{k+1} = B_k + \dfrac{(s_k - B_k y_k)}{s_k^t B_k y_k} s_k^t B_k$

Step 8: Stopping criteria **do step-2 up to step-6** Until $\|F(x_k)\| < tol$

Step 9: Numerical output.

Numerical Output

Broyden's method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001

Table 5: NS Vs NF and norm_2 of error for Pure Broyden's Method

Pure Broyden Method Norms			
K	NS	NF	Norm-2
1	1.14412280563536850000	5.65685424949238060000	5.656854
2	1.09342628886753480000	10.77032961426900700000	10.770330
3	1.07842458196501840000	21.88742104497466600000	21.887421

4	0.98138502716476683000	25.98008282531208500000	25.980083
5	1.04834339954118370000	7.03957885959067390000	7.039579
6	0.23062333540457639000	14.617160432410799000000	14.617160
7	0.21331703545724862000	2.90447621260659260000	2.904476
8	0.24944013657353503000	0.99087105265382125000	0.990871
9	0.23829655216459428000	0.11993798013063228000	0.119938
10	0.23709584498535291000	0.01066882238118191600	0.010669
11	0.23952803115626350000	0.00031694256037647067	0.000317
CPU Time Broyden Method = 426.9435368			
Norm of F 1.589988716789636e-05			

As shown in table 5, the output of the MATLAB code of Broyden method, it shows the norm error of solution, the step size and function are continuously decreases, it can converge for a sample problem with in 11 iteration. Moreover, we can observe that the number of iteration is longer than pure Newton method.

Figure 7.A

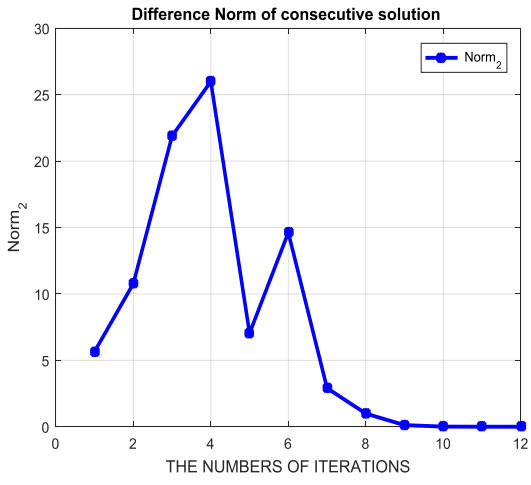


Figure 7.B

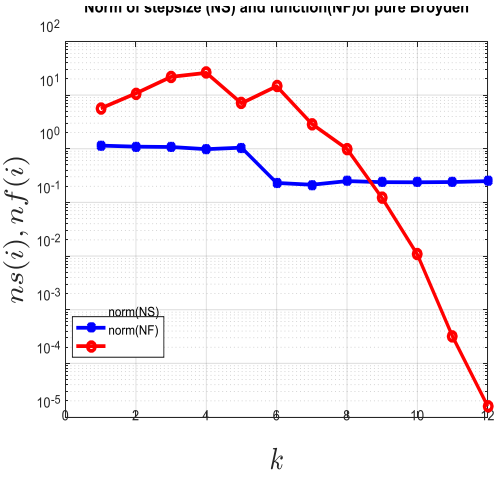


Figure 7: Pure Broyden's method of Norm_2 and NS Vs NF

As we seen in figure 7.A and 7.B, Broyden method converges for our sample problem within 11 iterations. However, when we see norm error of step size and norm error of function, as well as norm of the solution graph are not decreases continuously.

3.5. SAFEGUARDED BROYDEN'S METHOD

If the initial estimate of the root is poor, then Broyden's method may diverge. In these instances, the stability of Broyden's method can often be enhance, at a cost, by monitoring the iterates to ensure that they improve rather than deteriorate with each iteration.

Since the norm of the function value $\|f(x)\|$ is precisely zero at a root, one may view an iterate as yielding an improvement if it reduces this norm.[48]. If an iterate increases the norm of the function value, then one can cut the step length prescribed by the Broyden's method in half, and continue cutting it in half, until the revised iterate yields an improvement.

Cutting the step length in half, when necessary, prevents Broyden's method from taking a large step in the wrong direction, something that can occur early in execution if the starting value is poor or the function is irregular. Broyden methods that implement this feature are said to be **safeguarded Broyden method**.

Algorithm of Broyden's With Safeguard method

Step 1: Choose an initial estimate x_0 $\max iteration = 25$ $toi = 0.0001$ and set k:= 0: $B = identity\ matrix$.

Step2: Solve $S = B / F(x_i)$ to find step of size(identity and function F)

update $x_{k+1} = x_k + s_k$

Step3: *if* $step > \max tolerance$

Safeguard method $step_2 = step / 2$ (divided step size by two)

Step 4: else *if* $step <= \max tol$,Compute Broyden update

$s = x_{k+1} - x_k$

$y = F_{k+1} - F_k$

$$B = B_{k+1} + \frac{(y - B \times s)s'}{s' \times s}$$

Step 6: Numerical output.

Numerical output

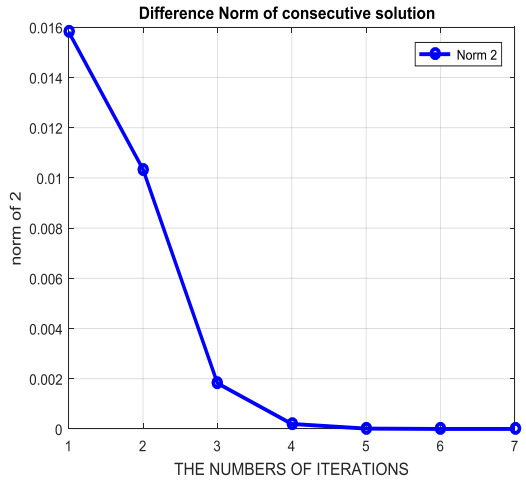
Broyden's safeguard method are applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 6: NS Vs NF and norm_2 for Broyden's with safeguard

Broyden with Safeguard Method Norms			
K	NS	NF	Norm_2
1	0.72048401940794082000	0.01581672496379229300	0.015817
2	0.00355267047390306650	0.01033568442185572000	0.010336
3	0.00178490139295727890	0.00181994106053278760	0.001820
4	0.00033226431362815191	0.00019938761959927399	0.000199
5	0.00003760450264711686	0.00001390010341066786	0.000014
6	0.00000274742290738359	0.00000024831133459377	0.000000
CPU Time for Broyden with safeguard Method is 0.0312002			
Norm of F 4			

As shown in table 6, the output of a MATLAB code for Safeguarded Broyden method, it shows the norm error of solution, the step size and function are continuously decreases, it can converge for a sample problem with in 6 iteration. Moreover, we can observe that the number of iteration is smaller than pure Broyden method.

Figure 8.A **Figure 8.B**



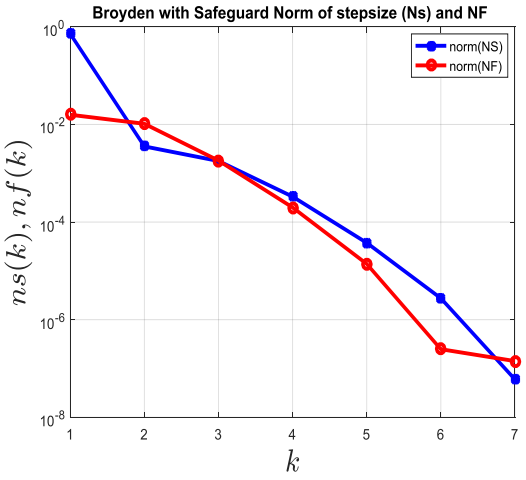


Figure 8: Broyden's with Safeguard Method for norm_2 and NS Vs NF.

As we seen in figure 8.A and 8.B, Broyden method converges to the solution within 6 iterations. But when we see norm error of step size and norm error of function, as well as norm of the solution graph are decreases continuously and move smoothly to the solution.

3.6. Damped Broyden's Method

The idea of damped Broyden's method is similar with the Damped Newton method. When the solution is not in the area of local convergence and has well results for global convergent.

As mentioned before, the superlinearly convergence of the Broyden's method is local. [49]It has no global convergence properties. This can be a problem for large time steps or small radius of convergent due to complicated algebraic equations. That is why we also need a damped Broyden iteration.

The different between the damped Broyden's and pure Broyden's methods is that directional variables, calculated is multiplied with a damping factor λ [48]. In order to alleviate this difficulty, we find a corrected direction $x_{(k+1)}$ by introducing a relaxation parameter λ as follows.

$$J(x_k)s_k = -F(x_k) \tag{31}$$

$$pp = \lambda * s$$

$$x_1 = x + pp,$$

$$y = x_1 - x,\ s = F_1 - F_0$$

$$B_{k+1} = \frac{B_k + (s_k - B_k y_k)y'}{y_k' B_k s_k} y_k' B_k \tag{32}$$

Where $0 < \lambda_{(k)} \leq 1$ is chosen such that

$$\left\|F(x_{(k+1)})\right\| < \left\|F(x_{(k)})\right\| \tag{33}$$

In practical problems this Damped Broyden's algorithm convergence, if that $\lambda_{(k)} = 1$, we obtain the original Broyden's method, and the process how to change λ is called line search procedure.

Algorithm: Damped Broyden's Method

Step 1: Choose an initial estimate x_0 $\max iteration = 25$ $toi = 0.0001$ and set k:= 0:

$\lambda = 0.1$ initial Damping factor

$\min relaxation\ facter = 1e-10$ (minimal permitted damping factor)

$convergent\ criter = 1e-4$ (max permitted damping factor)

$F(x)$ (nonlinear system of function)

$J(x)$ (Jacobian of the function)

Step 2: if i=1 up to k Compute step size Sk by starting very small number and get x_1

$$sk = sqrt(eps) * x(i)$$

$$xi = x(i) + sk$$

Compute Jacobian matrix

Step 3: compute step size $\rightarrow$ $s = -J / F1$, $x_{i+1} = x_i + lambda * s$ for s_k (derivative of Jacobian matrix & F)

While $err > convergent\ critera\ and\ k < \max iter$

Step 4: compute next iteration $\rightarrow$ $sk_2 = sqrt(eps) * x_{i+1}$, $x_{i+2} = x_{i+1} + sk_2$

Compute Jacobian matrix

Step 5: If $\max abs(lam) * s_{k+1} < \max abs(s)$

$lam = lamb / 2$ else

$pp = \lambda * s$

$x_1 = x + pp,$

Step 6: compute Broyden update $y = x_1 - x,\ s = F_1 - F_0$

$$B_{k+1} = \frac{B_k + (s_k - B_k y_k) y'}{y_k' B_k s_k} y_k' B_k$$

Step 7: Stopping criteria

Step 8: Numerical output

Numerical Output

Damped Broyden's method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

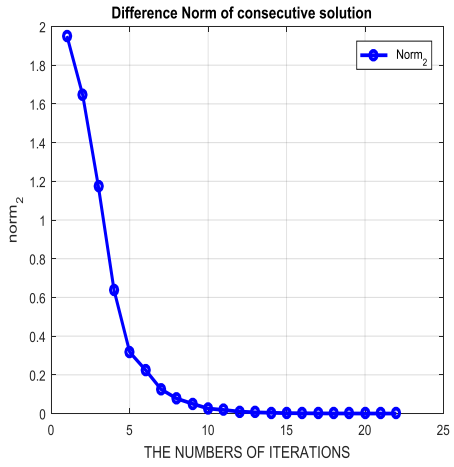
Table 7: NS Vs. NF and norm_2 of error for Damped Broyden's method.

Damped Broyden Method Norms			
K	NS	NF	Norm_2
1	0.06212660435224746900	3.37885722485194150000	1.950784
2	0.11542264062631814000	2.85202623592814190000	1.646618
3	0.19835120591793734000	2.03458863623574080000	1.174670
4	0.28636069868026998000	1.10484929988005050000	0.637885
5	0.18276889514168998000	0.54921888104419803000	0.317092
6	0.12748723909361417000	0.39113738319838903000	0.225823

7	0.06894640708160244800	0.21722908314778130000	0.125417
8	0.04306340026633177800	0.13382839102105981000	0.077266
9	0.02771076747085155600	0.08784401924739994300	0.050717
10	0.01447948351624241100	0.04467642533015685200	0.025794
11	0.01025967531514643100	0.03219487236178666300	0.018588
12	0.00554526267645081940	0.01738027847138223100	0.010035
13	0.00346226861582214470	0.01073968112271453700	0.006201
14	0.00222259642189659090	0.00702917878750619180	0.004058
15	0.00116751443309429210	0.00360129303506429020	0.002079
16	0.00082689696372455288	0.00260102405082797890	0.001502
17	0.00044375169346891699	0.00138895748020593050	0.000802
18	0.00028007333477501450	0.00086925268228148125	0.000502
19	0.00017804850147661242	0.00056293673663157442	0.000325
20	0.00009410462849888113	0.00029019530947488093	0.000168
21	0.00006663370010954759	0.00020976651637001582	0.000121
22	0.00003550255756444007	0.00011099816458086094	0.000064
Solver converged.			
CPU Time for Damped Broyden is 0.0156001			
Norm of F 1.1100e-04			

As shown in table 7, the output of a MATLAB code of Damped Broyden method, it shows the norm error of solution, step size and function are continuously decreases, it can converge for a sample problem with in 22 iteration. Moreover, we can observe that the number of iteration is longer than pure Broyden method.

Figure 9.A **Figure 9.B**



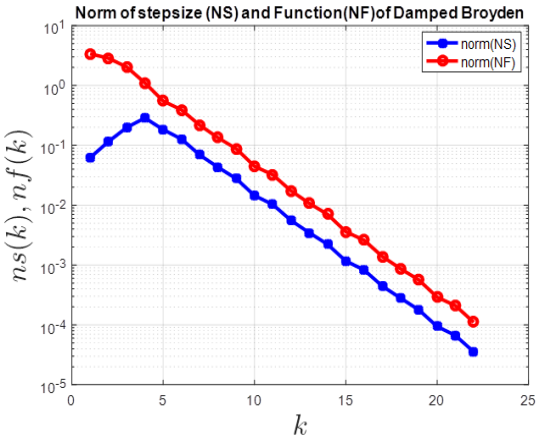


Figure 9: Damped Broyden's Method norm_2 and NS Vs NF.

As shown in figure 9A and 9B, Damped Broyden method converges to the solution within 22 iterations. Nevertheless, when we see norm error of step size and norm error of function, as well as norm of the solution graph are decreases continuously and move smoothly to the solution.

COMPARISONS OF NUMERICAL OUTPUT FOR SOLVING SYSTEM OF NONLINEAR ALGEBRAIC EQUATION

Iteration	Norm_2 error for solving system of Nonlinear Algebraic Equation		
	Pure Broyden method	Broyden with safeguarded	Damped Broyden method
1	5.656854	0.015817	1.950784
2	10.770330	0.010336	1.646618
3	21.887421	0.001820	1.174670
4	25.980083	0.000199	0.637885
5	7.039579	0.000014	0.317092
6	14.617160	0.000000	0.225823
7	2.904476		0.125417

8	0.990871		0.077266
9	0.119938		0.050717
10	0.010669		0.025794
11	0.000317		0.018588
12			0.010035
13			0.006201
14			0.004058
15			0.002079
16			0.001502
17			0.000802
18			0.000502
19			0.000325
20			0.000168
21			0.000121
22			0.000064
CPU time	426.9435368	0.0312002	0.0156001
Norm of F	1.589988716789636e-05	4	1.1100e-04

Table 8: Comparisons of Euclidian norm of Broyden, Safeguard and Damped Method

Generally, when we summarized these three methods, Broyden safeguard method have highly convergent property and take small number of iteration with small CPU time for computation. However, norm of Function is large. When we see Pure Broyden and Damped Broyden method take large number iteration than Broyden safeguarded. In order to reduce the wrong direction and to help the graph move smoothly of Broyden method, we use the Safeguarded and Damped method. However, Safeguarded method takes more Function norm of error than Pure Broyden and Damped Broyden method.

3.7.Modified Broyden's Methods for Solving SNLAE

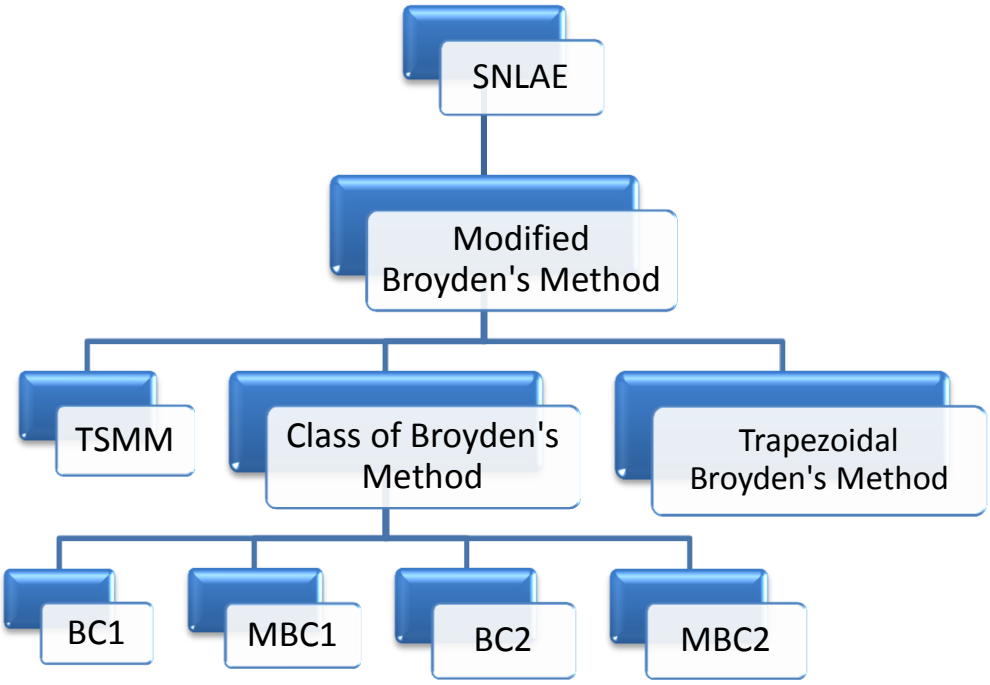


Figure 10: Schematic diagram of Quasi Broyden's method

3.7.1. TRAPEZOIDAL BROYDEN'S METHOD

In this section, we present a two point Broyden- like Trapezoidal method to solve system of nonlinear equations using the Trapezoidal rule. It used a predictor corrector approach where the classical Broyden is the predictor and the proposed method is the corrector [14]. [3] to obtain the classical Broyden's method

$$x_{K+1} = x_k - B(x_k)^{-1}F(x_k) \qquad (34)$$

$$f(x) = f(x_k) + \int_{x_k}^{x} f(t)dt \qquad (35)$$

With trapezoidal rule as

$$\int_{x}^{x_k} f'(t)dt \; \tfrac{1}{2}(x - x_k)[f'(x_k) + f'(x)] \qquad (36)$$

Now, if we replace $f(x_k), f'(x_K), by\ B(x_k), B(x_K)$ in (36) and use the same procedure and applying some algebra (35) becomes

$$x_{K+1} = x_k - 2[B(x_k) + B(x_{k+1})]^{-1} F(x_k) \quad (37)$$

In order to avoid the implicit nature of (37), we use the $(k+1)^{th}$ iteration of the Broyden's method in the right hand side. Thus

$$x_{K+1} = x_k - 2[B(x_k) + B(m_k)]^{-1} F(x_k) \quad , k = 0,1,.... \quad (38)$$

Where

$$m_k = x_k - B(x_k)^{-1} F(x_k)... \quad (39)$$

We call (38) the Trapezoidal Broydens method (TBM).

The Trapezoidal Broyden's method can be understood as a substitution of $B(x_k)$ in Broyden's method by $2[B(x_k) + B(m_k)]$.

Algorithm for TBM (Trapezoidal Broyden's Method)

STEP 1: input parameter initial guess x_0, let $k = 0$ and $B_0 = I$

STEP 2: Compute function $F(x_k)$,

STEP 3: Compute m_k were $m_k = x_k - B(x_k)^{-1} F(x_k)$

STEP 4: Compute $B(m_k)$ using

$$B(m_k) = B(x_k) - \frac{(y_k - B(x_k))s_k^T}{s_k^T s_k} s\ s$$

where $y_k = F(m_k) - F(x_k)$, $s_k = m_k - x_k$

STEP 5: Compute x_{K+1} using Trapezoidal method

$$x_{K+1} = x_k - 2[B(x_k) + B(m_k)]^{-1} F(x_k)$$

STEP 6: Compute Euclidean norms and norm of NS and NF

STEP 7: Stopping criteria

if $F(x_k) \le tol$ is satisfied stop. Else Set $k = k+1$ and go to Step 2

STEP 8: Numerical Output

Numerical output

Trapezoidal Broyden's method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 9 NS Vs NF and norm_2 error for Trapezoidal Broyden's method

Broyden-like Trapezoidal Method Norms			
	NS	NF	Norm_2 error
1	0.74535599249992990000	5.65685424949238060000	5.656854
2	0.10434772515659899000	9.57760559235930310000	9.577606
3	0.05796447486446168500	5.21617157367917450000	5.216172
4	0.02978805975845402700	2.73770208343053150000	2.737702
5	0.01487814706312486100	1.40778914070963260000	1.407789
6	0.00755185661980786560	0.72538342635716646000	0.725383
7	0.00395357947530490610	0.37992900669028562000	0.379929
8	0.00212572230669892670	0.20299076862806467000	0.202991
9	0.00116385406452617760	0.11033725825851720000	0.110337
10	0.00064446550498567371	0.06074298240708780900	0.060743
11	0.00035931310918142681	0.03373064211277639100	0.033731
12	0.00020115538897107170	0.01883543243496066400	0.018835
13	0.00011289132097714273	0.01055452252097550400	0.010555
14	0.00006344982473683699	0.00592684926411334980	0.005927
15	0.00003569303989675707	0.00333243947376049090	0.003332
16	0.00002008931798040811	0.00187511501545422480	0.001875

17	0.00001131052313863739	0.00105556499810218200	0.001056
18	0.00000636913751497457	0.00059436451185490254	0.000594
19	0.00000358695543756027	0.00033472201365295623	0.000335
20	0.00000202022233153537	0.00018851751655248302	0.000189
21	0.00000113785941249217	0.00010617917767962587	0.000106
22	0.00000064089589020608	0.00005980509779767604	0.000060
23	0.00000036098721470246	0.00003368551319134078	0.000034
24	0.00000020332898148835	0.00001897367304140568	0.000019
25	0.00000011452717160455	0.00001068713703535686	0.000011
Norm of f 1.019684589281344e-06			
CPU Time for Broyden-like trapezoidal Method is 0.0780005			

As shown in table 9, the output of the MATLAB code Trapezoidal Broyden's method, it shows the norm error of solution, step size and function are continuously decreases, it can converge for a sample problem with in 25 iteration. Moreover, we can observe that the number of iteration is longer than pure Broyden method.

Figure 11.A Figure 11.B

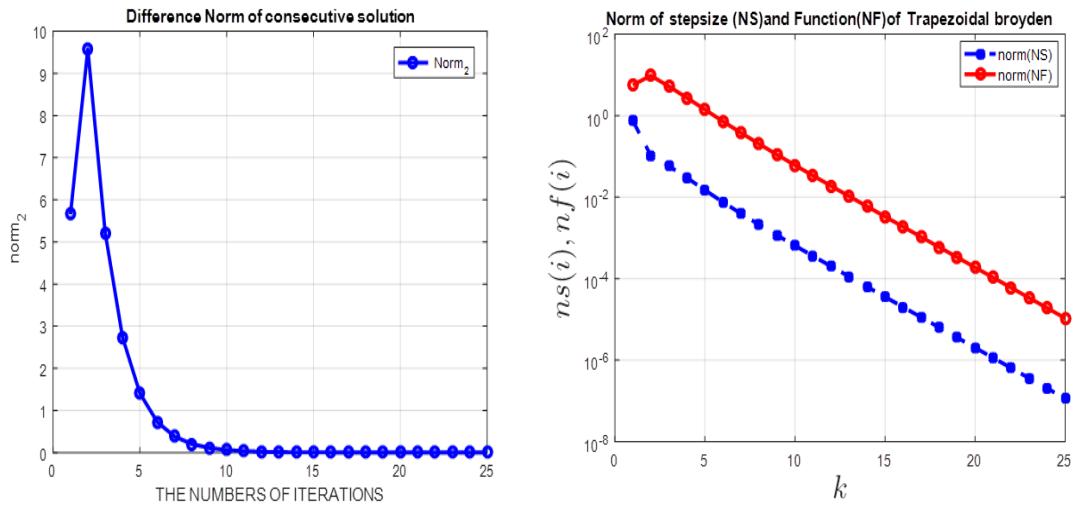


Figure 11: Trapezoidal method norms of Ns Vs NF and norm_2 Error

As shown in figure 11A and 11B, Trapezoidal Broyden method converges to the solution within 25 iterations. However, when we see norm error of step size and norm error of function, as well as norm of the solution graph are decreases continuously and move smoothly to the solution.

3.7.2. TRAPEZOIDAL, SIMPSON AND MIDPOINT METHOD (TSMM)

Broyden-like method is proposed using the weighted combination of the Trapezoidal, Simpson and Midpoint quadrature rules. [20]. Hence, a new hybrid Broyden method known as TSMM has been created based on these rules. The numerical tests confirm that TSMM is promising when subjected to comparison with other Broyden-like methods.

Broyden-like methods to solve systems of non-linear equations, using the Trapezoidal rule in order to reduce the number of iterations of the classical Broyden method.

Newton method originates from the Taylor's series expansion of the function (of a single variable) $f(x)$ about the point x_1 :

$$f(x) = f(x_1) + (x - x_1) f'(x_1) + \frac{1}{2!}(x + x_1)^2 f''(x_1) + ... \quad (40)$$

Where f , and its first and second derivatives, f' and f'' are calculated at x_1. For multiple variable function F ; from the above equation, it is obvious that

$$F(x) = F(x_k) + \int_{x_k}^{x} F'(t)dt \quad (41)$$

Approximating the integral in Equation (41) by the weighted combination of Trapezoidal, Simpson and Midpoint quadrature rules yields:

Replacing $F(x_k), F'(x_{K+1}) by\ B(x_k), B(x_{K+1})$

$$x_{K+1} = x_k - 24[5B(x_k) + 14B\left(\frac{x_k + m_k}{2}\right) + 5B(m_{k+1})]^{-1} F(x_k) \quad (42)$$

Where m_k is given as

$$m_k = x_k - B_k^{-1} F(x_k) \tag{43}$$

So we have

$$x_{K+1} = x_k - 24[5B(x_k) + 14B(Z_k) + 5B(m_k)]^{-1} F(x_k) \tag{44}$$

For

$$Z_k = \frac{x_k + m_k}{2} \tag{45}$$

Suppose we set, $B_k = [5B(x_k) + 14B(Z_k) + 5B(m_k)]$, then we have

$$x_{K+1} = x_k - 24B_k^{-1} F(x_k) \tag{46}$$

Then update by using Broyden two method.

$$B_{k+1} = B_k + (y - B_k * s) * s' / (s' * s) \tag{47}$$

Hence, with the formulation above and selecting Predictor-Corrector of Broyden method we have the following two-step iterative scheme of the Trapezoidal-Simpson-Midpoint method.

Algorithm for TSMM

Step 1: input parameter ; initial guess x_0, let $k = 0$ and $B_0 = I$,

Step 2: Compute (m_k). Inverse of Jacobain matrix $m_k = x_0 - B / F_0$

Step 3: Compute $B(m_k)$ using, Where; $y_k = F(m_k) - F(x_k)$

$s_k = m_k - x_k$

$$B(m_k) = B(x_k) + \frac{(y_k - B(x_k)s_k)s_k^T}{s_k^T s_k}$$

Step 4: Compute $B(Z_k)$ using Where; $Z_k = \frac{m_k + x_k}{2}$,

$g_k = F(z_k) - F(x_k)$

$b_k = z_k - x_k$

$B(Z_k) = B(x_k) + \frac{(g_k - B(x_k)b_k)b_k^T}{b_k^T b_k}$

$x = x_0 - 24*(5*B + 14*B_z + 5*B_m) / f(x_0)$

Step 5: Compute x_{K+1} $y = F' - F_0$ $s = x - x_0$

$B = B + (y - B*s)*s'/(s'*s)$

Step 6: Stopping criteria **:** if $F(x_k) \le 10^{-12}$ is satisfied stop, Else

Set $k = k + 1$ and go to step 2.

Step 7: Numerical Output

Numerical output

For comparison purpose, TSMM method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

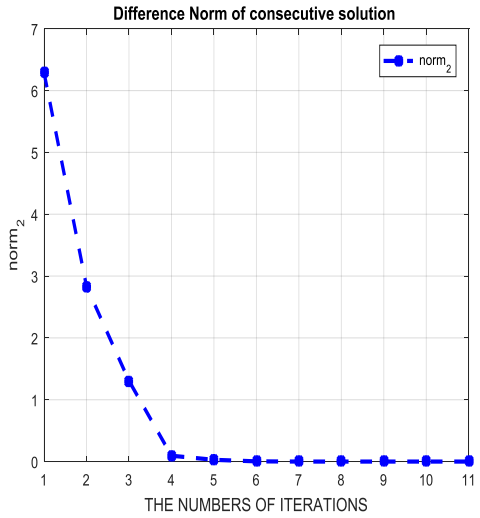
Table 10: NS Vs NF and norm_2 error for TSMM

Norms of TSMM METHOD			
k	NS	NF	Norm_2error
1	1.72453271718467230000	6.29040819551912200000	6.290408
2	1.03731261597933670000	2.83456123808169110000	2.834561
3	0.58827451248627050000	1.30553578786601390000	1.305536
4	0.29064433337367451000	0.09449204728564608600	0.094492
5	0.02308342604479965800	0.02838670030497882400	0.028387
6	0.00520416596209081780	0.00328688106936444100	0.003287

7	0.0005283996066625856 5	0.0000179257570452500 9	0.000018
8	0.0000028619636448591 1	0.0000000374783806635 0	0.000000
9	0.0000000062929759053 6	0.0000000001500374173 1	0.000000
10	0.0000000000215007072 4	0.0000000000018558800 5	0.000000
CPU Time for TSMM Method is 0.0468003			
Norm of f 1.765778221602602e-14			

As shown in table 10, the output of the MATLAB code of TSMM method, it shows the norm error of solution, step size and function are continuously decreases, it can converge for a sample problem with in 10 iteration. Moreover, we can observe that the number of iteration is smaller than Trapezoidal Broyden method.

Figure 12.A **Figure 12.B**



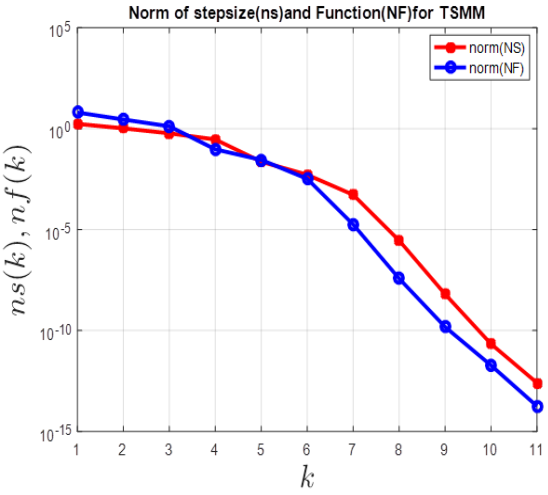


Figure 12: TSMM method of norm_2 and Ns Vs NF

As we seen in figure 12A and 12B, TSMM method converges to the solution within10 iterations. However, when we see norm error of step size and norm error of function, as well as norm of the solution graph are decreases continuously and move smoothly to the solution.

COMPARISONS OF NUMERICAL OUTPUT FOR SOLVING SNAE OF TSMM AND BROYDEN-LIKE TRAPEZOIDAL METHOD

Norm of NS Vs NF and norm_2 error for solving SNAE						
Norms of TSMM METHOD				Broyden-like Trapezoidal Method Norms		
K	NS	NF	Norm_2	NS	NF	Norm_2 error
1	1.7245327	6.2904081	6.290408	0.7453559	5.6568542	5.656854
2	1.0373126	2.8345612	2.834561	0.1043477	9.5776055	9.577606
3	0.5882745	1.3055357	1.305536	0.0579644	5.2161715	5.216172
4	0.2906443	0.0944920	0.094492	0.0297880	2.7377020	2.737702
5	0.0230834	0.0283867	0.028387	0.0148781	1.4077891	1.407789
6	0.0052041	0.0032868	0.003287	0.0075518	0.7253834	0.725383
7	0.0005283	0.0000179	0.000018	0.0039535	0.3799290	0.379929
8	0.0000028	0.0000000	0.000000	0.0021257	0.2029907	0.202991
9	0.0000000	0.0000000	0.000000	0.0011638	0.1103372	0.110337
10	0.0000000	0.0000000	0.000000	0.0006444	0.0607429	0.060743
11				0.0003593	0.0337306	0.033731
12				0.0002011	0.0188354	0.018835
13				0.0001128	0.0105545	0.010555
14				0.0000634	0.0059268	0.005927
15				0.0000356	0.0033324	0.003332
16				0.0000200	0.0018751	0.001875
17				0.0000113	0.0010555	0.001056
18				0.0000063	0.0005943	0.000594
19				0.0000035	0.0003347	0.000335
20				0.0000020	0.0001885	0.000189
21				0.0000011	0.0001061	0.000106
22				0.0000006	0.0000598	0.000060
23				0.0000003	0.0000336	0.000034
24				0.0000002	0.0000189	0.000019
25				0.0000001	0.0000106	0.000011
CPU Time for TSMM Method is 0.0468003				CPU Time for Broyden-like trapezoidal Method is 0.0780005		
Norm of f 1.765778221602602e-14				Norm of f 1.019684589281344e-06		

Table 11 Comparisons of norm_2 of Trapezoidal and TSMM Method

Generally, when we summarized Trapezoidal and TSMM Method, TSMM method has convergent property within small number of iteration and small CPU time. However, norm

of function is large. Nevertheless, Broyden-like Trapezoidal Method use a large number of iteration. In order to reduce the limitation of Trapezoidal method, we can merge Simpson, Midpoint with Trapezoidal to increase their efficiency.

3.7.3. **MODIFIED CLASSES OF BROYDEN** WITH CENTRAL FINITE DIFFERENCE

One of the problems of Broyden's method is that, it is not self-correcting property by itself. In order to avoid this we propose two efficient algorithms based on Broyden's methods by using the central finite difference and modification of Newton's method for solving systems of nonlinear algebraic equations.

In this section, we will see modify Broyden's methods to increase the accuracy and the order of convergence. These algorithms of modified Broyden's with central finite difference contain types 1 and 2.

The first type is an approximation matrix for $F'(x_k)$ (i.e. $B_k \cong F'_k$) and second method approximate the inverse of Jacobian matrix (i.e. $H_k \approx F_k^{'-1}$) [19]

Broyden's classes with central finite difference (BC)

Broyden can use central finite difference to approximate the Jacobian matrix

$$\frac{\partial f_j}{\partial x_k}\left(x^{(i)}\right) \cong \frac{f_j(x^{(i)} + e_k h) - f_j\left(x^{(i)}\right)}{h} \tag{48}$$

3.7.3.1. Broyden's classes 1 (BC1)

In the first type of Broyden's we will use central finite difference to approximate Jacobian matrix (i.e. $B_k \cong F'_k$), to improve the order of convergence and to avoid computing the Jacobian matrix.

By using Taylor's expansion and a similar explanation in[19]

$$F(x+h) = F(x) + F'(x)h + \frac{1}{2!}F''(\xi_1)h^2, \tag{49}$$

$$F(x-h) = F(x) - F'(x)h + \frac{1}{2!}F''(\xi_2)h^2. \tag{50}$$

Subtract (50) from (49), and by the mean value theorem of vector-valued function, we get

$$F(x+h)-F(x-h)=F'(x)2h+O(\|h\|)^2. \quad (51)$$

Equation (51) indicates that the secant equation of Broyden's method replaced by

$$F(x+h)-F(x-h)=B_k 2h, \quad (52)$$

From this, two assumptions Broyden's set on B_k:

I. Central secant equation $F(x+h)-F(x-h)=B_k 2h,$

II. No change condition $B_k q = B_{k-1} q, where \quad h^t q = 0.$

Now we need to determine $x,\ x-h$ and $x+h$.

Let $x = x^{(k)}$ and $2h = x^{(k)} - x^{(k-2)} = s^{(k)}$ then

$$x+h=\frac{3}{2}x^{(k)}-\frac{1}{2}x^{(k-2)} \ and \quad x-h=\frac{1}{2}(x^{(k)}-x^{(k-2)})$$

$$y^{(k)}=F\left(\frac{3}{2}x^{(k)}-\frac{1}{2}x^{(k-2)}\right)-F\left(\frac{1}{2}(x^{(k)}-x^{(k-2)})\right)$$

By (i) and (ii) , we get

$$B_k = B_{k-1}+\frac{(y^{(k)}-B_{k-1}s^{(k)})(s^{(k)})^t}{\left\|s^{(k)}\right\|^2},\ k=2,...,m \quad (53)$$

In (53) B_0 and B_1 are not defined, we can define both as Newton's method, like: $B_0 = F_0'$ and $B_1 = F_1'$. We can use Sherman-Morrison formula to make the above updating depend on the inverse of B_k,

$$B_{k+1}^{-1} = B_k^{-1} + \frac{\left(s_{k+1} - B_k^{-1} y_{k+1}\right) s_{k+1}^T B_i^{-1}}{s_{k+1}^T\ B_i^{-1} y_{k+1}} \quad (54)$$

Algorithm of BC1

Step 1. Input parameter

Step 2. Solve $x^{(1)} = x^{(0)} - J^{-1}(x^{(0)})F(x^{(0)})$

$x^{(2)} = x^{(1)} - J^{-1}(x^{(1)})F(x^{(1)})$

Step 3. Let $\hat{B}_0 = J(x^{(0)}),\ \hat{B}_1 = J(x^{(1)})$

Step 4. for k = 2 $s^{(k)} = x^{(k)} - x^{(k-2)}$

Step 5. $$y^{(k)} = F\left(\frac{3}{2}x^{(k)} - \frac{1}{2}x^{(k-2)}\right) - F\left(\frac{1}{2}(x^{(k)} - x^{(k-2)})\right)$$

Step 6. $$\hat{B}_k = \hat{B}_{k-1}^{-1} + \frac{(s^{(K)} - \hat{B}_{k-1}^{-1}y^{(k)})s^{(k)}\hat{B}_{k-1}^{-1}}{(s^{(k)})^t \hat{B}_{k-1}^{-1}y^{(k)}}$$

Step 7. $X^{(k+1)} = x^{(k)} - \hat{B}_k^{-1}F(x^{(k)})$

Step 8. If $\left\|x^{(k+1)} - x^{(k)}\right\|_2 \le Tol, Stop\ output\ x^{(k+1)}$

Otherwise $k = k+1$ Continue

Step 9: Numerical output

Numerical output

BC1 method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 12: Solution of X and Norm of NS Vs NF for BC1

BC1 Method					
k	x(1)	x(2)	x(3)	NS	NF
1	0.95719300	0.87247856	0.29367079	0.07864742	0.39662803
2	0.95831442	0.87343284	0.29334547	0.00150800	0.00588908
CPU Time BC1 Method = 0.0312002					
Norm of F 0.009321378802078					

As shown in table 12, the output of the above MATLAB code BC1 method, it shows the solution X, norm of step size and norm of function are continuously decreases, it can converge for a sample problem with in 2 iteration. Moreover, we can observe that the number of iteration is very small, this increase their accuracy and the order of convergence than other Broyden method.

Figure 13.A

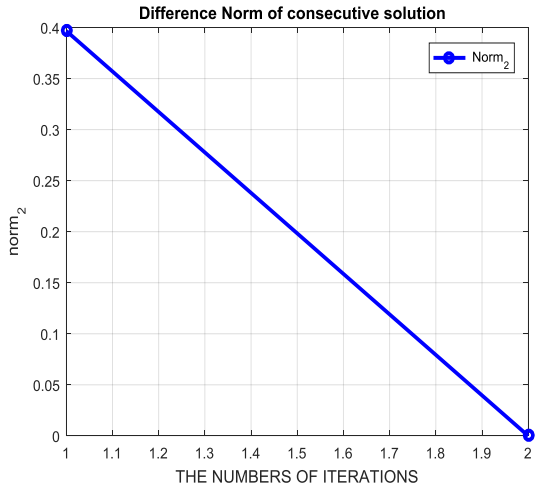


Figure 13.B

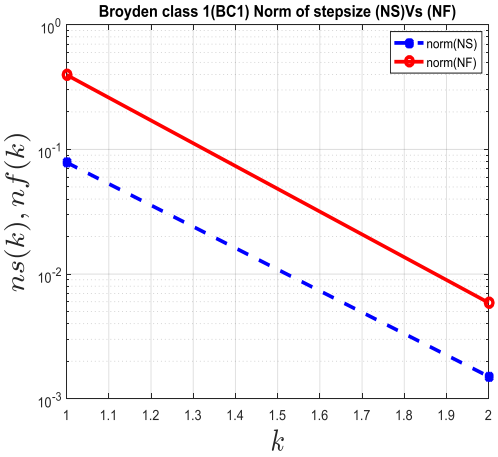


Figure 13: BC1 method norm_2 Error and norm of Ns Vs NF

As we seen in figure 13.A and 13.B, BC1 method converges to the solution within 2 iterations. In addition to that when we see norm error of step size and norm error of function are decreases continuously and move smoothly to the solution.

3.7.3.2. Modified Broyden's classes 1 (MBC1)

In this section, we use Newton's modification method (predictor-corrector), in the modified classes of Broyden's types one and two.[19] We follow the same procedures in [1,2] by using the following iteration formulas to obtain the approximate solution $x^{(K+1)}$.

$$\begin{aligned} \hat{X}^{(K+1)} &= x^{(k)} - \hat{B}_k^{-1} F(x^{(k)}), \\ X^{(k+1)} &= \hat{X}^{(K+1)} - \hat{B}_k^{-1} F(\hat{x}^{(k+1)}), \end{aligned} \tag{55}$$

Where $\hat{X}^{(K+1)}$ is the predictor of $X^{(k+1)}$

Set those equations on the algorithm of BC1, to obtain a new algorithm for MBC1 describes the algorithm of this modification.

Algorithm: Modified Broyden's Classes1 (MBC1)

Input parameter

Step1. Solve $x^{(1)} = x^{(0)} - J^{-1}(x^{(0)})F(x^{(0)})$

$x^{(2)} = x^{(1)} - J^{-1}(x^{(1)})F(x^{(1)})$

Step2. Let $\hat{B}_0 = J(x^{(0)}),$

Step3. for k = 2 $s^{(k)} = x^{(k)} - x^{(k-2)}$

$$y^{(k)} = F\left(\frac{3}{2}x^{(k)} - \frac{1}{2}x^{(k-2)}\right) - F\left(\frac{1}{2}(x^{(k)} - x^{(k-2)})\right)$$

$$B_k^{-1} = B_{k-1}^{-1} + \frac{(s^{(K)} - B_{k-1}^{-1}y^{(k)})s^{(k)}B_{k-1}^{-1}}{(s^{(k)})^t B_{k-1}^{-1}y^{(k)}}$$

Step4. Predictor $\hat{X}^{(K+1)} = x^{(k)} - \hat{B}_k^{-1} F(x^{(k)})$

Corrector $X^{(k+1)} = \hat{X}^{(K+1)} - \hat{B}_k^{-1} F(\hat{x}^{(k+1)})$

Step 5. If $\left\|x^{(k+1)} - x^{(k)}\right\|_2 \le Tol, Stop\, output\, x^{(k+1)}$

Numerical Output

For comparison purpose, MBC1method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 13: Solution of X and Norm of NS Vs NF for MBC1

MBC1 Method					
K	x(1)	x(2)	x(3)	NS	NF
1	0.95719300	0.87247856	0.29367079	0.07864742	0.39662803
2	0.95863463	0.87375316	0.29354387	0.00049448	0.00932138
CPU Time MBC1 Method = 0.0312002					
Norm of F 0.011712944354230					

As shown in table 13, the output of the MATLAB code MBC1 method, it shows the solution X, norm of step size and norm of function are continuously decreases, it can converge for a sample problem with in 2 iteration. Moreover, we can observe that the number of iteration is very small, increase their accuracy and the order of convergence than other modified Broyden method.

Figure 14.A

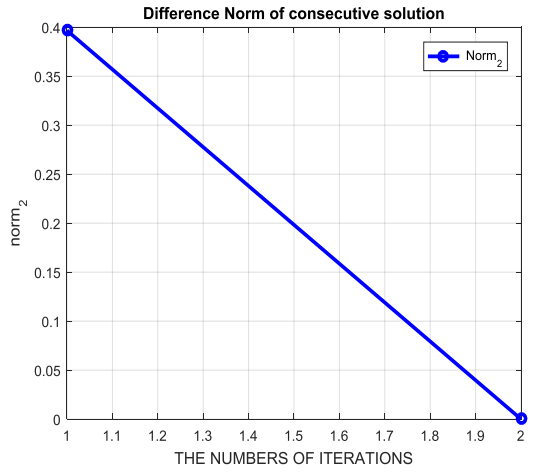


Figure 14.B

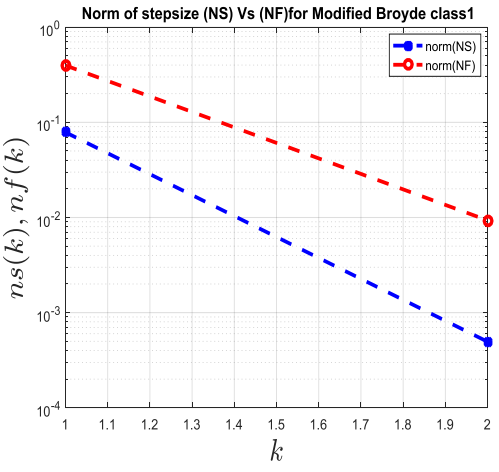


Figure 14: MBC1 method norm_2 Error and norms of Ns Vs NF

As we seen in figure 14.A and 14.B, MBC1method converges to the solution within2 iterations. In addition to that when we see norm error of step size and norm error of function graph are decreases continuously and move smoothly to the solution.

3.7.3.3. Broyden's Classes 2(BC2)

For this type, we use the central finite difference to approximate F_K^{-1}(*i.e*, $H_K \cong F_K^{-1}$) .One can easily show that the conditions (i) and (ii) in BC1 method, become [64]:

(i) Central secant equation $H(F(x+h)-f(x-h))=2h.$

(ii) No change condition $H_k q = H_{k-1} q$, where q $q^t y^{(k)} = 0$

Now, we use the same procedures in BC1 to determine *x, x − h and x + h* for BC2.

By (I) and (I) we obtain

$$H_k = H_{k-1} + \frac{\left(s_k - H_{k-1}s^{(k)}\right)}{\left\|y^{(k)}\right\|^2}\left(y^{(k)}\right)^t \qquad (56)$$

we can use H_0 and H_1 as Newton's method, $H_0 = F_0'^{-1}$ and $H_1 = F_1'^{-1}$

Algorithm: Broyden's Classes2 (BC2)

Step 1: Input parameter

Step2: Solve $x^{(1)} = x^{(0)} - J^{-1}(x^{(0)})F(x^{(0)})$

$x^{(2)} = x^{(1)} - J^{-1}(x^{(1)})F(x^{(1)})$

Step3: For k=2 $\hat{H}_0 = J(x^{(0)})^{-1}$ $\hat{H}_1 = J(x^{(1)})^{-1}$

Step4: $s^{(k)} = x^{(k)} - x^{(k-2)}$

Step5: $y^{(k)} = F\left(\frac{3}{2}x^{(k)} - \frac{1}{2}x^{(k-2)}\right) - F\left(\frac{1}{2}(x^{(k)} - x^{(k-2)})\right)$

Step6: $\hat{H}_k = \hat{H}_{k-1} + \frac{(s^{(k)} - \hat{H}_{k-1}s^{(k)})}{\|y^{(k)}\|^2}(y^{(k)})^t$

Step7: $x^{(k+1)} = x^{(k)} - \hat{H}_k F(x^{(k)})$

Step 8: If $\left\|x^{(k+1)} - x^{(k)}\right\|_2 \le Tol, Stop\, output\, x^{(k+1)}$

Otherwise $k = k+1$ Continue

Step 9: Numerical Output

Numerical Output

For comparison purpose, BC2method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 14: Solution of X and Norm of ns Vs nf for BC2

BC2 Method					
K	x(1)	x(2)	x(3)	NS	NF
1	0.95719300	0.87247856	0.29367079	0.07864742	0.39662803
2	0.95711349	0.87227500	0.29294075	0.00076205	0.00588908
CPU Time BC2 Method = 0.0468003					
Norm of F 0.002129441196115					

As shown in table 14, the output of the MATLAB code BC2 method, it shows the solution X, norm of step size and function are continuously decreases, it can converge for a sample problem with in 2 iteration. Moreover, we can observe that the number of iteration is very small, increase their accuracy and the order of convergence than other modified Broyden method.

Figure 15.A

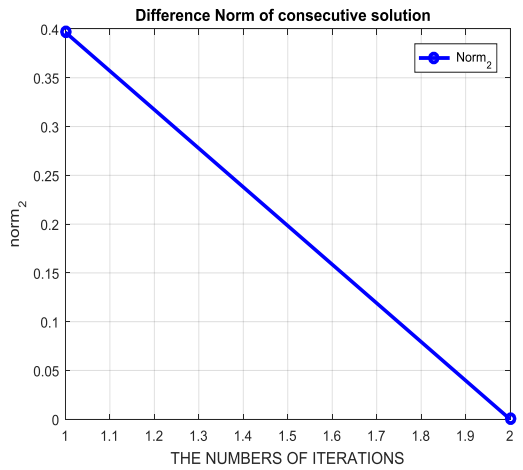


Figure 15.B

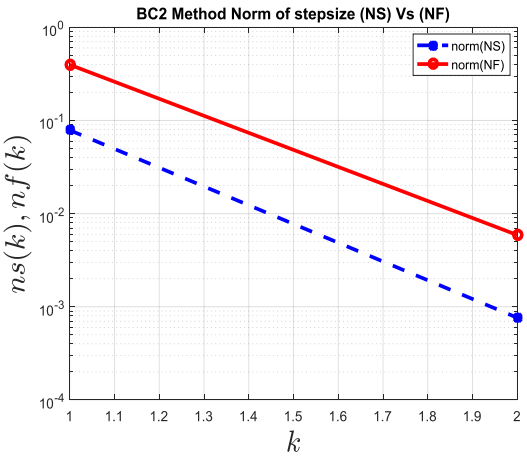


Figure 15: BC2 method norm_2 Error and norms of Ns Vs NF

As we seen in figure 15A and 15B, BC2 method converges to the solution within 2 iterations. In addition to that when we see norm error of step size and norm error of function decreases continuously and move smoothly to the solution.

3.7.3.4. Modified Broyden's class2 (MBC2)

We follow the same procedures as above by replacing $\hat{B}_k = \hat{H}_k$ and we obtain the following modification of type two of BC class [19]:

$$\begin{aligned} \hat{X}^{(K+1)} &= x^{(k)} - \hat{H}_k^{-1} F(x^{(k)}), \\ X^{(k+1)} &= \hat{X}^{(K+1)} - \hat{H}_k^{-1} F(\hat{x}^{(k+1)}). \end{aligned} \tag{57}$$

Algorithm: Modified Broyden's Classes 2 (MBC2)

Input parameter

Step1. Solve $x^{(1)} = x^{(0)} - J^{-1}(x^{(0)})F(x^{(0)})$

$x^{(2)} = x^{(1)} - J^{-1}(x^{(1)})F(x^{(1)})$

Step2. Let $\hat{H}_0 = J(x^{(0)})^{-1}$ $\hat{H}_1 = J(x^{(1)})^{-1}$

Step3. for k = 2 $s^{(k)} = x^{(k)} - x^{(k-2)}$

$$y^{(k)} = F\left(\frac{3}{2}x^{(k)} - \frac{1}{2}x^{(k-2)}\right) - F\left(\frac{1}{2}(x^{(k)} - x^{(k-2)})\right)$$

$$\hat{H}_k = \hat{H}_{k-1} + \frac{(s^{(K)} - H_{k-1}s^{(k)})}{\left\|y^{(k)}\right\|^2}(y^{(k)})^t$$

Step4. predictor $\hat{X}^{(K+1)} = x^{(k)} - \hat{H}_k F(x^{(k)})$

corrector $X^{(k+1)} = \hat{X}^{(K+1)} - \hat{H}_k F(\hat{x}^{(k+1)})$

Step 5. If $\left\|x^{(k+1)} - x^{(k)}\right\|_2 \le Tol, Stop\, output\, x^{(k+1)}$

Otherwise $k = k+1$ Continue

Step 6: Numerical Output

Numerical Output

For comparison purpose, MBC2 method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 15: Solution of X and norm of ns Vs nf for MBC2

MBC2 Method					
K	x(1)	x(2)	x(3)	NS	NF
1	0.95719300	0.87247856	0.29367079	0.07864742	0.39662803
2	0.95710251	0.87223254	0.29264728	0.00029673	0.00212944
CPU Time MBC2 Method = 0.0312002					
Norm of F 6.973505736540425e-04					

As shown in table 15, the output of the above MATLAB code MBC2 method, it shows the solution of X, norm of step size and function are continuously decreases, it can converge for a sample problem with in 2 iteration. Moreover, we can observe that the number of iteration is very small, increase their accuracy and the order of convergence than other modified Broyden method.

Figure 16.A

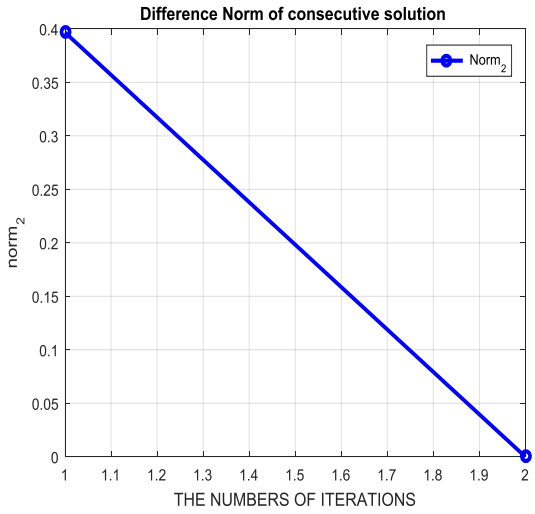


Figure 16.B

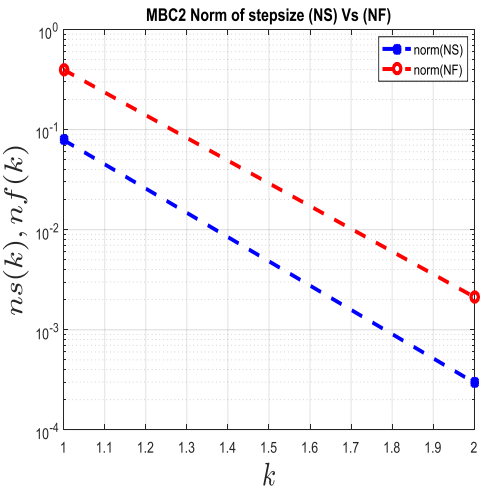


Figure 16: MBC2 method norm_2 Error and norms of Ns Vs NF and

As we seen in figure 16.A and 16.B, MBC2 method converges to the solution within 2 iterations. In addition to that when we see norm error of step size and norm error of function are decreases continuously and move smoothly to the solution.

COMPARISONS OF NUMERICAL OUTPUT FOR SOLVING SNAE OF MODIFIED BROYDEN CLASS METHODS

Table 16: Norms of Ns Vs NF of BC1, MBC1, BC2 and MBC2.

K	**BC1**		**MBC1**		**BC2**		**MBC2**	
	NS	**NF**	**NS**	**NF**	**NS**	**NF**	**NS**	**NF**
1	0.078674	0.396628	0.0786474	0.3966280	0.0786474	0.3966280	0.0786474	0.39662803
2	0.001508	0.005889	0.0004944	0.0093213	0.0007620	0.0058890	0.0002967	0.00212944
CPU	0.0312002		0.0312002		0.0468003		0.0312002	
Norm of F	0.009321378802078		0.011712944354230		0.002129441196115		6.973505736540425e-04	

CHAPTER FOUR

4. NEWTON HOMOTOPY ANALYSIS METHOD FOR SOLVING SNLAE

4.1. Homotopy Analysis Method

As is well-known, a disadvantage of the Newton method is that the initial approximation x_0, must be chosen sufficiently close to a true solution in order to guarantee their convergent. Finding a criterion for choosing x_0 is quite difficult and therefore effective and globally convergent algorithm is needed [26].

Liao [53] employed the basic ideas of Homotopy to propose a general method for nonlinear problems, namely the Homotopy analysis method (HAM) to get effective initial point.

The nonlinear system $f(x)=0$ is solved by a Homotopy method, in which a Homotopy function $H(x,\lambda)=f(x)-(1-\lambda)f(x_0)$ is introduced and the solution path of $H(x,\lambda)=0$ is followed from an obvious solution $(x_0,0)$ to the solution $(x^*,1)$ which we seek. An ordinary differential equation based on Newton Homotopy is used for following the solution path.

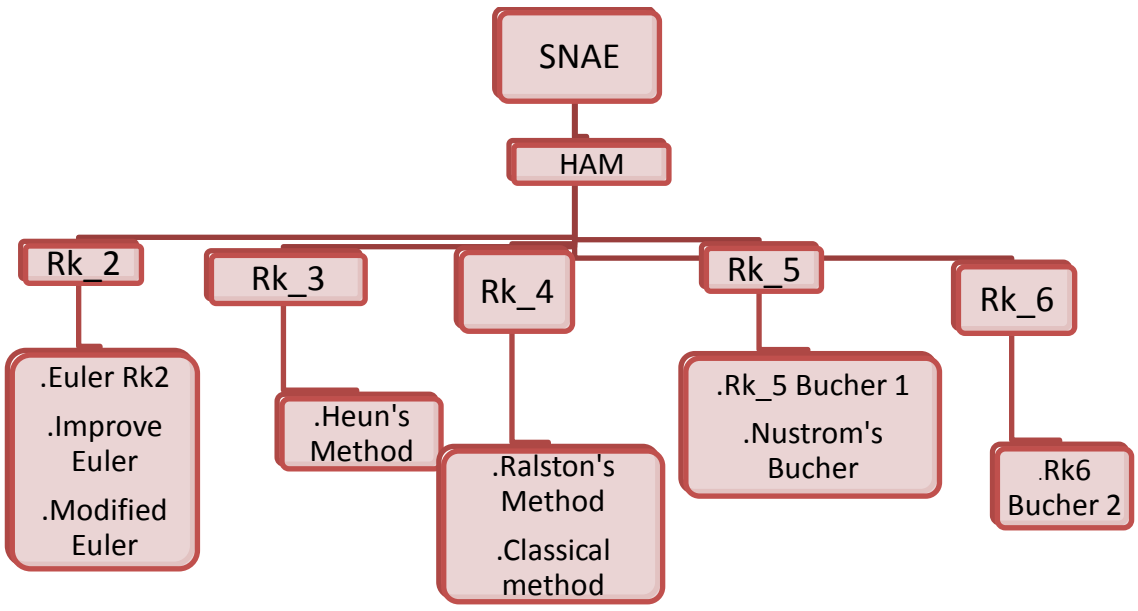


Figure 17: Schematic diagram of Homotopy analysis method

We introduce a parameter $\lambda \in [0,1]$ and consider an equation

$$H(x,\lambda) = \lambda f(x) + (1-\lambda) f(x_0) \qquad (58)$$

Where $x \in R^n, F; R^n \to R^n$ (Note that the fixed point problem can be easily reformulated as a zero finding problem.).We create the Homotopy function $H(x,\lambda)$ by embedding a parameter λ _ into $F(x)$ and thus obtaining an equation of higher dimension. The parameter λ is called the Homotopy parameter**.** .

4.1.1. Description of Homotopy Analysis Method

The Homotopy method for the system of nonlinear equation has the following function [31]

$H : [0,1] \times \Re^n \to \Re^n$

Is defined by

$$H(\lambda, x) = \lambda F(x) + (1-\lambda)(F(x) - F(x_0)) \qquad (59)$$

Here x^0 is an initial guess of $\varnothing(1) = x^*$ and λ is Homotopy parameters. Obviously at $[\lambda = 0]$ and $[\lambda = 1]$.

$$H(0,x) = F(x) - F(x^{(0)}), \quad H(1,x) = F(x) \qquad (60)$$

The function H is a Homotopy between $H(0,x) = F(x) - F(x^{(0)})$ and $H(1,x) = F(x)$. Accepting that $\varnothing : [0,1] \to \square^n, x = \varnothing(\lambda)$ is unique solution of the equation. $H(\lambda, x) = 0, \qquad \lambda \in [0,1]$.

The set $\{\varnothing(\lambda) / 0 \le \lambda \le 1\}$ can be viewed as a family of parameterized curves respect to λ in $\Re^n$ from $\varnothing(0)$ to $\varnothing(1) = x^*$ .The solution x^* of $F(x) = 0$ can be obtained by solving the following system of equations.

$$\varnothing'(\lambda) = -\left[J(\varnothing(\lambda))\right]^{-1} F(\varnothing(0)) \qquad 0 \le \lambda \le 1, \qquad (61)$$

With initial condition $\varnothing(0) = x^0$, where $J(\varnothing(\lambda))$ is Jacobian matrix of H with respect to x. This method will be referred to Newton Homotopy Analysis Method.

The main advantage of Homotopy analysis method is the relative freedom of choosing initial guess. However, the method spends a lot of time during each iteration to convergent. Therefore, for accelerating the convergent of this method, we suggest the combination of HAM with other method. HAM is a globally convergent method to solve nonlinear systems of algebraic or differential equation.

4.1.2. The Newton Homotopy Differential Equations (NHDE)

To derive the Homotopy differential equation, we can follow the following mathematical steps:

$$H(x,\lambda) = f(x) - (1-\lambda) f(x_0)$$
$$H(x,\lambda) = 0$$

By chain rule, we obtained

$$\frac{\partial H}{\partial x}\frac{dx}{d\lambda} + \frac{\partial H}{\partial \lambda} = 0 \qquad (62)$$

$$\frac{dx}{d\lambda} = \frac{-\frac{\partial H}{\partial \lambda}}{\frac{\partial H}{\partial x}} = \frac{-H_\lambda}{H_x} = f(x,\lambda)$$

This equation is a first order differential equation named as *Newton Homotopy Differential Equation* (NHDE). This equation can be solved numerically using Euler's or any Runge Kutta method to find x for parameter $\lambda \in [0,1]$,The numerical solution can be formulated as follow:

$x_k = x_{(k-1)} + hf(x,\lambda); \quad h = r\lambda$ h-is Euler step size

$$x_k = x_{(k-1)} + h(\frac{-H_\lambda}{H_x}); k = 0,1,2...n \qquad (63)$$

4.1.3. Homotopy Analysis Method with Runge-Kutta Steps

Runge-Kutta methods are a family of single-step, explicit, numerical techniques for solving a first-order ordinary differential equation. Various types of Runge-Kutta methods are classified according to their order.

The order identifies the number of points within the subinterval that are utilized for finding the value of the slope. For instance, second-order Runge-Kutta methods use the slope at two points; third-order methods use three-points, and so on. The classical Runge-Kutta method is of order four and uses four points.

Runge-Kutta methods give a more accurate solution compared to the simpler Euler's explicit method. The accuracy increases with increasing their order of Runge-Kutta method.

A general first order ordinary differential equation (ODE) system under the initial condition $y_0 = y(x_0)$ can be given as $y'(x) = f(x, y(x))$.

In the next section we are try to apply different RK methods to solve equation (62), NHDE.

In summary, the main steps of the algorithm of NHAM are as follows:

1. Write the given system in the form $f(x) = 0$.
2. Write the Newton homotopy $H(\lambda, x) = \lambda F(x) + (1-\lambda)(F(x) - F(x_0))$ for the given system.
3. Write NHDE
4. Choose *h*.
5. Solve NHDE using Euler's formula or other RK methods
6. Put the results in a table.

4.2. Homotopy Analyses Method with Forward Euler Steps

The initial condition provides us with a point on the true solution, so (x_0, y_0) is also the natural starting point for the approximation. To obtain an approximation to the solution at x_1 we compute the slope of the tangent at (x_0, y_0) as $y' = f(x_0, y_0)$. This gives us the tangent $T_0(x_0) = y_0 + (x - x_0)y_0'$ to the solution at x_0. As the approximation y_1 at x_1 we use the value of the tangent to which is given by,

$$y_1 = T_0(x_1) = y_0 + hy_0' = y_0 + hf(x_0, y_0) \qquad (64)$$

This gives us the next approximate solution point (x_1, y_1). To advance to the next point (x_2, y_2), we move along the tangent to the exact solution that passes through (x_1, y_1).

The derivative at this point is $y_1^{'} = f(x_1, y_1)$ and so the tangent is

$$T_0(x) = y_1 + (x - x_1)y_1^{'} = y_1 + (x - x_1), f(x_1, y_1) \quad (65)$$

The approximate solution at x_2 is therefore

$$y_2^{'} = y_1 + hf(x_1, y_1), \; x_k = x_{(k-1)} + h(\frac{-H_\lambda}{H_x}) \quad (66)$$

If we continue in the same way, we can compute an approximation y_2 to the solution at x_3, then an approximation y_4 at x_4, and so on.

Algorithm of NHAM with Forward Euler Step

Step 1 : Input parameter

U - initial guess of problem of problem

Maxiter - maximum number of iteration

dl - Change of Homotopy parameter

N - number of step in Euler equation

$h_i = 1/N$ - Euler step size , h is grade spacing

$x = zeros(k+1, n)$ - initialization

J - Jacobian matrix

X - Solution vector

Step 2: compute Jacobian matrix $J = jacbian(f)$

Step 3: compute step size of *dl*

$$dl = -J / f(x_0)$$

Step 4: Compute first slope and update solution

$$k = U + h * dl$$

Step 5: Display output

4.2.1. Numerical output of NHAM with Forward Euler (FE) Step

For comparison purpose, NHAM with FE step is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 17: Number of iteration for NHAM with Forward Euler Step

NHAM with FE Steps			
I	X(1)	X(2)	X(3)
1	2.000000000000000e+00	2.000000000000000e+00	2.000000000000000e+0
2	1.023315416034467e+00	9.496539948977500e-01	3.977973256556855e-01
3	9.574120567551337e-01	8.726000715905279e-01	2.929346098764011e-01
4	9.570972186197976e-01	8.722119677940901e-01	2.925043316636623e-01
5	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
6	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
7	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
8	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
9	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
CPU Time for NHAM with FE Steps =12.2616786			
Elapsed time is 9.692701 seconds.			

As shown in table 17, the output of the MATLAB code of NHAM with FE step, NHAM_FE can converge for a sample problem with in 9 iteration. Moreover, we can observe that it take large CPU time to reach the solution as compared to other numerical methods in this paper. However, the method can converge to a solution even the initial guess far from exact solution.

Figure18.A **Figure18.B**

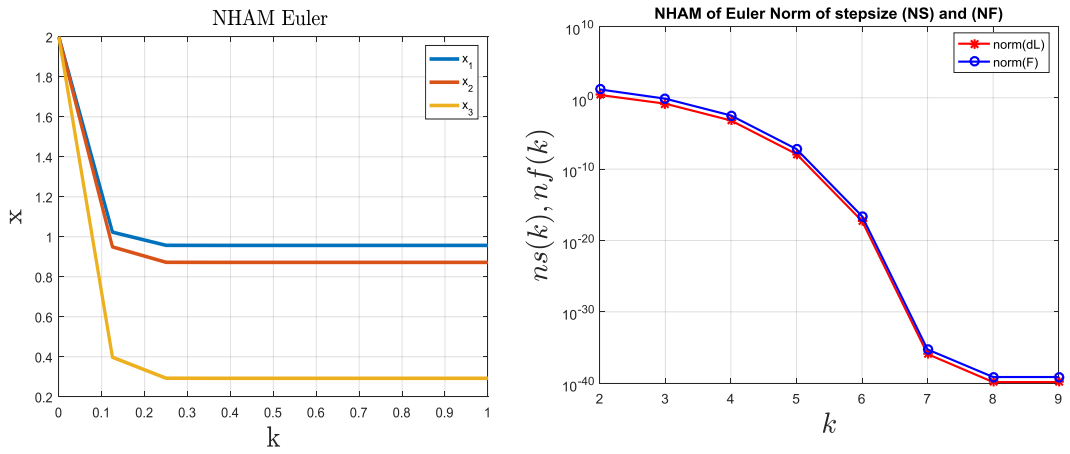


Figure18:NHAM_FE solution and norms of step size Vs norm of function

As shown in figure 18A, the output of solution(x) versus each Homotopy parameters. This shows as the value of parameter increases from 0 to 1, the accuracy of the solution increases and converge to a better accurate solution when value of parameter is 1. Moreover as shown in Figure 18B, which proves the convergence of NHAM_FE for sample problem of equation (13) and the norm errors of both the function and steps(NHDE) are getting smaller in between successive iterates and finally its norm errors becomes very small.

4.3. Homotopy Analyses Method with RK2 Step

A general first order ordinary differential equation (ODE) system under the initial condition $y_0 = y(x_0)$ can be given by, $y'(x) = f(x, y(x))$, this NHDE can be solve by using a classical second order Runge_Kutta method as [54] .

$$\left.\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + h, y_i + hk_i) \end{aligned}\right\} \rightarrow y_{i+1} = y_i + \frac{h}{2}(k_1 + k_2) \qquad (67)$$

In general, equation(63) can be solve by using RK4 or RK5 or RK6 methods that have been developed with similar steps of RK2 by only change Rk2 step by Runge kutta 4 or 5 or 6 steps. This alternative of a Runge-Kutta methods of order four (RK4), RK5 or RK6 that shows the number of linear systems that needed to be solved. The algorithm of these methods will be display in the next sections.

Algorithm of NHAM with Improved Euler Steps

Step 1 : Input parameter

U - initial guess of problem of problem

Maxiter - maximum number of iteration

dl - Change of Homotopy parameter

N - number of step in EulerRk_2 equation

$h_i = 1/N$ - EulerRk_2 step size , h is grade spacing

$x = zeros(k+1, n)$ - initialization

J - Jacobian matrix

X - Solution vector

Step 2: compute Jacobian matrix $J = jacbian(f)$

Step 3: compute step size of dl

$$dl = -J / f(x_0)$$

Step 4: Compute first slope and update solution

$$k_1 = U + h * dl$$

Step 5: Compute dl_2 *and* k_2 in iteration two

Step 6: Compute improve Euler Rk2 slope

$$u_2 = u + k_1 / 2 + k_2 / 2$$

Step 7: Numerical output

4.3.1. Numerical output with Improved Euler Steps

For comparison purpose, NHAM with Improved Euler steps is applied and tested on equation (13) and the numerical result and norm errors are displayed by using the MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table18: Number of iteration for NHAM with Improved Euler

NHAM with Improved Euler Step			
I	X(1)	X(2)	X(3)
1	2.000000000000000e+00	2.000000000000000e+00	2.000000000000000e+00
2	9.573731438721318e-01	8.725519431240861e-01	2.928796260550977e-01
3	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
4	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
5	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
6	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
7	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
8	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
9	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
CPU Time for NHAM with Improved Euler Step =15.4440990			
Elapsed time is 13.076105 seconds.			

As shown in Table 18, the output of the above MATLAB programing code of NHAM with RK2 step converges for sample problem(13) and terminates at the ninth iteration. The CPU-time indicates NHAM with RK2 takes more time than NHAM with Forward Euler.

Figure 19.A

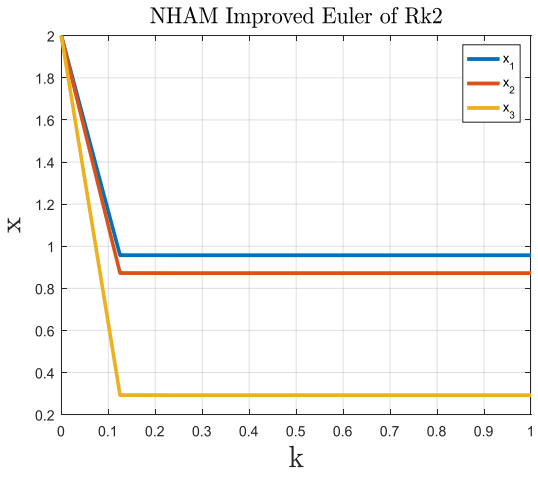


Figure 19.B

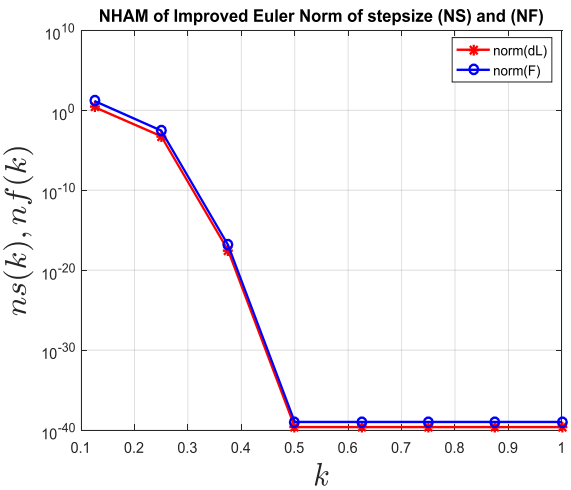


Figure 19: NHAM with Improve Euler solution and norms of step size Vs norm function

As shown in figure 19.A, the output or solution(x) versus each Homotopy parameters. This shows as the value of parameter increases from 0 to 1, the accuracy of the solution increases and converge to a better accurate solution when value of parameter is 1. Moreover as shown in Figure 19.B, which proves the convergence of NHAM_RK2 for sample problem of equation (13) and the norm errors of both the function and steps(NHDE) are getting smaller in between successive iterates. The Euclidean norm error of both Newton step and the function are lesser than NHAM with Forward Euler.

4.4.NHAM with Third order Runge-Kutta (RK3- NHAM) Step

Algorithm for NHAM with RK_3 Steps

Step 1 : Input parameter

U - Initial guess of problem of problem

Maxiter - Maximum number of iteration

dl - Change of Homotopy parameter

N - Number of step in Rk_3 equation

$h_i = 1/N$ - Rk_3 step size, h is grade spacing

$x = zeros(k+1, n)$ - Initialization

J - Jacobian matrix

X - Solution vector

Step 2: compute Jacobian matrix $J = jacbian(f)$

Step 3: compute step size of dl

$$dl = -J / f(x_0)$$

Step 4: Compute first slope and update solution

$$dl = -J \setminus f0$$

$$k_1 = h * dl$$

$$p_2 = u + \frac{1}{3} * k_1$$

$$dl1 = -J1 \setminus f0$$

Step 5 : Compute slope 2 $k_2 = h * dl1$

$$p_3 = u + \frac{2}{3} k_2$$

Step 6: Compute Heun's Rk3 step by $u = u + (p_1 + 3p_3)/4$

Step 8: Numerical output

4.4.1. Numerical output with classical RK_3 Steps

For comparison purpose, NHAM with Heun's RK_3 method is applied and tested on equation (13) and the numerical result and norm error are displayed by using a MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table19: Numerical solution problem(13) by NHAM with RK3 Steps

NHAM for Heun's Rk-3			
I	X(1)	X(2)	X(3)
1	2.000000000000000e+00	2.000000000000000e+00	2.000000000000000e+00
2	1.305316920978217e+00	1.256083459013939e+00	8.595417179644160e-01
3	1.055833867304566e+00	9.820358492341759e-01	4.522762423000100e-01
4	9.827894285143761e-01	9.008707198741364e-01	3.339577921631445e-01
5	9.635899919033010e-01	8.794599651977889e-01	3.029709597890476e-01
6	9.587248848103606e-01	8.740293133993387e-01	2.951275939185758e-01
7	9.575044116126927e-01	8.726666344454302e-01	2.931605581911890e-01
8	9.571990291166034e-01	8.723256488838026e-01	2.926684096490646e-01
9	9.571226669500028e-01	8.722403827178131e-01	2.925453481240047e-01
CPU Time for NHAM of Heun's problem =18.3613177			
Elapsed time is 16.153025 seconds.			

As shown in Table 19, the output of the MATLAB programing code of NHAM with RK_3 step converges for sample problem(13) and terminates at the ninth iteration. The CPU-time indicates NHAM with RK_3 steps takes more time than NHAM with RK_2 steps.

Figure 20.A **Figure 20.B**

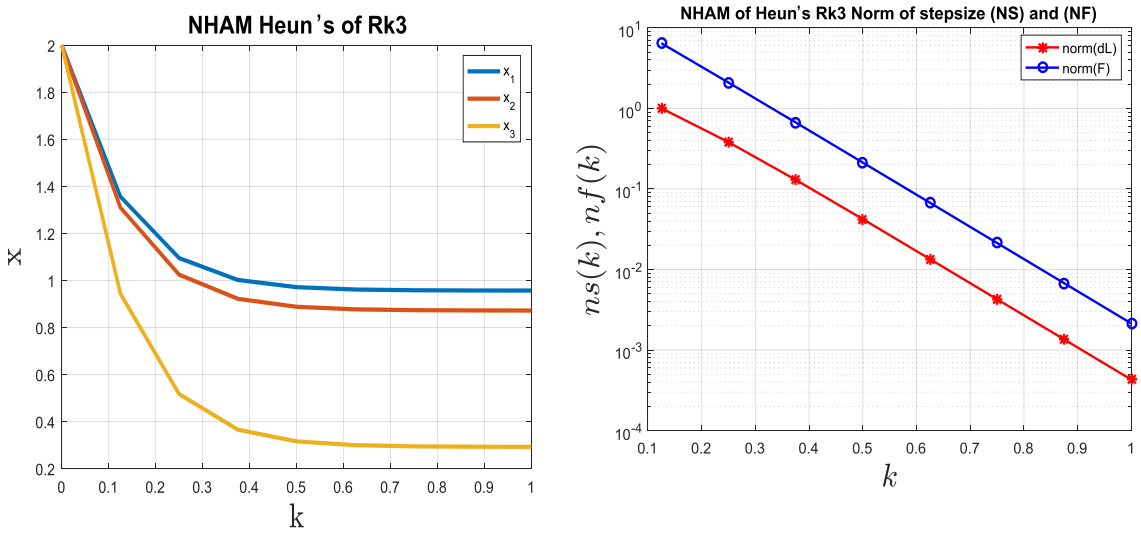


Figure20: NHAM_RK3 solution and norms of step size and norm of function Vs Iteration

As shown in figure 20.A, the output or solution(x) versus each Homotopy parameters. This shows as the value of parameter increases from 0 to 1, the accuracy of the solution increases and converge to a better accurate solution when value of parameter is 1. Moreover as shown in Figure 20.B, which proves the convergence of NHAM_RK3 for sample problem of equation (13) and the norm errors of both the function and steps(NHDE) are getting smaller in between successive iterates. The Euclidean norm error of both Newton step and the function are lesser than NHAM with RK_2 steps.

4.5. NHAM with Classical 4th order RK Steps

$$
\begin{aligned}
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h) \\
k_3 &= f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h) \\
k_4 &= f(x_i + h, y_i + k_3 h) \\
y_{i+1} &= y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h \\
&= y_i + \frac{1}{3}\left(\frac{k_1 + k_2}{2} + \frac{k_2 + k_3}{2} + \frac{k_3 + k_4}{2}\right)h
\end{aligned} \tag{74}
$$

Algorithm of NHAM with classical RK_4 Steps

Step 1 : Input parameter

U - Initial guess of problem of problem

Maxiter - Maximum number of iteration

dl - Change of Homotopy parameter

N - Number of step in Rk_4 equation

$h_i = 1/N$ - Rk_4 step size , h is grade spacing

$x = zeros(k+1, n)$ - Initialization

J - Jacobian matrix

X - Solution vector

Step 2: compute Jacobian matrix $J = jacbian(f)$

Step 3: compute *dl* for slope 2 and update solution

$$dl = -J \setminus f0$$
$$k_1 = h * dl$$
$$p = u + 1/2 * k_1$$

Step 4: compute slope 3 and update

$$dl1 = -J1 \setminus f0$$
$$k_2 = h * dl1$$
$$p_1 = u + 1/2 * k_2$$

Step 5: compute slope 4 and update solution

$$dl2 = -J2 \backslash f0$$

$$k_3 = h * dl2$$

$$p_2 = u + *k_3$$

$$u = u + (k_1 + 2*k_2 + 2*k_3 + k_4)/6$$

Step 6 : compute Classical Rk4 slop by $kk = u + \left(\frac{k_1 + k_2}{2} + \frac{k_2 + k_3}{2} + \frac{k_3 + k_4}{2}\right)/3$

Step 7: Numerical output

4.5.1. Numerical output with classical RK_4 Steps

For comparison purpose, NHAM with Classical RK_4 steps is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 20: NHAM number of iteration for Classical RK_4 method

NHAM with RK4 steps			
I	X(1)	X(2)	X(3)
1	2.000000000000000e+00	2.000000000000000e+00	2.000000000000000e+00
2	9.699577037009443e-01	8.873184241440124e-01	3.127106054037049e-01
.	..	…	…
6	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
7	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
8	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
9	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
CPU Time for NHAM with Classical RK4 Steps =22.8541465			
Elapsed time is 20.684354 seconds.			

As shown in Table 20, the output of the MATLAB programing code of NHAM with RK_4 step converges for sample problem(13) and terminates at the ninth iteration. However, after 6^{th} iteration change of solution is very small, i.e. it is almost reach with a better accuracy. The CPU-time indicates NHAM with RK_4 steps takes more time than NHAM with RK_3 steps.

Figure 21.A

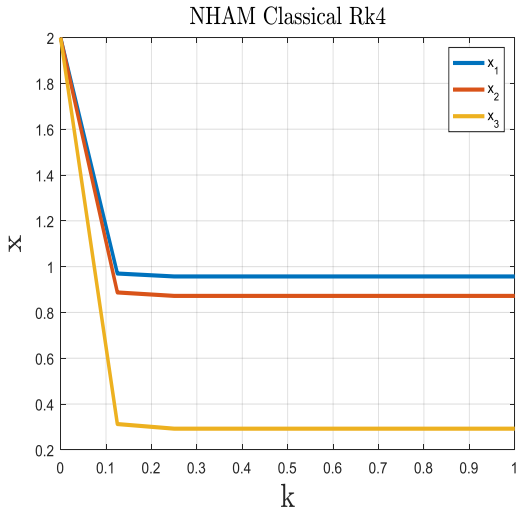


Figure 21.B

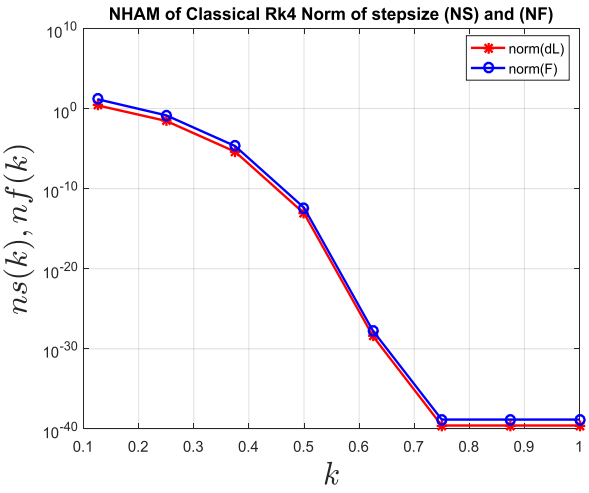


Figure 21: NHAM Classical Rk4 solution and norms of step size Vs norm of function

As shown in figure 21.A, the output or solution(x) versus each Homotopy parameters. This shows as the value of parameter increases from 0 to 1, the accuracy of the solution increases and converge to a better accurate solution when value of parameter is 1. Moreover as shown in Figure 21.B, which proves the convergence of NHAM_RK4 for sample problem of equation (13) and the norm errors of both the function and steps(NHDE) are getting smaller in between successive iterates. The Euclidean norm errors of both Newton step and the function are lesser than NHAM with RK_3 steps.

4.6. NHAM with Fifth order Runge-Kutta (Bucher1) steps

$$
\begin{aligned}
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + \frac{1}{4}h, y_i + \frac{1}{4}k_1 h) \\
k_3 &= f(x_i + \frac{1}{4}h, y_i + \frac{1}{8}k_1 h + \frac{1}{8}k_2 h) \\
k_4 &= f(x_i + \frac{1}{2}h, y_i - \frac{1}{2}k_2 h + k_3 h) \\
k_5 &= f(x_i + \frac{3}{4}h, y_i + \frac{3}{16}k_1 h + \frac{9}{16}k_4 h \\
k_6 &= f(x_i + h, y_i - \frac{3}{7}k_1 h + \frac{2}{7}k_2 h + \frac{12}{7}k_3 h - \frac{12}{7}k_4 h + \frac{8}{7}k_5 h) \\
y_{i+1} &= y_i + \frac{1}{90}(7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6)h
\end{aligned}
\tag{75}
$$

Algorithm of NHAM for Bucher1 RK_5 Method

Step 1 : Input parameter

U - Initial guess of problem of problem

Maxiter - Maximum number of iteration

dl - Change of Homotopy parameter

N - Number of step in Rk_5 equation

$h_i = 1/N$ - Rk_5 step size , h is grade spacing

$x = zeros(k+1, n)$ - Initialization

J - Jacobian matrix

X - Solution vector

Step 2: Compute Jacobian matrix $J = jacbian(f)$

Step 3: Compute *dl* by slope 1 and update the solution

$$dl = -J \setminus f0$$

$$k_1 = h * dl$$

$$p = u + k_1 * 0.25$$

$$dl1 = -J1 \backslash f0$$

Step 4: Compute slope 2 and update $k_2 = h * dl1$

$$p_1 = u + (k_1 + k_2)/8$$

Step 5: Compute slope 3 and update solution

$$dl2 = -J2 \backslash f0$$

$$k_3 = h * dl2$$

$$p_2 = u + (1/8) * k_2 + k_3$$

Step 6: Compute slope 4 and update solution

$$dl3 = -J3 \backslash f0$$

$$k_4 = h * dl3$$

$$p_3 = u + (3/16) * k_2 + (9/16) * k_4$$

Step 7: Compute slope 5 and update solution

$$dl4 = -J4 \backslash f0$$

$$k_4 = h * dl4$$

$$p_4 = u - (3/7) * k_1 + (2/7) * k_2 + (12/7) * k_3 - (12/7) * k_4 + (8/7) * k_5$$

Step 6: Compute slope 6

$$dl5 = -J5 \backslash f0$$

$$k_6 = h * dl5$$

Step 6: Compute Bucher1 Rk5 slop by

$$u = u + (7 * k_1 + 32 * k_3 + 12 * k_4 + 32 * k_5 + 7 * k_6)/90$$

$$kk = (7 * k_1 + 32 * k_3 + 12 * k_4 + 32 * k_5 + 7 * k_6)/90$$

Step 7: Numerical output

Numerical output with classical RK_5 Steps

For comparison purpose, NHAM Bucher1 RK_5 method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 21: Number of iteration for NHAM Bucher1 RK_5 method

Solution of NHAM_ Rk5			
I	X(1)	X(2)	X(3)
1	2.000000000000000e+00	2.000000000000000e+00	2.000000000000000e+00
2	1.023315416034467e+00	9.496539948977500e-01	3.977973256556855e-01
3	9.574120567551337e-01	8.726000715905279e-01	2.929346098764011e-01
…	…	…	…
7	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
8	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
9	9.570972114233326e-01	8.722119589038248e-01	2.925043254468138e-01
CPU Time for NHAM with RK5 Step =27.8305784			
Elapsed time is 28.470442 seconds.			

As shown in Table 21, the output of the MATLAB programing code of NHAM with RK_5 step converges for sample problem(13) and terminates at the ninth iteration. However, after 6^{th} iteration change of solution is very small, i.e. it has a better accuracy. However, the CPU-time indicates NHAM with RK_5 steps takes more time than NHAM with RK_4 steps.

Figure 22.A

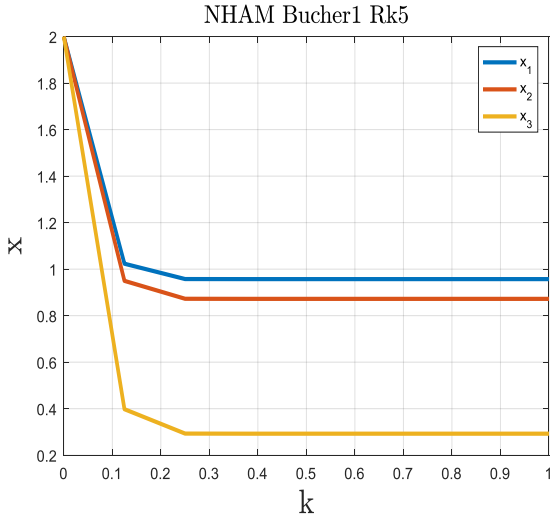


Figure 22.B

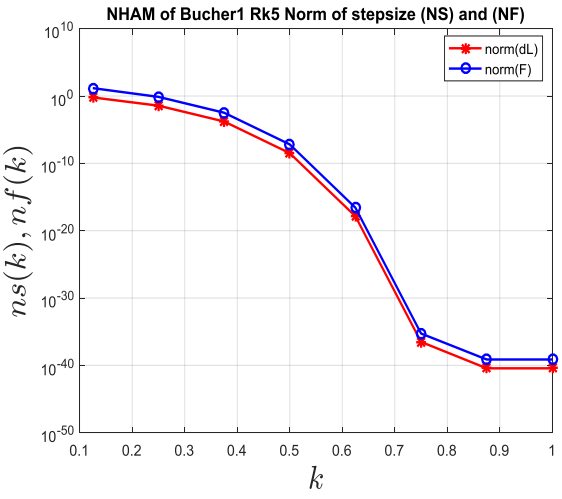


Figure 22: NHAM Bucher Rk5 solution and norms of step size Vs norm of function

As shown in figure 22.A, the output or solution(x) versus each Homotopy parameters. This shows as the value of parameter increases from 0 to 1, the accuracy of the solution increases and converge to a better accurate solution when value of parameter is 1. Moreover as shown in Figure 22.B, which proves the convergence of NHAM_RK5 for sample problem of equation (13) and the norm errors of both the function and steps(NHDE) are getting smaller in between successive iterates. The Euclidean norm errors of both Newton step and the function are lesser than NHAM with RK_4 steps.

4.7. NHAM with Sixth order Runge-Kutta (Bucher2) Steps

$$
\begin{aligned}
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + h, y_i + k_1 h) \\
k_3 &= f(x_i + \frac{2}{3}h, y_i + \frac{4}{9}k_1 h + \frac{2}{9}k_2 h) \\
k_4 &= f(x_i + \frac{1}{3}h, y_i + \frac{11}{36}k_1 h + \frac{1}{9}k_2 h - \frac{1}{12}k_3 h) \\
k_5 &= f(x_i - \frac{1}{3}h, y_i + \frac{151}{36}k_1 h + \frac{29}{9}k_2 h - \frac{7}{4}k_3 h - 6k_4 h) \\
k_6 &= f(x_i + \frac{4}{3}h, y_i - \frac{121}{9}k_1 h - \frac{116}{9}k_2 h + \frac{32}{3}k_3 h + 18k_k h - 2k_5 h) \\
k_7 &= f(x_i + h, y_i - \frac{5}{4}k_1 h - \frac{29}{23}k_2 h - \frac{397}{276}k_3 h + \frac{152}{69}k_4 h - \frac{10}{69}k_5 h + \frac{1}{69}k_6 h) \\
y_{i+1} &= y_i + \left(\frac{23}{160}k_1 + \frac{29}{80}k_3 + \frac{29}{80}k_4 - \frac{1}{160}k_5 - \frac{1}{160}k_6 + \frac{29}{160}k_7 \right) h
\end{aligned}
\tag{77}
$$

Algorithm for NHAM with RK_6 steps

Step 1 : Input parameter

U - Initial guess of problem of problem

Maxiter - Maximum number of iteration

dl - Change of Homotopy parameter

N - Number of step in Rk_6 equation

$h_i = 1/N$ - Rk_6 step size , h is grade spacing

$x = zeros(k+1, n)$ - Initialization

J - Jacobian matrix

X - Solution vector

Step 2: Compute Jacobian matrix $J = jacbian(f)$

Step 3: Compute dl for slope 1 and update solution

$$dl = -J \backslash f0$$
$$k_1 = h * dl$$
$$p = u + k_1$$

$$dl1 = -J1 \backslash f0$$

Step 4: Compute slope 2 and update $k_2 = h * dl1$

$$p_1 = u + 4/9 * k_1 + 2/9 * k_2$$

Step 5: Compute slope 3 and update solution

$$dl2 = -J2 \backslash f0$$
$$k_3 = h * dl2$$
$$p_2 = u + 11/36 * k_1 + 1/9 * k_2 - 1/12 * k_3$$

Step 6: Compute slope 4 and update solution

$$dl3 = -J3 \backslash f0$$
$$k_4 = h * dl3$$
$$p_3 = u + 151/36 * k_1 + 29/9 * k_2 - 7/4 * k_3 - 6 * k_4$$

Step 7: Compute slope 5 and update solution

$$dl4 = -J4 \backslash f0$$
$$k_5 = h * dl4$$
$$p_4 = u - 112/9 * k_1 - 116/9 * k_2 + 32/3 * k_3 + 18 * k_4 - 2 * k_5$$

Step 6: Compute slope 6 and update

$$dl5 = -J5 \backslash f0$$
$$k_6 = h * dl5$$
$$p_5 = u - 5/4 * k_1 - 29/23 * k_2 + 397/276 * k_3 + 152/69 * k_4 - 10/69 * k_5 + 1/69 * k_6$$

Step 7: Compute slope 7

$$dl6 = -J6 \backslash f0$$
$$k_7 = h * dl6$$

Step 8: Compute Bucher 2 Rk6 slope by

$$u = u + (23/160 * k_1 + 29/80 * k_3 + 29/80 * k_4 - 1/160 * k_5 - 1/160 * k_6 + 29/160 * k_7)$$
$$kk = (23/160 * k_1 + 29/80 * k_3 + 29/80 * k_4 - 1/160 * k_5 - 1/160 * k_6 + 29/160 * k_7)$$

Step 8: Display Numerical output

Numerical output with classical RK_6 Steps

For comparison purpose, NHAM with Bucher2 RK_6 step is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 22: Number of iteration for NHAM Bucher 2 RK_6 method

NHAM with Rk6 Steps			
I	X(1)	X(2)	X(3)
1	2.000000000000000e+00	2.000000000000000e+00	2.000000000000000e+00
2	9.800856654561335e-01	9.025865215722668e-01	3.270895002183018e-01
3	9.562725589272951e-01	8.711229158497873e-01	2.912454957891883e-01
4	9.571281843061597e-01	8.722528629027022e-01	2.925515828215656e-01
5	9.570960500085094e-01	8.722104250953708e-01	2.925025533674690e-01
6	9.570972549764845e-01	8.722120164217705e-01	2.925043918998907e-01
7	9.570972097900895e-01	8.722119567469020e-01	2.925043229548235e-01
8	9.570972114845792e-01	8.722119589847094e-01	2.925043255402634e-01
9	9.570972114210358e-01	8.722119589007916e-01	2.925043254433094e-01
CPU Time for NHAM with RK6 Steps =26.7853717			
Elapsed time is 23.316504 seconds.			

As shown in Table 22, the output of the MATLAB programing code of NHAM with RK_6 step converges for sample problem(13) and terminates at the ninth iteration. However, after 6^{th} iteration change of solution is very small, i.e. it has a better accuracy. However, the CPU-time indicates NHAM with RK_6 steps takes more time than NHAM with RK_5 steps.

Figure 23.A **Figure 23.B**

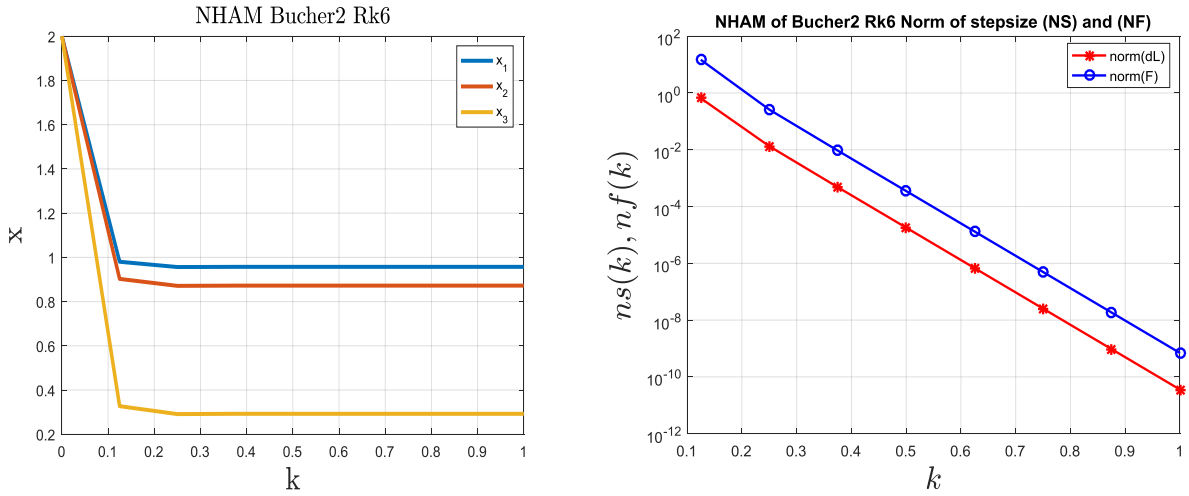


Figure 23: NHAM with Rk_6 solution and norms of step size Ns Vs NF

As shown in figure 23.A, the output or solution(x) versus each Homotopy parameters. This shows as the value of parameter increases from 0 to 1, the accuracy of the solution increases and converge to a better accurate solution when value of parameter is 1. Moreover as shown in Figure 23.B, which proves the convergence of NHAM_RK6 for sample problem of equation (13) and the norm errors of both the function and steps(NHDE) are getting smaller in between successive iterates. The Euclidean norm errors of both Newton step and the function of NHAM with RK_6 are lesser than NHAM with RK_5 steps.

CHAPTER FIVE

5. SOLVING SYSTEMS OF NONLINEAR ALGEBRAIC EQUATIONS USING OPTIMIZATION METHODS

Many problems in chemical engineering, are expressed mathematically as optimizations problems, and involve finding the particular x that minimizes some cost function $F(x)$. [52]Each component of x may vary either continuously or discretely.

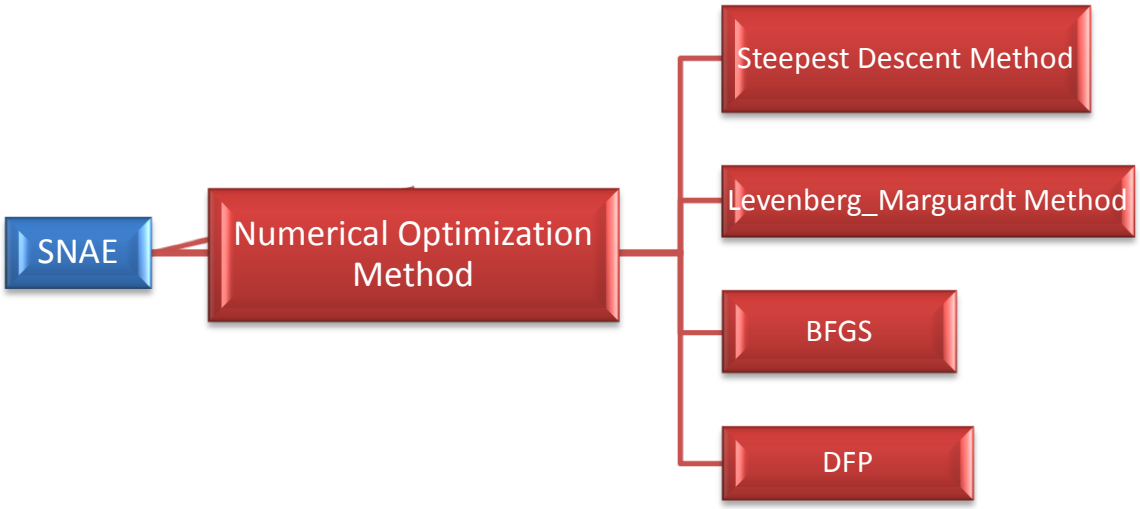


Figure 24: Schematic diagram of Optimization method

Optimization means finding the best solution among many feasible solutions that are available to us. Feasible solutions are those that satisfy all the constraints in the optimization problem. The best solution could be minimizing the cost of a process or maximizing the efficiency of a system.

In general, the optimizations classified into two type; constrained and unconstrained optimization. In this thesis concerned solving system of nonlinear algebraic equation by using unconstrained optimization that involves the minimization of real-value objective function f(x),that is finding[56]; $\min f(x) \; or \; \max(-f(x))$.

The term unconstrained means that **no restriction** is placed on the range of x. The methods can be broadly categorized in terms of the derivation information that is used. They are

gradient-based and non–gradient-based search methods. As the name suggests, gradient-based methods require gradient information in determining the search direction.

Search methods that use only function evaluations are most suitable for problems that are not smooth or have a number of discontinuities. Gradient methods are more efficient when the function to be minimized/ maximized is continuous in the first derivative.

In general the solution set of uncostrained optimization is basically grident based method, which is search direction method. [52]The grident based method are steepest descent method, Levenberg–Marquardt methods, Davidon–Fletcher–Powell (DFP) and Broyden–Fletcher–Goldfarb–Shanno (BFGS). The search direction computed by these methods uses the gradient information.

5.1. Numerical Optimization Method

At each iteration of numerical optimization method, there is need to determine two things [56]

1, Descent direction (d_k) ,

2, Step length (α_k):

$$x^{k+1} = x^k + \alpha_k d_k \qquad (78)$$

There are various methods for Step-length calculation. Such as Analytic line search, Equal interval search, Section search, Golden section search, Quadratic interpolation method and Approximation line search.

5.1.1. Golden Search Method

This method is applicable to an unconstrained minimization problem [5] such that the solution interval $[a,b]$ is known and the objective function $f(x)$ is unimodal within the interval; that is, the sign of its derivative $f'(x)$ changes at most once in $[a,b]$ so that $f(x)$ decreases/increases monotonically for $[a,x^0]/[x^0,b]$,where x^0 is the solution .

Golden Search converges linearly to the minimum with linear convergence rate $g \approx 0.618$.

Let see the methods that will be covered in this thesis.

5.2. STEEPEST DESCENT METHOD

First, we discuss the connection between solving systems of nonlinear equations and quadratic minimization problems. [57] The idea is to minimize the $2-norm$ of the residual.

To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the *positive* of the gradient, one approaches a local maximum of that function; the procedure is then known as **gradient ascent**.

Gradient descent is also known as **steepest descent**, or the **method of steepest descent**. Gradient descent should not be confused with the method of steepest descent for approximating integrals.

The gradient method is a descent method in which one begins to iterate at an arbitrary point x_0 and continues following the line of maximum descent of the paraboloid obtaining a succession of points x_1, x_2 ..... until you get a point close enough to the solution x^*.

The search direction ***Si*** that reduces the function value is a descent direction. It was discussed earlier that along the gradient direction, there is the maximum change in the function value. Thus, along the **negative gradient direction**, the function value **decreases** the most. The negative gradient direction is called the **steepest descent direction**. That is,

$$S_i = -\nabla f(x_i) \qquad (79)$$

In successive iterations, the design variables can be updated using the equation

$$x_{i+1} = x_i - \alpha \nabla f(x_i) \qquad (80)$$

where α is a positive scalar parameter that can be determined using the line search algorithm such as the golden section method.

The steepest descent method ensures a reduction in the function value at every iteration. If the starting point is far away from the minimum, the gradient will be higher and the function reduction will be maximized in each iteration. Because the gradient value of the function changes and decreases to a small value near the optimum, the function reduction is uneven and the method becomes sluggish (slow convergence) near the minimum.

Locally the steepest descent direction is the best direction in the sense that it reduces the objective function. Steepest Descent method converges only linearly to the solution, but it will usually converge even for poor initial approximations.

Algorithm for the Steepest Descent Method

Step 1: Input parameter

Step 2: Compute $\nabla s(x_0) = 2[J(x_0)]^{-1}F(x_0) \quad A_0 = (q_1,\ q_2,\ q_3....,q^n)$

Step 3: Compute $H_0 = \|A_0\|_2$

Step 4: Compute $H = \frac{A_0}{H_0}$, taking $\alpha_1 = 0,\ \alpha_2 = 0.5, \alpha_3 = 1$

Step 5: Compute $\nabla s(x_0) = 2[J(x_0)]^{-1}F(x_0)$

If $s_3 \geq s_1$ reduce the size α_3, until such a time we have $s_3 \geq s_1$

Step 6: Compute: $s_3 = s_1(x_0 - \alpha_3 H)$

Step 7: Compute: $h_1 = \frac{(s_2 - s_1)}{\alpha_2}$, $h_2 = \frac{(s_3 - s_2)}{\alpha_3 - \alpha_2}$ and $h_3 = \frac{(h_2 - h_1)}{\alpha_3}$

Step 8: Compute: $B_0 = 0.5(\alpha_2 - \frac{h_1}{h_3})$

Step 9: Compute: $s_0 = s(x_0 - B_o H)$

If $s_0 < s_3, set\ B = B_0$

Step 10: Compute: $x_{k+1} = (x_0 - BH)$

Step 11: Numerical Output

Numerical output

For comparison purpose, Steepest Descent method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 23: Steepest Descent solution(x) and norm of step size and norm of function

Optimization Steepest Descent Method					
K	x1	x2	x3	NS	NF
1	0.5773507	0.57735027	0.57735027	1.00000000	2.28210115
2	0.7854070	0.88614346	0.26614625	0.48527224	1.05533030
3	0.9199787	0.82731184	0.28729099	0.14838322	0.27388092
4	0.9337969	0.86869979	0.29127115	0.04381463	0.13092859
5	0.9521234	0.86471629	0.29285593	0.01882250	0.04312004
Norm of f 0.0431					
CPU Time Steepest descent Method = 0.0312002					

As shown in table 23, the numerical output of Steepest Descent method, it shows that the solution(x) converge for a sample problem with in 5 iteration. Norm of step size and norm of function are continuously decreases and converge for certain error tolerance

Figure 25.A **Figure 25.B**

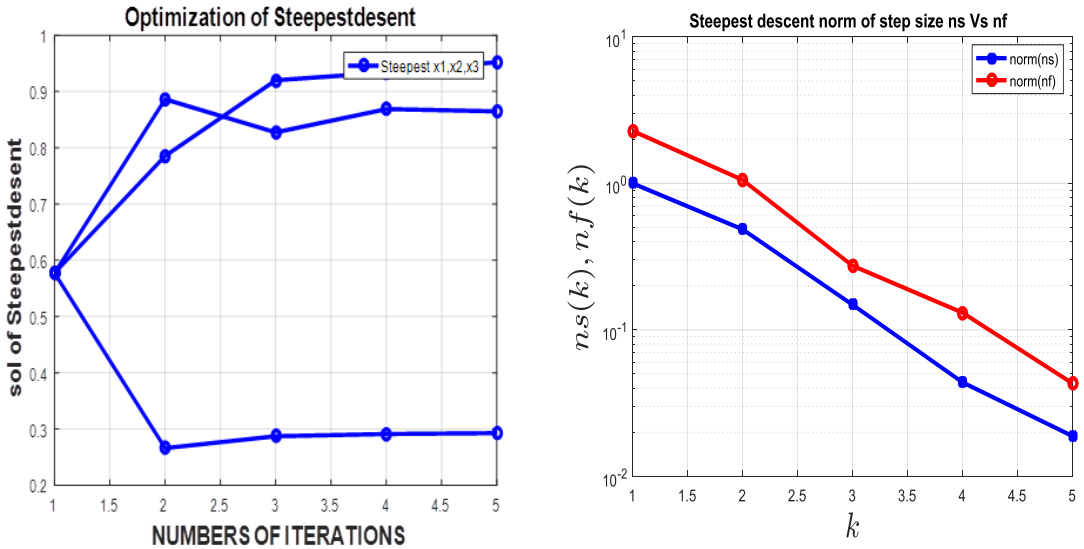


Figure25: Steepest Descent solution and norms of step size Vs norm of function

As we see in Figure 25, we observe that steepest descent method takes five iteration converge to the solution. Moreover, when we see the norms of step size and norm of function; it is stable and continuously decreases. However, if the initial guess is near or not to the solution, it will converge.

5.3. LEVENBERG–MARQUARDT METHOD

The advantage of the steepest descent method is that it reaches closer to the minimum of the function in a few iterations even when the starting guess is far away from the optimum. However, the method shows sluggishness near the optimum point. [57]On the contrary, Newton's method shows a faster convergence if the starting guess is close to the minimum point. Newton's method may not converge if the starting point is far away from the optimum point.

The Levenberg–Marquardt method is a kind of hybrid method that combines the strength of both the steepest descent and Newton's methods. The search direction in this method is given by

$$S_i = -[H + \lambda I]-1\nabla f\left(x_i\right) \quad (81)$$

where ***I*** is an identity matrix, H hessian matrix and λ is a scalar that is set to a high value at the start of the algorithm. The value of λ is altered during every iteration depending on whether the function value is decreasing or not. If the function value decreases in the iteration, λ decreases by a factor (less weightage on steepest descent direction). On the other hand, if the function value increases in the iteration, λ increases by a factor (more weightage on steepest descent direction).

Algorithm for the Levenberg–Marquardt Method

Step 1: Input parameter

x_i : (starting value of design variable) and α

β_1 : Tolerance of function value from previous iteration

β_2 : Tolerance on gradient value

Δx : required for gradient computation

Step 2: Compute $f(x_i)$, $\nabla f(x_i)$, and $[H]$ (function, gradient, and Hessian)

$S_i = -[H + \lambda I]-1\nabla f\left(x_i\right)$ (Search direction)

$x_{i+1} = x_i + S_i$ (Update the design vector)

Step 3 : If $f(x_{i+1}) < f(x_i)$

then change the value of λ as $\lambda / 2$

else change the value of λ as 2λ

$f(x_{i+1}) - f(x_i) > 11\_or \|\nabla f(x_i)\| > \beta_2$

then go to Step 3

else go to Step 4

Step 4: Converged. Print $x^* = x_{i+1}$

Step5: Converged. Print $f(x^*) = f(x_{i+1})$

Numerical output

For comparison purpose, Levenberg-Marquard method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 24: Levenberg-Marquard method solution(x) and norm of step size vs norm of function

Solution By Levenberg_Marquard Method					
K	x1	x2	x3	Ns	Nf
1	0.999605	0.985765	0.956121	50.596458	16.000000
2	0.997743	0.962870	0.881675	46.206042	13.767790
3	0.992131	0.931814	0.769796	39.162835	10.445394
4	0.980713	0.899456	0.628434	29.515861	6.463882
5	0.966384	0.876691	0.485350	18.872917	2.964095
6	0.957100	0.868601	0.375202	9.760890	0.898781
7	0.955486	0.869529	0.315933	3.828589	0.154229
8	0.956520	0.871479	0.296344	1.025244	0.011930
9	0.957016	0.872119	0.292840	0.164102	0.000321
10	0.957092	0.872206	0.292520	0.014348	0.000003
11	0.957097	0.872212	0.292505	0.000668	0.000000
Norm of f 7.705257153618898e-12					
CPU Time Levenberg_Marquard Method = 0.062					

As shown in table 24,the numerical output of Levenberg_Marquard method for our sample problem, it shows the method converges to a solution with eleven iterations. Both the norm

of step size and norm of function are continuously decreases; it can converge for a sample problem with in 11 iteration, with a better accuracy than Steepest Decent method.

Figure 26.A **Figure 26.B**

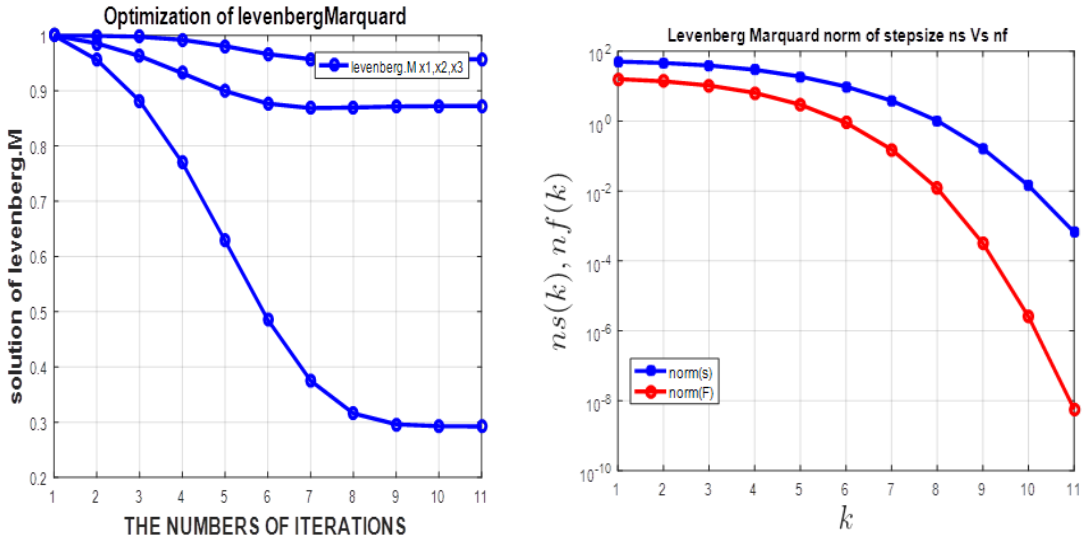


Figure26: Levenberg-Marquard solution(x) and norms of step size vs function.

As we see in Figure 26, we observe that Levenberg_Marquard method convergence in 11^{th} iteration. In addition to that when we see norm error of step size and norm error of function are decreases continuously and move smoothly and have a better accuracy than steepest decent method.

5.4. BROYDEN–FLETCHER–GOLDFARB–SHANNO (BFGS) METHOD

At each iteration of quasi-newton's methods, positively definite Hessian approximation B_k is update to a new approximation B_{k+1} using y_k and s_k defined by $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s^k = x^{k+1} - x^k$ respectively, for which the quasi-newton condition is satisfied. [57]To define the update matrix several formulas have been proposed. These formulas belong to be the Broyden family of updates which has certain useful properties when the line search structure is used. They are DFP, BFGS and SR1. Here only see BFGS and DEP method.

In the BFGS method, the Hessian is approximated using the variable metric. Matrix [**B**] given by the equation

$$[B]_{k+1} = [B] + \frac{y \nabla y^T}{\nabla y^T \Delta x} + \frac{\nabla f(x_k) \nabla f(x_k)^T}{\nabla f(x_k)^T s_k} \quad (82)$$

Search direction of BFGS

$$S_{i+1} = -[[B]_{i+1}]^{-1} \nabla f(x_{i+1}) \quad (83)$$

It is important to note that whereas the matrix [**B**] converges to the inverse of the Hessian in the DFP method, the matrix [**B**] converges to the Hessian itself in the BFGS method. As the BFGS method needs fewer restarts as compared to the DFP method, it is more popular than the DFP method.

Algorithm for the BFGS Method

Step 1: Input Parameter

x_i : (starting value of design variable) and α

β_1 : Tolerance of function value from previous iteration

β_2 : Tolerance on gradient value

Δx : required for gradient computation

[**B**] : initialize to identity matrix

Step 2: Compute $f(x_i)$ and $\nabla f(x_i)$ (function and gradient vector)

$S_i = -\nabla f(x_i)$ (Search direction)

$x_{i+1} = x_i + \alpha S_i$ (Update the design vector)

Minimize $f(x_{i+1})$ and determine α (use golden section method)

Step 3: Compute Δx and ∇y

$$[B]_{k+1} = [B] + \frac{y \nabla y^T}{\nabla y^T \Delta x} + \frac{\nabla f(x_k) \nabla f(x_k)^T}{\nabla f(x_k)^T s_k}$$

$$S_{i+1} = -[[B]_{i+1}]^{-1} \nabla f(x_{i+1})$$

Step 4: Minimize $f(x_{i+2})$ and determine α (use the golden section method)

if $|f(x_{i+2}) - f(x_{i+1})| > \beta_1$ $or \|\nabla f(x_{i+1})\| > \beta_2$

then go to Step 4

else go to Step 5

Step 5: Converged. Print $x^* = x_{i+2}, f(x^*) = f(x_{i+2})$

Numerical output

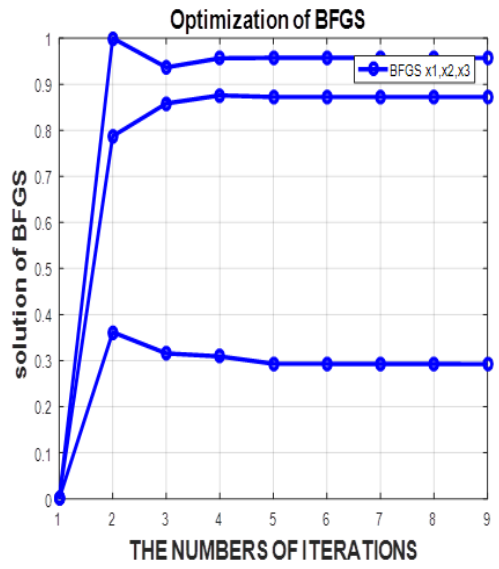
For comparison purpose, BFGS method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

Table 25: BFGS method solution x and norm of step size Vs norm of function

BFGS Method Solution for Our Problem					
K	x1	x2	x3	Ns (k)	Nf (k)
2	0.99999984	0.78725049	0.36175178	10.28730727	0.55589877
3	0.93667514	0.85829099	0.31585099	1.31613813	0.02175895
4	0.95635945	0.87580906	0.30913313	0.89150642	0.00716448
5	0.95740075	0.87216629	0.29326863	0.04316561	0.00001698
6	0.95731477	0.87217906	0.29305747	0.03112862	0.00000885
7	0.95725409	0.87218817	0.29290336	0.02245611	0.00000461
8	0.95721030	0.87219473	0.29279216	0.01619943	0.00000240
9	0.95707650	0.87221478	0.29245232	0.00291759	0.00000008
Norm of f 7.853335950309482e-08					
cpu Time BFGS Method = 0.0780005					

As shown in table 25, the output of BFGS method for our sample problem, it shows that the method converges to a solution with 9 iteration. In addition to norm of step size and norm of function are continuously decreases. Moreover, we can observe that the number of iteration is smaller to convergence than Levenberg- Marquardt method.

Figure 27.A **Figure 27.B**



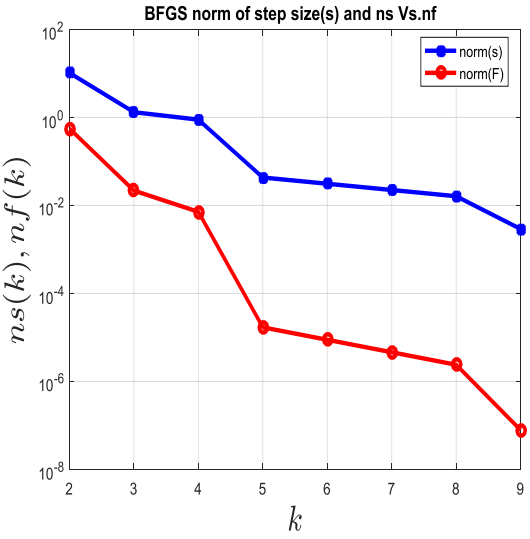


Figure27: BFGS Method solution of(x) and norms of step size Vs function

As we see in Figure 27, we observe that BFGS method convergence in 9 iteration. In addition to that, we can observe norm error of step size and norm error of function are decreases continuously and not move smoothly to the solution, there is a little beet instability to converge.

5.5. DAVIDON–FLETCHER–POWELL (DFP) METHOD

In the DFP method, the inverse of the Hessian is approximated by a matrix [**B**] and the search direction is given[57]

$$S_i = -[B]\nabla f(x_k) \qquad (84)$$

The information stored in the matrix [**B**] is called as the metric and because it changes with every iteration, the DFP method is known as the variable metric method. Because this method uses first-order derivatives and has the property of quadratic convergence, it is referred to as a *quasi-Newton* method. The inverse of the Hessian matrix can be approximated as

$$[B]_{k+1} = [B] + \frac{\Delta x \Delta x^T}{\Delta x^T \nabla y} - \frac{[B]_k \nabla y \nabla y^T [B]_k}{\nabla y^T [B]_k \nabla y} \quad (85)$$

where

$$\Delta x_2 = \Delta x_1 - \Delta x_0 \quad (86)$$

$$\nabla y_2 = \nabla y_1 - \nabla y_0 \quad (87)$$

The matrix [**B**] is initialized to the identity matrix.

Algorithm for the DFP Method

Step 1: Input parameter

x_i : (starting value of design variable) and α

β_1 : Tolerance of function value from previous iteration

β_2 : Tolerance on gradient value

[**B**] : initialize to identity matrix

Step 2: Compute $f(x_i)$ and $\nabla f(x_i)$ (function and gradient vector)

$S_i = -\nabla f(x_i)$ (Search direction)

$x_{i+1} = x_i + \alpha S_i$ (Update the design vector)

Minimize $f(x_{i+1})$ and determine α (use golden section method)

Step 3: Compute Δx and ∇y

$$[B]_{k+1} = [B] + \frac{\Delta x \Delta x^T}{\Delta x^T \nabla y} - \frac{[B]_k \nabla y \nabla y^T [B]_k}{\nabla y^T [B]_k \nabla y} -$$

$$S_i = -[B]\nabla f(x_k)$$

Minimize $f(x_{i+2})$ and determine α (use the golden section method)

if $|f(x_{i+2}) - f(x_{i+1})| > \beta_1$ or $\|\nabla f(x_{i+1})\| > \beta_2$

then go to Step 4

else go to Step 5

Step 4: Converged. Print $x^* = x_{i+2}$, $f(x^*) = f(x_{i+2})$

Numerical output

For comparison purpose, DFP method is applied and tested on equation (13) and the numerical result and norm error are displayed by using MATLAB program. By taking inputs: number of iteration is 25 and error tolerance of 0.0001.

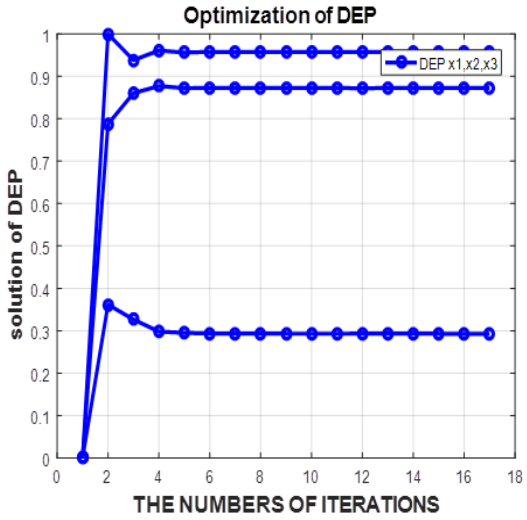
Table 26: DFP method solution(x) and norm of step size Vs norm of function

DFP Method Solution					
K	X(1)	X(2)	X(3)	Ns(k)	Nf(k)
2	0.99999984	0.78725049	0.36175178	10.28730727	0.55589877
3	0.93797703	0.86009602	0.32659130	1.65804523	0.03270315
4	0.96076741	0.87700593	0.29808424	0.43086829	0.00172174
5	0.95588267	0.87199794	0.29508263	0.13955456	0.00018230
6	0.95766512	0.87255147	0.29361139	0.06279332	0.00004039
7	0.95728630	0.87255958	0.29332510	0.04776783	0.00002104
8	0.95714995	0.87247391	0.29322306	0.04179513	0.00001517
9	0.95700913	0.87230554	0.29306616	0.03037530	0.00000814
10	0.95703868	0.87238340	0.29293641	0.02925939	0.00000625
11	0.95704496	0.87226132	0.29295909	0.02265740	0.00000495
12	0.95697645	0.87196208	0.29326293	0.03339821	0.00001322
13	0.95714250	0.87237565	0.29297327	0.02660696	0.00000632
14	0.95748836	0.87252646	0.29309281	0.03777557	0.00001459
15	0.95723510	0.87232990	0.29310400	0.02976568	0.00000925
16	0.95717841	0.87251693	0.29282753	0.02974087	0.00000621
17	0.95707498	0.87237267	0.29289972	0.02571971	0.00000506
Norm of f 5.0636e-06					
cpu Time DFP Method = 0.0780005					

As shown in table 26, the output of the MATLAB code for DFP method, it shows the solution (X) for our sample problem, in addition to that norm of step size and norm of function are continuously decreases to reach a small acceptable error, it can converge for a

sample problem with in 17 iteration. However, the number of iteration is very large as compared to Optimization methods.

Figure 28.A **Figure 28.B**



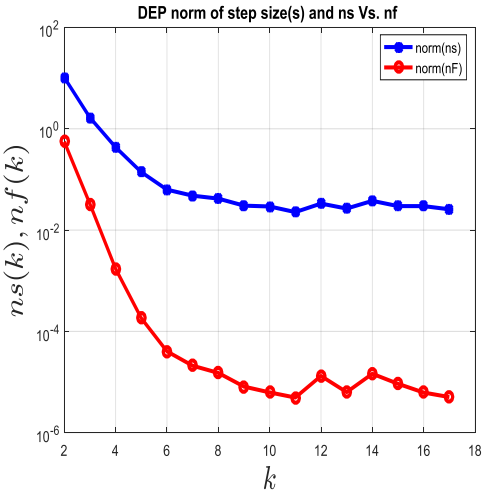


Figure 28: DFP method solution(x) and norms of step size Vs Function.

As we see in Figure 28, we observe that DFP method convergence in 17^{th} iteration and. In addition to that when we see norm errors of step size and norm error of function are decreases continuously and move to the solution.

COMPARISONS OF NUMERICAL OUTPUT FOR SOLVING SNAE OF OPTIMIZATION METHOD

K	Steepest Descent		Levenberg–Marquar		BFGS		DEP	
	NS	NF	NS	NF	NS	NF	NS	NF
1	50.5964	16.00000	50.596458	16.00000	10.287307	0.555898	10.28730	0.55589877
2	58.1292	19.85597	46.206042	13.76779	1.3161381	0.0217589	1.65804	0.03270315
3	9.04390	1.04247	39.162835	10.44539	0.8915064	0.0071644	0.43086	0.00172174
4	20.6930	43.75432	29.515861	6.463882	0.0431656	0.0000169	0.13955	0.00018230
5	42.3452	20.67610	18.872917	2.964095	0.0311286	0.0000088	0.06279	0.00004039
6	19.5616	6.26764	9.760890	0.898781	0.0224561	0.0000046	0.04776	0.00002104
7	12.9631	1.24293	3.828589	0.154229	0.0161994	0.0000024	0.04179	0.00001517
8	4.70853	0.26793	1.025244	0.011930	0.0029175	0.0000000	0.03037	0.00000814
9	2.78805	0.05282	0.164102	0.000321			0.02925	0.00000625
10	0.97341	0.01086	0.014348	0.000003			0.02265	0.00000495
11							0.03339	0.00001322
12							0.000668	0.00000632
13							0.03777	0.00001459
14							0.02976	0.00000925
15							0.02974	0.00000621
16							0.02571	0.00000506
CPU tiem	CPU Time Steepest descent Method = 0.03120		CPU Time Levenberg_Marquard Method = 0.062		Cpu Time BFGS Method = 0.0780005		CPU Time DEP Method = 0.0780005	
Norm of F	Norm of f 0.0023		Norm of f 7.853335950309482e-08		Norm of f 7.853335950309482e-08		Norm of f 5.0636e-06	

Table 27: Norm errors of Step Size and Norm of function of Steepest descent, Levenberg–Marq, BFGS and DEP Methods.

CHAPTER SIX

6. CONCLUSIONS

In general, in this thesis we used different techniques or method to solve system of nonlinear algebraic equation. The method is considered in different aspect such as with respect to their number of iteration, norm errors and efficiency of the proposed method.

We have seen Newton method and Broyden's method. The two methods are the most known method in finding the root of SNAE. They are linearly convergent. So in order to fix the borderline and to converge globally, we use safeguarded and Damped method for both method to increaser their efficiency. However the convergence of newton's and Broyden's methods are highly depended on the initial approximation. If they have bad initial guess the method may not convergent. So a good initial guess is a very important for the two methods.

In addition to increase the range of initial value and the efficiency of convergence, we present a new implementation method which is NHAM. It is a powerful analytical method by combine newton method with RK methods to find slop. The NHAM algorithms are very effective and which investigate highly accurate result in less number of iterations by adding the parameter λ, it control convergent rate as well as ensure to get the initial guess.

Finally we investigate optimization method to solve SNAE. In optimization method we use the algorithm of unconstrained methods to get the minimum objective function value by search direction method. The methods can converge if the initial point far or not and converge globally.

Reference

[1] Remain, C. (2013). Numerical Methods for solving systems of nonlinear equation. *Department of Mathematical Sciences*

[2] Woollett, E. L. (2009, January 29). Solving equation ch.4. *Maxima by example*.

[3]Mustafa Mamat, K. m. (2014). Trapezoidal Broyden's method for solving systems of Nonliner equation. 251-260,8(6).

[4] Roose, A., *Test Examples of Systems of Nonlinear Equations*: Version 3- 901990: Estonian Software and Computer Service Company

[5] Yang, W. Y. (2005). *Applied Numerical method using MATLAB* . Korea: Jcohn Wiley and Sons.

[6] V.Dukkipat, R. (2010). *Numerical Methods.* USA: New age international publishers

[7] Okorie Charity Ebelechukwu, B. O. (2018). Comparison of Some Iterative Methods of Solving Nonlinear Equation. *Interntional Journal of Theoretical & Appliede Mathematics, 4*, 22-28.

[8]CHYN, W. E. (2014). NUMERICAL METHODS FOR NONLINEAR SYSTEMS OF EQUATIONS. *University Technology of Malaysia*.

[9]Gerardo C.Velez-Lopez, L. H.-M. (2019). A Tool to Solve Nonlinear Algebraic equations Systems. *International Conferenceon Electrical Engineering*, 11-13.

[10]Atluri, C.-s. L. (2011). An iterative Algorithm for Solving a System of Nonlinear Algebraic Equation, using system of ODEs with an optimization. *Department ofCivil Engineering, 73*, 395- 431.

[11]Ypma, T. J. (1995, December). Historical Development of the Newton-Raphson Method. *Societybfor Industrial and Applied mathematics, 37*, 531-551.

[12]Nusrat. (2012, january). Some Derivative Free Iterative Methods for Solving Nonlinear Equations. *Academical Research International, 2*.

[13] (Aliyu Bhar Kisabo). Newton's Method for Solving Non-Linear System of Algebraic Equations (NLSAEs) *American Journal of Mathematical and Computer Modelling.* Vol. 2, No. 4, 2017, pp. 117-131

[14] Gyang, K. N. (2018, January 29). Numerical Solution of Nonlinear Systems of Algebraic Equations. *International Journal of Data Science and Analysis, 14*, 20-23.

[15] Alam, M. S. (2018). Iterative Methods to Solve Systems of Nonlinear. *Masters Theses & Secialist Projects*.

[16]Rapajic, Z. L. (2001). Convergenr Accelerationof A General Newton Method For Systems Of Nonlinear Equation. *Scientiae Mathematicae Japonicae Online, 4*, 835-841.

[17] Remain, C. (2013). Numerical Methods for solving systems of nonlinear equation. *Department of Mathematical Sciences*.

[18]1Kabiru Muhammad, M. M. (2013). A Broyden's-like Method for Solving Systems of Nonlinear Equations. *World Applied Sciences Journal*, 168-173.

[19] Mohammad H.Al-Towaiq*, Y. S. (2017). Two improved classes of Broyden's methods for solving nonlinear systems of equations. *Journal of Mathematics and Computer Science*, 22-31.

[20] A.Ramli, M.L. Abdullah, and M. Mamat, *Broyden's method for solving fuzzy nonlinear equations.* Advances in Fuzzy Systems, 2010. Art ID 763270, 6 pages.

[21] Hour, M. A.-T. (2016). Two Improved Methods Based on Broyden's Newton Methods for the solution of NONLINEAR SYSTEMS OF EQUATION. *Jordan University of Science and Technology*.

[22]J.E. Dennis, J. a. (2017). Quasi-Newton Methods, Motivation and Theory. *Society for Industrial Applied Mathematics*, 1-45.

[23]Satya N. Atluri*, C.-S. L.-L. (2009). AMODIFIED NEWTON METHOD FOR SOLVING NONLINEAR ALGEBRAIC EQUATION. *Journal of Marine Science and Technology, 17*, 238-247.

[24]Ejiejl, A. A. (2016). A Mpdified Newton's Method for solving Nonlinear Programing Prolems. *University of llorin*, 2-15.

[25]Rafat Alshorman, S. A.-S. (2013). Automatic Iterative Mehods for the Muliivariate SNAE. *World Academy of science, Engineering and Technology, 7*.

[26]M.M. Hosseini*, S. H. (2010). Improving homotopy analysis method for system of nonlinear algebri equations. *Journa of Advanced Research in Applied nathematicsl, 2*(4), 22-30.

[27]Nor Hanim Abd. Rahman, A. I. (2011, December). Newton Homotopy Solution for Nonlinear Equations using Maple14. *Journal of Science and Technology, 3*.

[28]Vazquez-Leal, H. (2013, June 17). Generalized homotopy method for solving nonlinear differental equations. *Computational Applied Mathmatics*.

[29]S. Abbasbandy a, *. Y. (2007). Newton-homotopy analysis method for nonlinear equations. *Applied Mathematics and Computation*, 1794-1800.

[30]Talib Hashim Hasan, M. S. (2011). Solving Nonlinear Algebraic Problem Using Newton Homotopy Differential Equation. . *Australian Journal if Basic and Applied Sciences*, 56-59

[31]1M. S. H. CHOWDHURY, 2. H. (2017). SOLVING LINEAR AND NON-LINEAR STIFF SYSTEM OF ODE BY MULTI STAGE HOMOTOPY PERTURBATION METHOD. *International Journal of Management and Applied Science, ISSN, 3*, 2394-7926.

[32].Izadias*, R. A. (2014). A New Approach for Solving Nonlinear System of Equation using Newton Method and HAM. *Australian Journal of Basic and Applied Science, 4*, 57-72.

[33]Baba, I. A. (2014). Unconstrained Optimization and NSEs Review and Analysis Using Excel. 1-120.

[34]M.Al-Baali*, L. G. (2012). ON THE BEHAVIOR OF DAMPED QUASI-NEWTON METHODS FOR UNCONSTRAINED OPTIMIZATION. *Iranian Journal of Operations Research, 3*, 1-10.

[35]Sona Taheri, M. M. (2012). SOLVING SYSTEM OF NONLINEAR EQUATION UING A GLOBALLY CONSTRACTION OPTIMIZATION ALORITHM. *Global Jorunal of Technology & Optimization, 3*.

[36]Mariola Jureccko, S. D. (2016). Solving System of Nonlinear Equations With The Use of Optimazation methods in Problem Related to the Wheel-Reil Contact. *Journal of Applied Mathematics & comutational Mechanicics*, 53-64.

[37]Sterck, H. D. (2012). Steepest descent preconditioning for nonlinear GMREs Optimization. *Numerical Linear Algebra With Applications*, 1-19.

[38]Trogdon, S. O. (2012). Nonlinear steepest descent and the numerical solution of Riemann-Hilbert problem. *Schoolof Mathematics and Statistics*, 3-27

[39]MOHD. ASRUL HERY BIN IBRAHIM, M. M. (2014). BFGS Method : A New Search Direction. *Universiti Sutan Zainal Abidin, 43*, 1591-1597.

[40]Xiangrong Li, X. W. (2014). A Limited Memory BFGS Method for Solving Large-Scale Symmetric Nonlinear Equation. *Hindawi Publishing Corporation*, 9..

[41]Waziri, M. Y. (2017). A Globally Convergent Hyperplane- BFGS for SNEs. *Bayero Jornal of pure and applied Sciences*, 615-622.

[42]Artacho, M. A. (2010). Local Convergence of the Levenberg-Marquardt method under Holder metric subregularity. *Mathematics Subject equation*, 1-30.

[43]Bergou, E. H. (924-951). Levenberg-Marquardt Method Based on Probabilistic Gradient Model and Inexact Subproblem Solution, with Applicatio to Data Assimilation. *Journal on Uncertality Quantification, 4*, 2016.

[44]Rafat Alshorman, S. A.-S. (2013). Automatic Iterative Mehods for the Muliivariate SNAE. *World Academy of science, Engineering and Technology*, *7*.

[45]Ros, A. (2010). Introduction to Numerical Analysis. 3-5.

[46]Kaw, A. (2009, December 23). Newton-Raphson method of solving Nonlinear equation.

[47]Reference Chapter 03.04 Newton-Raphson Method of Solving a Nonlinear Equation

[48] prof note hand out.

[50]Mariola Jureccko, S. D. (2016). Solving System of Nonlinear Equations With The Use of Optimazation methods in Problem Related to the Wheel-Reil Contact. *Journal of Applied Mathematics & comutational Mechanicics*, 53-64.

[51]B.Ghanbary, J. a. (2008). A New Techniqu For Solving Systems Of Nonlinear Equation. *Applied mathematical sciences, 2*, 2699-2703

[52]BEERS, K. J. (2007). Numerical Methods for Chemical Engineering(book). USA: USA by Cambride University Press.

[53]Faires, R. L. (2011). *Numerical analysis.* USA: Young stown stste university.

[54] VATANSEVER*, M. H. (2016). Differential Equation Solver Simulatur for Runge-Kutta Method. *University Journal of the Faculity of Engineering, 21*, 146-154.

[56]Baba, I. A. (2014). Unconstrained Optimization and NSEs Review and Analysis Using Excel. 23-33.

[57]Franics, T. a. (2015). OPtimization Algorithm and Appliction. *Indian Space Research organization*, 55-73.

[58]I.A.Osinuga, S. Y. (2017). CONSTRUCTIO OF A BRUYDEN-LIKE METHOID FOR NONLINEAR SYSTEMS OF EQUATIONS. *anale. Seria Informatica, XV*.

[59] C. G. Broyden, A class of methods for solving nonlinear simultaneous equations, 577–593.: Math. Comp., 19, 1965.

[60] Y. Abu-Hour, "Improved Classes of Broyden methods for solving a nonlinear systems of equations," Jordan University of Science and Technology, 2016.

[61] S.J. Liao, "The proposed homotopy analysis technique for the solution of nonlinear problems," Ph.D. Thesis, p. Shanghai Jiao Tong University, 1992.

[62] Jafri, M.D., Suleiman M., Majid, Z.A., Ibrahim Z.B., "Solving directly two point boundary value problems using direct multistep method. : 723-728.," Sains Malaysiana, vol. 38(5), pp. 723-728, 2009.

[63] W. C. Davidon, "Variable metric methods for minimization," SIAM J. Optim, vol. 1, pp. 1-17, 1991.

[64] C. G. Broyden, "The convergence of a class of double rank minimization algorithms: , 6th ed.: "J. Inst. Math, no. The new algorithm. pp. 222–231, , April 1970.

[65] R. Fletcher, "A new approach to variable metric algorithms," , 13th ed.: Computer J, pp. 317–322, 1970.

[66] S. M Robinson, Normal Maps Induced by Linear Transformations.:.: Math. Oper., Res. 17 691-714. oz.au., 1992.